
Rhythmyx

Implementing Content Editors

Version 5.7

Copyright and Licensing Statement

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

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

Copyright © 1999-2005 Percussion Software.
All rights reserved

Licenses and Source Code

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

Netscape Public License

Apache Software License

IBM Public License

Lesser GNU Public License

Other Copyrights

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

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

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

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

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

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

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

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

This product includes software developed by CDS Networks, Inc.

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

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

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

AuthorIT™ is a trademark of Optical Systems Corporation Ltd.

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

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

Percussion Software

600 Unicorn Park Drive

Woburn, MA 01801 U.S.A

781.438.9900

Internet E-Mail: technical_support@percussion.com

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

Contents

Introduction to Content Editors 5

Content Editor Definition Files.....	6
Local Definition Structure.....	6
Shared Definition Structure.....	10
Understanding the System Definition.....	12

Maintaining Content Editors 13

Content Editor Maintenance Dialogs.....	14
Content Editor Properties Dialog.....	14
Content Editor Settings Dialog.....	15
Rule Editor.....	20
Content Editor Maintenance.....	21
Creating a Content Editor Resource.....	21
Activating a Content Editor Resource for Edit.....	23
Content Type.....	23
Attaching Tables to a Content Editor.....	23
Maintaining Content Editor Settings.....	24

Maintaining Content Editor Fields 31

Field Maintenance Dialogs.....	32
New Field Properties Dialog.....	33
Field Properties Dialog.....	34
Child Editor Field Properties Dialog.....	35
Creating a New Field.....	36
Quick Field Creation.....	37
Field Maintenance.....	38
Editing a Field.....	38
Deleting a Field.....	38
Child Editors.....	38

Field Data 39

Source of Field.....	40
Type of Field.....	41
Show in Preview.....	42
Name of Field.....	43
Label of Field.....	44
Mnemonic.....	45
Error Label.....	46
Control.....	47
Adding Parameters to a Control.....	47
Specifying a URL for a Control.....	49
Managing Dependencies in a Control.....	51
Adding Choices to a Control.....	52

Data Type	55
Occurrence of Field	56
Format of Field	57
Default Value of Field	58
Treating Text as Binary	59
Clearing Binary Data from a Field.....	60
Show in Summary.....	61
Search Properties	62

Content Editor Field Controls 63

Content Editor Control Dialogs	64
Display Control Properties Dialog.....	64
URL Request Properties Dialog	68
Create Choice Lookup Request Dialog	71
Configuring a Content Editor Control	72

Adding Data Validation 75

Field-Level Validation.....	76
Transition-Dependent Field-Level Validation.....	78
Item-Level Validation.....	80
Sample Item Validation Exit	81
Sample Error Page.....	83

Visibility and Read-only Rules 85

Visibility Rules	86
Example: Hiding the Workflow Field in the Content Editor System Definition.....	86
Read-only Rules.....	89
Setting a Field in a Content Editor to Read-Only	90

Text Extraction 91

Implementing Text Extraction in Rhythmyx	92
Uploading External Binary Files into Rhythmyx.....	93
Creating a Content Editor that Extracts Text.....	94
Displaying Extracted Text in a Content Editor	97
sys_TextExtraction	98
Continuous Conversion Example	100
Migration Example	108

Customizing the ArticleWord Content Editor 111

How Word-based Content Editors Work.....	113
How to Create a Word-based Content Editor	114
Creating the Word Template File	116
Modifying the Style Sheet for Parsing Fields.....	116
Modifying the Content Assembler to Display Custom Word-based Content Editor Fields	118

Installing New Features of Rhythmyx Word Connector 119

Moving Rhythmyx Word Connector Files to the Correct Directory.....120
 Setting the Address in the Word Template Files122
 Processing Related Links.....124
 Copying the Template File to the Client Word Application126
 Updating the sys_FileWord Content Editor Control.....127

Appendices 129

Implementing a Content Editor Manually131
 Registering a New Content Type.....131
 Creating the Content Editor Definition.....131
 Updating the Content Type Registration143
 Creating the Content Editor Application144
 Validating the Content Editor Definition.....145
 Content Editor Control Reference147
 Control Header147
 Control Template Standards148
 Control Events148
 Standard Rhythmyx Controls149
 Creating an Internal Lookup Query195
 Content Editor XML Reference.....197
 Basic Objects197
 Content Editor Local Def250
 Content Editor Shared Def254

Index 257

CHAPTER 1

Introduction to Content Editors

Rhythmyx Content Editors are the tools Business Users employ to create, modify, and preview content. Content Editors display the fields defined for the Content Type with which it is associated, and make fields eligible for modification available to the user.

When a user takes an action, such as submitting a new item or editing an item, Rhythmyx passes a URL to the server with an attached command parameter indicating the type of action. Available actions include:

- New (displays content editor with default values)
- Edit (displays existing content item)
- Preview (displays a preview of the formatted content item)
- Modify (updates database with new information for the content item)
- Workflow (performs a workflow action on the content item)
- Clone (creates a duplicate of the content item)

The server uses the command parameter to determine what action to perform on the content item.

Rhythmyx then directs the user to a new page or updates the display. Business rules may affect the processing of the data before it is updated to the database or may influence the display formatting of updated data.

Content Editor Definition Files

Rhythmyx Content Editors are composed of three layers of data, each controlled by one or more XML definition files. The base level is system data, which is provided by Percussion Software. System data is common to all Content Editors, and is controlled by a single XML file for the entire system. This file is overwritten when Rhythmyx is upgraded; therefore, it should not be customized.

The intermediate level of data is shared data. Shared data is data that is shared by two or more Content Editors, but is not common to all Content Editors. A Rhythmyx content management system can include any number of shared field XML definition files; most implementations include at least one.

The final level is local data, which is data specific to each individual editor. Each Content Editor has its own XML file, which defines fields specific to the individual Content Editor, the shared fields that are included in the Content Editor, and any local overrides to the default names of system and shared field labels.

The following sections explain the details of each level of Content Editor definition. For detailed descriptions of the XML elements that comprise these files, see the appendix "Content Editor XML Reference".

Local Definition Structure

Each content editor is defined by a unique XML document that defines:

- the fields in the Content Editor;
- the individual behavior of each field;
- the collective behavior of all the fields in the Content Editor;
- and the functionality and look and feel of the editor itself.

The root node of the content editor is the `PSXContentEditor` element.

PSXContentEditor	
contentType	1
enableRelatedC...	yes
objectType	1
workflowId	1
PSXDataSet id=961	
SectionLinkList	
PSXValidationRules maxErrorsToStop=10	
PSXInputTransl...	
PSXOutputTran...	

NOTE: All illustrations use XML Spy's Grid View.

The field level behavior is defined in the `PSXDataSet`, which is the overall container for data definitions in the content editor. Nested within this element is the `PSXContentEditorPipe`, which is the direct container for the field-level definitions.

PSXDataSet	
id	961
name	ArticleContentEditor5488
description	
transactionType	none
PSXContentEditorPipe	id=0
PSXPageDataTank	id=960
PSXRequestor	directDataStream=no id=0

The `PSXContentEditorPipe` has two children, the `PSXContainerLocator` and the `PSXContentEditorMapper`. The `PSXContainerLocator` stores the definitions of the backend database tables where the data for the fields in the content editor are stored.

PSXContentEditorPipe	
id	0
name	cePipe
description	This is the ContentEditorLocaDef for the ArticleWord ContentType
InputDataExits	
PSXContainerLocator	
PSXContentEditorMapper	

The `PSXContentEditorMapper` contains one or more `PSXFieldSet` elements, each of which defines a set of fields specific to the content editor. The fields are defined using the `PSXField` element.

The `PSXUIDefinition` child of the `PSXContentEditorMapper` defines the user-interface controls for the fields in the content editor. The `PSXUIDefinition` includes one `PSXDisplayMapper` element for each `PSXFieldSet` element, and one `PSXDisplayMapping` element for each `PSXField` element, which defines the user interface for that field.

PSXContentEditorMapper			
SystemFieldExcludes			
PSXFieldSet			
name	main		
repeatability	zeroOrMore		
supportsSeque...	yes		
type	parent		
PSXField (9)			
	forceBinary	modificationType	name
1	no	user	bodycontent
2	no	user	defaultvariantid
3	no	user	authorname
4	no	user	sys_suffix
5	no	user	keywords
6	no	user	sys_pathname
7	no	user	abstractcontent
8	no	user	displaytitle
9	no	user	bodyformat
PSXUIDefinition			
PSXDisplayMapper			
fieldSetRef	main		
id	0		
PSXDisplayMapping (17)			
	FieldRef	PSXUISet	
1	sys_suffix	PSXUISet	
2	sys_pathname	PSXUISet	
3	sys_title	PSXUISet	
4	displaytitle	PSXUISet	
5	keywords	PSXUISet	
6	authorname	PSXUISet	
7	sys_contentstartdate		
8	sys_contentexpirydate	PSXUISet	
9	sys_reminderdate	PSXUISet	
10	abstractcontent	PSXUISet	
11	bodycontent	PSXUISet	
12	bodyformat	PSXUISet	
13	defaultvariantid	PSXUISet	
14	sys_communityid		
15	sys_lang		
16	sys_currentview		
17	sys_workflowid		

Note that the value of the `fieldSetRef` attribute of the `PSXDisplayMapper` must match the name attribute of the `PSXFieldSet` for which you are defining the user interface. Note also that `PSXDisplayMapping` includes a number of fields not included in the `PSXFieldSet`. These fields are derived from the system definition and any shared fields included in the `PSXFieldSet`. System fields are included in every Content Editor automatically, but you can specify individual system fields to exclude in the `SystemFieldExcludes` child of the `PSXContentEditorMapper`. Shared Fields are only included if you specify the shared field sets in a `PSXSharedFieldIncludes` element. Specify the shared field group to include in a `SharedFieldGroupName` element. If you want to exclude specific fields from a `SharedFieldGroup`, specify those fields in a `SharedFieldExcludes` child of the `SharedFieldGroup`.

The remaining immediate children of the `PSXContentEditor` node fall into two categories: editor behavior definition and item-level definitions.

Editor behavior definition specifies the functionality and the look and feel of the content editor. The `PSXCommandHandlerStylesheet` element defines the stylesheets that format the content editor page. The `PSXApplicationFlow` element determines the page to which the user should be directed after each non-query request to the server. These nodes are optional; any stylesheets or application flows defined here are overrides to the default stylesheets and application flows defined in the system definition. The `SectionLinkList` element defines links to other Rhythmyx components included in the content editor, such as lookups for Slot content and Variants for previews.

Item-level definitions define the processing of the data in the fields of the content editor as a whole. `PSXValidationRules` define the processing to ensure the data in the content editor meets the requirements specified, for example, that date fields are numeric and in the correct format. `PSXOutputTranslations` converts data from one form to another when the user requests the data from the database. `PSXInputTranslation` convert data from one form to another when the user updates data to the database.

Shared Definition Structure

Fields that are shared by two or more content editors are defined in one or more separate XML definition files that conform to the `sys_ContentEditorShared` DTD. Most systems include at least one shared definition file. A sample Shared definition file is installed with Rhythmyx in `<Rhythmyxroot>/Samples/SharedDef`. A SQL script to create a table to store data for the fields in the sample shared definition is also in this directory.

The shared field XML definition files share many elements with the local definition XML definition files, but have a slightly different structure because the sets of fields are shared. The root element of all shared field XML definition files is the `PSXContentEditorSharedDef`. This node contains one or more `PSXSharedFieldGroup` children. Each `PSXSharedFieldGroup` element defines a set of fields and each stores its data in a table specific to that shared field group. (Note that the you must create the table before implementing a Content Editor that uses the shared field group. The table must include columns for the Content ID and Revision ID; when joining this table to a Content Editor application, you must define joins to `CONTENTSTATUS.CONTENTID` and `CONTENTSTATUS.CURRENTREVISION`.)

PSXContentEditorSharedDef	
Comment	Generic Shared Field Group
PSXSharedFieldGroup name=shared	
Comment	Image Shared Field Group
PSXSharedFieldGroup name=sharedimage	
Comment	Binary Shared Field Group
PSXSharedFieldGroup name=sharedbinary	
Comment	Category Set Shared Field Group
PSXSharedFieldGroup name=sharedcategory	

The value of the name attribute of each `PSXSharedFieldGroup` element in the system must be unique. Shared field groups are included in Content Editors by specifying the name of the shared field group to include.

Each `PSXSharedFieldGroup` element has three children:

- The `PSXContainerLocator` node specifies the table used to store the data for the shared definition.
- The `PSXFieldSet` defines the set of fields in the field set. The value of the name attribute of the `PSXFieldSet` element must match the value of the name attribute of the `PSXSharedFieldGroup` and also of the `fieldSetRef` attribute of the `PSXDisplayMapper` node of the `PSXUIDefinition`. The individual fields in the field

- set are defined using the PSXField element.
- The PSXUIDefinition defines the user interface for the fields in the field set. The PSXUIDefinition node requires the child PSXDisplayMapper. The value of the fieldSetRef attribute of the PSXDisplayMapper element must match the value of the name attribute of the PSXFieldSet element and the PSXSharedFieldGroup element. For each PSXField child defined in the PSXFieldSet element, the PSXDisplayMapper must include a corresponding PSXDisplayMapping element that defines the default user interface for that field.

The screenshot shows a hierarchical configuration tree for a shared field group. The root node is **PSXSharedFieldGroup** with a **name** attribute set to `shared`. It contains a **PSXContainerLocator** child, which in turn contains a **PSXFieldSet**. The **PSXFieldSet** has attributes: **name** (shared), **repeatability** (oneOrMore), **supportsSe...** (no), and **type** (multiPropertySimpleChild). It contains seven **PSXField** elements, each with a **forceBinary** attribute set to `no` and a **modificationType** attribute set to `user`. Below the field set is a **PSXUIDefinition** node, which contains a **PSXDisplayMapper**. The **PSXDisplayMapper** has attributes: **fieldSetRef** (shared) and **id** (5). It contains seven **PSXDisplayMapping** elements, each with a **FieldRef** attribute pointing to a specific field name: `filename`, `displaytitle`, `keywords`, `description`, `callout`, `body`, and `default_variantid`.

A shared field group can include field rules that define the processing of the data in the shared field group as a whole. The **PSXValidationRules** child defines the rules for ensuring that the data in the field meets the requirements of the field, for example, that dates are all numeric and in the correct format. **PSXOutputTranslations** converts data from one form to another when the user requests the data from the database. **PSXInputTranslation** convert data from one form to another when the user updates data to the database.

Understanding the System Definition

The system definition XML file defines fields common to all editors. This file contains fields defined by Percussion Software, and is overwritten when the system is upgraded. It should never be customized at the installation site.

The system definition XML specifies the following fields:

Start Date: Determines the date the content item is eligible to be published to an active site.

Expiration Date: Determines the date the content will be removed from all active sites.

Reminder Date: Defines a date on which a reminder can be generated to Business Users.

System Title: Defines the text displayed in the Title bar for the content item when it is published or previewed.

Pub Date: Displays the date the content item was published.

Path Name: Determines the path to the directory where the file associated with the content item is stored.

Suffix: Defines the extension of the file (for example, .pdf or .ppt).

Community ID: Specifies the Community with which a Content Item is associated.

Workflow ID: Specifies the Workflow with which a Content Item is associated.

CHAPTER 2

Maintaining Content Editors

A Content Editor is a unique Rhythmyx resource that runs from an XML file that conforms to the `sys_ContentEditorLocalDef` DTD. The Rhythmyx Workbench includes a special dialog, the Content Editor Properties dialog, to maintain Content Editors.

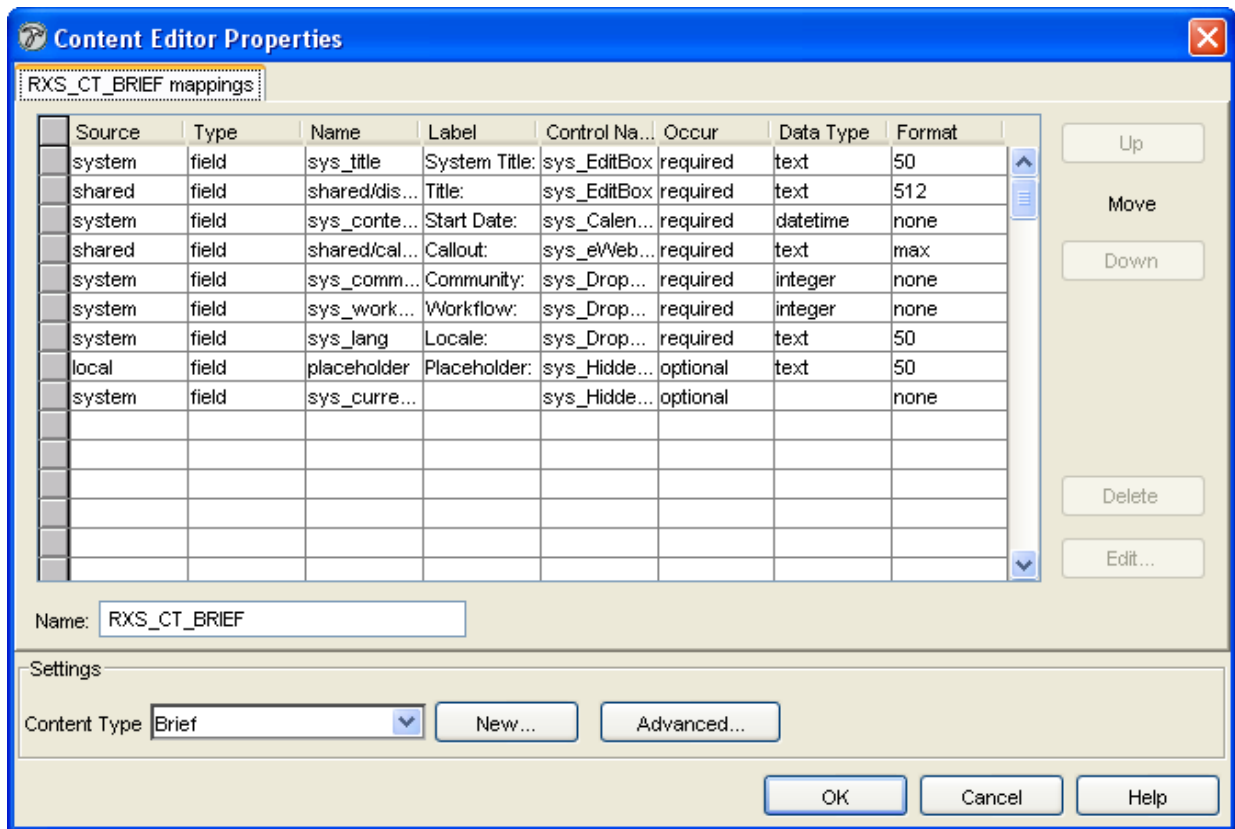


Figure 1: Content Editor Properties Dialog

You can also maintain a Content Editor XML manually, but this is recommended only for advanced users or to access features that are not currently available in the Content Editor Properties dialog.

Content Editor Maintenance Dialogs

Rhythmyx provides two dialogs to maintain editor-level data. The Content Editor Properties dialog is the main interface for maintaining Content Editors and their data. A variety of additional properties are available on the Content Editor Settings dialog.

Content Editor Properties Dialog

The Content Editor Properties dialog is the central interface for maintaining Content Editors. It provides options for maintaining data about the Content Editor as a whole. It also lists the fields available in the Content Editor and provides the ability to add or delete fields, as well as the ability to edit a limited set of data for the fields on the Content Editor.

Rhythmyx displays the Content Editor Properties dialog when you double-click on a Content Editor



resource:

Source	Type	Name	Label	Control Na...	Occur	Data Type	Format
system	field	sys_title	System Title:	sys_EditBox	required	text	50
shared	field	shared/dis...	Title:	sys_EditBox	required	text	512
system	field	sys_conte...	Start Date:	sys_Calen...	required	datetime	none
shared	field	shared/cal...	Callout:	sys_e/Web...	required	text	max
system	field	sys_comm...	Community:	sys_Drop...	required	integer	none
system	field	sys_work...	Workflow:	sys_Drop...	required	integer	none
system	field	sys_lang	Locale:	sys_Drop...	required	text	50
local	field	placeholder	Placeholder:	sys_Hidde...	optional	text	50
system	field	sys_curre...		sys_Hidde...	optional		none

Name:

Settings

Content Type:

Figure 2: Content Editor Properties Dialog

For details about the columns in the main table, see the appropriate topic in the chapter "Content Editor Field Data".

The Content Editor Properties dialog displays one tab for the main Content Editor, and an additional tab for each child editor (Detail Editor).

Field Descriptions

Name Required. The name of the Content Editor you are editing.

Content Type Drop list. The Content Type for which you are defining the Content Editor. To define a new Content Type, click the [New...] button.

Content Editor Settings Dialog

Use the Content Editor Settings dialog to specify settings for the Content Editor as a whole including:

- support for related content
- support for search
- input and output translation
- item-level validation

To access the Content Editor Settings dialog, on the Content Editor Properties dialog, click [**Advanced**].

The Content Editor Settings dialog consists of five tabs:

- General
- Workflow
- Input Translations
- Output Translations
- Item Validation

Content Editor Settings General Tab

Use the General tab of the Content Editor Settings dialog to define general settings for the Content Editor. Settings available on this this tab include:

- support for related content, and
- support for searches of Content Editor fields.

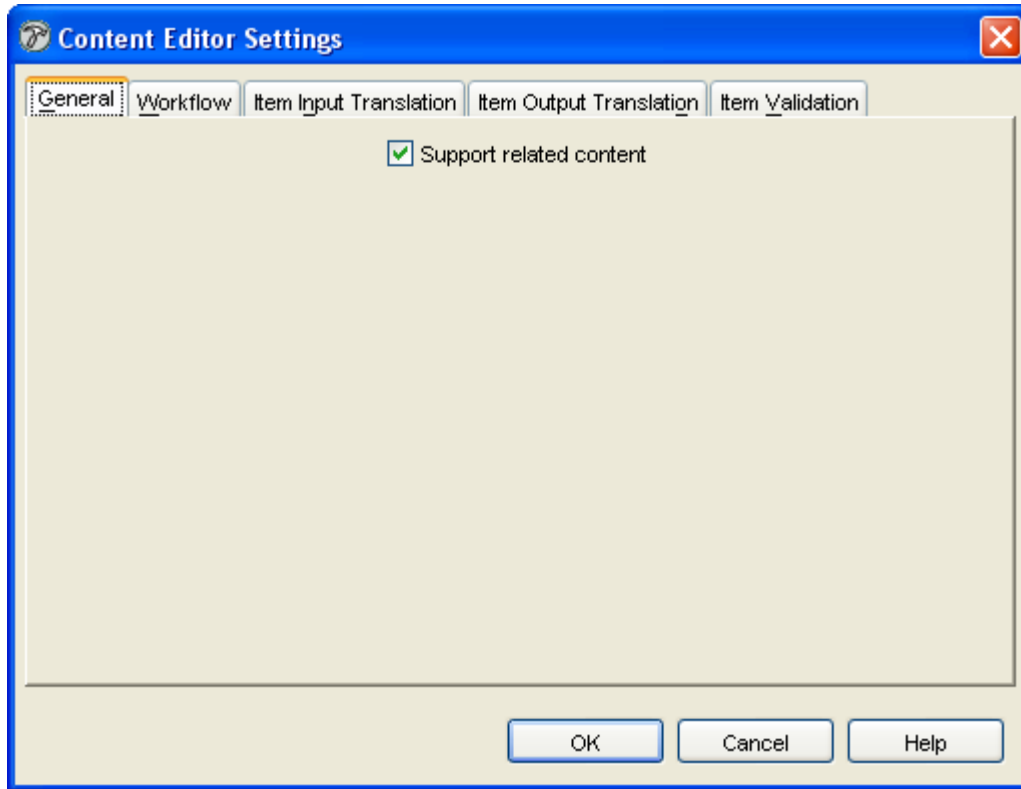


Figure 3: Content Editor Settings General Tab

The **Allow fields in this Content Editor to be searched** box is only available if you have installed the Rhythmyx full-text search engine. If this box is checked, fields in the Content Editor are eligible to be indexed for full-text search, and the **Allow this field to be searched** checkbox is checked for each field in the Content Editor by default, making each field eligible to be searched. You can uncheck the **Allow this field to be searched** checkbox for each field in the Content Editor individually.

If the **Allow fields in this Content Editor to be searched** box is unchecked, none of the fields in the Content Editor are eligible to be searched, and the **Allow this field to be searched** checkbox is unavailable for all fields.

If the **Support related content** box is checked, you can use Active Assembly to add related Content Items to the Content Items edited using this Content Editor. If this box unchecked, you cannot add related content to Content Items edited using this Content Editor.

Content Editor Settings Workflow Tab

Use the Workflow tab of the Content Editor Settings dialog to specify the default Workflow for Content Items created using the Content Editor, and any other Workflows that might be available for the Content Item.

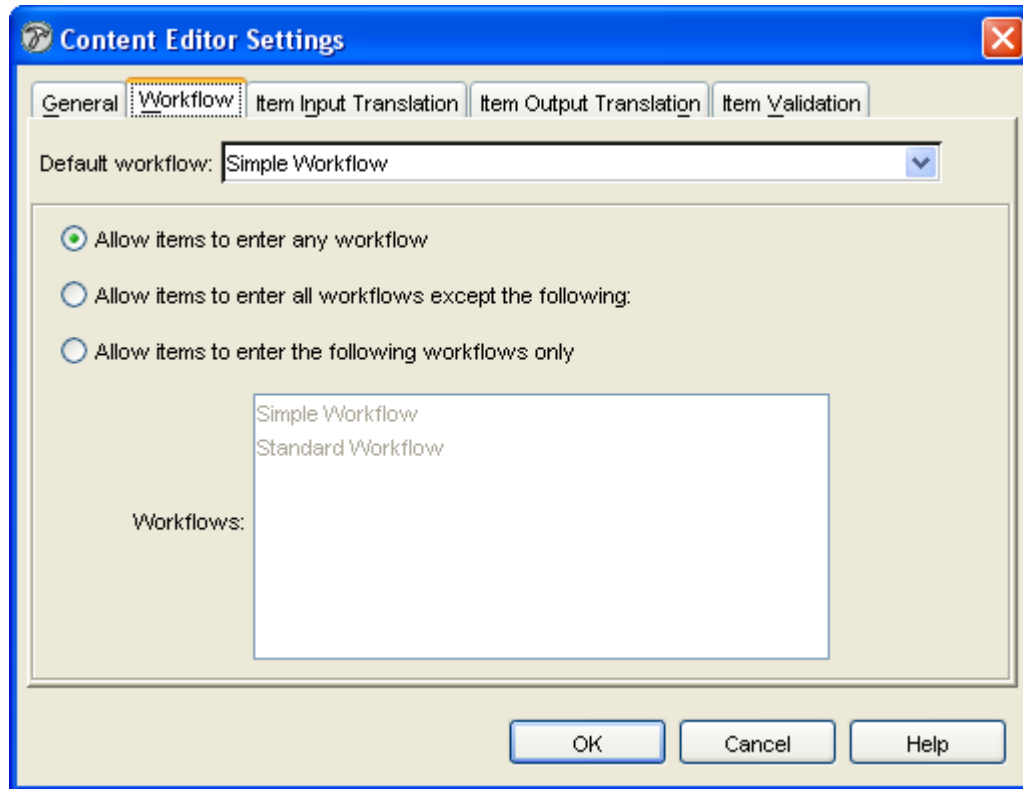


Figure 4: Content Editor Settings Workflow Tab

Field Descriptions

Default Workflow Drop list. Specifies the Workflow new Content Items enter by default when no Workflow has been specified.

Allow items to enter any workflow Radio button. When selected, Content Items created using this Content Editor can enter any Workflow defined in the system.

Allow items to enter all workflows except the following Radio button. When selected, Content Items created using this Content Editor can enter any Workflow defined in the system except those selected in the Workflows box.

Allow items to enter the following workflows only Radio button. When selected, Content Items created using this Content Editor can enter only the Workflows selected in the Workflows box.

Workflows Text box. List of Workflows in the system. Used to specify the Workflows available or disallowed for Content Items created using this Content Editor. You can selected multiple Workflows listed (use Control-click to select multiple workflows).

Content Editor Settings Content Item Input and Output Translations Tabs

Use the Content Item Input Translation tab to specify exits to convert data when uploading the data to the Repository (in other words, data input to the Repository). Use the Content Item Output Translation tab to specify exits to convert data when downloading it from the Repository (in other words, database output from the Repository).

Translations can convert data in a variety of ways, such as:

- combining the data in two Content Editor fields into one string to upload to a single database column;
- splitting the data from one Content Editor field into multiple strings to upload to different database columns;
- converting the encoding or format of the data;
- converting string data into a file or a file into string data.

Data conversions are performed by exits; input translation is performed by pre-exits and output translation is performed by post-exits. You can specify conditions that activate the extension to convert the Content Item data.

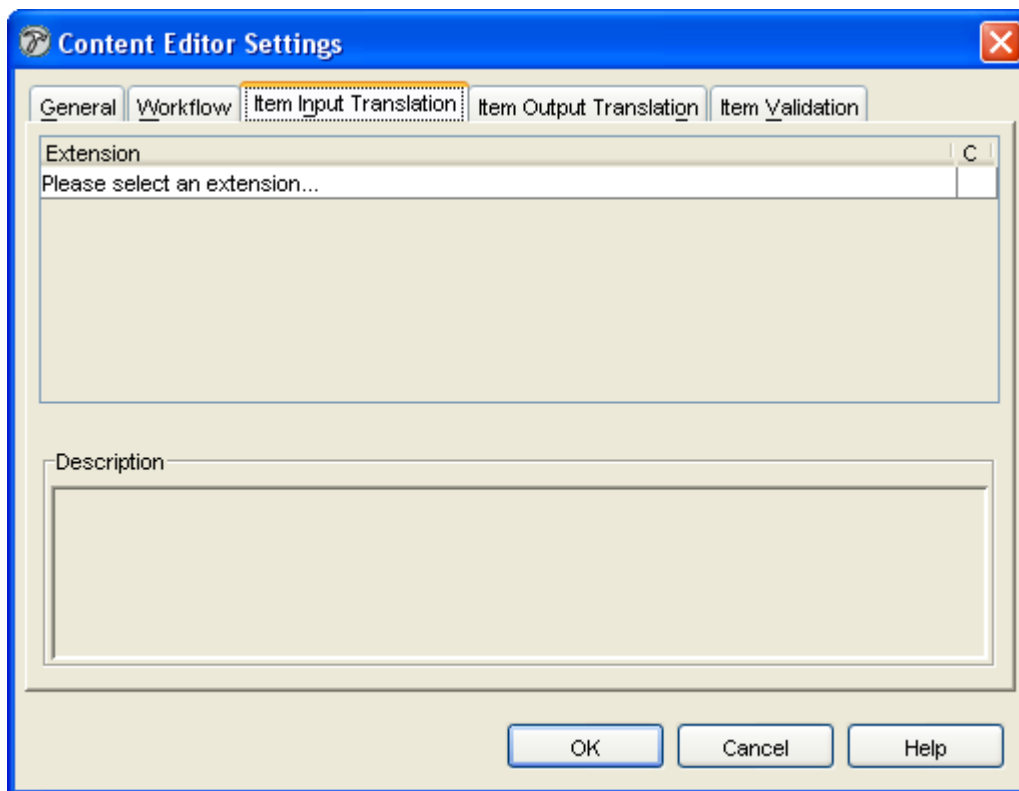


Figure 5: Content Editor Input Translations Tab

When you double-click in a row of the Extension table, the row is activated, and you can select an exit from the drop list. Options are all exits available in the system. The C column indicates whether you have specified a Condition to activate the translation exit.

Content Editor Settings Item Validation Tab

Use the Content Editor Settings Item Validation tab to specify post-exits to perform validation on the Content Item as a whole. Use item-level validation to compare the values in multiple fields, for example, such as when the values valid for one field depend on the value assigned to another field. (To validate that a specific field contained data and or that the data in the field was in the correct format, you would use field-level validation; for details see *Field-Level Validation* (on page 76)).

You can specify conditions for activating each validation post-exit. If you do not specify conditions for an exit, it is activated whenever you Transition or upload a the Content Item.

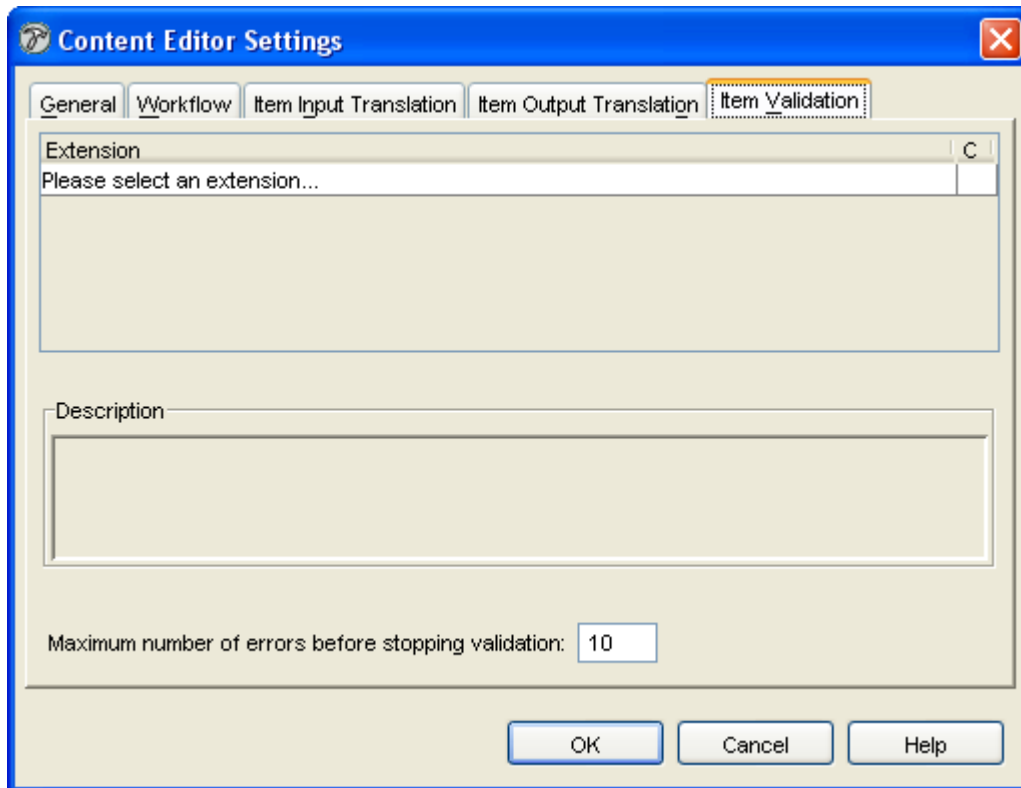


Figure 6: Content Editor Settings Item Validation Tab

When you click in a row in the Extension table, the row is activated and you can select an exit from the drop list. Options include all post-exits in the system.


The C column specifies whether a condition has been specified to activate that exit.

The **Maximum number of errors before stopping validation** field specifies the maximum number of errors generated during item validation before terminating validation and reporting the errors to the user.

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.

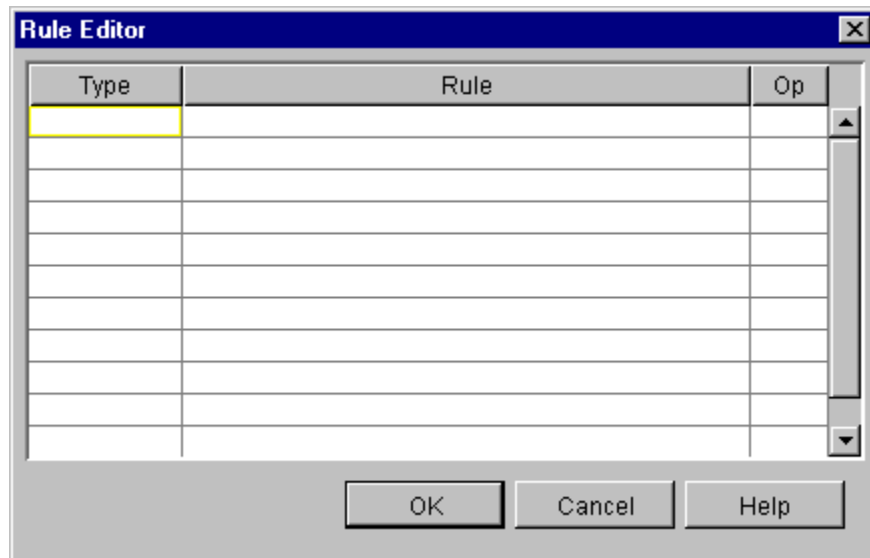


Figure 7: Rule Editor

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 be 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.

Content Editor Maintenance

This section describes:

- how to *create a Content Editor resource* (see "[Creating a Content Editor Resource](#)" on page 21);
- how to *activate a Content Editor for editing* (see "[Activating a Content Editor Resource for Edit](#)" on page 23);
- how to *define and maintain the Content Type for a Content Editor* (see "[Content Type](#)" on page 23);
- how to *attach tables to a Content Editor* (see "[Attaching Tables to a Content Editor](#)" on page 23); and
- how to *maintain settings for a Content Editor* (see "[Maintaining Content Editor Settings](#)" on page 24).

Creating a Content Editor Resource

You can create a content editor in the Rhythmyx workbench:

- from scratch;
- by defining a Content Editor XML file and dropping it into the Rhythmyx workbench; or
- by dropping the Content Editor database table into the Rhythmyx workbench.

Creating a Content Editor from Scratch

Each content editor you create must be based on a template. The template determines which content editor data is already defined. Rhythmyx includes the following default template:

- `sys_default.xml`: includes a small set of system fields:
 - `sys_title`
 - `sys_communityid`
 - `sys_lang`
 - `sys_currentview`
 - `sys_workflowid`

When you create a content editor from scratch, you are also defining the database table that will store the data for the content editor. Rhythmyx creates the database table when you save the content editor if you have defined the following data for the content editor.

- Name (becomes the name of the database table);
- at least one local field name (the name of each field becomes the name of the database table column that stores data for that field);
- a Data Type for each field you have defined (defines the data type of the database column); and
- for certain Data Types (such as *string*), a Data Format.

➤ **To create a content editor resource from scratch:**

- 1 In the Rhythmyx Workbench, start a new application.
- 2 In the Menu bar of the Rhythmyx Workbench, choose *Insert > Content Editor*.
Rhythmyx displays the Content Editor Templates dialog
- 3 Choose the template you want to use for this content editor.

Rhythmyx inserts the content editor into the active application.

Creating a Content Editor from an XML File

You can define an XML file manually, and then drop it into the workbench to create a content editor application. For more details, see *Implementing a Content Editor Manually* (on page 131).

NOTE: You must use a manually created XML file based on the `sys_ContentEditorLocalDef.dtd` file. You cannot use drag and drop with a template file. Template files are only intended to be used when adding a new Content Editor resource through the Menu bar.

Creating a Content Editor from a Database Table

If you have already defined a database table for a content type, you can create a content editor for that content type by dropping the table into the Rhythmyx workbench.

Note: You cannot create a Content Editor from a system table (tables Rhythmyx uses to operate). If you attempt to drop a system table, the pop-up menu does not include the option *Content Editor*.

To create a content editor from a database table:

- 1 Drag the database table into the application.
- 2 On the pop-up menu, choose *Content Editor*. (Note that if you dropped a system table, this option will not be available. Right-click to cancel the drop.)
- 3 Rhythmyx displays the Content Editor Templates dialog. Choose the template you want to use for the content editor.
- 4 Rhythmyx generates a content editor XML based on the template you chose and mapped to the table you dropped. This table becomes the primary content table for the content editor.
- 5 Edit the content editor and field properties.

Activating a Content Editor Resource for Edit

To activate a content editor resource for edit, double-click on it, or right-click on it and choose *Properties* from the pop-up menu. Rhythmyx will display the Content Editor Properties dialog.

Content Type

Each content editor must manage the data for a specific content type. In most cases, you will create a new content type when you create a new editor. In special circumstances, you can choose the content type for the editor from the Content Type drop list. To create a new content type:

- 1 On the Content Editor Properties dialog, click [New].
- 2 Rhythmyx displays the Create Content Type dialog.

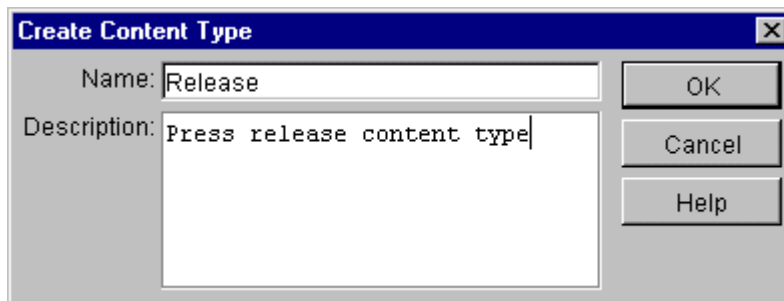


Figure 8: Create Content Type dialog

- 3 Enter the **Name** and optional **Description** of the new content type.
- 4 Click [OK].


Attaching Tables to a Content Editor


To add tables to a content editor, drag them from the data tab in the workbench and drop them on the content editor resource. Rhythmyx displays a pop-up menu with the available options.

Option	Comment
Primary editor content	This is the default selection if the editor does not yet have a primary table.
Multi-valued choice (such as check box group of drop list with multiple values)	This is the default choice if the table has a single, non-system column.
Child content with a 1 to many relationship to the parent	This is the default choice if no other choice is the default.
Child content with a 1 to 1 relationship to the parent	

Maintaining Content Editor Settings

To maintain settings for a Content Editor:


- 1 On the Content Editor Properties dialog, click the [**Advanced**] button.
Rhythmyx displays the Content Editor Settings dialog.
- 2 To specify general settings for the Content Editor:
 - a) Choose the General tab (this is the default tab for the dialog).
 - b) To enable searches on all fields of the Content Editor, check the **Allow fields in this content editor to be searched** checkbox. Note that you can disable search on specific individual fields in the Content Editor.
 - c) To allow Content Items edited with this Content Editor to create Active Assembly Relationships to other Content Items, check the Support Related Content checkbox.
- 3 To specify the Workflow settings for the Content Editor:
 - a) Choose the Workflow tab.
 - b) In the **Default Workflow** drop list, select the Workflow you want Content Items edited with this Content Editor to enter by default when no Workflow is specified.
 - c) If you want to allow Content Items to enter any Workflow defined in the system, select the **Allow items to enter any workflow** radio button.
 - d) If you want to allow Content Items to enter any Workflow except a specific set, select the **Allow items to enter all workflows except the following** radio button and select the Workflows you do not want available for the Content Items in the Workflows box. Use Control-click to select multiple Workflows.
 - e) If you want to allow Content Items to enter only a specific set of Workflows, select the **Allow items to enter the following workflows only** radio button and select the Workflows you want to be available for the Content Items in the Workflows box. Use Control-click to select multiple Workflows.
- 4 To add input translations (conversions of Content Item data uploaded to the database) to the Content Editor:
 - a) Choose the Item Input Translation tab.
 - b) Click in the first available row and select the exit you want to add as an input translation for the Content Editor.
Rhythmyx displays the Extension Parameter Editor.
 - c) Specify the values for the parameters of the exit. You can specify values manually or use the Value Selector. When you have specified values for each parameter of the exit, click the [**OK**] button to return to the Content Editor Settings dialog.
 - d) To specify a condition for activating the exit, click the  button.
Rhythmyx displays the Rule Editor.
 - e) Specify the Rules you want to activate the exit.

- 5 To add output translations (conversions of Content Item data downloaded from the database) to the Content Editor:
 - a) Choose the Item Output Translation tab.
 - b) Follow substeps b through f of Step 5 to add output translation exits and conditions.
- 6 To add item-level validation to the Content Editor:
 - a) Choose the Item Validation tab.
 - b) Click in the first available row and select the exit you want to add as an input translation for the Content Editor.
Rhythmyx displays the Extension Parameter Editor.
 - c) Specify the values for the parameters of the exit. You can specify values manually or use the Value Selector. When you have specified values for each parameter of the exit, click the [OK] button to return to the Content Editor Settings dialog.
 - d) To specify a condition for activating the exit, click the  button.
Rhythmyx displays the Rule Editor.
 - e) Specify the Rules you want to activate the exit.
 - f) The default value of Maximum number of errors before stopping validation is 10. You can change this value to any numeric value.

Defining Conditions for Exits, Effects, and Cloning Processes

When adding an Exit or 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.

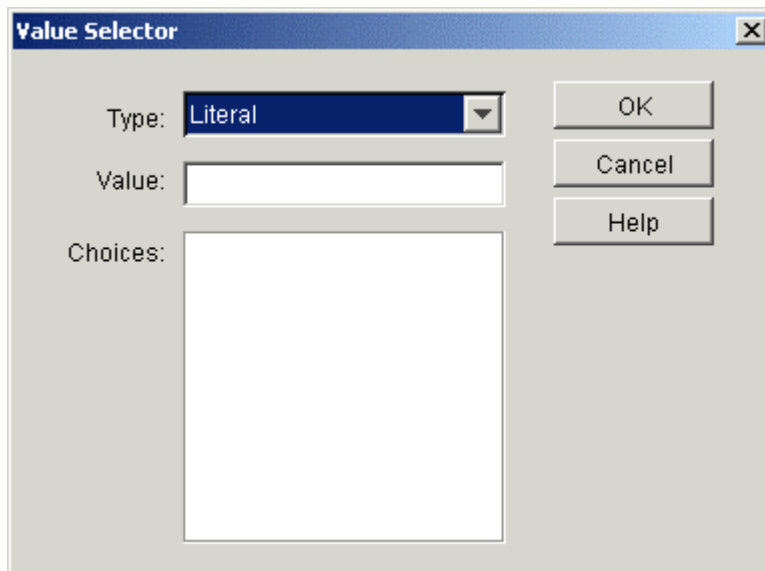
To define conditions for an extension or cloning process:

- 1 Double-click the  icon in the row of the extension 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, either 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.

Using the Value Selector

The Value Selector is available from many different fields to help you enter the data for the field. When you click in a field in which you can enter a value, Rhythmyx activates the field and displays a browse button (. . .). You can enter the value into the field manually, but using Value Selector is often faster. Click the browse button to activate the Value Selector.



Field Descriptions

Type Drop list. Specifies the classification of the Value. Options include:

Type	Description
Backend Column	Value is derived from a backend database column. Available options are all columns in all database tables associated with the Resource.
CGI Variable	Value is derived from the value of the specified CGI variable. Available options include all CGI variables.

Type	Description
Content Item Data	Value is derived from the specified Content Item Field. This option may return a Collection of Values. See Collections Processing, below, for details about the processing of Collections.
Content Item Status	<p>Value is derived from one of the following tables:</p> <ul style="list-style-type: none"> ▪ CONTENTSTATUS ▪ STATES ▪ WORKFLOWAPPS <p>Most commonly, you will want either the name (STATES.STATENAME) or ID (CONTENTSTATUS.CONTENTSTATEID) of the current State of the Content Item when using this class of values.</p>
Cookie	Value is derived from the specified Cookie. You must enter the name of the Cookie manually.
Date	Value is a date. Use SQL syntax when specifying the date value.
HTML Parameter	Value is derived from the specified HTML parameter. This Value type should only be selected if the HTML parameter could potentially store multiple values (for example, if the values are derived from a sys_CheckBoxGroup control). If the parameter only stores a single value, you should use the SIngle HTML Parameter type.
Literal	Value is the literal value you enter manually in the Value field.
Macro	Value is returned by the macro selected in the drop list.
Number	Value is the numeric value you enter manually in the Value field.
Relationship (Triggering)	Used only with Effects. Value is derived from the specified property of the Relationship that triggered the Effect. Available options are all properties (including User Properties) of the Relationship Type of the Relationship that triggered the Effect.
Relationship (Current)	Used only with Effects. Value is derived from the specified property of the Relationship current being processed. Available options are all properties (including User Properties) of the Relationship Type that is being processed.
Single HTML Parameter	Value is derived from the specified HTML parameter. Unless the parameter could potentially store multiple values (for example, if the values are derived from a sys_CheckBoxGroup control), you should use this Value type when specifying HTML parameters. If a parameter could store multiple values, use the HTML Parameter value type.
User Context	Value is derived from a user context variable, which stores information about the user logged in and the current session.

Value The selected value.

Choices The options available for the Value Type.

To use the Value Selector:

- 1 Click the browse button in the field for which you want to enter a value.

- 2 Rhythmyx displays the Value Selector dialog.
- 3 Select the Value **Type**.
- 4 Specify the Value.
 - For most Value Types, a list of available Values will be listed in the **Choices** field. Select the value you want to specify.
 - For a Literal or Number Value Type, enter the value. Numeric values should not include commas or periods.
 - For the Date Value Type, enter the SQL date expression.
- 5 Click [**OK**] to add your choice to the field.

Collections Processing

Content Value Data and Relationship Value Types may return collections of values rather than single values. The result of the processing depends on which side of the operand the collection is on.

Collection to Left of Operand

Operation	Behavior
=, !=	If the collection consists of a single item, it is processed; otherwise, returns <i>false</i> .
LIKE	Behaves as a set of conditional statements linked by ORs: (left1 LIKE right) OR (left2 LIKE right) OR (left3 LIKE right)...
NOT LIKE	Behaves as a set of conditional statements linked by ANDs: (left1 LIKE right) AND (left2 LIKE right) AND (left3 LIKE right)...
<, <=, >, >=	Behaves as a set of conditional statements linked by ANDs: (left1 LIKE right) AND (left2 LIKE right) AND (left3 LIKE right)...
IS NULL	If the collection is empty, returns <i>true</i> ; otherwise, returns <i>false</i> .
IS NOT NULL	If the collection is empty, returns <i>false</i> ; otherwise, returns <i>true</i> .
BETWEEN, NOT BETWEEN	Illegal if the right side of the equation consists of a single value. Displays an error message.
IN, NOT IN	If the collection contains exactly 1 entry, it is processed; otherwise, results in an error.

Collection to Right of Operand

Operation	Behavior
=, !=	If the collection consists of a single item, it is processed; otherwise, returns <i>false</i> .
LIKE	Behaves as a set of conditional statements linked by ORs: (left1 LIKE right) OR (left2 LIKE right) OR (left3 LIKE right)...
NOT LIKE	Behaves as a set of conditional statements linked by ANDs: (left1 LIKE right) AND (left2 LIKE right) AND (left3 LIKE right)...

Operation	Behavior
<, <=, >, >=	Behaves as a set of conditional statements liked by ANDs: (left1 LIKE right) AND (left2 LIKE right) AND (left3 LIKE right)...
IS NULL	Not allowed. Results in an error.
IS NOT NULL	Not allowed. Results in an error.
BETWEEN, NOT BETWEEN	If the collection contains exactly 2 entries, it is processed; otherwise, results in an error.
IN, NOT IN	The collection is processed.

Collection on Both Sides of Operand

Operation	Behavior
=, !=	If both collections consists of exactly 1 item, they are processed; otherwise returns <i>false</i> .
LIKE	If each collection consists of 1 or more items, each item in the left collection is compared to each item in the right collection; if, for any pair of items, a match is found, returns <i>true</i> ; otherwise returns <i>false</i> .
NOT LIKE	If each collection consists of 1 or more items, each item in the left collection is compared to each item in the right collection; if, for any pair of items, a match is found, returns <i>false</i> ; otherwise returns <i>true</i> .
<, <=, >, >=	Illegal operand; results in an error.
IS NULL	Illegal operand; results in an error.
IS NOT NULL	Illegal operand; results in an error.
BETWEEN, NOT BETWEEN	If the Left collection contains exactly 1 entry and the right collection contains exactly 2 entries, the equation is processed; otherwise, results in an error.
IN, NOT IN	Processed similar to LIKE and NOT LIKE.

CHAPTER 3

Maintaining Content Editor Fields

Each content editor defines a set of fields. Each field either modifies the data in a backend table column or points to a child editor.

NOTE: You can also make a field read-only by editing the content editor XML file directly. See *Implementing a Content Editor Manually* (on page [131](#)) for details.

When you create an editor by dropping a database table, Rhythmyx creates a field for each non-system field in the table you dropped. The name of the column (all lower-case) becomes the name of the field and the HTML display name for the field. The default control assigned to fields a parent in is the standard text control (`sys_EditBox`). If you chose *Multi-valued choice* or *Child content with a 1 to 1 relationship to the parent* when you dropped the table, the default control is the first control in the list that supports the data type *array*. If you chose *Child content with a 1 to many relationship to the parent* when you dropped the table, the default control is the first control in the list that supports the data type *table*.

The fields are defined in the content editor in the following sequence:

- Primary fields (in the order defined in the table)
- Choice sets
- Multi-values, single choice fields, in table order
- Children with a one to many relationship to the parent
- System fields

Field Maintenance Dialogs

A limited set of data is available for maintenance directly in the Content Editor Properties dialog. To access complete data for a Content Editor field, use one of the field maintenance dialogs:

- New Field Properties dialog
- Field Properties dialog
- Child Editor Field Properties dialog

New Field Properties Dialog

Rhythmyx displays the New Field Properties dialog when you click **[Edit]** after selecting a field on the Content Editor Properties dialog that has not yet been linked to a column in a Repository database table. If you are creating a new field, Rhythmyx creates a column in the content type table when you save the content editor.

The fields on this dialog define data for the Content Editor field. For details about each field, see the appropriate topic in *Field Data* (on page 39).

The screenshot shows the 'New Field Properties' dialog box with the following details:

- Field Properties:**
 - Field Name:
 - Data Type:
 - Format:
 - Default Value:
 - Treat data as binary
- Display Properties:**
 - Label:
 - Mnemonic:
 - Error Label:
 - Control:
 - Show in summary
 - Show in preview
 - Show 'clear field' checkbox in binary control
- Search Properties:**
 - Allow this field to be searched
 - Enable Transformation (don't index formatting, such as HTML tags)
 - Visible for global query
 - Punctuation is part of word (use for filenames, etc.)
- Validation:**
 - Occurrence:
 - Quantity:

Figure 9: New Field Properties dialog

Field Properties Dialog

Rhythmyx displays the Field Properties dialog when you click **[Edit]** after selecting a field on the Content Editor Properties dialog that is linked to a column in a Repository database table, if the **Type** of the field is *field*.

The screenshot shows the 'Field Properties' dialog box with the following details:

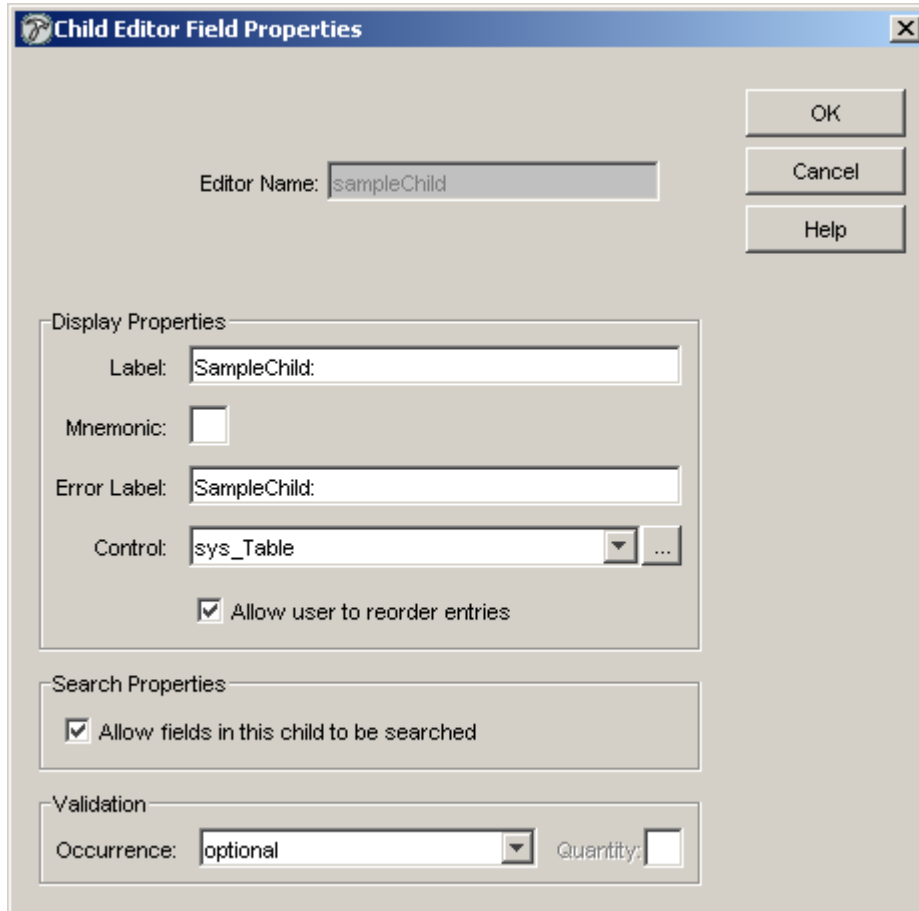
- Field Properties:**
 - Field Name: sys_title
 - Default Value: [Empty]
 - Treat data as binary
- Display Properties:**
 - Label: System Title
 - Mnemonic: [Empty]
 - Error Label: [Empty]
 - Control: sys_EditBox
 - Show in summary
 - Show in preview
 - Show 'clear field' checkbox in binary control
- Search Properties:**
 - Allow this field to be searched
- Validation:**
 - Occurrence: required
 - Quantity: [Empty]

Buttons on the right: OK, Cancel, Help.

Figure 10: Field Properties dialog

Child Editor Field Properties Dialog

Rhythmyx displays the Child Editor Field Properties dialog when you click **[Edit]** after selecting a field on the Content Editor Properties dialog that is linked to a column in a Repository database table, if the **Type** of the field is *child*.



The dialog box, titled "Child Editor Field Properties", contains the following elements:

- Editor Name:** A text field containing "sampleChild".
- Display Properties:**
 - Label:** A text field containing "SampleChild:".
 - Mnemonic:** An unchecked checkbox.
 - Error Label:** A text field containing "SampleChild:".
 - Control:** A dropdown menu showing "sys_Table" and a button with three dots.
 - Allow user to reorder entries:** A checked checkbox.
- Search Properties:**
 - Allow fields in this child to be searched:** A checked checkbox.
- Validation:**
 - Occurrence:** A dropdown menu showing "optional".
 - Quantity:** A text field.

Buttons for "OK", "Cancel", and "Help" are located in the top right corner.

Figure 11: Child Editor Field Properties dialog

Field Descriptions

Editor Name: Name of the field on the editor for the child editor is defined.

Allow user to reorder entries Check this box to allow users to change the order of entries in the child editor.

Allow fields in this child to be searched Check this box to make fields in the child editor eligible to be indexed for full-text search. Checking this box also checks the Allow this field to be searched box for each field on the child editor by default. If this box is unchecked, none of the fields on the child editor are eligible to be searched.

Creating a New Field

To create a new field:

- 1 On the Content Editor Properties dialog, click in the **Source** column of an empty row and choose the content editor definition file where the field will be stored. Options are *local def* (adds the field to the content editor local definition), *shared def* (adds a field from a shared definition file; the field must exist in the shared definition file; if it does not already exist in that field, you must add it to a shared definition file), and *system def* (adds a field from the system definition file; the field must already exist in the system definition file).
- 2 Click in the **Type** column of the same row and select the type of data stored in this field. Options are field and child.
- 3 Double-click in the **Name** column of the same row and enter a name for this field. This name should match the database table name of the column that will store the data for the field. If you have not attached a table to the editor, this name becomes the name of the column. Spaces are not allowed; use underscores instead.
- 4 Click the [Edit] button.
Rhythmyx displays the New Field Properties dialog.
- 5 Choose the **Data Type** from the drop list. Options include: *text*, *number*, *binary*, *date*, *time*, and *datetime*. NOTE: This field is not available for fields in child editors.
- 6 If the **Data Type** is *text* or *binary*, enter the maximum number of characters or bytes for the field in the **Format** field. The default is *max*. If Data Type has any other value, the **Format** field is not available.
- 7 Enter a **Default Value** for the field.
 - If you want to use a system value, click the browse button [...] and choose *Other Value*. Rhythmyx displays the Value Selector dialog, where you can choose the system value you want to assign to the field.
 - If you want the system to calculate the default value for the field, click the browse button [...] and choose *User Defined Function*. Rhythmyx displays the Function Properties dialog, where you can choose or enter the function to calculate the value.
- 8 If the field will store large text files, check the **Treat text as binary** box.
- 9 Enter a **Label** and **Error Label** for the field. If you do not define an **Error Label**, the value defaults to the same value as **Label**.
- 10 Choose a **Control** to use to display the field.
- 11 In **Mnemonic**, enter the keystroke for accessing the field. In the Content Editor, a user must enter ALT + the keystroke to access the field.

- 12 Check options to show the field in summary view and preview, and to clear binary data.
 - The **Show in Summary** option is only available for fields in child editors, and only when **Treat Data as Binary** is not checked. Check this option if you want Rhythmyx to display a summary of the field in the parent editor. Typically, this field is checked unless it stores a large amount of text data.
 - Check **Show in Preview** if you want Rhythmyx to display the field when displaying the content editor in Preview mode. This option is not available if **Treat Data as Binary** is checked. In most cases, you will check this option.
 - The **Show 'clear field' checkbox in binary control** option is only available if **Treat Data as Binary** is checked. Check this option if you want the content editor to display a box the user can check to clear the database table column before uploading binary data.
- 13 Choose an **Occurrence** option. Options include: *optional* and *required* for parent editors and *optional*, *required*, and *count* for child editors. If you choose *count*, you must enter the **Quantity**.
- 14 Click **[OK]** to save the field record.

Quick Field Creation

You can create a field quickly by entering data in the columns of the Content Editor Properties dialog. You can define the **Source**, **Type**, **Name**, **Label**, **Control**, **Occurrence**, **Data Type**, and **Format** on this dialog. To define the default value or error label, and to access display options, binary field clearing options, and advanced occurrence settings, you must edit the field.

Field Maintenance

Field maintenance includes creating new fields, *editing an existing field* (see "Editing a Field" on page 38), or *deleting a field* (on page 38). You can also *maintain some data about the child editor field on the parent editor* (see "Child Editors" on page 38).

Editing a Field

To edit a field, select the field you want to edit and click the **[Edit]** button. Rhythmyx will display the Field Properties dialog. You can modify any field displayed.

Deleting a Field

To delete a field, select the field you want to delete (you can select multiple adjacent fields) and click the **[Delete]** button.

Child Editors

A child editor is an editor that processes content that has a one-to-many relationship with the parent (in other words, the parent editor may result in several child items from the child editor). The left-most tab in the Content Editor Properties dialog is the mapper for the parent editor. Each tab to the right is the mapper for a child editor. The order that the child editors are rendered on the parent editor is determined by the order in which they are defined on the parent editor.

CHAPTER 4

Field Data

Each Content Editor field is composed of a set of data that defines the properties of the field. This chapter describes each of the properties available to a Content Editor field.

Source of Field

The Source of the field specifies the content editor XML definition document where the content editor field is defined.

Content editor definition documents fall into three classes. Each implementation of Rhythmyx includes a single system field document, which conforms to the `sys_ContentEditorSystemDef` DTD. This document defines all fields that will be common to all content editors in the implementation, such as creation and expiration date fields. System fields are included in each editor by default unless specifically excluded.

To define a system field, select `system def` for the Source. To exclude a system field, select the field and click **[Delete]**. The field will not be included in the content editor, but it still exists in the system.

Generally, each implementation also includes one shared field definition document (although there can theoretically be more than one shared field definition XML). This document defines fields that are shared by more than one content editor. To include a shared field, select *shared def* for the Source.

You cannot edit either the system field definition or the shared field definition in the Content Editor Properties dialog. You can only include from these two documents or exclude them. To edit a system or shared field definition, you must edit the field manually in the XML file.

Each individual editor is defined by a local definition document, which conforms to the `sys_ContentEditorLocalDef` DTD. To define a local field, select *local def* for the Source. Each content editor requires at least one local field.

Type of Field

The value in the Type field Indicates whether the data for this content editor field is maintained in this editor or in a child editor. If the Type is *field*, the data for the field is maintained using this editor. If the Type is *child*, the data for the field is maintained using a child editor.

Show in Preview

If you check this box (the default), the field will appear in the content editor in preview mode. If the box is unchecked, the field will not appear in preview mode.

Name of Field

If the value in the **Type** field is *field*, this field defines the Rhythmyx name for the content editor field.

The value in the **Name** field is the name Rhythmyx will use when processing the content editor field. It is not the name displayed for the content editor field, which is either the **Label** or the **Error Label**.

If you are creating a content editor from scratch, and generating the database table for the editor, Rhythmyx will create a column in the database table with the same name when you save the content editor. If you change the name of the field later, the name of the database column will not change, but the field will still be linked to the same column.

If the value in the **Type** field is *child*, the value in this field is the name of the child editor where the data for content editor field is maintained.

Label of Field

This is the label Rhythmyx will display for the content editor field when rendering the content editor. If you created the editor by dropping a table, Rhythmyx generates the **Label** by converting all underscores in the **Name** to spaces and capitalizing each letter preceded by a space, as well as the first letter in the string.

Mnemonic

This field holds the keystroke that accesses the field. The user must enter ALT + Mnemonic to access the field.

Error Label

This label will override the default **Label** if Rhythmyx generates a validation or translation error when processing the field data. If this field is blank, the value defaults to the same value as the **Label** field.

Control

This field specifies the control used to display the content editor field. For details about configuring controls for a field, see *Content Editor Field Controls* (on page 63). For technical details about controls in general and about the controls available with Rhythmyx by default, see *Content Editor Control Reference* (on page 147) in the Appendix.

Adding Parameters to a Control

Some controls require parameters to operate. The Inline Link Properties are used in fields where you want to enable inline links and inline images. These fields must use either the `sys_EditBox` or `sys_EditLive` controls. To allow inline links in a field, check the **mayHaveInlineLinks** box. To clean up inline references to purged Content Items, check the **cleanupBrokenInlineLinks** box.

To add parameters to a control:

- 1 Click the browse button [...] in the Control column of the Content Editor Properties dialog or next to the **Control** field in the Field Properties dialog.

Rhythmyx displays the Display Control Properties for (field) dialog.

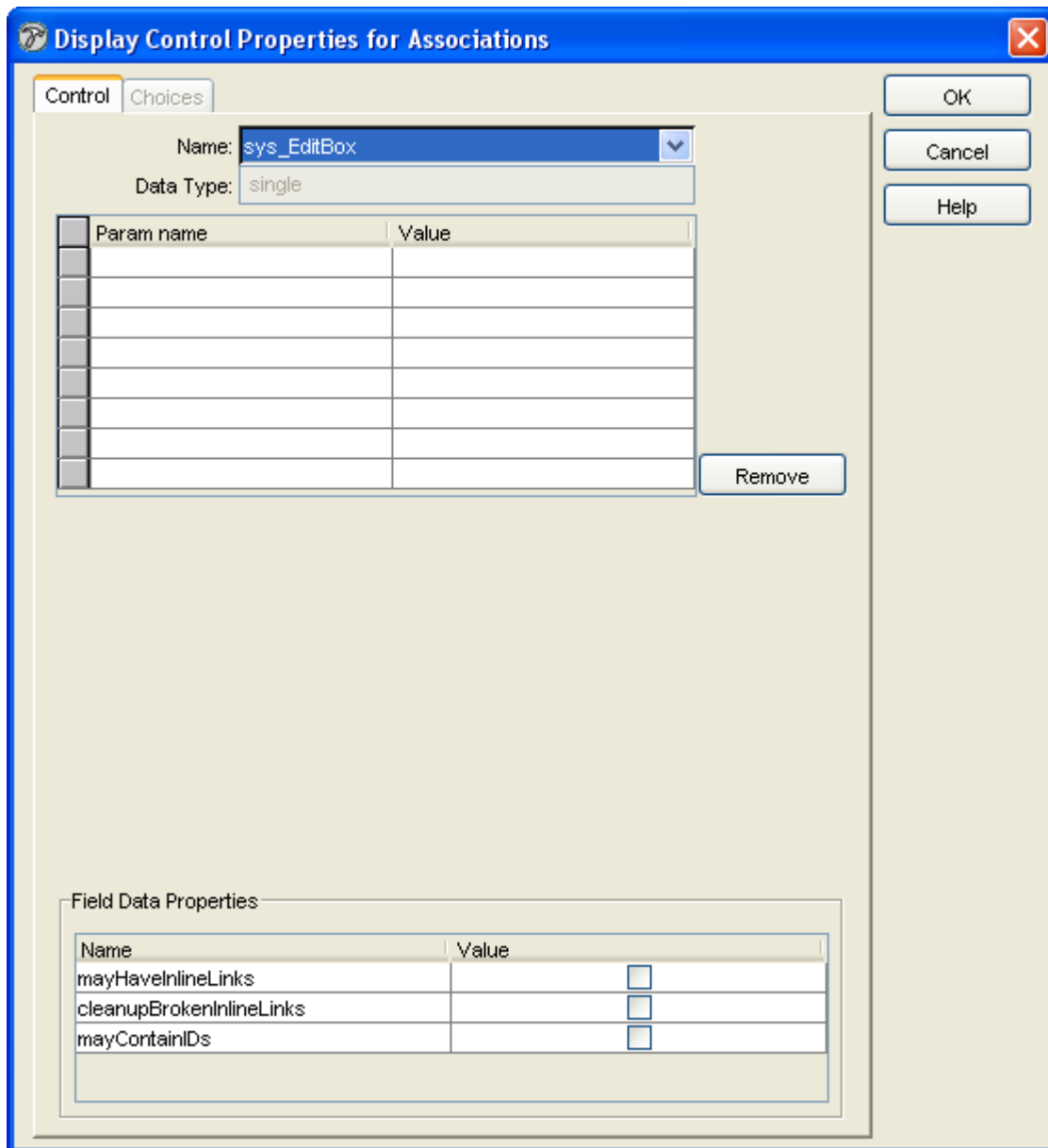


Figure 12: Display Control Properties for Associations dialog

- 2 Click in a row in the **Param name** column and select a parameter from the drop list.
- 3 Click in the same row in the **Value** column.
- 4 If the control definition specifies a choice list for this parameter, Rhythmyx displays the choices. Choose the choice for the parameter.
- 5 If the control definition does not specify a choice list for the parameter, choose an option from the drop list. Options available include: *Link* (displays the URL Request Properties dialog, where you define a URL to an application that calculates the parameter), *User Defined Function* (displays the standard Function Properties dialog), and *Other Value* (Displays the Value Selector dialog).

- 6 Repeat steps 2 through 5 for each parameter.
- 7 If the control includes a choice list, you must *define the choices* (see "[Adding Choices to a Control](#)" on page 52).
- 8 Click [OK] to save your control definition.

Specifying a URL for a Control

A URL for a control parameter can be either an internal request or an external request. An internal request is used when all processing of the parameter is internal to Rhythmyx. If the client browser processes the request, you must use an external request.

Use the URL Request Properties dialog to specify the URL for a control.

Specifying an External Request

To specify an external request:

- 1 On the URL Request Properties dialog, choose the **External** radio button.

The screenshot shows the 'URL Request Properties' dialog box. The 'Type' section has two radio buttons: 'External (links for browser use)' is selected, and 'Internal (links to Rhythmyx server, typically from stylesheet processor)' is unselected. Below the radio buttons is a 'Base href' text field. At the bottom of the dialog is a table with two columns: 'Param name' and 'Value'. The table has several empty rows. To the right of the dialog are buttons for 'OK', 'Cancel', and 'Help'. At the bottom right is a 'Remove' button.

Param name	Value

Figure 13: URL Request Properties dialog for external URLs

- 2 In the **Base href** field, enter the relative path to the Rhythmyx application that will generate the URL.
- 3 Click in a blank row in the **Param** name column and enter the name of a parameter in the application.

- 4 Click in the **Value** column of the same row and select the Value from the Value Selector dialog (or enter Literal text directly into the field).
- 5 Repeat steps 3 and 4 for all parameters in the application.
- 6 Click [**OK**] to save the URL request.

Specifying an Internal Request

To specify an internal request:

- 1 On the URL Request Properties dialog, choose the **Internal** radio button.

The screenshot shows the 'URL Request Properties' dialog box. The 'Type' section has the 'Internal' radio button selected. The 'Application name' dropdown is set to 'DTD'. The 'Resource name' dropdown is empty. Below these are several rows in a table for parameters, with columns for 'Param name' and 'Value'. A 'Remove' button is located at the bottom right of the table area. On the right side of the dialog, there are 'OK', 'Cancel', and 'Help' buttons.

Params:	Param name	Value

Figure 14: URL Request Properties dialog for internal requests

- 2 Choose the **Application name** of the Rhythmyx application that will build the URL from the drop list.
- 3 Choose the **Resource name** of the resource in the application that build the URL from the drop list.
- 4 Click in a blank row of the **Param name** column and enter the name of a parameter in the application.
- 5 Click in the **Value** column of the same row and select the Value from the Value Selector dialog (or enter Literal text directly into the field).
- 6 Repeat step 5 for all parameters in the resource.
- 7 Click [**OK**] to save the URL request.

Managing Dependencies in a Control

NOTE: Most users will not have to change the dependencies in controls. Only refer to the following topic if you are an advanced user.

Dependencies are pre-exits that a control requires. They are defined in `../sys_resources/sys_Template.xml`. The DTD for dependencies is `../DTD/sys_LibraryControlDef.dtd`.

If a control has dependencies, the Display Control Properties for <Field> dialog includes the **[Dependencies]** button. When you click this button, Rhythmyx displays a drop list showing all dependencies for the control. Dependencies appear as `exit_name (parameter1, parameter2, ...)`.

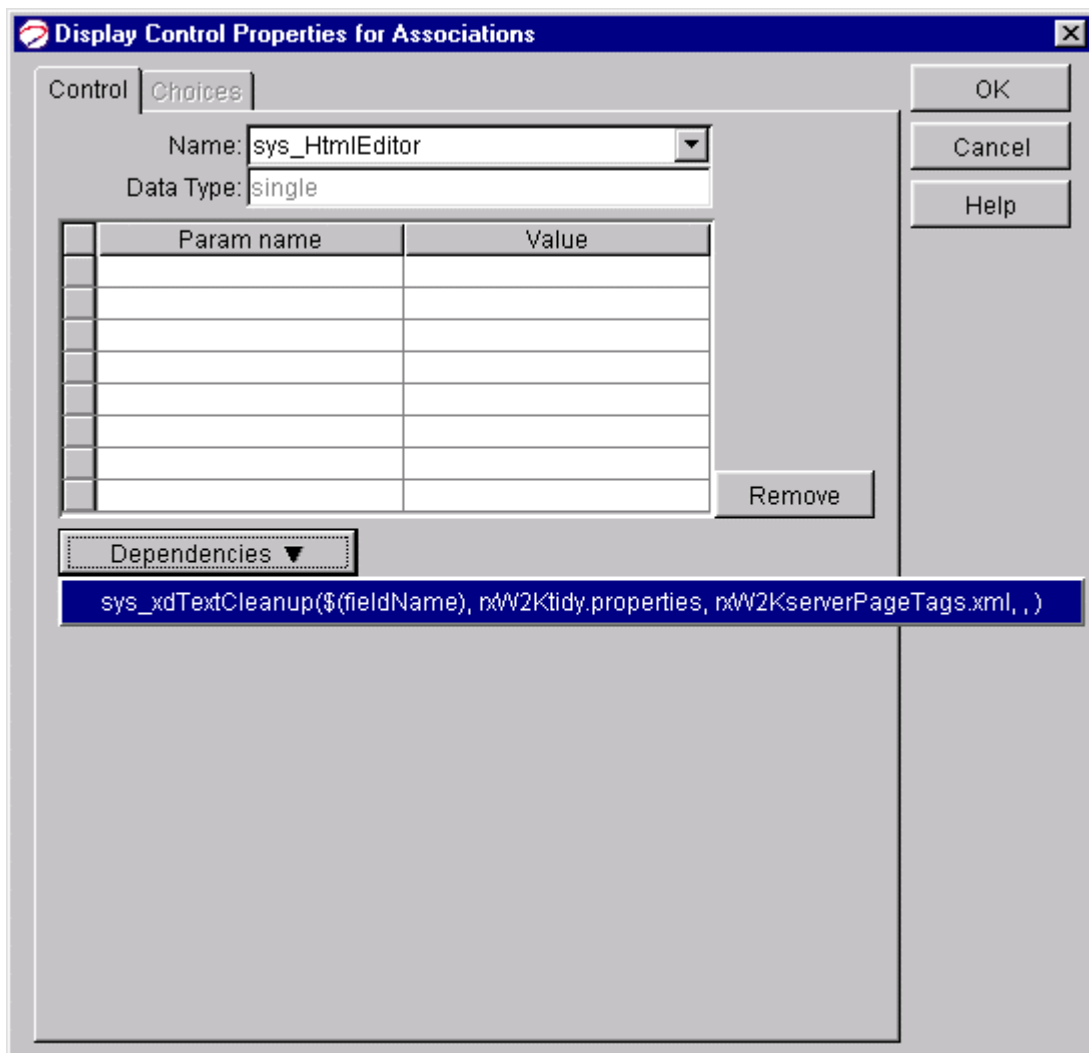


Figure 15: Display Control Properties for Associations Dialog with Dependency

If the dependency includes no editable parameters, the dependency name appears as a disabled link. If the dependency includes editable parameters, the dependency name appears as an enabled link. Rhythmyx displays the Exit Properties dialog when you click the dependency name.

If the occurrence setting of the dependency in the control library metadata is `single occurrence`, all instances of the dependency in a content editor use the same parameters. If the occurrence setting of the dependency is `multiple occurrence`, you must define the parameters for each instance of the dependency in the content editor.

Adding Choices to a Control

If a control provides a list of choices to the user, you will need to specify how those choices are generated.

To specify choices for a control:

- 1 On the Display Control Properties for (control) dialog, click the Choices tab.

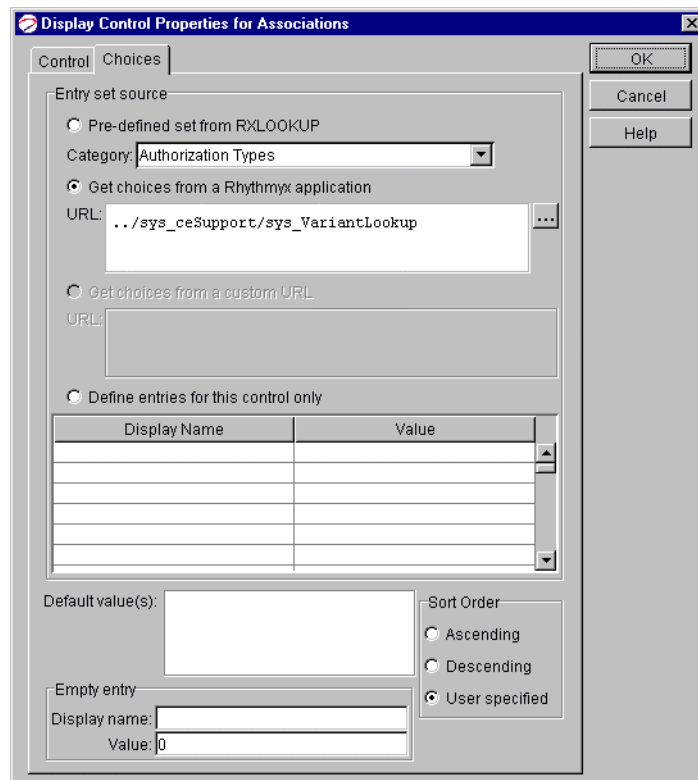


Figure 16: Display Control Properties for Associations dialog showing Choices tab selected

- 2 Choose the Entry-set source. Options are:
 - **Pre-defined set from RXLOOKUP Table.** If you choose this option, you must choose the category of the choices from the drop list.
 - **Get choices from a Rhythmyx application.** If you choose this option, you must specify the URL of the application that generates the choice list.
 - (The **Get choices from a custom URL** option is disabled for all systems except legacy systems that used this option in the past. This option caused errors in some implementations and has been deprecated. If you use this option, you should migrate to the **Get choices from a Rhythmyx application** option.)

- **Define choices for this control only.** If you choose this option, you must define the choice list for this control by entering a Display Name (the name displayed in the browser when the control is rendered) and Value (the value Rhythmyx uses when processing the choice) for each choice.
- 3 Check the radio button specifying the Sort Order of the choices. Options are
 - **Ascending** (ascending alphanumeric order)
 - **Descending** (descending alphanumeric order)
 - **User-Specified** (choices appear in the order specified by the SequenceID if derived from the RXLOOKUP Table, or in the order defined or received if defined for the control or generated by a Rhythmyx application)
 - 4 If **RXLOOKUP Table** is checked in the Entry-set sources box, select the Default value(s) of the choice list. Otherwise, enter the Default Value(s).
 - 5 In the **Empty Entry** box, enter the Display name (name displayed when the editor is rendered if there is no default entry for the choice list) and Value (value Rhythmyx uses for processing if there is no default entry for the choice list).

Specifying the URL of a Choices Application

If you choose to derive choices for a choice list in a content editor field from a Rhythmyx application, you must specify the URL of the application. You can enter the URL manually in the **URL** field on the Choices tab of the Display Control Properties for <control> dialog or you can use the Create Choice Lookup Request dialog. To create a URL in the Create Choice Lookup Request dialog:

- 1 Click the browse button [. . .] next to the URL field on the Display Control Properties for <control> dialog.
- 2 Rhythmyx displays the Create Choice Lookup Request dialog.
- 3 Choose the **Application Name** of the Rhythmyx application that generates the choice list.
- 4 Choose the **Resource Name** of the resource within that application that generates the choice list.
- 5 Click in a row in the **Param name** column and select a parameter from the drop list.
- 6 Click in the same row in the **Value** column and enter the value or click the browse button [. . .] to display the Value Selector dialog.
- 7 To remove a parameter, click the [**Remove**] button.
- 8 Click [**OK**] to save the URL specification.

Create Choice Lookup Request Dialog

Use the Create Choice Lookup Request dialog to automate entry of the URL to the Rhythmyx application that creates choices for a choice list in a content editor control. To access the Create Choice Lookup Dialog, click the browse button [...] next to the **URL** field on the on the Choices tab of the Display Control Properties for <control> dialog.

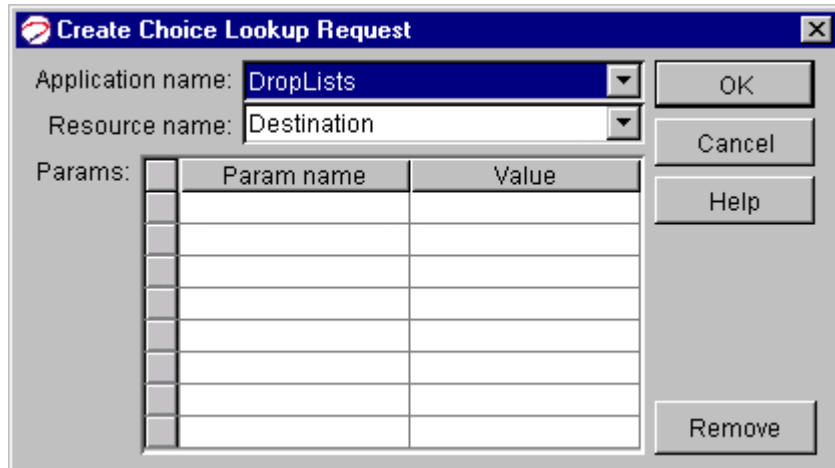


Figure 17: Create Choice Lookup Request Dialog

Field Descriptions

Application name Drop list. Name of the Rhythmyx application from which the list of choices is derived. Options include all applications defined in your system.

Resource name Drop list. Name of the resource within the Rhythmyx application from which the list of choices is derived. Options include all resources in the specified application.

Params Table. Parameters for the lookup request.

Name Name of the parameter.

Value Value of the Parameter. You can specify the value manually or use the *Value Selector* (see "[Using the Value Selector](#)" on page 26).

Data Type

The Data Type specifies the generic type of data stored in the content editor field. You can only define the Data Type for a new field. Options are *text*, *number*, *binary*, *date*, *time*, and *datetime*.

Occurrence of Field

The value in this field defines whether the content editor field is required or optional, and for child fields, how many times it must repeat. If the value in the Type field for this content editor field is *field*, options are *required* and *optional*. If the value in the Type field for this content editor field is *child*, options are *required*, *optional*, and *count*. If the value of Occurrence is *count*, you must enter the **Quantity** to define how many times the field must appear.

NOTE: When defining a field in the Content Editor Properties dialog, enter the quantity in the **Occurrence** column, which sets the value of Occurrence to *count*.

Format of Field

The Format is required if the Data Type is *text* or *binary*. This field defines the size of the field by specifying the maximum number of characters (text; default is 50) or bytes (binary; default is max, which mean the field stores the greatest amount of data available for the data type in the RDBMS).

Default Value of Field

The Default Value property defines the default value Rhythmyx will display in the content editor field. You can enter a literal value directly, specify a value using the Value Selector, or specify a UDF that will generate the default value.

Treating Text as Binary

The Rhythmyx server treats fields with binary data differently than it treats text data. When you add a new content item or update a content item that includes binary data in a field, the binary data is stored in the database. If the field does not contain any data, the database column is not changed. When editing a binary field, the binary data is not stored in the content editor. You must request the binary data file using the `binary` command. You may want to treat large text fields the same way. To treat a text field as though it contains binary data, check **Treat text as binary**.

Clearing Binary Data from a Field

If a field maintains binary data, the user may need to clear the backend database column before saving new data to it. To provide this option to the user, check **Show 'clear' field checkbox in binary control**. The field will include a checkbox the user can check to clear the database before saving binary data.

You can also make this option available if you checked the **Treat text as binary** box.

Show in Summary

This checkbox is available only if the field is in a child editor and is not a binary field or treated as a binary field. If you check this box, this field will be shown in the summary view on the parent editor.

Typically, this box is unchecked for large text fields that do not make sense in summary view. For all other fields, it is usually checked.

Search Properties

Search properties define whether the field is eligible to be searched, and controls how the full-text search editor indexes the field for searching. Content Items are indexed for searching when they are created and any time they are uploaded to the Repository. (You can also reindex manually; see the topic "Server Console Commands for Search" in the Rhythmyx Server Administrator Help.

The **Allow this field to be searched** checkbox specifies whether the field is eligible to be searched. If this box is checked, the field is indexed for searching. If the box is unchecked, the field is not indexed for searching. In addition, the other search properties are unavailable if this box is unchecked.

NOTE: If you uncheck this box after already indexing some Content Items, those Content Items will be returned in search queries that match data for the field. If you re-index, the Content Items will no longer be included in the index, and they will not be returned.

The **Enable Transformation** checkbox controls whether the content of the field is converted to raw text before it is added to the index. Transforming the text removes extraneous data, such as HTML markup, leaving only the document text. For fields with certain data types (such as BLOB), this box is checked by default and cannot be unchecked. In general, you want to check this box for fields that contain binary or rich-text data (such as a field that uses the `sys_file` control or uses a rich text editor such as the `sys_EditLive DHTML` editor. Checking this box for fields that only store plain text data (such as fields that use the `sys_editbox` control) will not affect the results of indexing, but will slow performance.

The **Visible for global query** checkbox defines whether the field is searched when a user enters a query string into the Search for field on the search dialog. If the **Allow this field to be searched** checkbox is checked, this box is also checked by default. Some fields, however, should not be searched automatically; examples include the Content ID or Workflow. Fields with this box unchecked can be searched using the Advanced search capability in the Content Explorer.

The **Punctuation is part of word** checkbox controls how the search engine parses words during indexing. If this field is unchecked, punctuation marks are treated as word separators rather than as part of a word. If the box is checked, punctuation marks are treated as part of the word. Check this box for fields that will contain values such as file names or product IDs that include hyphens or other punctuation marks. Note that if the field stores dates, this box is unchecked and unavailable for editing.

If the value of the **Type** for the field is *child*, then only one checkbox, **Allow fields in this child to be indexed**, is included. If this box is checked, fields in the child editor are eligible to be indexed for full-text search, and the **Allow this field to be searched** checkbox is checked for each field in the child editor by default, making each field eligible to be searched. You can uncheck the **Allow this field to be searched** checkbox for each field in the child editor individually.

If the **Allow fields in this child to be indexed** box is unchecked, none of the fields in the Content Editor are eligible to be searched, and the **Allow this field to be searched** checkbox is unavailable for all fields in the child editor.

CHAPTER 5

Content Editor Field Controls


The control specified for a field defines the interface for the field and its behavior. In general, you select a control by clicking in the **Control Name** column in the Content Editor Properties dialog or the **Control Name** field in Field Properties dialog, and selecting the control you want to assign to the field.

Content Editor Control Dialogs

When you assign a control to a Content Editor field, use the Control Properties for <control> dialog to define. If a control includes a limited set of optional values, you may need to use the URL Request Properties dialog to define the URL from which the values for the field are derived.

Display Control Properties Dialog

Use the Display Control Properties for <control> dialog to define the properties of the control. To access the Display Control Properties dialog:

- on the Content Editor Properties dialog, click in the Control column of the row of the field for which you want to maintain the control, then click the browse button .
- On the Field Properties dialog click the browse button next to the **Control** field.

The Display Control Properties dialog includes two tabs: Control and Choices.

Control Tab

Use the Control tab of the Display Control Properties dialog to define the properties for the control.

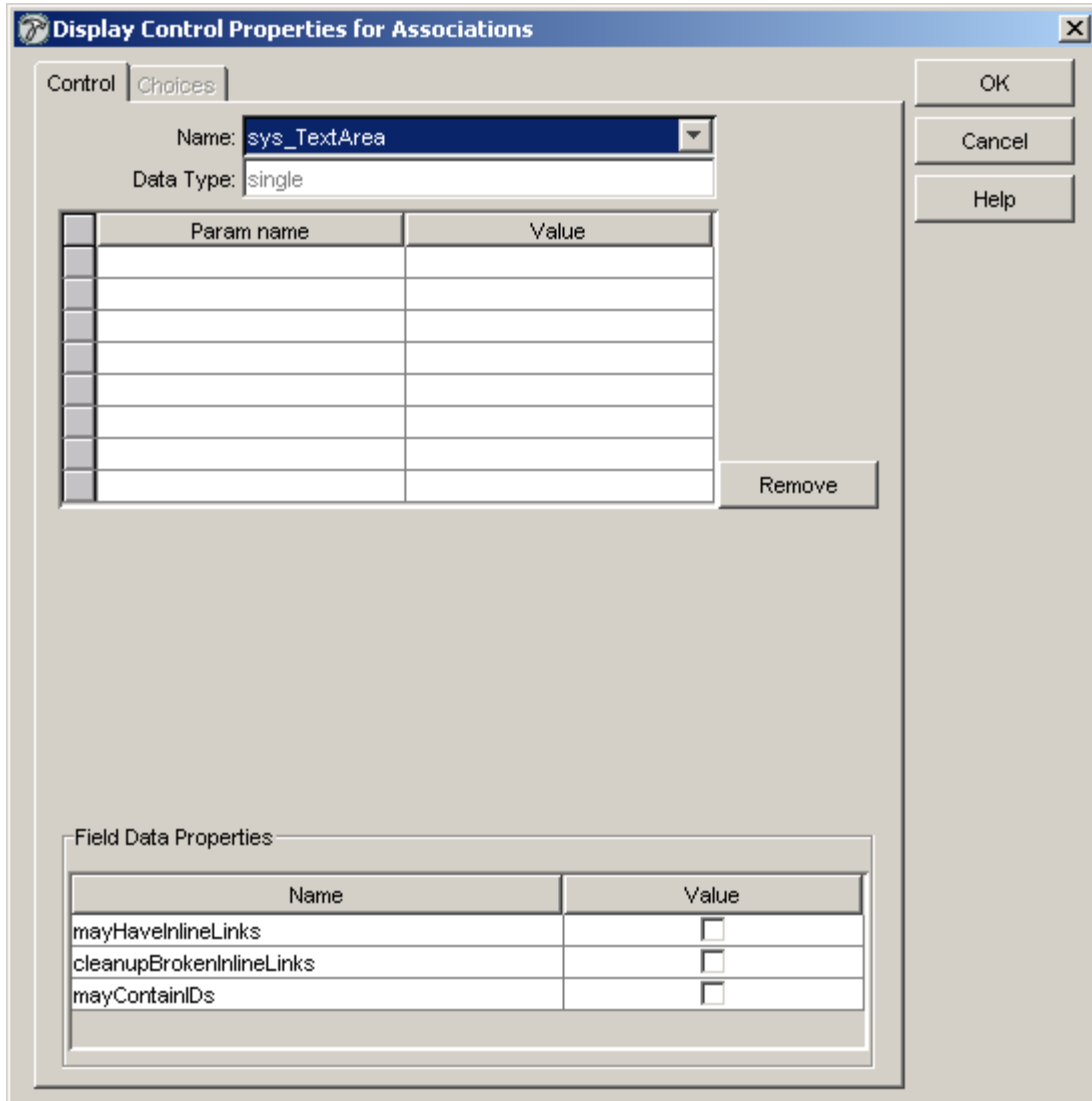


Figure 18: Control Tab of Display Control Properties Dialog

Field Descriptions

Name Drop list. The name of the control being specified for the field.

Data Type Read only. Specifies the dimension of the control, which defines the form of data the control expects. Available values include:

Value	Description
single	Data is zero or one value.

Value	Description
array	Data is a sequence of 0 or more values.
table	Data is a table of values.

Param name Drop list. The name of a parameter for the control.

Value The value of the parameter. May be derived from a Rhythmyx application, from a UDF, or assigned using the Value Selector.

Field Data Properties List of addition properties for the field.

mayHaveInlinelinks Specifies whether the control supports inline links. Only useful for rich-text editor controls, such as `sys_EditLive`.

cleanupBrokenInlineLinks Specifies that Rhythmyx should clean up inline references to purged Content Items.

mayContainIDs Specifies that the value of the field may include identifiers of other parts of the implementation. For example, the field may specify the URL of an automated index query. Among the parameters of the query may be a Variant ID. This checkbox is used by Multi-Server Manager so it can detect these identifiers when discovering dependencies while building an archive.

Choices Tab

The choices tab is only available for controls that can take set of values. Use this tab to specify how the control derives the options.

Display Control Properties for Associations

Control Choices

Entry set source

Pre-defined set from RXLOOKUP

Category: Authorization Types

Get choices from a Rhythmyx application

URL:

Get choices from a custom URL

URL:

Define entries for this control only

Display Name	Value

Default value(s): All Content
All Public Content
Custom

Sort Order

Ascending

Descending

User specified

Empty entry

Display name:

Value: 0

OK
Cancel
Help

Figure 19: Choices Tab of Display Control Properties Dialog

The three options for deriving the values for the control are selected using radio buttons:

- Predefined set from RXLOOKUP

This option derives the possible values for the control from the RXLOOKUP table in the Repository. You must specify the **Category** of the lookup used to derive the options in the associated drop list. This option is most useful if the control uses a defined set of values that are shared by several Content Editors.

- Get choices from a Rhythmyx application

This option derives the possible values for the control from a Rhythmyx application. You must specify the URL of the application used to generate the values for the lookup. This option is most useful if the set of options is generated dynamically; for example, if the set of options is derived from data maintained in other Content Editors.

- Define entries for this control only

This option derives the set of values from a list defined locally in the control. Use the table associated with this radio button to define the choices for this control. Each option must consist of both a **Display Name** (the name displayed to the user for the choice) and a **Value** (the values used internally in Rhythmyx for processing).

NOTE: The fourth option (Get choices from a custom URL) is only available on legacy systems that used this option in the past. This option caused errors in some implementations and has been deprecated. If you use this option, you should migrate to the Get choices from a Rhythmyx application option.

Default Values specify the default values for the control. This option is only valid if you have chosen the **Predefined set from RXLOOKUP** or **Define entries for this control only** radio buttons. You can select multiple values using Control-click.

Sort order defines the order in which the options are sorted. Options include:

- Ascending
- Descending
- User specified (choices appear in the order specified by the SequenceID if derived from the RXLOOKUP Table, or in the order defined or received if defined for the control or generated by a Rhythmyx application)

Use the Empty Entry options to specify the **Display Name** and **Value** for a null entry, such as a drop list that does not have a default value when first displayed..

URL Request Properties Dialog

Use the URL Request Properties dialog to specify the URL from which the value of the parameter is derived. A URL for a control parameter can be either an internal request or an external request. An internal request is used when all processing of the parameter is internal to Rhythmyx. If the client browser processes the request, you must use an external request.

URL Request Properties Dialog for External Requests

Use this version of the dialog when the client browser processes the request. To use this version of the dialog, click the **External** radio button.

The dialog box is titled "URL Request Properties". It contains the following elements:

- Type:** A group box containing two radio buttons:
 - External (links for browser use)
 - Internal (links to Rhythmyx server, typically from stylesheet processor)
- Base href:** A text input field.
- Params:** A table with two columns: "Param name" and "Value". The table is currently empty.
- Buttons:** "OK", "Cancel", "Help", and "Remove" are located on the right side of the dialog.

Figure 20: URL Request Properties Dialog with External Radio Button Selected

The **Base href** field specifies the relative path to the source of the data.

The params table specifies a set of parameters to append to the request. The **Param name** specifies the name of the parameter, the **Value** specifies the value of the parameter.

URL Request Properties Dialog for Internal Requests

Use this version of the dialog when all processing of the parameter is internal to Rhythmyx. To use this version of the dialog, click the **Internal** radio button.

The dialog box is titled "URL Request Properties". It contains the following elements:

- Type:** A group box containing two radio buttons. The "Internal (links to Rhythmyx server, typically from stylesheet processor)" radio button is selected.
- Application name:** A drop-down menu currently showing "DTD".
- Resource name:** An empty drop-down menu.
- Params:** A table with two columns: "Param name" and "Value". The table is currently empty.
- Buttons:** "OK", "Cancel", "Help", and "Remove" buttons are located on the right side of the dialog.

Figure 21: URL Request Properties Dialog with External Radio Button Selected

Field Descriptions

Application name Drop list. Name of the Rhythmyx application from which the list of choices is derived. Options include all applications defined in your system.

Resource name Drop list. Name of the resource within the Rhythmyx application from which the list of choices is derived. Options include all resources in the specified application.

Params Table. Parameters for the lookup request.

Name Name of the parameter.

Value Value of the Parameter. You can specify the value manually or use the *Value Selector* (see "[Using the Value Selector](#)" on page 26).

Create Choice Lookup Request Dialog

Use the Create Choice Lookup Request dialog to automate entry of the URL to the Rhythmyx application that creates choices for a choice list in a content editor control. To access the Create Choice Lookup Dialog, click the browse button [...] next to the **URL** field on the Choices tab of the Display Control Properties for <control> dialog.

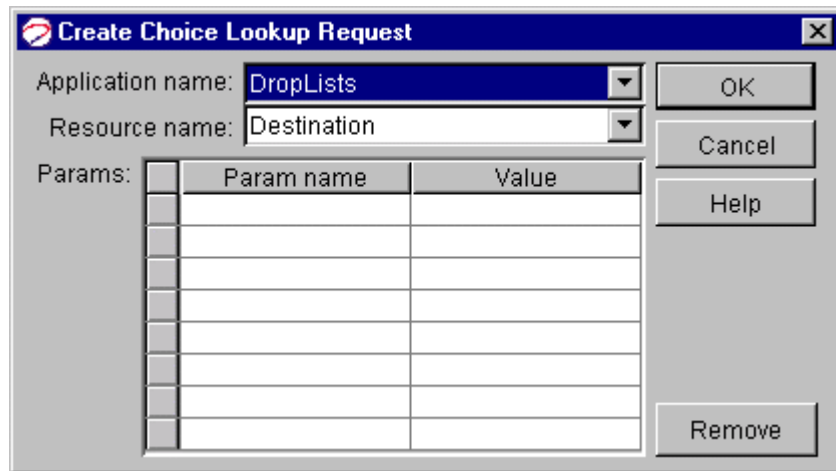


Figure 22: Create Choice Lookup Request Dialog

Field Descriptions

Application name Drop list. Name of the Rhythmyx application from which the list of choices is derived. Options include all applications defined in your system.

Resource name Drop list. Name of the resource within the Rhythmyx application from which the list of choices is derived. Options include all resources in the specified application.

Params Table. Parameters for the lookup request.

Name Name of the parameter.

Value Value of the Parameter. You can specify the value manually or use the *Value Selector* (see "Using the Value Selector" on page 26).

Configuring a Content Editor Control

To configure a Content Editor control:

- 1 On the Content Editor Properties dialog, in the Control column of the row of the Content Editor field for which you want to configure the control, double-click, then click the browse button;
OR
On the New Fields Properties dialog or the Field Properties dialog, click the browse button next to the **Control** field
Rhythmyx displays the Display Control Properties dialog.
- 2 In the **Name** drop list, select the control you want to assign to the field. See Appendix II, *Content Editor Control Reference* (on page 147), for a list of standard controls included in Rhythmyx. You can also implement your own specialized controls.
- 3 Click in the **Param name** column of the first empty row in the parameters table and select the parameter for which you want to specify a value from the drop list. A list of the parameters for each control is included with the reference information for the control in Appendix II, Content Editor Control Reference.
- 4 If you want to derive the value for the parameter from a Rhythmyx resource or other URL:
 - a) On the popup menu, click *Link*.
Rhythmyx displays the URL Request Properties dialog.
 - b) If the request for the value is processed by the browser, click the **External** radio button. Specify the **Base href** of the request URL.
 - c) If the request for the value is processed internally in the Rhythmyx server, click the **Internal** radio button. Select the Rhythmyx **Application** and **Resource** from which to derive the value.
 - d) To add a parameter to the request, click in the **Param name** column of the first empty row in the Params table and enter the name of the parameter. Then click in the **Value** column and use the *Value Selector* (see "Using the Value Selector" on page 26) to specify the value for the parameter.
- 5 If you want to derive the value for the parameter from a UDF:
 - a) On the popup menu, click *User Defined Function*.
 - b) Rhythmyx displays the Function Properties dialog.
 - c) Select the **Function** you want to use to provide the value for the parameter from the drop list.
 - d) To specify values for the UDF parameters, click in the Value column of row of the parameter for which you want to supply a value. You can enter a value manually or use the Value Selector to specify the value.

- 6 If you want to specify the value directly, on the popup menu, choose *Other Value* and use the *Value Selector* (see "[Using the Value Selector](#)" on page 26) to specify the value.
- 7 To remove a parameter from the control, select the parameter you want to remove and click the [**Remove**] button.
- 8 If you are configuring the sys_EditLive control (or a custom control that supports inline links):
 - if you want to support inline links in the control, check the **mayHaveInlineLinks** checkbox.
 - to clean up inline references to purged Content Items, check the **cleanupBrokenInlineLinks** checkbox.
- 9 If the value of the field may include identifiers of other parts of your implementation, check the **mayContainIDs** checkbox. For example, if the control is used for the URL of the query for an auto index, the parameters in the URL may include a Variant ID. You would need to flag this field so Multi-Server Manager could discover these identifiers when building deployment archives.
- 10 Click [**OK**] to save your configuration.

Adding Data Validation

When you create a new content editor, you can add data validation for fields entered in the user interface. Data validation occurs either at the field level or the item level.

Field-level validation checks whether the content in an individual field has the required format. For example, use field-level validation to check if a date field is numeric and uses the format `yyyymmdd`. Rhythmyx performs field validation each time it inserts or updates a document.

Item-level validation checks the relationship between fields in a content item. For example, use item-level validation to check that a user enters a file name in a `FileName` field when the user sets an `IncludeFile` field to "yes." Rhythmyx performs item validation each time a workflow transition occurs.

Field-Level Validation

In most cases, you can set up field-level validation within the `PSXField` definition in the content editor XML. However, if you want to perform extensive or complicated field-level validation, you can use a UDF to perform field-level validation. The UDF must return a Boolean object that is "true" if the field is valid and "false" if the field is not valid.

NOTE: The Content Editor Properties dialog and Field Properties dialogs do not include interfaces for defining field-level translations. To add field-level validation, you must edit the local definition XML for the Content Editor.

To set up field-level validation within a content editor, add a `<PSXFieldValidationRules>` element within the `<PSXField>` definition. In the `<PSXFieldValidationRules>` element, you can include

- `<PSXRule>` elements that specify validation conditions
- a `<PSXApplyWhen>` element that specifies when to apply the condition
- an `<ErrorMessage>` element to specify the error message to return when the validation condition is not met.

In the following example, a validation rule is set up for the `startdate` field. The validation rule requires that the field not be null.

```
<PSXField name="startdate" showInSummary="yes" showInPreview="yes"
forceBinary="no">
  <DataLocator>
    <PSXBackEndColumn id="0">
      <tableAlias>RX_DATE</tableAlias>
      <column>STARTDATE</column>
      <columnAlias/>
    </PSXBackEndColumn>
  </DataLocator>
  <DataType/>
  <OccurrenceSettings dimension="required" multiValuedType="delimited"
delimiter=";" />
  <FieldRules>
    <PSXFieldValidationRules name="isValidStartDate">
      <PSXRule>
        <PSXConditional id="1">
          <variable>
            <PSXSingleHtmlParameter id="0">
              <name>startdate</name>
            </PSXSingleHtmlParameter>
          </variable>
          <operator>IS NOT NULL</operator>
          <value>
            <PSXTextLiteral id="0">
              <text/>
            </PSXTextLiteral>
          </value>
        </PSXConditional>
      </PSXRule>
    </PSXFieldValidationRules>
  </FieldRules>
</PSXField>
```



```

<PSXApplyWhen ifFieldEmpty="yes">
  <PSXRule>
    <PSXConditional id="2">
      <variable>
        <PSXTextLiteral id="0">
          <text>1</text>
        </PSXTextLiteral>
      </variable>
      <operator>=</operator>
      <value>
        <PSXTextLiteral id="0">
          <text>1</text>
        </PSXTextLiteral>
      </value>
    </PSXConditional>
  </PSXRule>
</PSXApplyWhen>
<ErrorMessage>
  <PSXDisplayText>This field cannot be empty.</PSXDisplayText>
</ErrorMessage>
</PSXFieldValidationRules>
</FieldRules>
</PSXField>

```

In the `<PSXRule>` element of this example, the `<PSXConditional>` element specifies that the field `startdate` cannot be null. The `<ErrorMessage>` element specifies that Rhythmyx display the error message "This field cannot be empty" when the condition is not met. By default the XSL displays the message at the top of the returned page and displays the field name in red.

The following field(s) produced validation errors:	
Content Title:	This field cannot be empty
Start Date:	This field cannot be empty




Content Title:	<input type="text"/>
Display Title:	<input type="text" value="Lorem ipsum"/>
Keywords:	<input type="text" value="Lorem ipsum"/>
Author:	<input type="text" value="Thucydides Hieronedentes"/>
Start Date:	<input type="text"/> 
End Date:	<input type="text"/> 
Reminder Date:	<input type="text"/> 
Abstract:	<input type="text" value="Lorem ipsum dolor sit amet, consectetuer adipiscing"/>

Figure 23: Rhythmyx Content Editor displaying a validation error

By default, `<PSXApplyWhen>` only checks fields with data entered; if you want to validate when the field is empty, set the `ifFieldEmpty` attribute to "yes" (`ifFieldEmpty="yes"`). In the preceding example, the `<PSXApplyWhen>` element specifies that the content editor apply the rule when the `startdate` field is empty.

Transition-Dependent Field-Level Validation

You can set up transition-dependent field-level validations using attributes of the `<OccurrenceSettings>` element.

- You can specify that a field is required (must have a value) for all transitions by setting the `<OccurrenceSettings>` dimension attribute to "required":

```
<PSXField name="firstname" showInSummary="yes"
showInPreview="yes" forceBinary="no">
  <DataLocator>
    <PSXBackEndColumn id="0">
      <tableAlias>RX_PERSON</tableAlias>
      <column>FIRSTNAME</column>
      <columnAlias/>
    </PSXBackEndColumn>
  </DataLocator>
  <DataType/>
  <OccurrenceSettings dimension="required"
multiValuedType="delimited" delimiter=";" />
</PSXField>
```
- You can specify that a field is required (must have a value) for a specific transition by setting its `<OccurrenceSettings>` `transitionId` attribute to the transition ID and its dimension attribute to "required". You can include more than one `<OccurrenceSettings>` attributes, but each must have a different `transitionId` and only one can have no `transitionId`. When a transition occurs with an ID that is not an attribute of an `<OccurrenceSettings>` element, the transition uses the `<OccurrenceSettings>` element with no `transitionId`.

In the following example, the `firstname` field is not required for transitions 1, 2, and 3, but it is required for transitions 4, 5, and 6.

```
<PSXField name="firstname" showInSummary="yes"
showInPreview="yes" forceBinary="no">
  <DataLocator>
    <PSXBackEndColumn id="0">
      <tableAlias>RX_PERSON</tableAlias>
      <column>FIRSTNAME</column>
      <columnAlias/>
    </PSXBackEndColumn>
  </DataLocator>
  <DataType/>
  <OccurrenceSettings dimension="optional"
multiValuedType="delimited" delimiter=";" />
  <OccurrenceSettings dimension="required"
multiValuedType="delimited" delimiter=";" transitionId=4/>
  <OccurrenceSettings dimension="required"
multiValuedType="delimited" delimiter=";" transitionId=5/>
```

```
<OccurrenceSettings dimension="required"  
multiValuedType="delimited" delimiter=";" transitionId=6/>  
</PSXField>
```

NOTE: While you can set OccurrenceSettings in the Field Properties dialog, you cannot set the Transition ID. If you want to validate a field for a specific Transition, you must edit the local definition XML.

Item-Level Validation

Item-level validation occurs after field-level validation when a user performs a workflow action, but before transitions occur.

When a user performs a workflow action, Rhythmyx checks whether any validation exits have been assigned to the Content Editor. If conditions have been assigned to the Content Type, Rhythmyx evaluates any conditions that have been assigned to trigger the validation. If the Content Item meets the conditions, Rhythmyx passes the Content Item XML document to the exit that validates the item.

The validation exit must implement the `IPSResultDocumentProcessor` interface. The exit receives an XML document conforming to the `sys_ContentEditor` document. This document contains the parent editor information as well as all children. If item validation passes, the exit should return the same document. If it fails, the exit should return a new document conforming to the `sys_ItemValidation.dtd`. Use the `com.percussion.util.PSItemErrorDoc` class to create the error document.

Use the *Item Validation tab of the Content Editor Settings dialog* (see "[Content Editor Settings Item Validation Tab](#)" on page 19) to assign item-level validation exits to a Content Editor. For details about assigning a validation exit, see *Maintaining Content Editor Settings* (on page 24).

You can also assign item-level validation exits manually. The call to the Java exit appears in a `<PSXValidationRules>` element, which is a child of the `<PSXContentEditor>` element and a child of the `<PSXSharedFieldGroup>` element.

The following XML fragment shows an example of a call to a Java post-exit that performs item validation. Within the `<PSXValidationRules>` element, include a call to one or more Java exits in `<PSXExtensionCall>` elements.

```
<PSXValidationRules maxErrorsToStop="10">
  <PSXConditionalExit maxErrorsToStop="10">
    <PSXExtensionCallSet id="0">
      <PSXExtensionCall id="0">
        <name>Java/global/percussion/contenteditor/
          ItemValidationTest</name>
        <PSXExtensionParamValue id="0">
          <value>
            <PSXTextLiteral id="0">
              <text>3</text>
            </PSXTextLiteral>
          </value>
        </PSXExtensionParamValue>
      </PSXExtensionCall>
    </PSXExtensionCallSet>
  </PSXConditionalExit>
```

In `<PSXConditionalExit>`, the optional `<maxErrorsToStop>` attribute specifies the maximum number of errors that the exit must find before returning error messages. In `<PSXValidationRules>` the optional parameter `<maxErrorsToStop>` indicates the total number of errors that all item validation exits must find before the resource stops validating and returns the error page to the user. If the exits do not find the number of errors indicated in `<maxErrorsToStop>`, the resource completes the process of validating the entire item and returns any errors that it finds.

Sample Item Validation Exit

You can model your item validation exit on the following sample, which returns an error page.

The following example takes one parameter that specifies the number of validation errors to produce. It does not do an actual validation, but it shows how to produce an error document.

```

/*[ PSFixXmlRows.java
]*****
*
* COPYRIGHT(c)2001 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.ce;
import com.percussion.extension.IPSExtensionDef;
import com.percussion.extension.IPSResultDocumentProcessor;
import com.percussion.extension.PSExtensionException;
import com.percussion.extension.PSExtensionProcessingException;
import com.percussion.server.PSConsole;
import com.percussion.server.IPSRequestContext;
import com.percussion.util.PSItemErrorDoc;
import com.percussion.xml.PSXmlDocumentBuilder;
import java.io.File;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
/**
 * Sample item validation. Returns an error page. Does not include
validation code.
 */
public class PSItemValidationTest implements IPSResultDocumentProcessor
{
/**
 * Implementation of the method defined by the interface.
 *
 * @param params[0] the first and only parameter this exit takes
specifies
 * the number of item Validation errors that this test exit should
 * produce.
 */
public Document processResultDocument(Object[] params,
IPSRequestContext request, Document resultDoc)
throws PSExtensionProcessingException
{
Document doc = resultDoc;
try
{
if (params != null && params.length >= 1)
{
int errors = Integer.parseInt(params[0].toString());
if (errors > 0)

```

```
        {
            // create a new error document
            doc = PSXmlDocumentBuilder.createXmlDocument();
            for (int i=0; i<errors; i++)
            {
                String submitName = SUBMIT_NAME + " " + i;
                String displayName = DISPLAY_NAME + " " + i;
                Object[] args =
                {
                    submitName,
                    displayName
                };
                PSItemErrorDoc.addError(doc, submitName,
                    displayName, STRING_PATTERN, args);
            }
        }
    }
}
catch (Throwable t)
{
    PSConsole.printMsg("Exit:" + ms_fullExtensionName, t);
}
return doc;
}
/*
 * Implementation of the method defined by the interface
 */
public void init(IPSExtensionDef extensionDef, File file)
    throws PSExtensionException
{
    try
    {
        ms_fullExtensionName = extensionDef.getRef().toString();
    }
    catch (Throwable t)
    {
        throw new PSExtensionException(extensionDef.getRef(),
            t.getLocalizedMessage());
    }
}
/* Implementation of method defined by the interface */
public boolean canModifyStyleSheet()
{
    return false;
}
/* The fully qualified name of this extension. */
static private String ms_fullExtensionName = "";
/* The submit name prefix displayed in the error page.*/
private static String SUBMIT_NAME = "Submit Name";
/* The display name prefix shown on the error page. */
private static String DISPLAY_NAME = "Display Name";
/**
 * The item error message taking 2 parameters:
 * [0]: the submit name
 * [1]: the display name
 */
```

```
private static String STRING_PATTERN =
    "Item error for submit name {0}. The display name is: {1}";
```

Sample Error Page

The sample item validation exit returns the following error page, which conforms to `sys_ItemValidation.dtd`:

Item validation errors		
http://www.cdn-429923371mvs/aaItemValidation/cItemValidation.html?sys_command=preview&sys_contentid=30&sys_revision=1		
Field Display Name	Field Submit Name	Error Message
Display Name 0	Submit Name 0	Item error for submit name Submit Name 0. The display name is: Display Name 0
Display Name 1	Submit Name 1	Item error for submit name Submit Name 1. The display name is: Display Name 1
Display Name 2	Submit Name 2	Item error for submit name Submit Name 2. The display name is: Display Name 2

Figure 24: Sample Error Page

The link at the top of the page takes the user to the original item that caused the errors. The **Field Display Name** column lists the field that appears on the screen, the **Field Submit Name** lists the field that the content editor uses internally, and the **Error Message** column lists the error messages.

To change the appearance of the page, change the default stylesheet, which is specified in the following node in `ContentEditorSystemDef.xml`:

```
<CommandName>workflow</CommandName>
  <PSXParam name="com.percussion.defaultItemError">
    <DataLocator>
      <PSXTextLiteral id="274">

        <text>file:../rx_resources/stylesheet/errors/defaultItemError.xsl</text>
      </PSXTextLiteral>
    </DataLocator>
  </PSXParam>
```


CHAPTER 7

Visibility and Read-only Rules

Visibility and Read-only rules determine whether a user has access to each field in the Content Editor. Visibility rules control whether the Content Editor displays a field when rendering it. Read-only rules control whether a field that is displayed is available for editing or is rendered in read-only mode.

Visibility Rules

You may want to hide a field from users for any number of reasons. For example, Content Editors must include a field specifying the Workflow assigned to each Content Item. If you assign the workflow automatically, the user has no reason to see this field. Therefore, you can define Visibility rules that prevent the field from being displayed to the user. You may have other criteria that determine whether a user can see a specific field. For example, you may want to control visibility based on the State of the Workflow, the user's Role, or even the user's Community.

The PSXVisibilityRules child of the FieldRules element of the field definition is an optional element used to define Visibility rules. If this element is not present for a field, the field is automatically included in the output XML. If this element is present, you can define specific circumstances in which the field will be displayed.

The PSXVisibilityRules element requires at least one PSXRule child, which defines the set of conditions to be tested. If the conditions evaluate to *true*, the field will be included in the output XML. If the conditions evaluate to *false*, the inclusion of the field is defined by the value of the datahiding attribute of the PSXVisibilityRules element. If the value of this attribute is *xml*, the field will not be included in the output XML. If the value of this attribute is *xsl*, then the field is included in the output XML, and the XSL that controls the rendering of the field determines whether to display or hide the field.

The PSXVisibilityRules element can include multiple PSXRule children. The value of the boolean attribute of the PSXRule element defines boolean processing among multiple rules. This attribute can take the following values:

and

or

The *and* operator takes precedence over the *or* operator.

Example: Hiding the Workflow Field in the Content Editor System Definition

The field `sys_workflowid` is a required field defined in the Content Editor System Definition (`./rxconfig/Server/Content Editors/ContentEditorSystemDef.xml`). The following code shows the default definition of this field as installed.:

```
<PSXField defaultSearchLabel="Workflow" forceBinary="no"
modificationType="userCreate" name="sys_workflowid" showInPreview="yes"
showInSummary="yes">
  <DataLocator>
    <PSXBackEndColumn id="282">
      <tableAlias>CONTENTSTATUS</tableAlias>
      <column>WORKFLOWAPPID</column>
      <columnAlias/>
    </PSXBackEndColumn>
  </DataLocator>
  <OccurrenceSettings delimiter=";" dimension="optional"
multiValuedType="delimited"/>
  <FieldRules>
```

```

<PSXVisibilityRules dataHiding="xsl">
  <PSXRule boolean="and">
    <PSXConditional id="0">
      <variable>
        <PSXTextLiteral id="0">
          <text>1</text>
        </PSXTextLiteral>
      </variable>
      <operator>=</operator>
      <value>
        <PSXTextLiteral id="0">
          <text>2</text>
        </PSXTextLiteral>
      </value>
      <boolean>AND</boolean>
    </PSXConditional>
  </PSXRule>
</PSXVisibilityRules>
</FieldRules>
</PSXField>

```

Note the Visibility rule for this field, which tests whether the value 1 is equal to 2. Since this rule will always evaluate to false, the field will not be displayed in any Content Editor in the system. (Note also that the datahiding attribute of this PSXVisibilityRules element is *xsl*. The `sys_workflowid` field is required for processing of the Content Item, so it must always be included in the output XML. To avoid displaying this field, omit a template for it in the XSL.)

If you decide that you want to change the processing and show this field in all Content Editors, you can either remove the rule, or change the conditions to make `1=1`. This condition evaluates to true, and includes the field in the output XML.

Alternatively, you might want to make this field visible only in certain Content Editors. In that case, rather than modifying the system definition, you would override the definition of the `sys_workflowid` field in the individual Content Editor. The following code shows an example:

```

<PSXField defaultSearchLabel="Workflow" forceBinary="no"
modificationType="userCreate" name="sys_workflowid" showInPreview="yes"
showInSummary="yes">
  <OccurrenceSettings delimiter=";" dimension="optional"
multiValuedType="delimited"/>
  <FieldRules>
    <PSXVisibilityRules dataHiding="xsl">
      <PSXRule boolean="and">
        <PSXConditional id="0">
          <variable>
            <PSXTextLiteral id="0">
<text>1</text>
            </PSXTextLiteral>
          </variable>
          <operator>=</operator>
          <value>
            <PSXTextLiteral id="0">
              <text>1</text>
            </PSXTextLiteral>
          </value>

```

```
        <boolean>AND</boolean>
      </PSXConditional>
    </PSXRule>
  </PSXVisibilityRules>
</FieldRules>
</PSXField>
```

In the UI definition, you will also want to change the control used for this field. The default control for the field is `sys_hidden`. The recommended control for this field if visible is the `sys_DropDownSingle` control.

Read-only Rules

You may want to make a field visible, but only allow users to read it, not edit it. For example, you may want users at different stages in the Workflow to know the values in the specific field, but not allow them to change those values.

Read-only rules define whether a field that is displayed is eligible for editing. If you set a field to read-only, the text of the field is displayed inline as normal HTML text and is not eligible to be edited. Fields not set to read-only are rendered using the control specified for the field and are eligible to be edited.

To set a field to read-only, add a `ReadOnlyRules` child to the `PSXUISet` in the `DisplayMapping` for the field. The `ReadOnlyRules` element requires at least one `PSXRule` child. Use this child to specify the conditions under which the field will be set to read-only. If the conditions evaluate to true, the field will be rendered as read-only. If the conditions evaluate at false, the field will be rendered with the control defined for it in the field definition.

The `ReadOnlyRules` element can include multiple `PSXRule` children. The value of the boolean attribute of the `PSXRule` element defines boolean processing among multiple rules. This attribute can take the following values:

and

or

The *and* operator takes precedence over the *or* operator.

The following code illustrates a generic read-only rule:

```
<PSXDisplayMapping>
  <FieldRef>fieldname</FieldRef>
  <PSXUISet>
    <Label>
      <PSXDisplayText>Display Text:</PSXDisplayText>
    </Label>
    <PSXControlRef id="9999" name="sys_EditBox"/>
    <ReadOnlyRules>
      <PSXRule boolean="and">
        <PSXConditional id="1">
          <variable>
            <PSXTextLiteral id="1">
              <text>1</text>
            </PSXTextLiteral>
          </variable>
          <operator>IS NOT NULL</operator>
          <value>
            <PSXTextLiteral id="1">
              <text/>
            </PSXTextLiteral>
          </value>
          <boolean>AND</boolean>
        </PSXConditional>
      </PSXRule>
    </ReadOnlyRules>
  </PSXUISet>
</PSXDisplayMapping>
```

```
</PSXUISet>  
</PSXDisplayMapping>
```

Setting a Field in a Content Editor to Read-Only

In the Workbench, you cannot change an existing field in a Content Editor to read-only or add a new field to a Content Editor and set it as read-only. You must enter the Content Editor XML and add a `<ReadOnlyRules>` element.

To set a Content Editor field to read-only:

- 1 In `<Rhythmyxroot>\ObjectStore`, open the XML file for the content editor.
- 2 In the `<PSXDisplayMapping>` element, add `<ReadOnlyRules>` in the `<PSXUISet>` element. You can copy the `<ReadOnlyRules>` element that follows:

```
<PSXDisplayMapping>  
  <FieldRef>fieldname</FieldRef>  
  <PSXUISet>  
    <Label>  
      <PSXDisplayText>Display Text:</PSXDisplayText>  
    </Label>  
    <PSXControlRef id="9999" name="sys_EditBox"/>  
    <ReadOnlyRules>  
      <PSXRule boolean="and">  
        <PSXConditional id="1">  
          <variable>  
            <PSXTextLiteral id="1">  
              <text>1</text>  
            </PSXTextLiteral>  
          </variable>  
          <operator>IS NOT NULL</operator>  
          <value>  
            <PSXTextLiteral id="1">  
              <text/>  
            </PSXTextLiteral>  
          </value>  
          <boolean>AND</boolean>  
        </PSXConditional>  
      </PSXRule>  
    </ReadOnlyRules>  
  </PSXUISet>  
</PSXDisplayMapping>
```

CHAPTER 8

Text Extraction

Rhythmyx's text extraction feature lets you extract text from binary files created in third-party applications (for example, Adobe Acrobat PDFs) to create Rhythmyx Content Items. A Content Editor extension extracts the text and metadata in these files, formats them as text or HTML markup, and inserts the formatted data into Content Item fields. You can attach additional extensions to perform data translations or to insert text into other Content Editor fields.

The text extraction feature uses functionality provided by the Convera RetrievalWare software included with Rhythmyx when it is licensed to include the full text search. In order to use text extraction, you must install the Full Text Search feature when you install Rhythmyx. For information about the Full Text Search feature, see "Searching for Content Using the Full Text Search Engine" in the online *Rhythmyx Content Explorer Help*.

Implementing Text Extraction in Rhythmyx

The process of performing text extraction involves downloading external binary files to a Content Editor in Rhythmyx that is configured with the text extraction exit and optionally, other input translation exits.

To implement text extraction in Rhythmyx:

- 1** *Configure a method for uploading the external binary files to Rhythmyx* (see "[Uploading External Binary Files into Rhythmyx](#)" on page 93).
- 2** *Create a Content Editor that extracts text* (see "[Creating a Content Editor that Extracts Text](#)" on page 94).

Uploading External Binary Files into Rhythmyx

To apply text extraction to external binary files, add the `sys_TextExtraction` exit to the Content Editor application that will perform text extraction. The Content Editor must include a field that stores the binary file and a field that stores the data extracted from the file.

In some cases it is most efficient to upload files individually through a file upload control in a Content Editor. In other scenarios, it is most efficient to upload one or more binary files to Rhythmyx by inserting or storing them in a WebDAV-enabled folder. In this case, you must edit the WebDAV configuration file associated with the folder to convert the file into the Rhythmyx Content Type that performs text extraction.

Note: The `sys_TextExtraction` exit does not limit the size of text extracted. However, a user's browser, Web Server, database, ODBC driver, or `sys_EditLive` settings may limit the size of content in the field specified to store extracted text. An error message alerts users and prevents them from saving the Content Item if the text extracted exceeds the field's size limitation.

For information about adding a file-upload control to a field in a Content Editor, see the topic “[sys_file](#) (on page 183)” in the Workbench help set.

For information about using and configuring WebDAV with Rhythmyx see the document, *Implementing WebDAV in Rhythmyx*.

Creating a Content Editor that Extracts Text

To create a Content Editor that extracts text:

- 1 Create a Content Editor in the Rhythmyx Workbench.
- 2 Add or choose fields to store the uploaded file, the extracted data, an extraction error message, and optionally, the file type. Use the following guidelines:
 - Create a field to upload and store a file. Use the properties:
 - **Control Name:** `sys_file` (to store binary files)
 - **Data Type:** `binary`
 - **Format:** `max`
 - Create a field to store the file type. Note: The `sys_FileInfo` exit extracts and stores the file type; it requires the name of this field to be the name of the field that stores the uploaded file concatenated with `_type` (for example, `fileupload` and `fileupload_type`). Use the properties:
 - **Control Name:** `sys_EditBox`
 - **Data Type:** `text`
 - **Format:** `50` (This value is large enough to store long Mime Type names).
 - (Optional) Create a field to hold error text. Use the properties:
 - **Control Name:** `sys_EditBox`
 - **Data Type:** `text`
 - **Format:** `255` (This value is large enough to store long error messages.)
 - Create a field to hold the extracted text:

If you want the extracted text to include HTML formatting, use the properties:

- **Control Name:** `sys_EditLive` (You can also use `sys_TextArea`. You may encounter size limitations if you use `sys_EditBox`.)
- **Data Type:** `text`
- **Format:** `max` (You can specify a numeric size if you add an exit that truncates the data to a specific size.)

If you want the extracted text to be formatted as text, use the properties:

- **Control Name:** sys_TextArea
- **Data Type:** text
- **Format:** max (You can specify a numeric size if you add an exit that truncates the data to a specific size.)

For examples of how output formats and Content Editor controls affect the appearance of the text in your Content Editor, see *Displaying Extracted Text in a Content Editor* (on page 97).

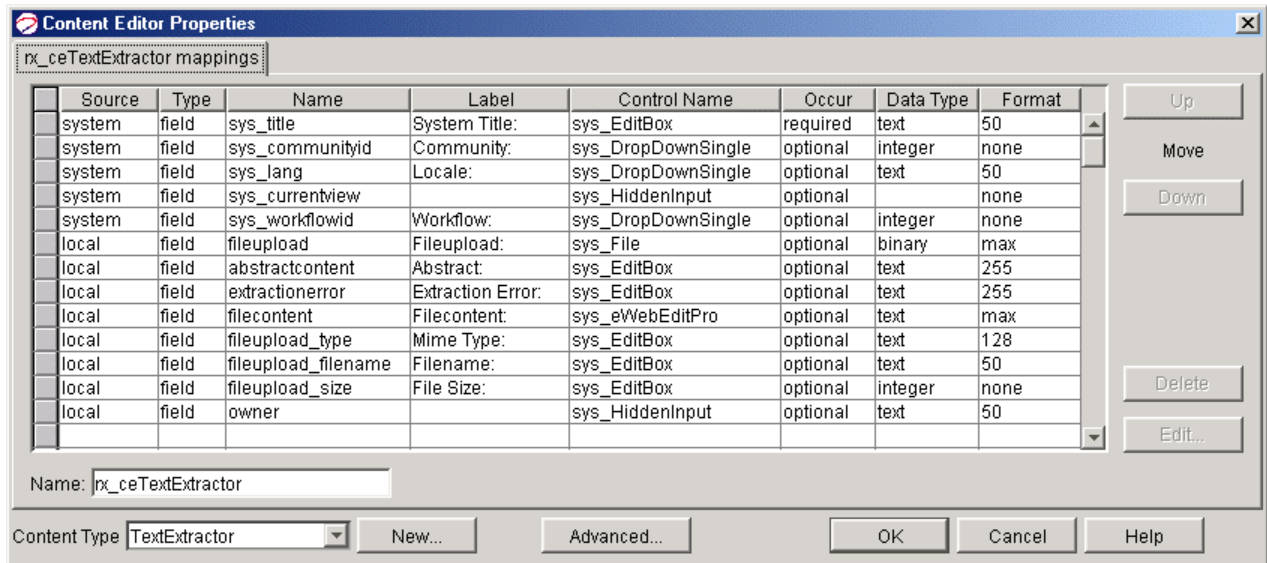


Figure 25: Content Editor Properties for an example Text Extraction Content Type

- 3 Click [**Advanced**] to open the Content Editor Settings dialog.
- 4 In the Item Input Translation tab:

- Choose the *sys_TextExtraction* (on page 98) extension. Configure the extension parameters for the specific text extraction.

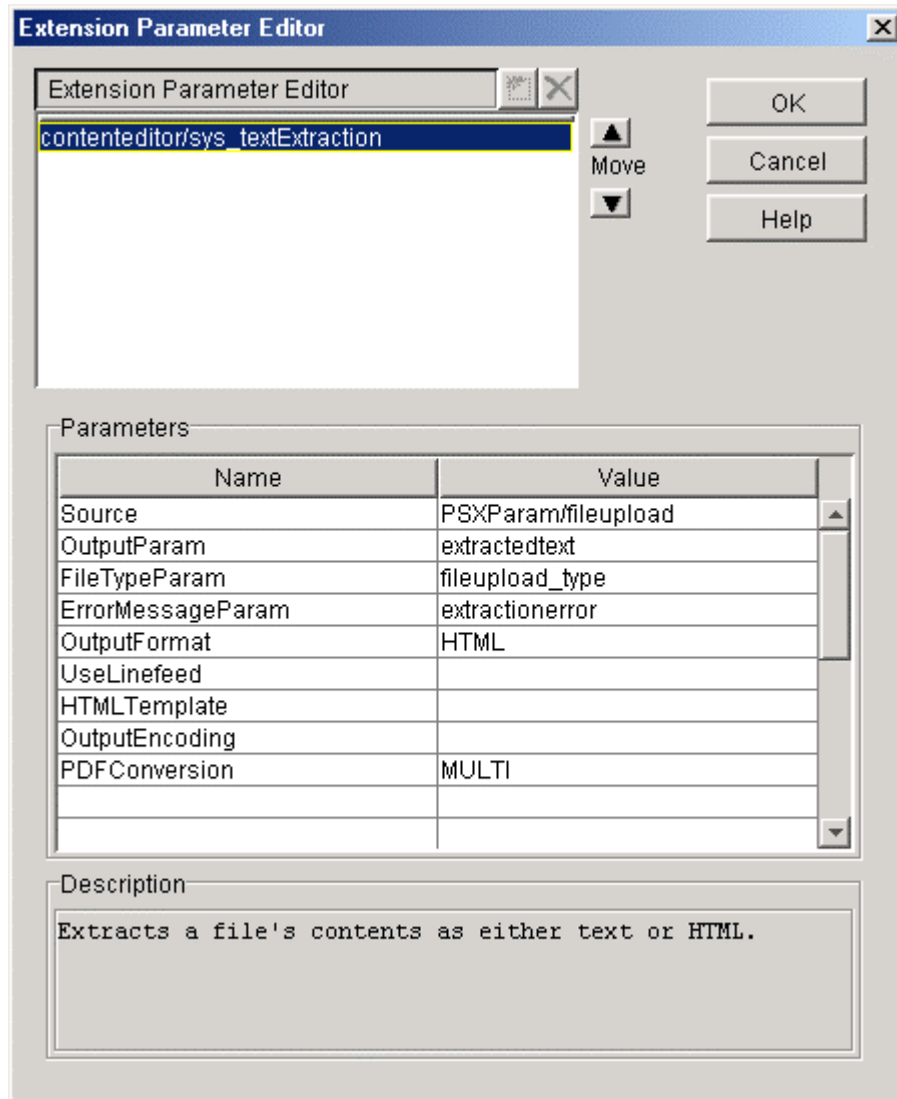


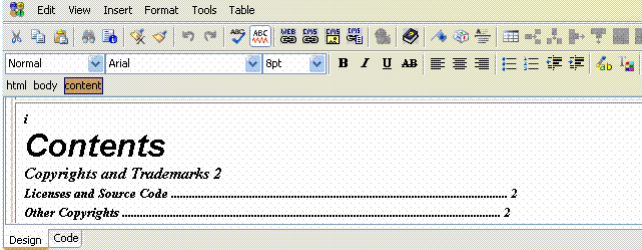
Figure 26: Example configuration of the *sys_TextExtraction* exit

These parameter values correspond to the fields in the Content Editor shown in the graphic for Step 2.

Optionally, choose additional extensions to translate data in the uploaded file before it is inserted into the Rhythmyx Content Item. When you add the *sys_File* control, Rhythmyx automatically includes the *sys_FileInfo* exit. This exit inserts the filename, file size, and file extension into Content Editor fields.

Displaying Extracted Text in a Content Editor

The way the extracted text appears in a Content Editor depends on the type of control that stores the extracted field and the output format specified in the `sys_TextExtraction` exit. The following table shows the same extracted text displayed using three different combinations of Content Editor controls and output formats.

Control Name for Extracted Text	Output Format for Extracted Text	Extracted Text in Content Editor
sys_EditLive	HTML	
sys_TextArea	HTML	<p>Extracted Text:</p> <pre data-bbox="760 1146 1349 1220"> size="6">Contents</p><p>Copyrights and Trademarks 2 </p><p>Licenses and Source </pre>
sys_TextArea	TEXT	<p>Extracted Text:</p> <pre data-bbox="760 1293 1349 1360"> Contents Copyrights and Trademarks 2 Licenses and Source Code </pre>

sys_TextExtraction

Name: sys_TextExtraction

Context: Java/global/percussion/contenteditor/

Description: This pre-exit extracts the text and metadata in a binary file uploaded to a Rhythmyx Content Editor and inserts the extracted data into a Content Editor field (or fields). The exit formats the extracted text as text or HTML markup. For information about performing text extraction with this exit, see *Text Extraction* (on page 91) in the document *Implementing Content Editors*.

Class name: com.percussion.content.PSFileConverterExit

Interface: com.percussion.extension.IPSRequestPreProcessor

Parameters

Name	Data Type	Description
Source	java.lang.String	Source file parameter. Enter the parameter that holds the source file. Required. Note: If a file upload control uploads the file, it inserts the file object into the Content Editor field. If Web Services upload the file (if you use WebDAV), they insert the base64 encoded data contained in the file into the Content Editor field. Therefore, if the Content Editor field does not hold a file object, the exit assumes it is base64 encoded data and treats it as such.
OutputParam	java.lang.String	Name of a parameter or the Content Editor field that stores the extracted data. Required.
FileTypeParam	java.lang.String	Name of a parameter or the Content Editor field that stores the original file's Mime type. Optional.
ErrorMessageParam	java.lang.String	Name of a parameter or the Content Editor field that stores error messages. When used, the Content Item is saved. Optional, but if not supplied, the extension throws exceptions for errors and does not save the Content Item. Note: If you are updating a Content Item, and you specify this field, if an error occurs, the exit saves the changed Content Item and the originally extracted text is lost.
OutputFormat	java.lang.String	Either TEXT or HTML. Case-insensitive. TEXT– Formats the output as plain text. HTML– Adds HTML tags to the output to attempt to duplicate its appearance in the original file.
UseLinefeed	java.lang.String	Indicates how to process line endings when OutputFormat is text. Optional. Y or y – Process line endings as line feeds. Other values/not specified – Process line endings as carriage returns.

Name	Data Type	Description
HTMLTemplate	java.lang.String	<p>Optional override for path of template that converts text to HTML. Default is: <Rhythmyx root>/sys_search/rware/rx/resource/sys_html_rx.tpt</p> <p>Note: For advanced users only. This feature is not documented.</p>
OutputEncoding	java.lang.String	<p>Encoding to use for output character set. Default is WINDOWS-1252 for text and UTF-8 for HTML. UTF-8 works for all HTML output character sets. If your OutputFormat is text and you are using a multi-byte character type, you must specify the correct output encoding. Valid values are:</p> <p>WINDOWS-1252 – standard Windows encoding Shift_JIS – encoding for Japanese characters EUC_KR – encoding for Korean characters GB2312 – Encoding for Simple Chinese characters Big5 – Encoding for traditional Chinese characters</p> <p>Note: Multi-byte characters are commonly used to represent ideograms in Asian languages such as Chinese.</p>
PDFConversion	java.lang.String	<p>How to perform conversion if file type is .pdf. Optional.</p> <p>SINGLE – (default) Does not process multi-byte characters; creates output using system default character set; can process multiple columns in the source pdf file; ignores OutputFormat specified and uses text.</p> <p>MULTI – Process multi-byte characters; creates output using OutputEncoding, if supplied; cannot process multiple columns in the source pdf file; uses OutputFormat specified.</p> <p>Note: Multi-byte characters are commonly used to represent ideograms in Asian languages such as Chinese.</p>

Continuous Conversion Example

In continuous conversion, Rhythmyx publishes content that a user has created in another application and wants to continue modifying in the other application. When the file is downloaded to Rhythmyx the body data is extracted as text and inserted into a Content Editor field. However, the file contents are not modified in Rhythmyx; they are always modified in the third-party application. After modification, Rhythmyx reloads the files and updates the original Content Items.

In our example, a Marketing Department maintains a library of PDF documents for customers. On the company Web Site, the Department wants to include a list of the documents that links to their contents. When the content of a PDF changes, the Department uploads the updated PDF to Rhythmyx again. Rhythmyx automatically updates the content of the original Content Item.

As an implementer in the Marketing Department, you use the following procedure to implement continuous conversion of the PDFs:

- 1 In the Rhythmyx Workbench, create a Content Editor for a Content Type to store the uploaded PDF files. It includes fields for the uploaded file, the extracted text, the file type, and an error message if file upload fails. The procedure for creating the Content Editor is as follows:
 - a) Choose the `sys_Default.xml` template and name the new resource `rx_TextExtract`.
 - b) Double-click the template to open the Content Editor Properties dialog and add the fields:
 - `fileupload` – The `sys_TextExtraction` exit will use `fileupload` to store the uploaded file. Properties: Control Name=`sys_file`, Data Type=`binary`, Format=`max`.
 - `fileupload_type` – The `sys_TextExtraction` exit and WebDAV will use `fileupload_type` to store the mime type. Properties: Control Name=`sys_EditBox`, Data Type=`text`, Format=`50`.
 - `extractionerror` - The `sys_TextExtraction` exit will use `extractionerror` to store the text of the first error encountered during extraction. Properties: Control Name=`sys_editBox`, Data Type=`text`, Format=`255`.
 - `filecontent` - The `sys_TextExtraction` exit will use `filecontent` to store the extracted text. Properties: Control Name=`sys_EditLive`, Data Type=`text`, Format=`max`.
 - `owner` - WebDAV will use `owner` to store the user that has the file locked. Properties: Name=`sys_HiddenInput`, Data Type=`text`, Format=`50`.
 - `fileupload_size` – WebDAV will use `fileupload_size` to hold the file size. Properties: Control Name=`sys_EditBox`, Data Type – `integer`, Format=`none`.
 - `abstractcontent` – This field can display an abstract of the Content Item; it is not required by `sys_TextExtraction` or WebDAV. Use either a custom input translation exit to fill it or allow the user to fill it through the Content Editor. Properties: Control Name=`sys_EditBox`, Data Type=`text`, Format=`255`.
 - `fileupload_name` – This field displays the uploaded file name. Include it for your own reference. It is populated by the `sys_file` control. Properties: Control Name=`sys_EditBox`, Data Type=`text`, Format=`50`.

c) Click [New] and specify the Content Type TextExtractor.

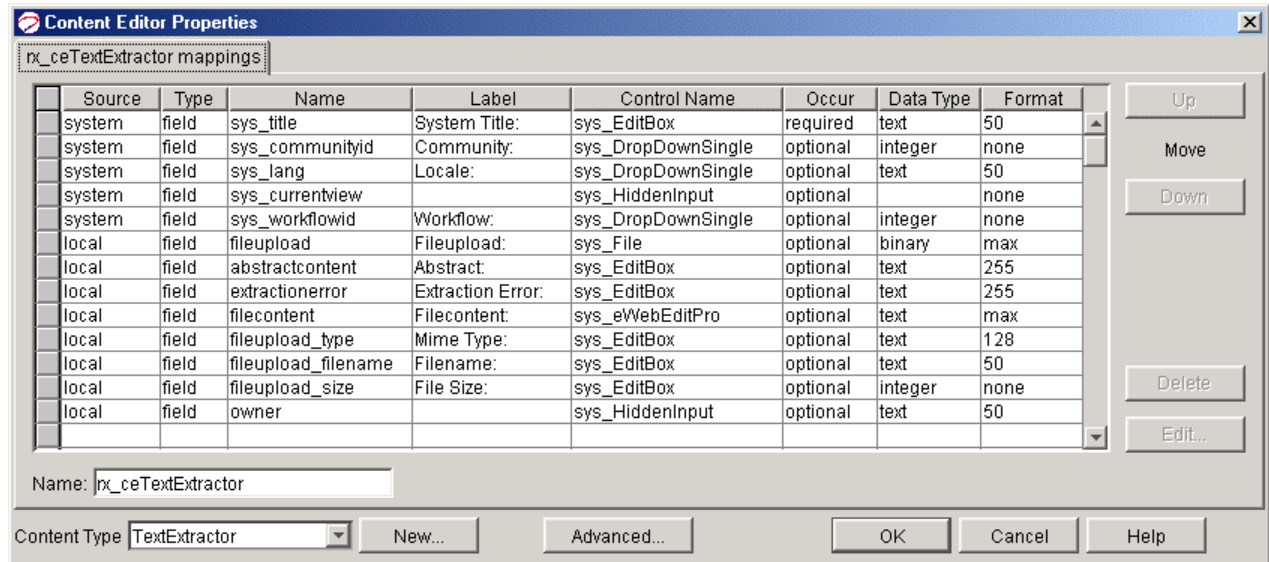


Figure 27: Content Editor Properties for Text Extractor

- d) Click [Advanced] to open the Content Editor Settings dialog.
- e) On the Item Input Translation tab, choose the sys_textExtraction extension.

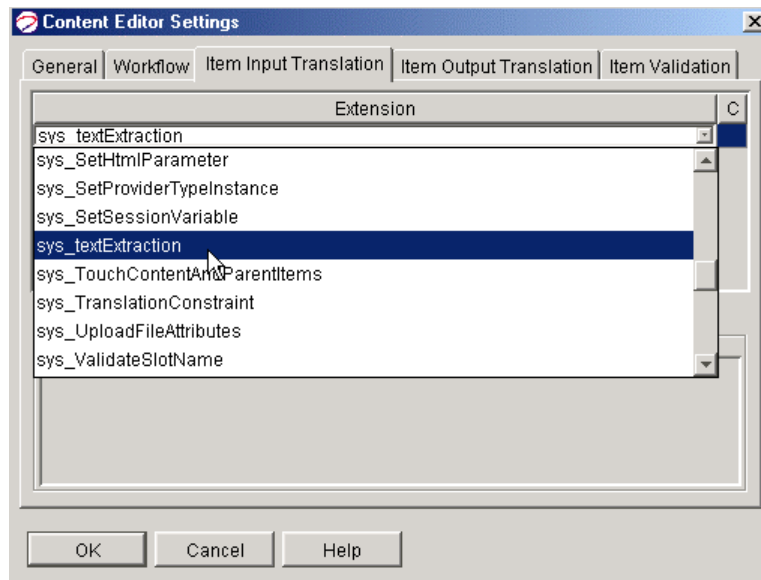


Figure 28: Choosing sys_textExtraction in the Content Editor Settings dialog

Set the following values for the parameters:

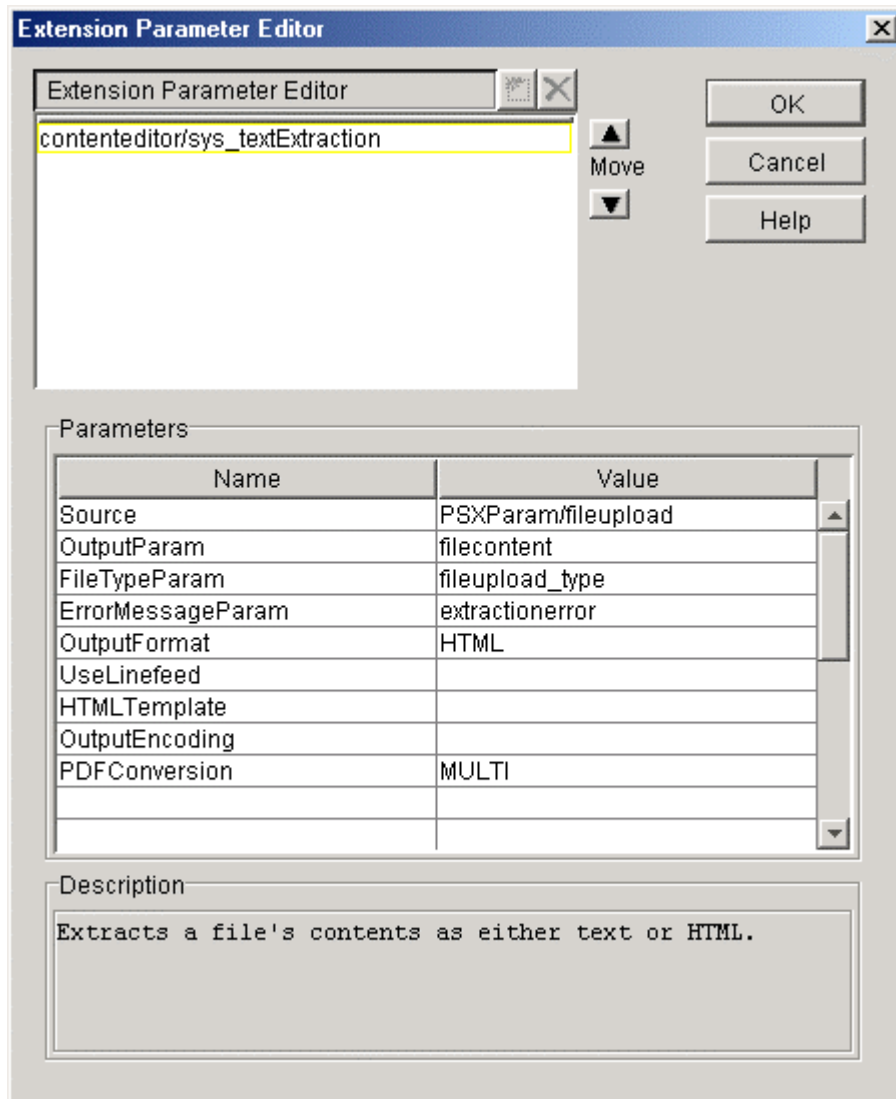


Figure 29: Parameter Settings for sys_textExtraction

Name	Value	Description
Source	PSXParam/fileupload	The field in the Content Editor that holds the uploaded file. May be expressed as any value type.
OutputParam	filecontent	The field in the Content Editor that stores the extracted data.
FileTypeParam	fileupload_type	The field in the Content Editor that stores the file type.

Name	Value	Description
ErrorMessageParam	extractionerror	The field in the Content Editor that stores the error message if data does not extract. By inserting the error message into a field in the Content Editor, you let the text extraction process convert all files, even if some do not convert properly.
OutputFormat	HTML	Specifies that converted data is formatted as HTML. Since the converted data is inserted into an HTML editor control, HTML is necessary so the control can format the data as closely as possible to its original appearance.
UseLineFeed	---	Not used because OutputFormat is HTML.
HTMLTemplate	---	Not used because standard HTML template is not overridden.
OutputEncoding	---	Not specified because default character encoding is used.
PDFConversion	Multi	multi is specified so OutputFormat can be HTML

Add a condition to the exit specifying that the exit only runs when a file is uploaded. If you do not include this condition, if a user edits metadata fields and saves the Content Item, the `sys_textExtraction` exit attempts to run and results in an error.

The condition specifies that the field that stores the uploaded file IS NOT NULL:

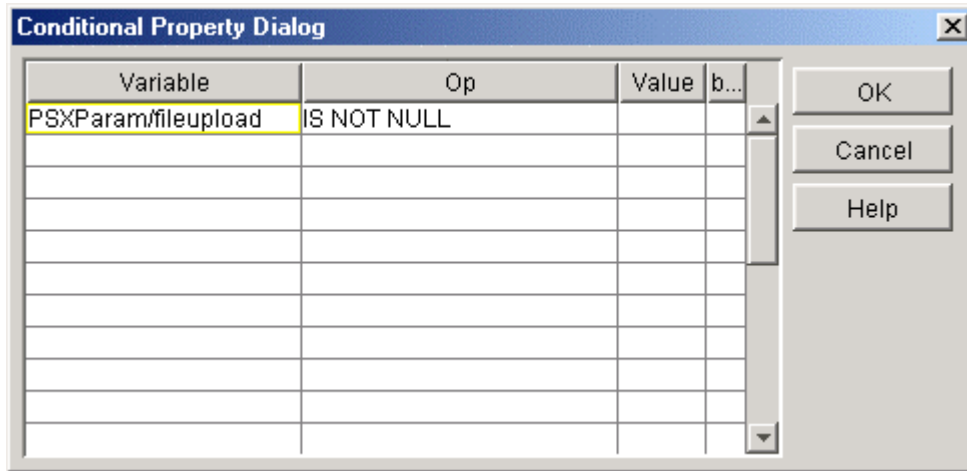


Figure 30: Conditional setting for `sys_textExtraction`

- f) At this point, you can add custom input translation exits before or after the `sys_textExtraction` exit to perform additional processing. For example, you may add an exit to parse the first sentence from the extracted data and insert the parsed text into the `abstractcontent` field.
- g) Save the Content Editor as `rx_ceTextExtractor` and close it.

NOTE: The new Content Type is automatically registered in Rhythmyx when you save the Content Editor, but you must associate it with the Communities that you want to have access to it.

- 2 Define a servlet named *Marketing PDFs* in the WebDAV deployment descriptor, `<Rhythmyx root>/AppServer/webapps/rxwebdav/WEB-INF/web.xml`:

```
<servlet>
  <servlet-name>Marketing PDFs</servlet-name>
  <display-name>Rhythmyx WebDAV Router</display-name>
  <description>Rhythmyx WebDAV Router</description>
  <servlet-class>com.percussion.webdav.PSWebdavServlet</servlet-
class>
  <init-param>
    <param-name>RxWebDAVConfig</param-name>
    <param-value>/RxWebdavConfig.xml</param-value>
    <description>The webdav configuration file path, which is
relative to the current web application</description>
  </init-param>
</servlet>
```

Also, add a servlet-mapping element for the new servlet in the WebDAV deployment descriptor:

```
<servlet-mapping>
  <servlet-name>Marketing PDFs</servlet-name>
  <url-pattern>/Marketing PDFs/*</url-pattern>
</servlet-mapping>
```

- 3 Edit the WebDAV configuration file, <Rhythmyx root>\AppServer\webapps\rxwebdav\RxWebdavConfig.xml, as follows. Only a default Content Type is necessary because all of the input files will be PDFs and they will all be converted to TextExtract Content Types.

```
<?xml version="1.0" encoding="UTF-8"?>
<PSXWebdavConfigDef root="//Folders/Marketing PDFs"
communityname="default" communityid="10" locale="en-us">
  <PSXWebdavContentType id="301" name="TextExtractor"
contentfield="fileupload" ownerfield="owner" default="true">
    <PropertyMap>
      <PSXPropertyFieldNameMapping name="getcontenttype">
        <FieldName>fileupload_type</FieldName>
      </PSXPropertyFieldNameMapping>

      <PSXPropertyFieldNameMapping name="getcontentlength">
        <FieldName>fileupload_size</FieldName>
      </PSXPropertyFieldNameMapping>
    </PropertyMap>
  </PSXWebdavContentType>
</PSXWebdavConfigDef>
```

- 4 Set up WebDAV-enabled folders in Rhythmyx Content Explorer and Windows Explorer for storing the PDFs.



Figure 31: WebDAV-enabled Folder in Content Explorer



Figure 32: Web Folder in Windows Explorer

- 5 Copy the PDFs into the WebDAV-enabled folder on Windows Explorer.

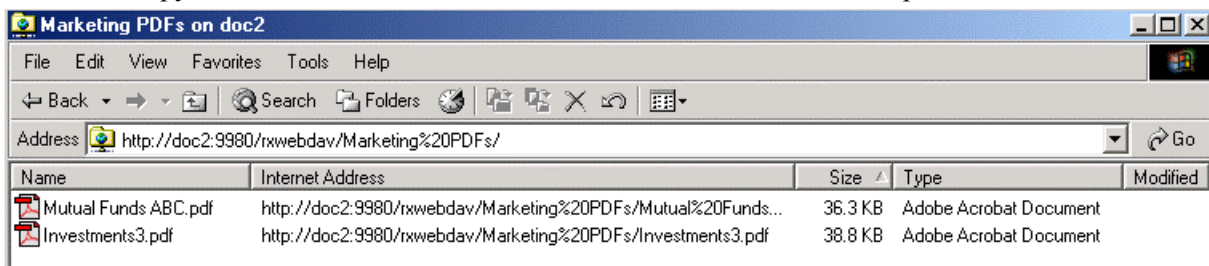
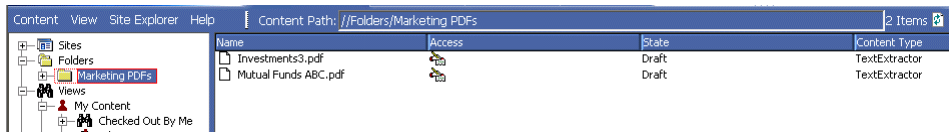


Figure 33: PDF files in Web Folder

They now exist in the WebDAV-enabled folders in Content Explorer, and Rhythmyx automatically converts them into TextExtractor Content Items.



An opened TextExtractor Content Item appears as:

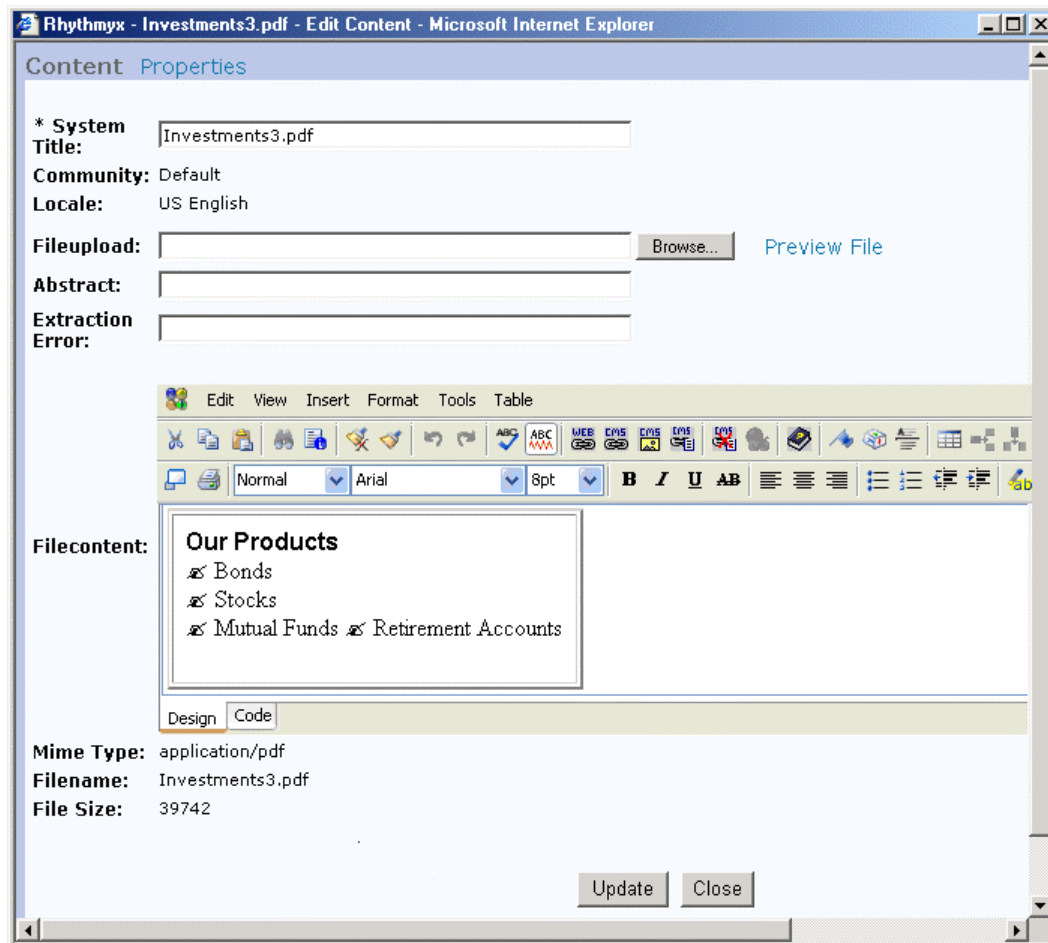


Figure 34: Opened Text Extractor Content Item

- 6 In this example, the body content of files is not edited in Rhythmyx (depending on your system requirements, you may choose to edit metadata fields). The original file is updated in its native application and re-uploaded to Rhythmyx, where it will automatically overwrite the originally uploaded Content Item.

If a content creator updates the file saved as Investments3.pdf in the original application, and recopies the file into the Marketing PDFs folder under My Network Places in Windows Explorer, the changed file overwrites the original Investments3.pdf. When you open it in Rhythmyx, the Filecontent field displays the changed content:

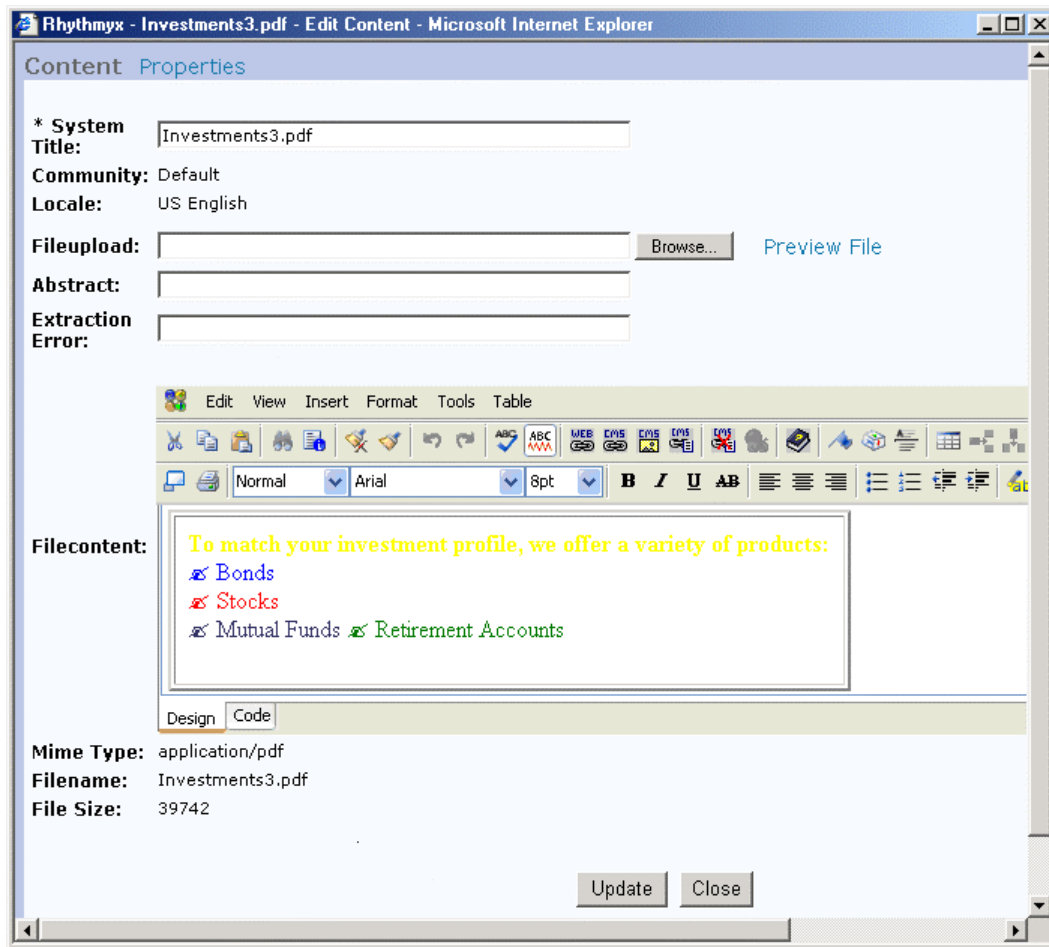


Figure 35: Content updated in original application

Migration Example

In migration, text extraction transforms binary files that existed prior to Rhythmyx implementation into XML Content Items. When the files are uploaded to Rhythmyx, some of the data is extracted and inserted into Content Editor fields. In the future, these Content Items will be edited in Rhythmyx; they will not be returned to the original third-party application for editing.

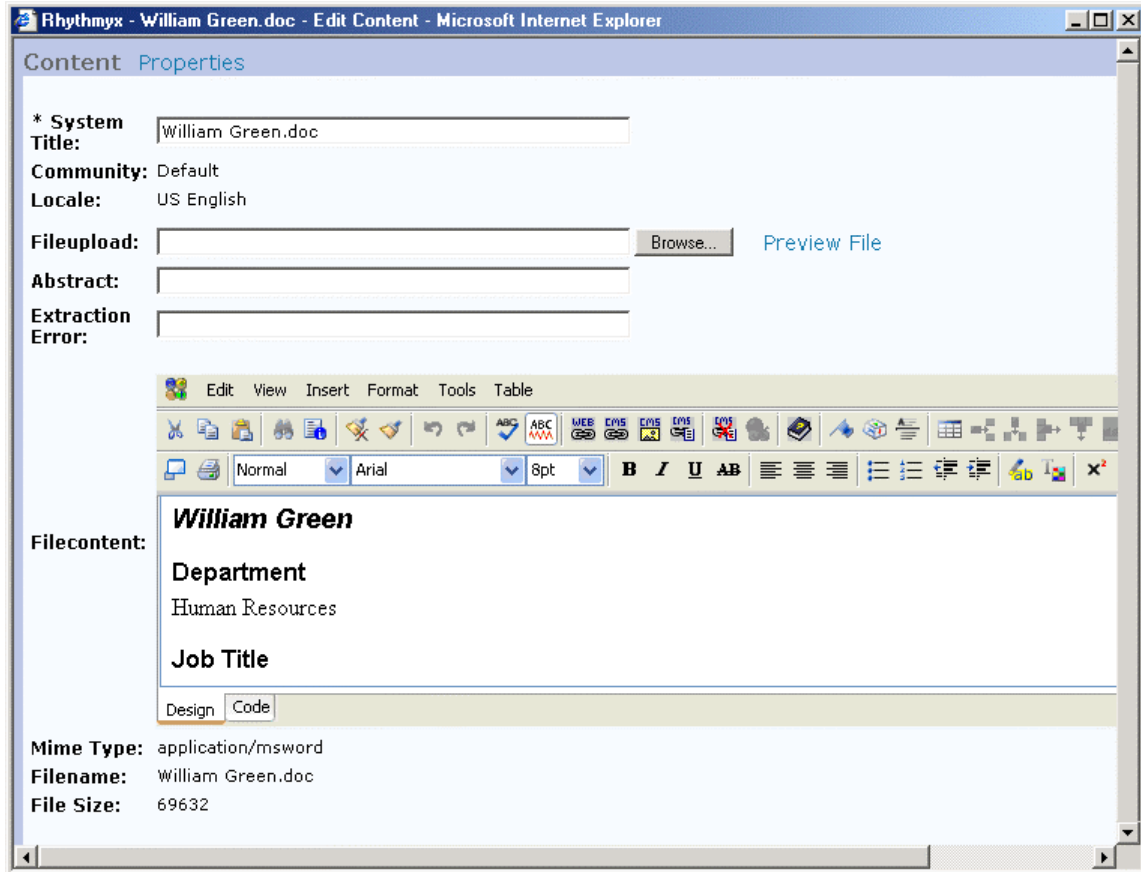


Figure 36: Uploaded Text Extractor Content Item edit in Rhythmyx

In our example, a company has stored its employee profiles in Word doc files prior to implementing Rhythmyx. Now that the company has implemented Rhythmyx, it wants to convert these files into Rhythmyx Content Items and have the ability to process them in the future as Rhythmyx Content Items.

The procedure for implementing text extraction for migration of the Employee Profiles is essentially the same as the procedure for implementing continuous conversion of Marketing PDFs:

- 1 Follow Step 1 of the *Continuous Conversion Example* (on page 100) to set up a TextExtractor Content Editor to store the uploaded Word files. Since users may edit the Filecontent field in Rhythmyx, errors will result if you do not include the conditional described in this step. *Note:* You can use the same procedure because the TextExtractor Content Editor does not specify a specific type of file.

- 2 Follow Step 2 of the *Continuous Conversion Example* (on page 100) [link] to define a Servlet in the WebDAV deployment descriptor. Give the Servlet the name Employee Profiles.
- 3 Follow Step 3 of the *Continuous Conversion Example* (on page 100) to edit the WebDAV configuration file, but set *root* to:


```
root="//Folders/Employee Profiles"
```
- 4 Set up WebDAV-enabled folders in Rhythmyx Content Explorer and Windows Explorer for storing the Word files.



Figure 37: WebDAV-enabled Folder in Content Explorer

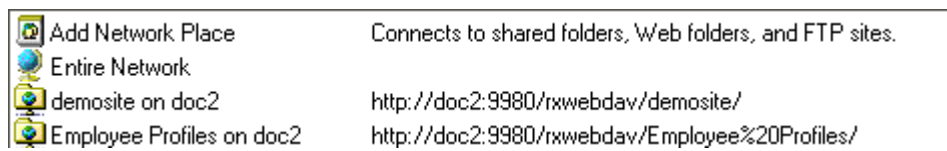


Figure 38: Web Folders in Windows Explorer

- 5 Copy the Word docs into the Web Folder on Windows Explorer.

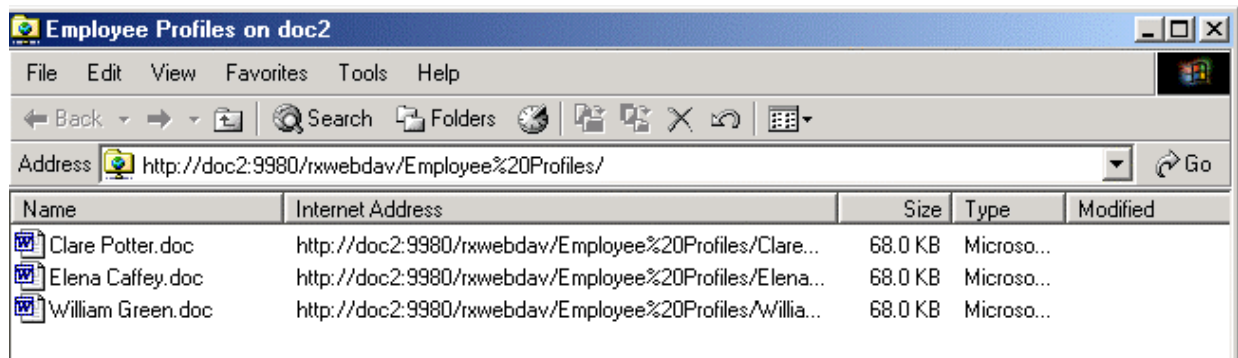


Figure 39: Word document in Web Folder on Windows Explorer

They now exist in the WebDAV-enabled folders in Content Explorer, and Rhythmyx automatically converts them into TextExtractor Content Items.



Figure 40: WebDAV-enabled Folder in Rhythmyx

- 6 In this example, the file has become a Rhythmyx Content Item, so its body content is edited in Rhythmyx.

A Content Creator may open the file in Rhythmyx, edit it, and save it.

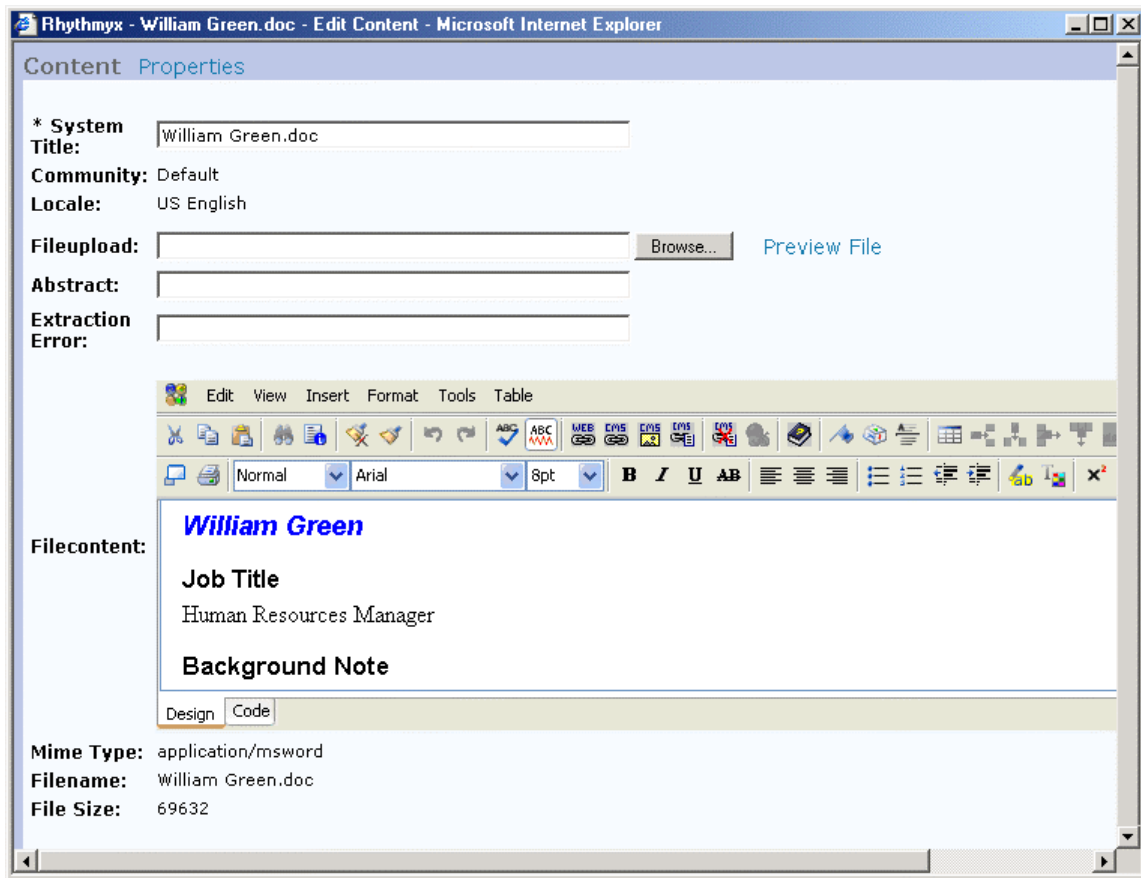


Figure 41: Uploaded Text Extractor Content changed

Customizing the ArticleWord Content Editor

Rhythmyx provides the Article Word Content Editor as the default Content Editor that uses the Rhythmyx Word Connector. This topic explains how to use the Article Word Content Editor. The next topic explains how to customize it to create your own Word-based Content Editor.

The default Article Word editor opens a Word document with the `rxwordsample.doc` template. This template is stored on the Rhythmyx server in the `..\rx_ceArticleWord` directory.

NOTE: The deprecated template is `..\sys_resources\word\Rhythmyx.doc`.

This template includes the following styles:

- ArticleAbstract
- ArticleAuthor
- ArticleDisplayTitle
- ArticleBody

Each of these styles is associated with a field in the Article Word content editor file (`articleword.xml`) and a column in the `RXARTICLEWORD` backend table. The content that the user enters in each style in the Word document becomes the value stored in the corresponding column in the `RXARTICLEWORD` table.

Rhythmyx saves files uploaded using the *Save to Rhythmyx* option as `.doc` files regardless of the file suffix indicated by the user. It saves the files to the `BODYSOURCE` column in the `RXARTICLEWORD` table.

Rhythmyx uses an ActiveX control to download content items from the repository to MS Word, and a Word Macro to upload the files to the Rhythmyx server.

For information about using the default template styles and configuring a system to use ActiveX, see "Word Prerequisites" in the Rhythmyx Content Explorer online Help.

How Word-based Content Editors Work

The Rhythmyx Word Connector uses Word-based Content Editors to let users create Content Items in Word and upload them to Rhythmyx. Users can then access Word from Rhythmyx to edit these Content Items.

A Word-based Content Editor opens a Word document with the `rxwordsample.dot` template. This template is stored on the Rhythmyx server in the `..\sys_resources\word\` directory.

NOTE: The deprecated template is `..\sys_resources\word\Rhythmyx.dot`.

By default, this template includes the following styles:

- ArticleAbstract
- ArticleAuthor
- ArticleDisplayTitle
- ArticleBody

Each of these styles (or each of the styles you include) is associated with a field in the Word-based Content Editor. The content that the user enters in each style in the Word document becomes the value stored in the corresponding column in the backend table for the Content Editor.

The template causes various Rhythmyx features to appear in Word including a *Save to Rhythmyx* option. Rhythmyx saves files uploaded using the *Save to Rhythmyx* option as `.doc` files regardless of the file suffix indicated by the user. It saves the files to a column named BODYSOURCE in the backend table.

Rhythmyx uses an ActiveX control to download content items from the repository to MS Word, and a Word Macro to upload the files to the Rhythmyx server.

For information about using the default template styles and configuring a system to use ActiveX, see "Word Prerequisites" in the Rhythmyx Content Explorer online Help.

How to Create a Word-based Content Editor

To create a Word-based Content Editor, follow the usual procedure for creating a new Content Editor in the Rhythmyx Workbench and add the fields and files that the Word Connector requires:

- 1 Create a new Content Editor and add shared and system fields. See *Creating a Content Editor from Scratch* (on page 21) in the *Implementing Content Editors* document for help adding new fields.
- 2 Add fields that will be entered in Word as local fields with a **data type** of text. See the section *New Field Properties Dialog* (on page 33) in the *Implementing Content Editors* document for help.
- 3 Rhythmyx uses the `sys_FileWord` control to upload files from Microsoft Word. The `sys_FileWord` control uses the fields `bodysource`, `bodysource_encoding`, `bodysource_filename`, and `bodycontent` to upload and store information. Add these as local fields and define them as follows:

Name	Control	Data Type	Format
<code>bodysource</code>	<code>sys_FileWord</code>	binary	max
<code>bodysource_encoding</code>	<code>sys_HiddenInput</code>	text	50
<code>bodysource_filename</code>	<code>sys_EditBox</code>	text	50
<code>bodycontent</code>	<code>sys_HiddenInput</code>	text	max

- 4 On the Content Editor application, attach the following pre-exits with the specified parameters in the order listed:

Exit	Parameter Name	Parameter Value
generic/sys_copyParameter	source	<code>bodysource_filename</code>
	destination	filename
generic/sys_copyParameter	source	<code>bodysource_clear</code>
	destination	<code>body_clear</code>
xmlDOM/sys_xdTextToDom	sourceName	<code>bodysource</code>
	DOMname	XMLDOM
	tidyProperties	<code>rxW2Ktidy.properties</code>
	serverPageTags	<code>rxW2KserverPageTags.xml</code>
	encodingDefault	UTF8
xmlDOM/sys_xdTransformDOMToText	Validate	(leave blank)
	SourceName	XMLDOM
	StyleSheet	<code>parsebody.xsl</code>
	DestName	body

Exit	Parameter Name	Parameter Value
xmldom/sys_xdTransformDOM	SourceName	XMLDOM
	StyleSheet	parsemeta.xsl
	DestName	XMLDOM
xmldom/sys_xdDOMToParams	SourceName	XMLDOM
	AppendParameter	(leave blank)

- 5 Include the following parameters and values as links (external) with the `sys_FileWord` control. Replace the application names with your Content Editor application name, the resource names with your Content Editor resource name, and the template name with your template name:

Parameter	Value
ContentBodyURL	<code>sys_MakeAbsLink(..\rxs_GenericWord_ce/genericword.html, sys_contentid, PSXParam/sys_contentid, sys_revision, PSXParam/sys_revision, sys_command, binary, sys_submitname, bodysource, psessionid, PSXUserContext/SessionId)</code>
RxContentEditorURL	<code>sys_MakeAbsLink(..\rxs_GenericWord_ce/genericword.xml, sys_contentid, PSXParam/sys_contentid, sys_revision, PSXParam/sys_revision, sys_command, edit, psessionid, PSXUserContext/SessionId)</code>
WordTemplateURL	<code>sys_MakeAbsLink(..\rxs_GenericWord_ce/rxs_word.dot, psessionid, PSXUserContext/SessionId, , , , ,)</code>

- 6 Save the Content Editor.
- 7 In the folder for the Content Editor in the Rhythmyx root, copy the `contentfilter.xsl`, `ParseMeta.xsl` and `ParseBody.xsl` files from `<Rhythmyx root>\sys_resources\word`.
- 8 **Create a Word template (.dot) file** (see "[Creating the Word Template File](#)" on page 116).
- 9 **Modify the `ParseMeta.xsl` or `ParseBody.xsl` files to reflect the new Word styles and fields added in the .dot and XML files.** (see "[Modifying the Style Sheet for Parsing Fields](#)" on page 116)
- 10 **Modify formatting of the body content fields so that they display correctly on assembled pages.** (see "[Modifying the Content Assembler to Display Custom Word-based Content Editor Fields](#)" on page 118)

Creating the Word Template File

Creating a Word-based Content Editor involves specifying fields that will be filled by information entered in Word. The Word template file must include a style for each of these fields.

Copy the current Word template file:

```
..\sys_resources\word\rxwordsample.dot
```

and store it with the name of the new content editor in the folder for the Content Editor in the Rhythmyx root, for example:

```
..\rxWord_ce\rxWord.dot
```

Open the template file in Word and do the following:

- 1 Add a style to correspond with each field in your new Content Editor that you want to be entered in Word. You may specify more than one style for your body field.

By default, rxwordsample.dot includes the styles:

- ArticleDisplayTitle
- ArticleAuthor
- ArticleBody
- ArticleAbstract

You can use or modify any of these default styles.

- 2 Change the custom property `RxContentEditorURL` in the template to point to the new Content Editor.

For instructions on creating or modifying a Word template file, see *Create a Document Template* in the Microsoft Word Help. For instructions on changing a custom property in Word, see *Modify a Custom File Property* in the Microsoft Word Help.

NOTE: The deprecated Word template is `..\sys_resources\word\Rhythmyx.dot`.

Modifying the Style Sheet for Parsing Fields

The `ParseMeta.xml` file performs formatting on Word-based Content Editor fields containing metadata and inserts them into the appropriate database columns in the backend table for the Content Editor. Any fields that are not included in `ParseMeta.xml` are formatted by `ParseBody.xml` and inserted into the body field for the Content Editor.

When you create a Word-based Content Editor, modify `ParseMeta.xml`, `ParseBody.xml`, and `contentfilter.xml` to use the filenames, fields, and styles that your Content Editor uses:

- 1 In `ParseMeta.xml`:
 - change the reference to `contentfilter.xml` to point to your Content Editor directory:
`<xsl:import href="../../../rx_WordContent_ce/contentfilter.xml" />`

- Change metadata field names and styles to those that your Content Editor and Word template use. For example, in the following code, change the field name `<ArticleTitle>` to a name of a new field in the Content Editor. Change the style `ArticleDisplayTitle` to a new style in the Word template.

```

<ArticleTitle>
  <xsl:choose>
    <xsl:when
test="string(//html:body[1]//html:p[@class='ArticleDisplay
Title'])">
      <xsl:apply-templates
select="//html:body[1]//html:p[@class='ArticleDisplayTitle
']" />
    </xsl:when>
    <xsl:when
test="string(//html:body[1]//html:p[@class='MsoTitle'])">
      <xsl:apply-templates
select="//html:body[1]//html:p[@class='MsoTitle']" />
    </xsl:when>
    <xsl:otherwise>COULD NOT FIND A TITLE</xsl:otherwise>
  </xsl:choose>
</ArticleTitle>

```

2 In ParseBody.xsl:

- change the reference to `contentfilter.xsl` to point to your Content Editor directory:

```

<xsl:import href="../../rx_WordContent_ce/
contentfilter.xsl" />

```

3 In contentfilter.xsl:

- Change metadata style names to those that your Word template uses. For example, in the following code, change `ArticleDisplayTitle` to a new style in the Word template:

```

<xsl:template match="html:p[@class='ArticleDisplayTitle']"
mode="contentfilter" />

```

Modifying the Content Assembler to Display Custom Word-based Content Editor Fields

Content fields that you add to the custom Word-based Content Editor may require additional processing so that they appear in the desired format on the assembled output page.

Where you add the additional formatting depends on the complexity of the style and how you want to use the output. Use the following guidelines:

- If the style that you want to apply is simple, include the formatting in your output CSS. Simple styles affect the appearance of text, for example, by changing the font or color.
- If the style is complex, add the formatting to `ParseBody.xsl` or `rx_InlineLinks.xsl`. Complex styles affect the appearance of content on a page, for example, by adding tables or line breaks.
 1. If you plan to use the style in various output formats, add a generic style for the content in `ParseBody.xsl`. `ParseBody.xsl` stores the content in the generic style prior to output assembly. When output assembly occurs, the various output mechanisms reformat the content as required.
 2. If you plan to output the style as an HTML page, add the formatting style that you want to appear on the final Web page to `rx_resources/stylesheets/assemblers/rx_InlineLinks.xsl`.

Installing New Features of Rhythmyx Word Connector

The current build of Rhythmyx may include changes to the Rhythmyx Word Connector since your previous installation. You can install Rhythmyx either as an upgrade or as a new installation. If you install it as an upgrade, the deprecated Article Word Content Editor and custom Word-based Content Editors will function as they did previously, but you can install new features. If you install Rhythmyx as a new installation or you upgrade and install the new Rhythmyx Word Connector features, you must set the Rhythmyx server address in the new template file.

Moving Rhythmyx Accelerator for Word Files to the Correct Directory (see "[Moving Rhythmyx Word Connector Files to the Correct Directory](#)" on page 120)

Setting the Address in the Word Template Properties (see "[Setting the Address in the Word Template Files](#)" on page 122)

Processing Related Links (see "[Moving Rhythmyx Word Connector Files to the Correct Directory](#)" on page 120)

Copying the Template File to the Client's Word Application (see "[Copying the Template File to the Client Word Application](#)" on page 126)

Updating the sys_FileWord Content Editor Control (on page 127)

Moving Rhythmyx Word Connector Files to the Correct Directory

During development, Percussion Software may modify the rxwordsample.dot file and various .cab files used to support Rhythmyx functionality in Word. When you upgrade to a new version of Rhythmyx (and in some cases, when you upgrade to a new build, such as to fix a bug), you may need to move the Rhythmyx Word Connector files from the install directory (sys_resources/word) to the local directory (rx_resources/word) to make them available to download to your users.

You may also need to develop your Rhythmyx-enabled Word templates again based on the new version of the rxwordsample.dot template provided by Percussion Software to access new features included in the new template. Then move the custom files corresponding to the default files in the instructions below.

NOTE: Be sure you back up or rename your existing files before copying the new files.

NOTE to users of Rhythmyx Version 4.0: If you install Rhythmyx Build 20020326 as a new installation, the installer copies the files associated with the Rhythmyx Word Connector into the correct directories. If you install Rhythmyx Build 20020326 as an upgrade you must move some of the Rhythmyx Word Connector files to a new directory in order to access the new features in the Article Word Content Editor and custom Word-based Content Editors.

NOTE: If you have created a custom Word-based Content Editor, move the custom files corresponding to the default files in the instructions below.

To get the new Word Connector features after upgrading:

- 1 Copy the following files from

```
../sys_resources/word
```

to

```
../<Word-based Content Editor folder>.
```

```
rxwordsample.dot
```

```
parsebody.xsl
```

```
parsemeta.xsl
```

```
contentfilter.xsl
```

If your system is still using the deprecated Rhythmyx.dot template, in the Content Editor definition files for your Word-based Content Editors:

locate the <PSXParam name="WordTemplateURL"> node and change:

```
<PSXTextLiteral id="0">
```

```
<text>../sys_resources/word/Rhythmyx.dot</text>
```

```
</PSXTextLiteral>
```

to:

```
<PSXTextLiteral id="0"> <text>../<Word-based Content Editor  
folder>/rxwordsample.dot</text>  
</PSXTextLiteral>
```

- 2** Copy the file: rxwordocx.cab

from: ../sys_resources/word

to: ../rx_resources/word (If the directory “/word” does not exist, create it.)

- 3** If your system is using rxword.cab instead of rxwordocx.cab, in the file:

../sys_resources/stylesheets/sys_Templates.xml

locate the <object id="word"> node and change:

codebase = ../sys_resources/word/rxword.cab

to:

codebase = ../rx_resources/word/rxwordocx.cab

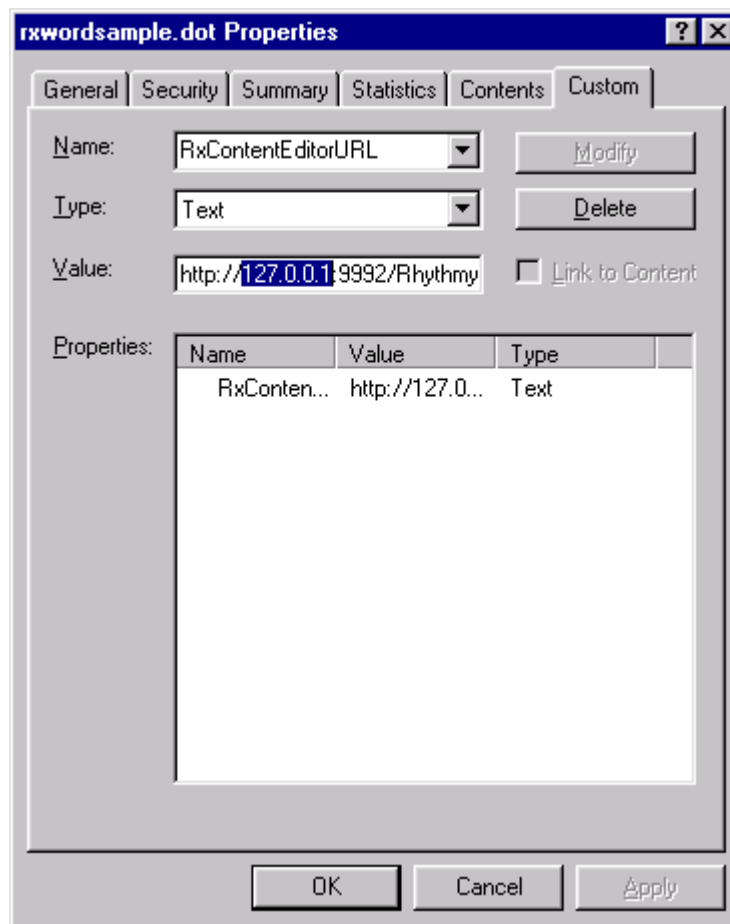
- 4** Stop and restart the Rhythmyx server.

Setting the Address in the Word Template Files

If you install Rhythmyx as a new installation, or if you upgrade and follow the instructions in *Installing New Features of Rhythmyx Accelerator for Word* (see "[Installing New Features of Rhythmyx Word Connector](#)" on page 119) to access the new features, you must change the value of the address in the template file's properties.

To change the template file address value:

- 1 Right click `../sys_resources/word/rxwordsample.dot` and select Properties in the drop menu. Rhythmyx opens the Template Properties dialog.
- 2 Click the Custom tab.




- 3 Click the **Properties Name** to make the values editable in the fields.
- 4 Change the **Value** of the host address from the default of 127.0.0.1 to the address of the Rhythmyx server.
- 5 Click [OK].

- 6 If you have already moved the new version of rxwordsample.dot into any of your Word-based Content Editor folders, repeat the procedure for those copies of the file.

Processing Related Links

If you install Rhythmyx as a new installation, or if you upgrade and follow the instructions in *Installing New Features of Rhythmyx Accelerator for Word* (see "[Installing New Features of Rhythmyx Word Connector](#)" on page 119) to access the new features, add the `sys_xdProcessRelatedLinks` exit to the Word-based Content Editor applications. The `sys_xdProcessRelatedLinks` exit creates the inline links and URLs for inline images for the published site.

To add the `sys_xdProcessRelatedLinks` exit to the Word-based Content Editor application:

- 1 Open the Word-based Content Editor application in the Workbench.
- 2 Double-click the exit on the Content Editor resource.
Rhythmyx opens the Exit Properties dialog.
- 3 Click the Insert New Entry button .
- 4 In the drop list, select `xml/dom/sys_xdProcessRelatedLinks`.
- 5 Enter XMLDOM as the value of the SourceObject parameter.

- 6 Use the arrow buttons to move `xmlDom/sys_xdProcessRelatedLinks` to the position after `xmlDom/sys_xdTextToDom`.

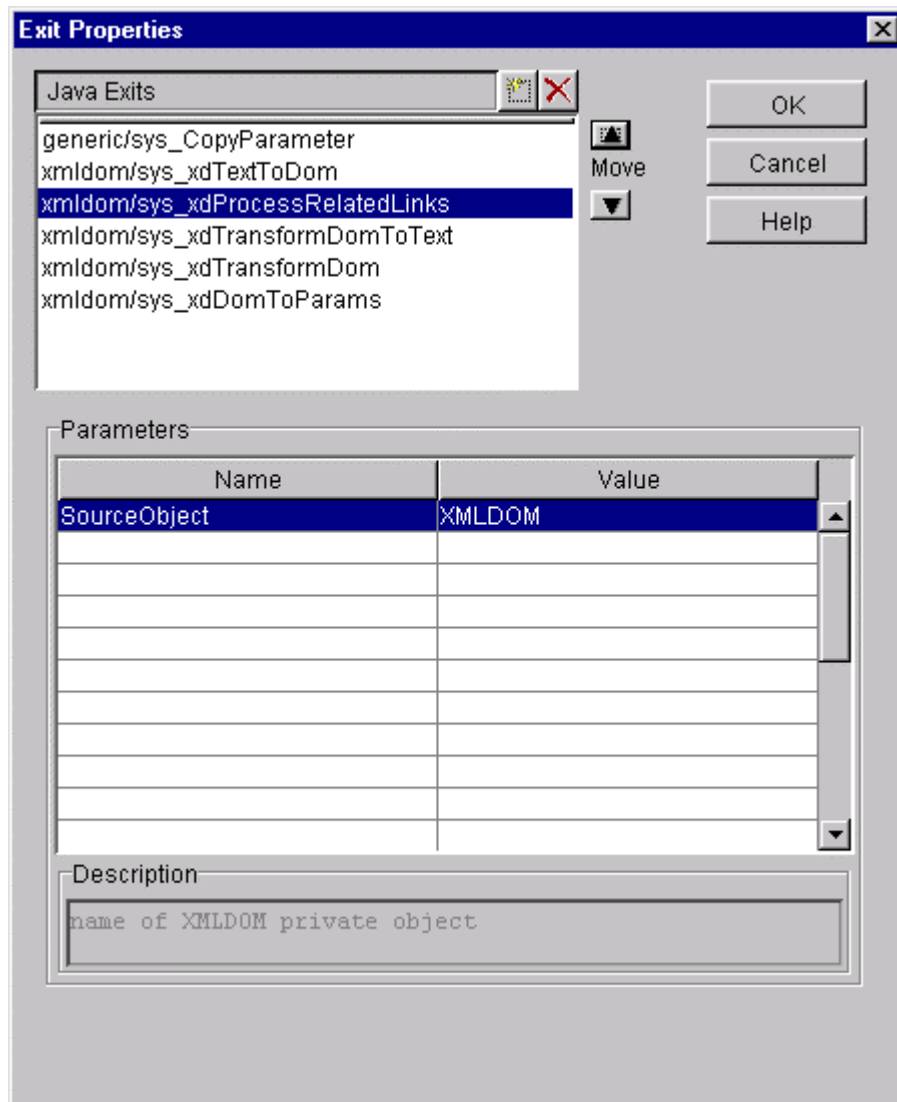


Figure 42: Exit Properties dialog

- 7 Click [OK].
- 8 Save the application.

Copying the Template File to the Client Word Application

After you have configured and copied Rhythmyx files to use the new Rhythmyx Accelerator for Word features, ensure that users attach the new `rxwordsample.dot` template to their Rhythmyx Word documents. You can install the template in one of two ways:

- Require users to open their first Rhythmyx Word document after upgrade through the **[Launch Word]** button in the Word-based Content Editor to automatically attach the template to Word. If they open their first Rhythmyx Word document directly in Word, the new template will not be attached.
- Send users the `rxwordsample.dot` template and instruct them to copy it into their Word template directories.

Updating the `sys_FileWord` Content Editor Control

During development or to fix bugs, Percussion Software may modify the `sys_FileWord` control, requiring you to update the existing copy of the control. The updated control will be in `sys_resources\stylesheets\sys_templates.xml`. Copy the `<controlMeta>` node (and all of its children) of the `sys_FileWord` template control in this stylesheet and use it to replace the `<controlMeta>` node of the `sys_FileWord` template control in `rx_resources\stylesheets\rx_templates.xml`.

Appendices

Implementing a Content Editor Manually

You should be able to complete most content editor development using the Content Editor Properties dialog in the Rhythmyx Workbench. A few features are not yet available in this dialog, however, and to access them, you must edit the content editor XML definition file manually.

Editing the content editor definition manually is recommended only to advanced implementers of Rhythmyx.

Registering a New Content Type

You must register each new content type in the Rhythmyx System Administrator. Registering the content type generates a content type ID that Rhythmyx uses to identify it. The content type registration also defines the URLs Rhythmyx uses to locate the content editor applications. These URLs do not exist when you initially register the content type. You must update the registration with this information after you create the content editor application.

To register a new content type:

- 1 Log into the Rhythmyx Content Manager with a user in the Admin Role.
- 2 In the banner, click [System](#).
Rhythmyx displays the System Administrator page.
- 3 In the navigation bar, under Content Types, click [By Name](#).
Rhythmyx displays the Content Types Editor, showing the existing content types.
- 4 Click [New Content Type](#).
Rhythmyx displays the New Content Type page.
- 5 Enter the **Name** and **Description**. Enter some dummy data in the **New Request URL** and **Query Request URL** fields. The data for these fields is defined later in the process. You will update them after you create the content editor application.
- 6 Click [**Save**] to save the content type registration.

Rhythmyx returns to the Content Types Editor page. The new registration appears at the end of the list.

The number in parentheses to the right of the content type name is the content type ID. Enter this value as the value of the `contentType` attribute of the `PSXContentEditor` element in the content editor XML definition file.

Creating the Content Editor Definition

Creating a content editor consists of the following processes:

- 1 *Selecting a template for the content editor definition* (see "[Selecting a Content Editor Definition Template](#)" on page 132).

- 2 *Creating the content editor definition file* (see "[Creating a Content Editor File Based on an Existing Definition File](#)" on page 132).
- 3 *Defining the database connection* (see "[Defining the Database Definition](#)" on page 133).
- 4 *Defining the content editor fields* (see "[Defining the Content Editor Mapper](#)" on page 134).
- 5 *Add Related Content links, if appropriate* (see "[Adding Related Content Links to a Content Editor](#)" on page 139).
- 6 *Add interface components* (see "[Adding Components to the Content Editor](#)" on page 142).

Selecting a Content Editor Definition Template

A content editor definition file is a complex XML document. Therefore, when creating a new content type it is usually easier to modify an existing content editor definition file rather than to create a new one from scratch. Using an existing Content Editor Definition also provides the following benefits:

- Backend database credentials are already populated. Therefore you only need to modify the table references for individual fields.
- Examples of field, field set, and user-interface definitions already exist as templates for your fields.
- Required fields that the user does not need to edit are already populated.
- Custom controls, such as Related Content, are already included.

NOTE: You must have already created a Content Editor or imported a Content Editor into your system in order to have a Content Editor Definition file available.

Creating a Content Editor File Based on an Existing Definition File

To create a content editor based on an existing definition file:

- 1 Open the content editor definition for an existing content type in the XML editor application of your choice. Use an XML specific editor with DTD validation built in.
- 2 Save the file with a new name.
- 3 In the XML, change the `contentType` attribute of the root element (`PSXContentEditor`) to the content type id generated when you registered the content type.
- 4 If the content type will use a different workflow, change the `workflowId` attribute to the of workflow ID the workflow you want this content type to use.

For example, your system could include a Content Editor named Article associated with a Content Type ID of 1 and a Workflow ID of 1. If you used the Article Content Editor as your template, and the Content Type ID you generated was 302, and the Workflow ID of the Workflow you wanted the Content Editor to use was 156, you would change
`<PSXContentEditor contentType="1" workflowId="1">` to
`<PSXContentEditor contentType="302" workflowId="156">`.

- 5 You can change the information in the `<name>` and `<description>` elements of the `PSXContentEditor` element to describe the new content editor. This information is optional and does not affect the appearance or operation of the content editor.

Defining the Database Definition

The `PSXContainerLocator` element stores the information Rhythmyx uses to connect the content editor to the back-end database tables where the data for the content editor is stored. This element requires one `PSXTableSet` child element, which defines the connection to the database.

The `PSXTableSet` element takes two children. The first is `PSXTableLocator`, which occurs only once in each `PSXTableSet`. The children of `PSXTableLocator` define the connection to the database. If you copied the content editor definition file from a working system, the data for these children will already be defined. Otherwise, you will have to define them.

- The `driver` element specifies the database driver; for example, `odbc` or `oracle:thin`.
- The `server` element defines the address of the database server. The value of the `server` element is the connection string; for example: `@127.0.0.1:1521:ORCL`.
- The `userID` element stores the user ID the system uses to access the database.
- The `password` element stores the password the system uses to access the database.
- The `Database` element stores the name of the database.
- The `Origin` element specifies the owner of the database or schema.

The second child of the `PSXTableSet` element is `PSXTableRef`. This element defines the tables for the content editor. The `PSXTableRef` element occurs once for the parent table of the content editor and once for each child table.

Example database definition:

```
<PSXContainerLocator>
  <PSXTableSet>
    <PSXTableLocator alias="">
      <PSXBackEndCredential id="0">
        <alias>Cred1</alias>
        <comment/>
        <driver>odbc</driver>
        <server>rxmaster</server>
        <userId>rxuser</userId>
        <password encrypted="yes"/>
      </PSXBackEndCredential>
      <Database>rxmaster</Database>
      <Origin>dbo</Origin>
    </PSXTableLocator>
    <PSXTableRef name="RXSECTION" alias="RXSECTION"/>
  </PSXTableSet>
</PSXContainerLocator>
```

Defining the Content Editor Mapper

The `PSXContentEditorMapper` element is the parent of the elements that define the fields in the content editor and the user-interface for those fields. This element has four children:

- The `PSXFieldSet` element defines the set of fields unique to the content editor.
- The `SharedFieldIncludes` element specifies fields included in the content editor that are shared with other content editors.
- The `SystemFieldExcludes` specifies which system fields should be excluded from the content editor. (When you create a content editor, all of the system fields are automatically included.)
- The `PSXUIDefinition` element links each field to the interface control used to display it.

Field Sets

Local fields are defined in one or more `PSXFieldSet` elements.

A content editor includes one field set for each table that stores data for the content type. One of these field sets must be the parent field set and it must be associated with the parent table. This field set includes one field definition for each column in the parent table. Any other field sets are child field sets, each of which must be associated with a child table. Usually, each child field set includes one field definition for each column in the associated child table.

The order in which the field sets (and the field definitions within them) are defined does not affect their display on the content editor interface. The user-interface definition controls the display order.

The `PSXFieldSet` element has the following attributes:

Attribute	Meaning
Name	Identifies the field set for display mapping. This attribute is required if the <code>Type</code> attribute has any value other than <code>parent</code> . The value of the <code>Name</code> attribute must be unique among all field set and field names in the content editor definition file. While this attribute is optional for parent field sets, they typically take the default value <code>main</code> .

Attribute	Meaning
Type	<p>Defines whether the field set is a parent or child field set, and if a child, the type of child field set. This attribute can take one of the following values:</p> <ul style="list-style-type: none"> ▪ Parent <p>Defines the field set as the parent field set for the content editor. This field set must be associated with the parent table. Only one field set in the content editor defining can have this value.</p> ▪ simpleChild <p>Child field set that is edited within the parent field set's row editor. Field sets of this type can only include one field, whose possible values are defined in the user-interface control for the field.</p> ▪ complexChild <p>Child field set that is displayed in summary view within the parent row editor. Field sets of this type can contain one or more columns and an arbitrary number of rows. In a summary view, data is summarized and read-only.</p> ▪ MultiPropertySimpleChild <p>Child field set is edited within the parent field set's row editor. Field sets of this type can contain one or more columns, but only one row.</p>
repeatability	<p>Defines how many times the rows of the field set can appear in the content editor. This attribute can take one of the following values:</p> <ul style="list-style-type: none"> ▪ zeroOrMore <p>The field set may appear once, more than once, or not at all. The parent field set should always have this value for the repeatability attribute.</p> ▪ oneOrMore <p>The field set must appear once, but may appear more than once.</p> ▪ count <p>The field set must appear a specified number of times. If the value of the <code>repeatability</code> attribute is <code>count</code>, the optional <code>count</code> attribute must be included to define the number of rows required.</p>
count	<p>Required only if the value of the <code>repeatability</code> attribute is <code>count</code>. Defines the number of rows that must appear for the field set.</p>
supportsSequencing	<p>Only applicable if type is not <code>parent</code>. Indicates whether the end user can put the rows associated with the parent in a specific order.</p>

Example Parent:

```
<PSXFieldSet name="main" type="parent" repeatability="zeroOrMore"
  supportsSequencing="yes">
```

Example Child:

```
<PSXFieldSet name="product" type="simpleChild" repeatability="oneOrMore"
  supportsSequencing="no">
```

Defining Fields

Each individual field in the content editor interacts with a single column in the database. Use the `PSXField` element to define each field. The behavior of the field is defined by both its attributes and its child elements.

Attributes

Attribute	Meaning
name	Defines the name of the field. Must be unique among all fields in the content editor, including shared and system field names.
showInSummary	Applies only to fields in <code>complexChild</code> fieldsets. This attribute determines whether to display the field in summary view. If the value of this attribute is <code>yes</code> , Rhythmyx displays the field in summary areas on the content editor. For fields with lengthy text that may be unclear in a summary view, set this attribute to <code>no</code> . It is automatically set to <code>no</code> for fields whose datatype is binary. Rhythmyx ignores this attribute except when generating summary table data. The default value of this attribute is <code>yes</code> .
showInPreview	Defines whether the content editor displays the field in preview mode. This attribute allows you to hide fields with lengthy text that may be unclear in Preview mode. By default, all fields except those containing binary data are displayed. The default value of this attribute is <code>yes</code> .
forceBinary	This attribute allows you to override the default behavior of the server. Normally, the server only considers binary fields such as Image or LONG RAW as binary. Set this flag to <code>yes</code> if you want the server to treat another data type (such as CLOB in Oracle or Text in SQLServer) as binary. Defaults to <code>no</code> . Set to <code>yes</code> for all binary fields.

Child Elements

Name	Appearance	Description
DataLocator	Once	Specifies the location of the data for this field.
DataType	Zero or one	Defines the type of data (character data versus binary). For future use.
DefaultValue	Zero or one	Defines the default value of the field. If a value is specified for this element, that value will be included in the output document when the content editor is displayed for a new item.

Name	Appearance	Description
OccurrenceSettings	Zero or one	Specifies how many times the field should occur (in other words the business rules).
FieldRules	Zero or one	Defines set of rules regarding validation, translation, and visibility for field.

Example

```
<PSXField name="displayname" showInSummary="yes" showInPreview="yes"
forceBinary="no">
  <DataLocator>
    <PSXBackEndColumn id="0">
      <tableAlias>RXHEADLINE</tableAlias>
      <column>DISPLAYNAME</column>
      <columnAlias/>
    </PSXBackEndColumn>
  </DataLocator>
  <DataType/>
  <OccurrenceSettings dimension="optional" multiValuedType="delimited"
delimiter=";" />
</PSXField>
```

Including Shared Fields

Several content editors may share a number of fields. For example, if you want to associate every content item with a division and product line, all of your content editors can share those fields. Rather than repeating the definition of the fields in each content editor, you can define sets of shared fields in one or more XML files that are separate from the content editor file. These shared field definition files use the same database definition and the same XML field and field set elements as other content editor files.

Use the `SharedFieldGroupName` child element to define the shared field groups you want to include in the content editor definition.

This is the XML for including the shared field group *relatedcontent*:

```
<SharedFieldIncludes>
  <SharedFieldGroupName>relatedcontent</SharedFieldGroupName>
</SharedFieldIncludes>
```

Excluding System Fields

Rhythmyx includes all system fields in each content editor by default. Rhythmyx includes the following system fields at installation:

System Field	Description
sys_contentstartdate	Date the content item is eligible to be published.
sys_contentexpirydate	Date the content item is no longer eligible to be published.
sys_title	Title of the content item in the CMS.
sys_pubdate	Date the content item was last published.
sys_pathname	Path, relative to the root of the web server, where the file will be published.

System Field	Description
sys_suffix	The extension of the file to be published. For example: htm, html, or asp.

Use the `SystemFieldExcludes` element to specify any system fields you do not want used in the content editor. Specify the individual fields in `FieldRef` child elements. The example below shows the XML code for excluding the `sys_pubdate` field:

```
<SystemFieldExcludes>
  <FieldRef>sys_pubdate</FieldRef>
</SystemFieldExcludes>
```

Defining the Interface

The `PSXUIDefinition` element contains the definitions of the user-interface controls for the fields in the content editor. Each control defines the user interface for entering or editing the contents of a single field on the browser form. For example, drop-down lists and checkboxes are types of controls. The `PSXUIDefinition` includes one `PSXDisplayMapper` element for each `PSXFieldSet` element, and one `PSXDisplayMapping` element for each `PSXField` element.

The `FieldRef` child of the `PSXDisplayMapping` element specifies the field for which you are defining the user interface, while the `PSXUISet` defines the interface label and control for the field. The `PSXUISet` element includes the following children:

Name	Appearance	Description
Label	Zero or one	Defines the label for the field in the user interface. This text is visible the user of the editor.
PSXControlRefE	Zero or one	Refers to a control used when displaying the data in this field. Controls are defined in an XSL field, so no validation is performed on this name until a request is processed. Default controls include: sys_EditBox sys_File sys_HiddenInput sys_TextArea sys_CalendarSimple sys_HtmlEditor sys_Table sys_DropDownSingle sys_CheckBoxGroup

Name	Appearance	Description
ErrorLabel	Zero or one	Defines the label displayed when the field fails a validation test. Replaces the standard label in the output document in this case.
PSXChoices	Zero or one	Defines a choice list for the field. Only used for SDMP field sets.
ReadOnlyRules	Zero or one	Defines the rules that determine whether a field should be displayed as read-only.
PSXCustomActionGroup	Zero or one	Defines overrides to the default editor behavior, removing existing buttons and adding new buttons.

Example PSXIUDefinition

```

<PSXIUDefinition>
  <PSXDisplayMapper id="0" fieldSetRef="main">
    <PSXDisplayMapping>
      <FieldRef>DISPLAYTITLE</FieldRef>
      <PSXUISet>
        <Label>
          <PSXDisplayText>Display Title:</PSXDisplayText>
        </Label>
        <PSXControlRef name="sys_EditBox"/>
      </PSXUISet>
    </PSXDisplayMapping>
  </PSXDisplayMapper>
</PSXIUDefinition>

```

Adding Related Content Links to a Content Editor

Related content is other content items that are associated with the content item being edited. For example, when editing a content item that stores a news article, you might want to include an image, links to other articles of related interest, and recent news developments. Each of these would be associated with the article as related content.

To add the ability to link to related content to an editor:

- 1 In the XML document for the content editor, include the shared field group `relatedcontent`. For example:

```

<SharedFieldIncludes>
  <SharedFieldGroupName>relatedcontent</SharedFieldGroupName>
</SharedFieldIncludes>

```
- 2 Include a reference to the Related Content control in a `<PSXDisplayMapping>` element that refers to the related content shared field group. Do not include a label. For example:

```

<PSXDisplayMapping>

```

```
<FieldRef>relatedcontent</FieldRef>
  <PSXUISet>
    <PSXControlRef name = "sys_RelatedContentTable"/>
  </PSXUISet>
</PSXDisplayMapping >
```

- 3** In the <SectionLinkList>, add a section link to the related content lookup in a <PSXUrlRequest> child element.

```
<PSXUrlRequest name = "RelatedLookupURL">
  <PSXExtensionCall id = "0">
<name>Java/global/percussion/generic/sys_MakeIntLink</name>
  <PSXExtensionParamValue id = "0">
    <value>
      <PSXTextLiteral id = "0">
        <text>../sys_rcSupport/relatedcontent.xml</text>
      </PSXTextLiteral>
    </value>
  </PSXExtensionParamValue>
  <PSXExtensionParamValue id="0">
    <value>
      <PSXTextLiteral id="0">
        <text>sys_contentid</text>
      </PSXTextLiteral>
    </value>
  </PSXExtensionParamValue>
  <PSXExtensionParamValue id="0">
    <value>
      <PSXHtmlParameter id="0">
        <name>sys_contentid</name>
      </PSXHtmlParameter>
    </value>
  </PSXExtensionParamValue>
  <PSXExtensionParamValue id="0">
    <value>
      <PSXTextLiteral id="0">
        <text>sys_revision</text>
      </PSXTextLiteral>
    </value>
  </PSXExtensionParamValue>
  <PSXExtensionParamValue id="0">
    <value>
      <PSXHtmlParameter id="0">
        <name>sys_revision</name>
      </PSXHtmlParameter>
    </value>
  </PSXExtensionParamValue>
  </PSXExtensionCall>
</PSXUrlRequest>
```

- 4** In the <SectionLinkList>, add a section link to the variant list in a <PSXUrlRequest> element.

```
<PSXUrlRequest name = "VariantListURL">
  <PSXExtensionCall id = "0">
<name>Java/global/percussion/generic/sys_MakeIntLink</name>
  <PSXExtensionParamValue id = "0">
    <value>
```



```

        <PSXTextLiteral id = "0">
            <text>../sys_rcSupport/
                variantlistwithslots.xml</text>
        </PSXTextLiteral>
    </value>
</PSXExtensionParamValue>
<PSXExtensionParamValue id="0">
    <value>
        <PSXTextLiteral id="0">
            <text>sys_contentid</text>
        </PSXTextLiteral>
    </value>
</PSXExtensionParamValue>
<PSXExtensionParamValue id="0">
    <value>
        <PSXHtmlParameter id="0">
            <name>sys_contentid</name>
        </PSXHtmlParameter>
    </value>
</PSXExtensionParamValue>
</PSXExtensionCall>
<PSXUrlRequest>

```

- 5** In the <SectionLinkList>, add a section link to the content slot lookup in a <PSXUrlRequest> element.

```

<PSXUrlRequest name="ContentSlotLookupURL">
    <PSXExtensionCall id="0">
        <name>Java/global/percussion/generic/sys_MakeIntLink</name>
        <PSXExtensionParamValue id="0">
            <value>
                <PSXTextLiteral id="0">
                    <text>../sys_rcSupport/contentslotvariantlist.xml</text>
                </PSXTextLiteral>
            </value>
        </PSXExtensionParamValue>
        <PSXExtensionParamValue id="0">
            <value>
                <PSXTextLiteral id="0">
                    <text>sys_contentid</text>
                </PSXTextLiteral>
            </value>
        </PSXExtensionParamValue>
        <PSXExtensionParamValue id="0">
            <value>
                <PSXHtmlParameter id="0">
                    <name>sys_contentid</name>
                </PSXHtmlParameter>
            </value>
        </PSXExtensionParamValue>
    </PSXExtensionCall>
</PSXUrlRequest>

```

Adding Components to the Content Editor

The Rhythmyx user interface is composed of several components independent of the content editor application itself, such as the banner, user-status display and **[Help]** button. To include these components:

- 1 Go to the `<SectionLinkList>` element.
- 2 If the `<PSXUrlRequest>` element for "bannerincludeurl" exists, replace it with the following XML fragment. If it does not exist, add the following XML fragment.

```
<PSXUrlRequest name="bannerincludeurl">
  <PSXExtensionCall id="0">
    <name>Java/global/percussion/generic/sys_MakeIntLink</name>
    <PSXExtensionParamValue id="0">
      <value>
        <PSXTextLiteral id="0">
          <text>../sys_ComponentSupport/component.xml</text>
        </PSXTextLiteral>
      </value>
    </PSXExtensionParamValue>
    <PSXExtensionParamValue id="0">
      <value>
        <PSXTextLiteral id="0">
          <text>sys_componentname</text>
        </PSXTextLiteral>
      </value>
    </PSXExtensionParamValue>
    <PSXExtensionParamValue id="0">
      <value>
        <PSXTextLiteral id="0">
          <text>cmp_banner</text>
        </PSXTextLiteral>
      </value>
    </PSXExtensionParamValue>
  </PSXExtensionCall>
</PSXUrlRequest>
```

- 3 If the `<PSXUrlRequest>` element for "userstatusincludeurl" exists, replace it with the following XML fragment. If it does not exist, add the following XML fragment.

```
<PSXUrlRequest name="userstatusincludeurl">
  <PSXExtensionCall id="0">
    <name>Java/global/percussion/generic/sys_MakeIntLink</name>
    <PSXExtensionParamValue id="0">
      <value>
        <PSXTextLiteral id="0">
          <text>../sys_ComponentSupport/component.xml</text>
        </PSXTextLiteral>
      </value>
    </PSXExtensionParamValue>
    <PSXExtensionParamValue id="0">
      <value>
        <PSXTextLiteral id="0">
          <text>sys_componentname</text>
        </PSXTextLiteral>
      </value>
    </PSXExtensionParamValue>
  </PSXExtensionCall>
</PSXUrlRequest>
```

```

    <PSXExtensionParamValue id="0">
      <value>
        <PSXTextLiteral id="0">
          <text>cmp_userstatus</text>
        </PSXTextLiteral>
      </value>
    </PSXExtensionParamValue>
  </PSXExtensionCall>
</PSXUrlRequest>

```

- 4** If the <PSXUrlRequest> element for "helpincludeurl" exists, replace it with the following XML fragment. If it does not exist, add the following XML fragment.

```

<PSXUrlRequest name="helpincludeurl">
  <PSXExtensionCall id="0">
<name>Java/global/percussion/generic/sys_MakeIntLink</name>
    <PSXExtensionParamValue id="0">
      <value>
        <PSXTextLiteral id="0">
          <text>../sys_ComponentSupport/component.xml</text>
        </PSXTextLiteral>
      </value>
    </PSXExtensionParamValue>
    <PSXExtensionParamValue id="0">
      <value>
        <PSXTextLiteral id="0">
          <text>sys_componentname</text>
        </PSXTextLiteral>
      </value>
    </PSXExtensionParamValue>
    <PSXExtensionParamValue id="0">
      <value>
        <PSXTextLiteral id="0">
          <text>ca_help</text>
        </PSXTextLiteral>
      </value>
    </PSXExtensionParamValue>
  </PSXExtensionCall>
</PSXUrlRequest>

```

Updating the Content Type Registration

Once you have a working content editor application, you can update the New Request and Query Request URL fields.

To update a content type registration:

- 1** In the content editor Resource Properties dialog, copy the content editor URL to the clipboard.
- 2** Log into the Rhythmyx Content Management System with a user in the Admin Role.
- 3** In the banner, click System.
Rhythmyx displays the System Administrator page.
- 4** In the left menu, click Content Types.

Rhythmyx displays the Content Types Editor, showing the existing content types.

- 5 Click on the content type you want to update.

Rhythmyx displays the Edit Content Type page.

- 6 Paste the URL from the content editor into the **New Request URL** and **Query Request URL** fields.
- 7 Click [**Save**] to save the content type registration. Rhythmyx will display a confirmation dialog. Click [**Yes**] to confirm the save action or [**No**] to abort it.


Creating the Content Editor Application

Once you have validated your content editor definition, you are ready to create the content editor application in the Rhythmyx workbench.

To create a content editor application:

- 1 Start the Rhythmyx workbench and create a new application.
- 2 Click the Files tab and navigate to the directory where you saved your content editor definition.
- 3 Drag the content editor definition file and drop it into the application window.
- 4 Rhythmyx will display a popup menu. Select Content Editor. If your content editor definition is valid, Rhythmyx will display the content editor resource:



- 5 Map the XML to the database.
- 6 Create a purge resource named "purge." Set the "pipe properties" of the purge resource to "purge." The purge resource deletes all rows of all tables (other than system tables) that hold content for the specific content id.
- 7 Save your application.
- 8 Click the Start button  in the button bar to start the application.
- 9 Right click on the content editor and choose Request Properties.
Rhythmyx displays the Resource Request Properties dialog.
- 10 Click the [**Copy to Clipboard**] button to copy the sample URL.
- 11 Open a web browser, paste the contents of the clipboard to the address field and open the page.

If you have defined the content editor correctly, the browser will display a login dialog. If you enter a valid user name and password into this dialog, Rhythmyx should display the content editor.

Validating the Content Editor Definition

When you finish creating the content editor definition, you will need to validate it against the content editor DTD (`sys_ContentEditorLocalDef.dtd`) to ensure that it conforms to the requirements of the DTD. If the content editor definition does not conform to this DTD, it will generate errors when you drop it into the Rhythmyx workbench. The DTD is located in the `/Rhythmyx/DTD/` directory. Note: Solaris users (and others) should not change directory name or case.

Most XML editor applications include DTD validation functions. Several XML editors for the Windows platform are available to be downloaded from the Internet, either as freeware or as limited-time trials. Search the Web to find the tools available.

APPENDIX II

Content Editor Control Reference

The content editor stylesheets refer to a set of controls that are defined in two XSL stylesheets: `rx_Templates.xml` and `sys_Templates.xml`. The `sys_templates` file stores the standard controls that are installed with Rhythmyx and is stored in the `/<rxroot>/ sys_resources /Stylesheets` directory. The `user_templates` file stores controls defined for the specific installation and is stored in the `/<rxroot>/ rx_resources /Stylesheets` directory. Users should not modify controls defined in `sys_Templates.xml`.

The control stylesheets are imported into the content editor using the following code:

```
<xsl:import href="sys_resources/Stylesheets/sys_templates.xml"/>
<xsl:import href="rx_resources/Stylehsheets/rx_templates.xml"/>
```

All controls provided by Percussion Software begin `sys_`; for example, `sys_DatePicker`. User-developed controls should not begin with this prefix.

Using `<xsl:import>` defines the precedence between the templates such that any template that exists in `user_templates.xml` overrides a template of the same name that exists in `sys_templates.xml`. When developing local control templates, do not use control names that begin "sys_" unless intentionally overriding an existing control.

Control Header

The control header stores the metadata that defines the control, including the name and description of the control, any parameters, associated files, or exits required for the control to function and process data correctly. The formal definition of the controls is defined in the `sys_LibraryControlDef.dtd`.

The header must be added to the `sys_template.xml` or `user_template.xml` immediately before the first `<xsl:template>` block related to the control. Rhythmyx uses this header when selecting controls. If the control header is missing or invalid, Rhythmyx cannot select the control. The control will continue to work unless it requires external script files, however.

All control definitions exist in the "psxctl" namespace. The full declaration of this namespace is:

```
xmlns:psxctl="URM:percussion.com/control"
```

Any files required for the control to function must be listed in the `AssociatedFiles` element of the header. The children of this element describes the file and specifies its location.

Any exits required by the control must be specified in the `Dependencies` element. The attributes of this element specify whether the extension requires additional setup and whether you must add additional iterations of the exit for each appearance of the control. The child elements specify the exit to call and any parameters you must specify for it. You must add these exits to the content editor resource in the content editor application.

Control Template Standards

A control template must meet the following standards:

- The template must match on a `<Control>` element with a specific name. The main templates must use the "psxcontrol" mode. For example:

```
<xsl:template match="Control[@name='sys_DatePicker']"
mode="psxcontrol"
```

- Controls should be written to conform to the shape of the table, and should not contain fixed-width formats.
- All controls use the same cascading stylesheet styles that are used in the editors.
- The `datadisplay` style will be used unless some special effects are required.
- The `datacell11` and `datacell12` styles can be used for alternating rows in complex controls.
- The `columnhead2` style will be used for labels.
- All controls must be capable of rendering both "read-only" and "writable" forms. The forms do not have to resemble each other. The read-only form of the control must also a HTML form element that returns the current field value; for example, `<input type="hidden" name="sample" value="blank" />`.

Control Events

Individual form elements do not have "load" and "submit" events, and therefore certain controls will need JavaScript event code on the Form and Document level. To add JavaScript code to a control, build another `<xsl:template>` with a mode that matches the event name.

The output of any event template should:

- be a single string;
- be well-formed;
- end with a semi-colon.

Multiple template events are concatenated together into a single `onLoad` or `onSubmit` attribute.

Control templates that do not implement these events can either provide an empty template (for example:

```
<xsl:template match = "Control[@name='sys_picker']" mode="psxcontrol -
body-onload"/>)
```

or no template at all. Providing an empty template can be faster because it shortcuts the search for a template match.

To prevent events from being rendered as text items if the event is empty, the system control library includes a default empty template for each defined event. For example:

```
<xsl:template match="Control" mode="psxcontrol -docload"/>
```

Currently, the following events are defined within Rhythmyx:

HTML Event	Mode Name
document.load	psxcontrol-body-onload

HTML Event	Mode Name
form.submit	psxcontrol-form-onsubmit

Use the AssociatedFileList element to add the JavaScript file. The following example is from the sys_CalendarSimple control

```

<psxctl:AssociatedFileList>
  <psxctl:FileDescriptor name="calPopup.js" type="script"
  mimetype="text/javascript">

  <psxctl:FileLocation>../rx_resources/js/calPopup.js</psxctl:FileLocat
  ion>
  <psxctl:Timestamp></psxctl:Timestamp>
  </psxctl:FileDescriptor>
</psxctl:AssociatedFileList>

```

Standard Rhythmyx Controls

Eleven standard controls are provided with Rhythmyx.

Each control has a name and a dimension. The dimension describes the form of the data expected by the control. Options are

Value	Description
single	Data is zero or one value.
array	Data is a sequence of 0 or more values.
table	Data is a table of values.

Each control can take a series of parameters. Each parameter included has a name, a data type and a parameter type. The data type defines the type of data expected for the parameter. Options include String, Date, Datetime, and Number.

The parameter type can take one of three values: generic, img, and jsript. The parameter type is used with the parametersToAttributes template. This template copies parameters into the HTML. The defaults specified in the control metadata are used except where the content editor XML definition file overrides the defaults. Only parameters that are listed in the control meta are copied. Multiple parameter types are available because a control may need to configure more than one HTML tag.

The description of the parameter describes what the parameter is for. A parameter may or may not include a default value.

sys_CalendarSimple



Figure 43: Example `sys_CalendarSimple`

The `sys_CalendarSimple` control is a combination of an editbox and a button (calendar icon). When a user clicks the calendar icon, Rhythmyx displays a popup calendar control they can use to select a date. Each field has its own control. The dimension is *single*.

The text field allows for manual entry of a date. Data entered into this field must conform to standard date patterns.

- "yyyy-MMMM-dd 'at' hh:mm:ss aaa",
- "yyyy-MMMM-dd HH:mm:ss",
- "yyyy.MMMM.dd 'at' hh:mm:ss aaa",
- "yyyy.MMMM.dd HH:mm:ss",
- "yyyyMMdd HH:mm:ss",
- "yyyy.MMMM.dd 'at' hh:mm aaa",
- "yyyy-MM-dd G 'at' HH:mm:ss",
- "yyyy-MM-dd HH:mm:ss.SSS",
- "yyyy-MM-dd HH:mm:ss",
- "yyyy.MM.dd G 'at' HH:mm:ss",
- "yyyy.MM.dd HH:mm:ss.SSS",
- "yyyy.MM.dd HH:mm:ss",
- "yyyy/MM/dd G 'at' HH:mm:ss",
- "yyyy/MM/dd HH:mm:ss.SSS",
- "yyyy/MM/dd HH:mm:ss",
- "yyyy/MM/dd HH:mm",
- "yyyy-MM-dd",
- "yyyy.MM.dd",
- "yyyy/MM/dd",
- "yyyy-MMMM-dd",
- "yyyy.MMMM.dd",
- "EEE, d MMM yyyy HH:mm:ss",
- "EEEE, MMM d, yyyy",
- "MMM d, yyyy",
- "MMM yyyy",
- "yyyy",
- "HH:mm:ss",
- "HH:mm"

If these patterns are not matched, we try Java's default for the locale of the server to match the date. Patterns not matched result in a error. Rhythmyx uses the SimpleDateFormat (<http://java.sun.com/j2se/1.3/docs/api/java/text/SimpleDateFormat.html>) class to format and parse dates.

Parameter	Data Type	Parameter Type	Description	Default
id	String	Generic	XHTML 1.0 attribute applied input tag	None
class	String	Generic	XHTML 1.0 attribute applied input tag	None
style	String	Generic	XHTML 1.0 attribute applied input tag	None
tabindex	Number	Generic	XHTML 1.0 attribute applied input tag	None
alt	String	Image	Alt for the calendar selector icon.	Calendar Pop-up
src	String	Image	href for the calendar selector icon	../rx_resources/images/cal.gif
height	String	Image	Height of the calendar selector icon	20
width	String	Image	Width of the calendar picker icon	20
formname	String	JavaScript	Name of the form that contains this control	EditForm
time	String	Generic	Defines whether the Calendar display includes the time. If the value is yes, the time calendar displays the time. If the value is no, the calendar does not display the time. If the parameter has any other value, it is treated as though the value is yes.	no

Example Field Definition

The sys_CalendarSimple control almost always is assigned to a system field.

Example UI Definition

```
<PSXDisplayMapping>
  <FieldRef>sys_contentstartdate</FieldRef>
  <PSXUISet>
    <Label>
      <PSXDisplayText>Start Date:</PSXDisplayText>
```

```

    </Label>
    <PSXControlRef name="sys_CalendarSimple"/>
  </PSXUISet>
</PSXDisplayMapping>

```

sys_CheckBoxGroup

The `sys_CheckBoxGroup` displays a group of check boxes that give the end user the ability to select multiple values at the same time. A checkbox group must be multidimensional, so the values for the group should always be stored in a child table. The child table should consist of at least three columns: one for `contentid`, one for `revisionid` and one for the value to be stored. You should only define the value column in the field definition. The server will populate the `contentid` and `revisionid` fields automatically. The dimension is *array*.



Figure 44: Example `sys_CheckBoxGroup`

When implementing this control, add the child table to the list of tables for the content editor:

```

<PSXTableSet>
  <PSXTableLocator>
    . . . .
  </PSXTableLocator>
  <PSXTableRef name="RXBRIEF" alias="RXBRIEF"/>
  <PSXTableRef name="CHECKTABLE" alias="CHECKTABLE" />
</PSXTableSet>

```

Parameters

Parameter	Data Type	Parameter Type	Description	Default
<code>id</code>	String	Generic	XHTML 1.0 attribute	None
<code>class</code>	String	Generic	XHTML 1.0 attribute	None
<code>style</code>	String	Generic	XHTML 1.0 attribute	None
<code>columncount</code>	String	Generic	Defines the number of columns in which the browser will display the check boxes. If the value of this parameter is 0 or 1, the browser renders the checkboxes in one column. If the value of the parameter is anything other than 0 or 1, the browser renders the checkboxes in the specified number of columns.	1

Parameter	Data Type	Parameter Type	Description	Default
columnwidth	String	Generic	Specifies the width of the column in pixels or percentage.	100%

Example Field Definition

```
<PSXFieldSet name="productused" type="simpleChild"
repeatability="oneOrMore" supportsSequencing="no">
  <PSXField name="productusedvalue" showInSummary="yes"
showInPreview="yes" forceBinary="no">
    <DataLocator>
      <PSXBackEndColumn id="0">
        <tableAlias> ProductsUsed</tableAlias>
        <column>PRODUCTID</column>
        <columnAlias>PRODUCTIDUSED</columnAlias>
      </PSXBackEndColumn>
    </DataLocator>
    <DataType/>
    <OccurrenceSettings dimension="optional"
multiValuedType="delimited" delimiter=";" />
  </PSXField>
</PSXFieldSet>
```

Example UI Definitions

Use the type attribute of the PSXChoices element in the UI definition to specify how you are storing the checkbox values so that Rhythmyx can retrieve them.

To store checkbox values in the lookup table (RXLOOKUP), use the value `global` for the type attribute. Rhythmyx gets the values when it builds the document by using the value in the Key element to determine the lookup key for the table. Use the Rhythmyx System Administrator to maintain the Keywords in this table. The key is a system-generated value, so manual addition of values to this table is not recommended.

Example UI Definition for `global`:

```
<PSXDisplayMapping>
  <FieldRef>productused</FieldRef>
  <PSXUISet>
    <Label>
      <PSXDisplayText>Product Used:</PSXDisplayText>
    </Label>
    <PSXControlRef name="sys_CheckBoxGroup"/>
    <PSXChoices type="global" sortOrder="ascending">
      <Key>3</Key>
    </PSXChoices>
  </PSXUISet>
  <PSXDisplayMapper id="8" fieldSetRef="productused">
    <PSXDisplayMapping>
      <FieldRef>productusedvalue</FieldRef>
      <PSXUISet/>
    </PSXDisplayMapping>
  </PSXDisplayMapper>
</PSXDisplayMapping>
```

To store checkbox values in the content editor definition as a list of PSXEntry elements, use the value `local` for the `type` attribute.

Example UI Definition for `local`:

```
<PSXDisplayMapping>
  <FieldRef>local</FieldRef>
  <PSXUISet>
    <Label>
      <PSXDisplayText>Local choices:</PSXDisplayText>
    </Label>
    <PSXControlRef name=" sys_CheckBoxGroup "/>
      <PSXChoices type="local" sortOrder="user">
        <PSXEntry sequence="0" default="no">
          <PSXDisplayText>Computing Machinery</PSXDisplayText>
          <Value>acm</Value>
        </PSXEntry>
        <PSXEntry sequence="1" default="no">
          <PSXDisplayText> Society of Electrical
Engineers</PSXDisplayText>
          <Value>ieeee</Value>
        </PSXEntry>
        <PSXEntry sequence="3" default="no">
          <PSXDisplayText>Society for Prevention</PSXDisplayText>
          <Value>spca</Value>
        </PSXEntry>
      </PSXChoices>
    </PSXUISet>
  </PSXDisplayMapping>
```

`sys_CheckBoxGroup` To store checkbox values in the content editor definition as a list of PSXEntry elements, use the value `local` for the `type` attribute.

Example UI Definition for `local`:

```
<PSXDisplayMapping>
  <FieldRef>local</FieldRef>
  <PSXUISet>
    <Label>
      <PSXDisplayText>Local choices:</PSXDisplayText>
    </Label>
    <PSXControlRef name=" sys_CheckBoxGroup "/>
      <PSXChoices type="local" sortOrder="user">
        <PSXEntry sequence="0" default="no">
          <PSXDisplayText>Computing Machinery</PSXDisplayText>
          <Value>acm</Value>
        </PSXEntry>
        <PSXEntry sequence="1" default="no">
          <PSXDisplayText> Society of Electrical
Engineers</PSXDisplayText>
          <Value>ieeee</Value>
        </PSXEntry>
        <PSXEntry sequence="3" default="no">
          <PSXDisplayText>Society for Prevention</PSXDisplayText>
          <Value>spca</Value>
        </PSXEntry>
      </PSXChoices>
    </PSXUISet>
  </PSXDisplayMapping>
```

To store or generate checkbox values in another control or stylesheet, use the value `lookup` for the type attribute. Define a URL with the control or stylesheet's address in a `PSXUrlRequest` child element.

Example UI Definition for `lookup`:

```
<PSXDisplayMapping>
  <FieldRef>productused</FieldRef>
  <PSXUISet>
    <Label>
      <PSXDisplayText>Product Used:</PSXDisplayText>
    </Label>
    <PSXControlRef name="sys_CheckBoxGroup"/>
    <PSXChoices type="lookup" sortOrder="ascending">
      <PSXUrlRequest>
        <PSXExtensionCall id="10">
          <name>Java/global/percussion/generic/
            sys_MakeIntLink</name>
          <PSXExtensionParamValue id="1">
            <value>
              <PSXTextLiteral id="1">
                <text>../rx_ceSupport/ ProductUsed.xml</text>
              </PSXTextLiteral>
            </value>
          </PSXExtensionParamValue>
          <PSXExtensionParamValue id="1">
            <value>
              <PSXTextLiteral id="1">
                <text>contentid</text>
              </PSXTextLiteral>
            </value>
          </PSXExtensionParamValue>
          <PSXExtensionParamValue id="1">
            <value>
              <PSXHtmlParameter id="1">
                <name>sys_contentid</name>
              </PSXHtmlParameter>
            </value>
          </PSXExtensionParamValue>
          <PSXExtensionParamValue id="1">
            <value>
              <PSXTextLiteral id="1">
                <text>sys_revision</text>
              </PSXTextLiteral>
            </value>
          </PSXExtensionParamValue>
          <PSXExtensionParamValue id="1">
            <value>
              <PSXHtmlParameter id="1">
                <name>sys_revision</name>
              </PSXHtmlParameter>
            </value>
          </PSXExtensionParamValue>
        </PSXExtensionCall>
      </PSXUrlRequest>
    </PSXChoices>
  </PSXUISet>
</PSXDisplayMapping>
```

```
</PSXUISet>
<PSXDisplayMapper id="8" fieldSetRef="productused">
  <PSXDisplayMapping>
    <FieldRef>productusedvalue</FieldRef>
  </PSXDisplayMapping>
</PSXDisplayMapper>
</PSXDisplayMapping>
```

sys_CheckBoxGroup To store or generate checkbox values in another control or stylesheet, use the value `lookup` for the type attribute. Define a URL with the control or stylesheet's address in a `PSXUrlRequest` child element.

Example UI Definition for `lookup`:

```
<PSXDisplayMapping>
  <FieldRef>productused</FieldRef>
  <PSXUISet>
    <Label>
      <PSXDisplayText>Product Used:</PSXDisplayText>
    </Label>
    <PSXControlRef name="sys_CheckBoxGroup"/>
    <PSXChoices type="lookup" sortOrder="ascending">
      <PSXUrlRequest>
        <PSXExtensionCall id="10">
          <name>Java/global/percussion/generic/
            sys_MakeIntLink</name>
          <PSXExtensionParamValue id="1">
            <value>
              <PSXTextLiteral id="1">
                <text>../rx_ceSupport/ ProductUsed.xml</text>
              </PSXTextLiteral>
            </value>
          </PSXExtensionParamValue>
          <PSXExtensionParamValue id="1">
            <value>
              <PSXTextLiteral id="1">
                <text>contentid</text>
              </PSXTextLiteral>
            </value>
          </PSXExtensionParamValue>
          <PSXExtensionParamValue id="1">
            <value>
              <PSXHtmlParameter id="1">
                <name>sys_contentid</name>
              </PSXHtmlParameter>
            </value>
          </PSXExtensionParamValue>
          <PSXExtensionParamValue id="1">
            <value>
              <PSXTextLiteral id="1">
                <text>sys_revision</text>
              </PSXTextLiteral>
            </value>
          </PSXExtensionParamValue>
          <PSXExtensionParamValue id="1">
            <value>
              <PSXHtmlParameter id="1">
                <name>sys_revision</name>
              </PSXHtmlParameter>
            </value>
          </PSXExtensionParamValue>
        </PSXExtensionCall>
      </PSXUrlRequest>
    </PSXChoices>
  </PSXUISet>
</PSXDisplayMapping>
```



```

        </PSXHtmlParameter>
    </value>
    </PSXExtensionParamValue>
</PSXExtensionCall>
</PSXUrlRequest>
</PSXChoices>
</PSXUISet>
<PSXDisplayMapper id="8" fieldSetRef="productused">
    <PSXDisplayMapping>
        <FieldRef>productusedvalue</FieldRef>
    </PSXDisplayMapping>
</PSXDisplayMapper>
</PSXDisplayMapping>

```

To store checkbox values in a Rhythmyx table other than the RXLOOKUP table, use the value `internalLookup` for the type attribute. Define a URL in a `PSXUrlRequest` child element so that Rhythmyx can get the entries through an internal request to the specified URL. The lookup query must conform to the `sys_Lookup.dtd`. You can add it to the content editor application or place it in a separate application. See *Creating an Internal Lookup Query* (on page 195) for a procedure for adding the lookup query.

Example UI Definition for internal lookup:

```

<PSXDisplayMapping>
    <FieldRef>productused</FieldRef>
    <PSXUISet>
        <Label>
            <PSXDisplayText>Product Used:</PSXDisplayText>
        </Label>
        <PSXControlRef name="sys_CheckBoxGroup"/>
        <PSXChoices type="internallookup" sortOrder="ascending">
            <PSXUrlRequest> <PSXExtensionCall id="10">
                <name>Java/global/percussion/generic/
                sys_MakeIntLink</name>
                <PSXExtensionParamValue id="1">
                    <value>
                        <PSXTextLiteral id="1">
                            <text>../rx_ceSupport/ ProductUsed.xml</text>
                        </PSXTextLiteral>
                    </value>
                </PSXExtensionParamValue>
                <PSXExtensionParamValue id="1">
                    <value>
                        <PSXTextLiteral id="1">
                            <text>contentid</text>
                        </PSXTextLiteral>
                    </value>
                </PSXExtensionParamValue>
                <PSXExtensionParamValue id="1">
                    <value>
                        <PSXHtmlParameter id="1">
                            <name>sys_contentid</name>
                        </PSXHtmlParameter>
                    </value>
                </PSXExtensionParamValue>
            </PSXExtensionParamValue id="1">

```

```

        <value>
        <PSXTextLiteral id="1">
            <text>sys_revision</text>
        </PSXTextLiteral>
        </value>
    </PSXExtensionParamValue>
    <PSXExtensionParamValue id="1">
        <value>
        <PSXHtmlParameter id="1">
            <name>sys_revision</name>
        </PSXHtmlParameter>
        </value>
    </PSXExtensionParamValue>
    </PSXExtensionCall>
</PSXUrlRequest>
</PSXChoices>
</PSXUISet>
<PSXDisplayMapper id="8" fieldSetRef="productused">
    <PSXDisplayMapping>
        <FieldRef>productusedvalue</FieldRef>
        <PSXUISet/>
    </PSXDisplayMapping>
</PSXDisplayMapper>

```

</PSXDisplayMapping>sys_CheckBoxGroup To store checkbox values in a Rhythmyx table other than the RXLOOKUP table, use the value `internalLookup` for the type attribute. Define a URL in a `PSXUrlRequest` child element so that Rhythmyx can get the entries through an internal request to the specified URL. The lookup query must conform to the `sys_Lookup.dtd`. You can add it to the content editor application or place it in a separate application. See *Creating an Internal Lookup Query* (on page 195) for a procedure for adding the lookup query.

Example UI Definition for internal lookup:

```

<PSXDisplayMapping>
    <FieldRef>productused</FieldRef>
    <PSXUISet>
        <Label>
            <PSXDisplayText>Product Used:</PSXDisplayText>
        </Label>
    <PSXControlRef name="sys_CheckBoxGroup"/>
    <PSXChoices type="internallookup" sortOrder="ascending">
        <PSXUrlRequest> <PSXExtensionCall id="10">
            <name>Java/global/percussion/generic/
            sys_MakeIntLink</name>
            <PSXExtensionParamValue id="1">
                <value>
                <PSXTextLiteral id="1">
                    <text>../rx_ceSupport/ ProductUsed.xml</text>
                </PSXTextLiteral>
                </value>
            </PSXExtensionParamValue>
            <PSXExtensionParamValue id="1">
                <value>
                <PSXTextLiteral id="1">
                    <text>contentid</text>
                </PSXTextLiteral>
                </value>
            </PSXExtensionParamValue>
        </PSXExtensionCall>
    </PSXChoices>

```

```

    </PSXExtensionParamValue>
    <PSXExtensionParamValue id="1">
      <value>
        <PSXHtmlParameter id="1">
          <name>sys_contentid</name>
        </PSXHtmlParameter>
      </value>
    </PSXExtensionParamValue>
    <PSXExtensionParamValue id="1">
      <value>
        <PSXTextLiteral id="1">
          <text>sys_revision</text>
        </PSXTextLiteral>
      </value>
    </PSXExtensionParamValue>
    <PSXExtensionParamValue id="1">
      <value>
        <PSXHtmlParameter id="1">
          <name>sys_revision</name>
        </PSXHtmlParameter>
      </value>
    </PSXExtensionParamValue>
  </PSXExtensionCall>
</PSXUrlRequest>
</PSXChoices>
</PSXUISet>
<PSXDisplayMapper id="8" fieldSetRef="productused">
  <PSXDisplayMapping>
    <FieldRef>productusedvalue</FieldRef>
    <PSXUISet/>
  </PSXDisplayMapping>
</PSXDisplayMapper>
</PSXDisplayMapping>'

```

sys_DropDownSingle

The `sys_DropDownSingle` is a basic DropDown Html control. When a user clicks on the control, Rhythmyx displays a list of potential values for the field. The user can select one of these values to populate the field. Use the `PSXNullEntry` element to define the behavior of the field when no value has been selected. The dimension is *single*.

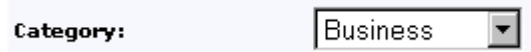


Figure 45: Example `sys_DropDownSingle`

Parameters

Parameter	Data Type	Parameter Type	Description	Default
id	String	Generic	XHTML 1.0 attribute	None
class	String	Generic	XHTML 1.0 attribute	None
style	String	Generic	XHTML 1.0 attribute	None
size	Number	Generic	XHTML 1.0 attribute	None
multiple	String	Generic	XHTML 1.0 attribute	None
tabindex	Number	Generic	XHTML 1.0 attribute	None
disabled	String	Generic	XHTML 1.0 attribute	None

Example Field Definition

```
<PSXField name="editor" showInSummary="yes" showInPreview="yes"
forceBinary="no">
  <DataLocator>
    <PSXBackEndColumn id="0">
      <tableAlias>BOOKS</tableAlias>
      <column>EDITOR</column>
      <columnAlias/>
    </PSXBackEndColumn>
  </DataLocator>
  <DataType/>
  <OccurrenceSettings dimension="optional" multiValuedType="delimited"
delimiter=";" />
</PSXField>
```

Example UI Definition

Use the type attribute of the PSXChoices element in the UI definition to specify how you are storing the drop down values so that Rhythmyx can retrieve them.

To store drop down values in the lookup table (RXLOOKUP), use the value global for the type attribute. Rhythmyx gets the values when it builds the document by using the value in the Key element to determine the lookup key for the table. Use the Rhythmyx System Administrator to maintain the Keywords in this table. The key is a system-generated value, so manual addition of values to this table is not recommended.

Example UI Definition for global:

```
<PSXDisplayMapping>
  <FieldRef>global</FieldRef>
  <PSXUISet>
    <Label>
      <PSXDisplayText>Global choices:</PSXDisplayText>
    </Label>
    <PSXControlRef name="sys_DropDownSingle"/>
    <PSXChoices type="global" sortOrder="ascending">
      <Key>3</Key>
      <PSXNullEntry sortOrder="first"
includeWhen="onlyIfNull">
        <PSXEntry sequence="0" default="no">
```

```

        <PSXDisplayText> -- choose -- </PSXDisplayText>
        <Value>0</Value>
    </PSXEntry>
</PSXNullEntry>
</PSXChoices>
</PSXUISet>
</PSXDisplayMapping>

```

To store drop down values in the content editor definition as a list of PSXEntry elements, use the value `local` for the type attribute.

Example UI Definition for local:

```

<PSXDisplayMapping>
    <FieldRef>local</FieldRef>
    <PSXUISet>
        <Label>
            <PSXDisplayText>Local choices:</PSXDisplayText>
        </Label>
        <PSXControlRef name="sys_DropDownSingle"/>
        <PSXChoices type="local" sortOrder="user">
            <PSXEntry sequence="0" default="no">
                <PSXDisplayText>Computing Machinery</PSXDisplayText>
                <Value>acm</Value>
            </PSXEntry>
            <PSXEntry sequence="1" default="no">
                <PSXDisplayText> Society of Electrical
Engineers</PSXDisplayText>
                <Value>ieee</Value>
            </PSXEntry>
            <PSXEntry sequence="3" default="no">
                <PSXDisplayText>Society for Prevention</PSXDisplayText>
                <Value>spca</Value>
            </PSXEntry>
            <PSXNullEntry sortOrder="last" includeWhen="always">
                <PSXEntry sequence="0" default="no">
                    <PSXDisplayText> -- choose -- </PSXDisplayText>
                    <Value>0</Value>
                </PSXEntry>
            </PSXNullEntry>
            <DefaultSelected>
                <PSXDefaultSelected type="nullEntry"/>
            </DefaultSelected>
        </PSXChoices>
    </PSXUISet>
</PSXDisplayMapping>

```

To store drop down values in a Rhythmyx table other than the RXLOOKUP table, use the value `internalLookup` for the type attribute. Define a URL in a PSXUrlRequest child element so that Rhythmyx can get the entries through an internal request to the specified URL. The lookup query must conform to the `sys_Lookup.dtd`. You can add it to the content editor application or place it in a separate application. See *Creating an Internal Lookup Query* (on page 195) for a procedure for adding the lookup query.

Example UI Definition for internal lookup:

```

<PSXDisplayMapping>

```

```
<FieldRef>lookup</FieldRef>
  <PSXUISet>
    <Label>
      <PSXDisplayText>Lookup choices:</PSXDisplayText>
    </Label>
    <PSXControlRef name="sys_DropDownSingle"/>
      <PSXChoices type=" internalLookup " sortOrder="user">
        <PSXUrlRequest>
          <Href>http://38.164.160.56:9992/Rhythmyx/sys_wfLookups/
            extroles.html</Href>
          <PSXParam name="workflowid">
            <DataLocator>
              <PSXTextLiteral id="0">
                <text>1</text>
              </PSXTextLiteral>
            </DataLocator>
          </PSXParam>
        </PSXUrlRequest>
      </PSXChoices>
    </PSXUISet>
  </PSXDisplayMapping>
```

To store or generate drop down values in another control or stylesheet, use the value `lookup` for the `type` attribute. Define a URL with the control or stylesheet's address in a `PSXUrlRequest` child element.

Example UI Definition for `lookup`:

```
<PSXDisplayMapping>
  <FieldRef>lookup</FieldRef>
  <PSXUISet>
    <Label>
      <PSXDisplayText>Lookup choices:</PSXDisplayText>
    </Label>
    <PSXControlRef name="sys_DropDownSingle"/>
      <PSXChoices type="lookup " sortOrder="user">
        <PSXUrlRequest>
          <Href>http://38.164.160.56:9992/Rhythmyx/sys_wfLookups/
            extroles.html</Href>
          <PSXParam name="workflowid">
            <DataLocator>
              <PSXTextLiteral id="0">
                <text>1</text>
              </PSXTextLiteral>
            </DataLocator>
          </PSXParam>
        </PSXUrlRequest>
      </PSXChoices>
    </PSXUISet>
  </PSXDisplayMapping>
```

sys_EditBox

The `sys_EditBox` control is used to input data in a standard one-line edit box. This control corresponds to a single, one-dimensional field. The dimension is *single*.



Figure 46: Example `sys_EditBox`

Parameters

Parameter	Data Type	Parameter Type	Description	Default
id	String	Generic	XHTML 1.0 attribute	None
class	String	Generic	XHTML 1.0 attribute	None
style	String	Generic	XHTML 1.0 attribute	None
size	String	Generic	XHTML 1.0 attribute	50
maxlength	Number	Generic	XHTML 1.0 attribute	None
tabindex	Number	Generic	XHTML 1.0 attribute	None

Example Field Definition

```
<PSXField name="displaytitle" showInSummary="yes" showInPreview="yes"
  forceBinary="no">
  <DataLocator>
    <PSXBackendColumn id="0">
      <tableAlias>RXARTICLE</tableAlias>
      <column>DISPLAYTITLE</column>
      <columnAlias/>
    </PSXBackendColumn>
  </DataLocator>
  <DataType/>
  <OccurrenceSettings dimension="optional" multiValuedType="delimited"
    delimiter=";" />
</PSXField>
```

Example UISet

```
<PSXDisplayMapping>
  <FieldRef>displaytitle</FieldRef>
  <PSXUISet>
    <Label>
      <PSXDisplayText>Display Title:</PSXDisplayText>
    </Label>
```

```
<PSXControlRef name="sys_EditBox">
  <PSXParam name="maxlength">
    <DataLocator>
      <PSXTextLiteral id="1">
        <text>30</text>
      </PSXTextLiteral>
    </DataLocator>
  </PSXParam>
</PSXControlRef>
```

EditLive for Java Editor

Ephox's EditLive for Java (ELJ) HTML editor is now the default HTML editor for Rhythmyx content editors.

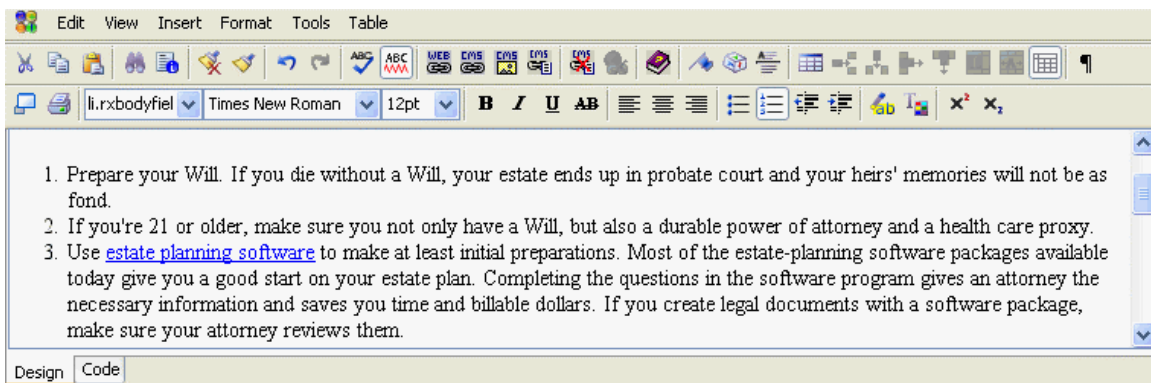


Figure 47: *sys_EditLive Control*

Customers who are upgrading and have previously used the `sys_eWebEditPro` control may continue to use it as a deprecated feature.

ELJ is provided by special license; check your Rhythmyx licensing agreement to confirm that you are licensed to use ELJ. Note that you must be running JRE Version 1.4.207 or higher to run the `sys_EditLive` control (JRE Version 1.4.207 or higher is required for Rhythmyx Version 5.7).

An XML configuration file (`elj_config.xml`) drives the functionality of the `<Rhythmyxroot>/rx_resources/ephox` and `<Rhythmyxroot>/sys_resources/ephox`. Only customize the file in `<Rhythmyxroot>/rx_resources/ephox`. On upgrade, Rhythmyx overwrites the file in `<Rhythmyxroot>/sys_resources/ephox`. To take advantage of any upgrades, you must copy the `elj_config.xml` file in `sys_resources/ephox` to `rx_resources/ephox` (or copy the changed portions of the file to your file in the `rx_resources/ephox` folder). You may create multiple custom files, but when your control runs, it can only reference one of them.

ELJ editor, defining the controls and styles available to the end user. You can customize this configuration file to add new functionality or to remove existing functionality.

Rhythmyx installs to the default configuration file to both Percussion Software will provide instructions for modifying the installation in `sys_resources/ephox` to take advantage of upgrades to the ELJ editor.


NOTE: The deprecated `sys_eWebEditPro` documentation is now located on the Rhythmyx extranet at <http://www.percussion.com/support/rhythmyx-extranet/>

sys_EditLive Control

Sys_EditLive is a multiple-line text entry control in which the user can type and edit text. It displays a DHTML editor that allows a user to enter text and apply standard formatting, such as changing the font or the alignment.

Sys_EditLive also includes a feature that allows users to copy content from a Microsoft Word file and paste it into sys_EditLive. The appearance of the content remains the same and sys_EditLive generates the corresponding HTML markup.

The default Rhythmyx installation of this editor includes built-in support for inserting inline links and images in addition to the standard features of the ELJ editor. (For details about the standard features and

Rhythmyx features of ELJ, click the help button  in the control). This control works with all browsers that Rhythmyx supports.

Parameters

Each sys_EditLive control includes the following parameters. The default values are set in the file <Rhythmyx root>/sys_resources/stylesheets/sys_templates.xml.

Parameter	Data Type	Parameter Type	Description	Default
Width	String	Generic	This parameter specifies the width of the inline frame. This parameter may be either a pixel or a percentage of the available horizontal .	760
Height	String	Generic	This parameter specifies the height of the inline frame. This parameter may be either a pixel or a percentage of the available vertical.	250
config_src_url	String	Generic	This parameter specifies the location of the config.xml that the control will use for configuration.	../rx_resources/ephox/elj_config.xml
config_download	String	Generic	This parameter specifies the location of the download directory.	../rx_resources/ephox/editlivejava
InlineLinkSlot	String	Generic	This parameter specifies the id of inline link slot. The search dialog for the inline link slot shows the content types that have a variant associated with the slot.	103

Parameter	Data Type	Parameter Type	Description	Default
InlineImageSlot	String	Generic	This parameter specifies the id of inline image slot. The search dialog for the inline image slot shows the content types that have a variant associated with the slot.	104
InlineVariantSlot	String	Generic	This parameter specifies the id of inline variant slot. The search dialog for the inline variant slot shows the content types that have a variant associated with the inline variant slot.	105
DebugLevel	String	Generic	This parameter specifies the debug level for the EditLive Applet. The allowed levels are (fatal, error, warn, info, debug, http)	info

You can change the values of sys_EditLive parameters for any individual Content Editor field.

To change the value of a sys_EditLive parameter for a Content Editor field:

- 1** In the Rhythmyx Workbench, access the Content Editor Properties dialog for the Content Editor for which you want to change the field parameter value.
- 2** In the field using the sys_EditLive control, double-click on sys_EditLive to display the browse button (...) next to it.
- 3** Click the browse button (...).

Rhythmyx displays the Display Control Properties for Associations dialog.

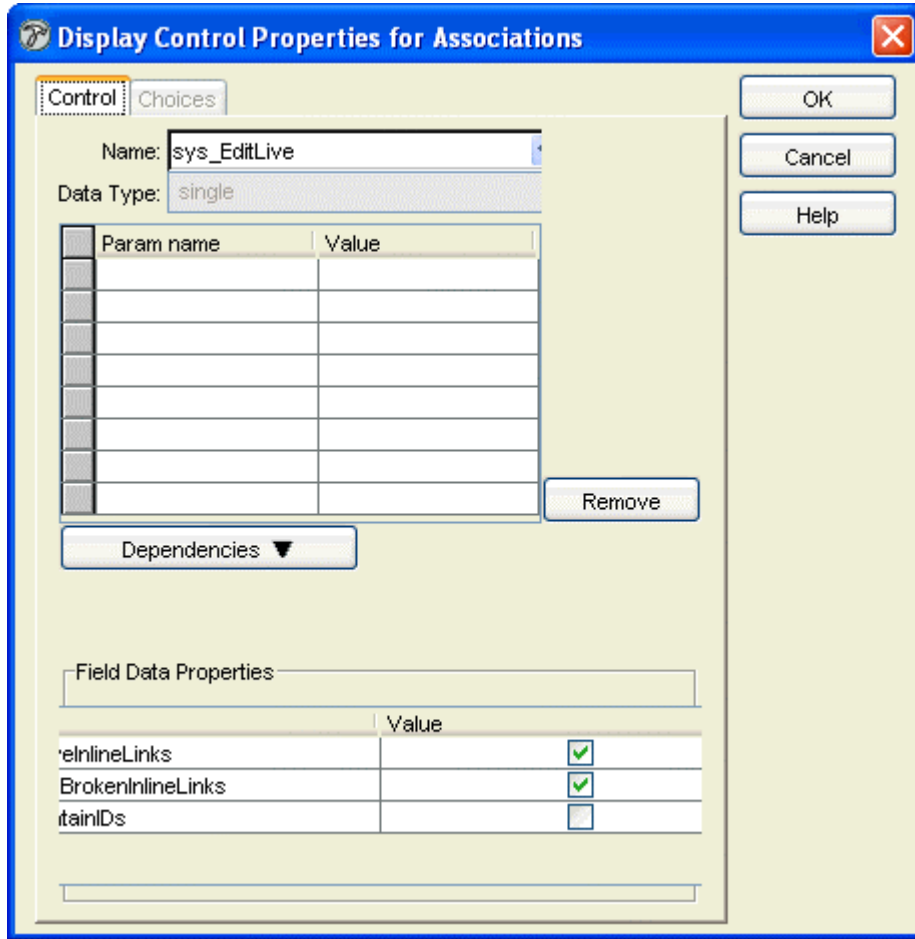


Figure 48: Display Control Properties for Associations dialog

Parameters that take the default values from the sys_templates.xml file are not shown in the Param name/Value table.

- 1 Click in the **Param name** column and choose the parameter whose value you want to change from the drop list.

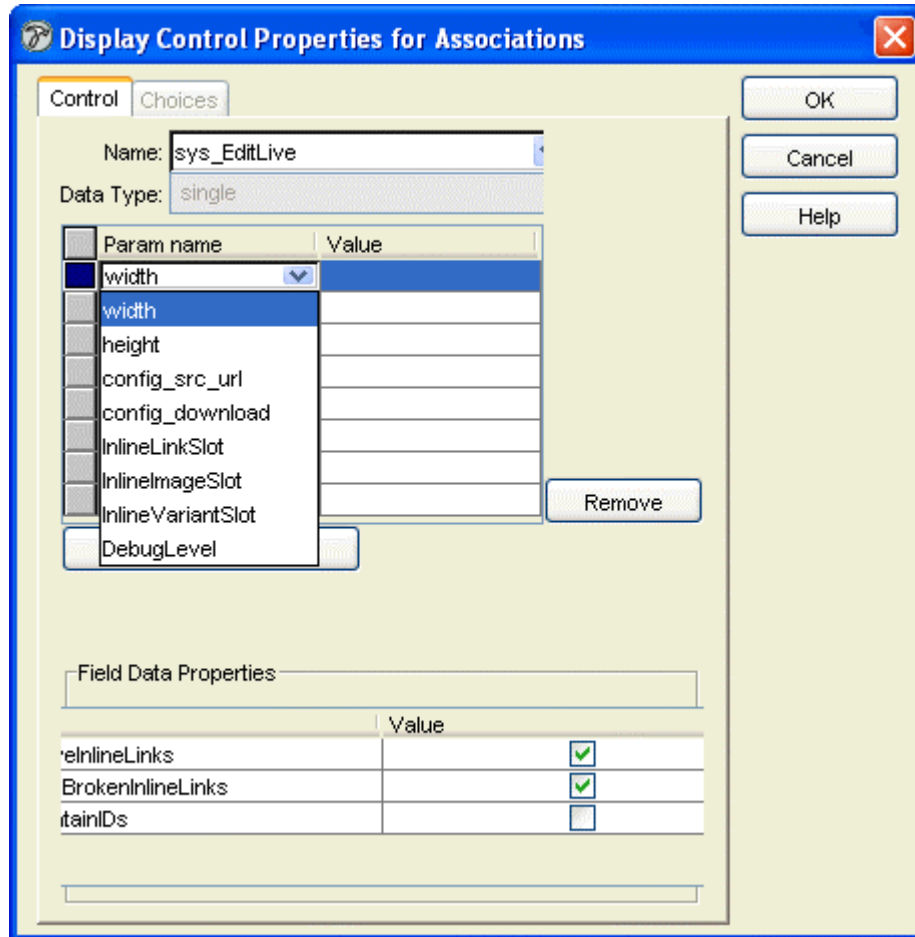


Figure 49: *sys_EditLive* Parameters

- 2 Click in the **Value** column of the same row and enter the value you want to use for this instance of the control.

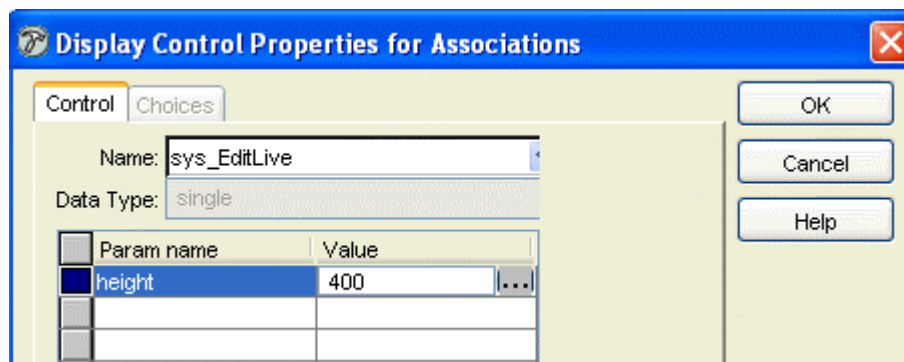


Figure 50: *Change to Ephox* parameter

- 3 On the Display Control Properties for Associations dialog, click [OK].
- 4 On the Field Properties dialog, click [OK].

- 5 On the Content Editor Properties dialog, click **[OK]**.
- 6 When you open the Content Editor in Content Explorer, the field should reflect the change made to the parameter. Note: You may have to choose *View > Refresh* in Content Explorer before seeing the change.

Adding the `sys_EditLive` Control to a Content Editor

To add the `sys_EditLive` editor to a content editor, select `sys_EditLive` as the Control Name for the field for which you want to use the ELJ editor. No additional implementation is required when implementing a content editor through the Content Editor Properties dialog. Rhythmyx automatically adds the `sys_xdTextCleanup` exit to the Content Editor, and automatically configures its required parameter.

Content Assemblers and the ELJ HTML Editor

All Content Assemblers that assemble content edited using the `sys_EditLive` control must include the `sys_xdTextToTree` exit.

Customizing the ELJ Editor

You can customize both the parameters of the `sys_EditLive` control and the configuration files of the ELJ editor itself.

Customizing the `sys_EditLive` control

The parameters of the `sys_EditLive` control define the height and width of the display of the editor and the path to configuration file (`elj_config.xml`). You can customize these parameters in the control definition (in the Display Control Properties for `<field>` dialog). If you customize the configuration file for the ELJ editor, update the `config_src_url` parameter of each instance of the `sys_EditLive` control to point to the correct configuration file.

Customizing ELJ Configuration

The ELJ editor is a robust and highly customizable HTML editor.

Most customizations of the ELJ editor involve modifications to the configuration file (`elj_config.xml` in the Rhythmyx implementation). Do not modify the default configuration file, which is located in the `<Rhythmyxroot>/sys_resources/ephox` directory. Instead, modify the copy in `<Rhythmyxroot>/rx_resources/ephox`. You may create multiple custom configuration files and give them different names or store them in different directories.

To customize the control, you may want to add javascript functions that extend its capabilities. See *Adding Custom Menu and Toolbar Actions* (on page 171) for instructions on adding custom javascript functions. Several instances of the control can use the same configuration XML file (shared configuration file), or you can use a local configuration file for each instance of the editor; you can also use a shared configuration file for some instances and a local configuration file for other instances. As a best practice, store the files in the following manner:

- The default configuration file is stored in the directory `sys_resources/ephox`. This configuration file should not be modified.
- Shared configuration files should be stored in a directory with the path `rx_resources/[path]/ephox`, where `[path]` is the path to a subdirectory that logically categorizes the file. For example, you might want to use the name of your project as part of the path; for a project with the name *sample*, the path would be `rx_resources/sample/ephox`.
- Local configuration files should be stored in a subdirectory of the Content Editor application. For example, if you have a local configuration file for a Press Release content editor, the configuration file would be stored in the subdirectory `Rhythmyxroot/pressrelease/ephox`.

To define an instance of the `sys_EditLive` control to use a customized configuration file:

- 1 In the Rhythmyx Workbench, access the Content Editor Properties dialog for the Content Editor in which you want to use the ELJ editor.
- 2 Select the field in which you want to use the ELJ editor and click **[Edit]**.
Rhythmyx displays the Field Properties dialog. (If you are defining a new field, Rhythmyx displays the New Field Properties dialog.)
- 3 In the **Control** field, choose `sys_EditLive`.
- 4 Click the browse button (...) next to the **Control** field.
Rhythmyx displays the Display Control Properties for <field> dialog.
- 5 Click in the **Param name** column and choose `config_src_url`.
- 6 Click in the **Value** column of the same row and enter the relative URL of the configuration file you want to use for this instance of the control as a literal value.
- 7 On the Display Control Properties for <field> dialog, click **[OK]**.
- 8 On the Field Properties dialog, click **[OK]**.
- 9 On the Content Editor Properties dialog, click **[OK]**.

The changes will take effect the next time you start your application. To see your changes, save the application, log in to Rhythmyx, and activate the editor.

For guidance on customizing (and localizing) the sys_EditLive editor, consult the Ephox EditLive! for Java Developer's Guide in the developer section of the Ephox Web Site (www.ephox.com) (<http://www.ephox.com>)).

Adding Custom Menu and Toolbar Actions

Rhythmyx provides you with xml code that you can use to create custom actions for your sys_EditLive editor. The xml code is located in <Rhythmyx root>/rx_resources/ephox/rx_ephox_custom.xml.

To add the toolbar button and/or menu choice associated with the custom action, you must modify your config file (elj_config.xml by default). To add the custom action, you must add a javascript function that uses the EditLive Java API to the rx_ephox_custom.xml file.

Rhythmyx adds your modified code to the sys_EditLive template in sys_Templates.xml, which incorporates it into the control.

To create a custom sys_EditLive function:

This procedure uses the example of an action that opens a window showing the source code between the body tags in the sys_EditLive control.

- 1 Modify your config file (elj_config.xml by default) to show the new menu item and/or toolbar button. The EditLive JavaScript API defines the elements <customMenuItem> and <customToolbarButton> which you configure as shown in this step to add the new Menu item and/or Toolbar button.

- a) Find the <menu> sub-element for the menu that you want to add the action to in the <menubar> element in the configuration file and add a <customMenuItem> element for the action. Below, the <customMenuItem> element is shown in bold. Copy the format of this sample element.

```

<menu name="ephox_editmenu">
<menuItem name="Undo"/>
<menuItem name="Redo"/>
<menuSeparator/>
<menuItem name="Cut"/>
<menuItem name="Copy"/>
<menuItem name="Paste"/>
<menuItem name="PasteSpecial"/>
<menuSeparator/>
<menuItem name="Select"/>
<menuItem name="SelectAll"/>
<menuSeparator/>
<menuItem name="Find"/>
<menuSeparator/>
<customMenuItem action="showbodysource"
imageURL="..rx_resources/ephox/images/bSource.gif" name="ShowBodySource" rxconfig="yes"
text="Shows Body Source" value="RxEphoxShowBodySource"/>
</menu>

```

- b) Find the <toolbar name="Command"> sub-element in the <tools> element in the configuration file and add a <customToolbarButton> element for the action. Below, the <customToolbarButton> element is in bold. Copy the format of this sample element.

```

<toolbar name="Command">
<toolbarButton name="Cut"/>

```

```

<toolbarButton name="Copy"/>
<toolbarButton name="Paste"/>
<toolbarSeparator/>

```

....

```

<customToolbarButton action="showbodysource" imageURL="./rx_resources/ephox/images/bSource.gif"
name=" ShowBodySource " rxconfig="yes" text=" Shows Body Source " value=" RxEphoxShowBodySource
"/>

```

```

</toolbar>

```

- 2 Add the JavaScript function to rx_ephox_custom.xml. Replace RxEphoxDummyFunction with your own. In our example, the custom JavaScriptFunction is:

```

<![CDATA [
    function RxEphoxShowBodySource_]]><xsl:value-of select="$name"/><![CDATA[()
    {
        // Get EditLive editor instance
        var EditorName = "]]><xsl:value-of select="@paramName" /><![CDATA[";
        var editor = getEditor(EditorName);
        //Get a reference to the EditLive applet
        var ephox = editor.objectref;

        var body = ephox.GetBody('rxShowBody_]]><xsl:value-of select="$name"/><![CDATA[, false]; // call back
function
    }

    function rxShowBody_]]><xsl:value-of select="$name"/><![CDATA[(body)
    {
        alert(body);
    }

]]>

```

For additional information about adding custom functions, see the Ephox EditLive! for Java Developer's Guide on the developer section of the Ephox Web Site (www.ephox.com (<http://www.ephox.com>)).

Best Practices: sys_EditLive

To simplify maintenance and promote effective technical support, observe the following Best Practices when working with the ELJ editor and the sys_EditLive control:

- Keep shared configuration files (configuration files used by more than one instance of the control) in directories with the name Rhythmyxroot/rx_resources/[path]/ephox, where [path] defines a category (such as the name of a project or customer). For example, if you are working on a project named *sample*, the directory should be Rhythmyxroot/rx_resources/sample/ephox.
- If only one editor is going to use a configuration file, store the file in a subdirectory of the editor application directory. If you decide to use this configuration file for other editors, move it to a shared directory and update the config_src_url parameters of the instances of the control that use that configuration file.
- When disabling a command or parameters of a command (such as lists of fonts or font sizes), hide the disabled elements first by commenting them out (<!-- text --!>), then test and refine your development. Remove the disabled commands and parameters when testing is complete to minimize clutter in the files and simplify future modification.

Upgrading from sys_eWebEditPro to sys_EditLive

If you upgrade to Rhythmyx 5.7, you receive the sys_EditLive control in addition to the sys_eWebEditPro control. Both options appear in the Control Name drop list in the Content Editor Properties dialog.

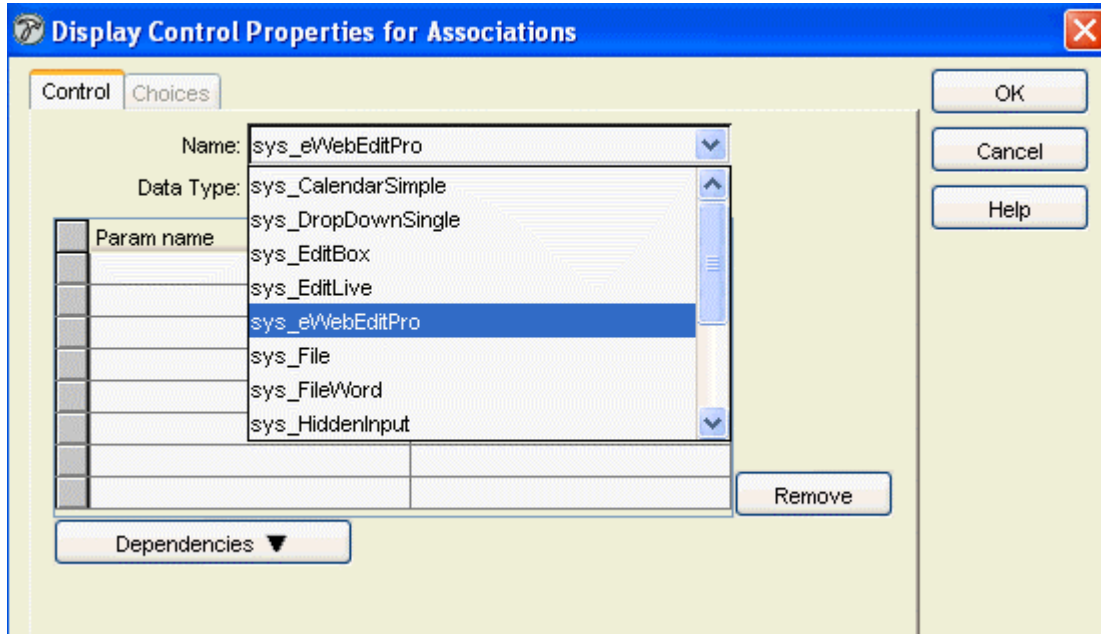


Figure 51: sys_EditLive and sys_eWebEditPro in the same editor

Content Editors fields that already use the sys_eWebEditPro control will continue to use it unless you change them manually. When you change the control from sys_eWebEditPro to sys_EditLive, sys_EditLive automatically adopts the field's sys_eWebEditPro values for the following parameters (by default, these parameters have the same values in sys_EditLive and sys_eWebEditPro):

- width
- height
- inlineLinkSlot
- inlineImageSlot
- inlineWidthSlot

For more information about the parameters, see *sys_EditLive Control* (on page 165) Control.

To manually replace eWebEditPro with ELJ in a Content Editor:

- 1 In the Rhythmyx Workbench, open the Content Editor application whose editor you want to change and double-click on the Content Editor resource.
Rhythmyx displays the Content Editor Properties dialog with the current configuration data for the Content Editor.
- 2 Select the Content Editor field whose control you want to change and click the **[Edit]** button.
Rhythmyx displays the Field Properties dialog.
- 3 In the Control field, click the drop list and choose **sys_EditLive**.

Rhythmyx automatically sets common parameters to the same values used for the `sys_eWebEditPro` control that was used for the field.

- 4 If you want to use a customized configuration file, or modify other parameters:
 - a) Click the browse button next to the Control field.
Rhythmyx displays the Display Control Properties for <field> dialog.
 - b) Enter the parameters and associated values you want to assign to the control.
 - c) Click **[OK]** to save your edits.
- 5 On the Field Properties dialog, click **[OK]** to save the modifications you made to the field.
- 6 On the Content Editor Properties dialog click **[OK]** to save the modification you made to the Content Editor.

The changes will take effect the next time you start your application. To see your changes, stop and restart the application, log into Rhythmyx, and activate the editor.

NOTE:

You cannot mix use of the `sys_eWebEditPro` and `sys_EditLive` controls in a single Content Editor. If you mix use them, when you attempt to save the Content Editor, the following error dialog appears:

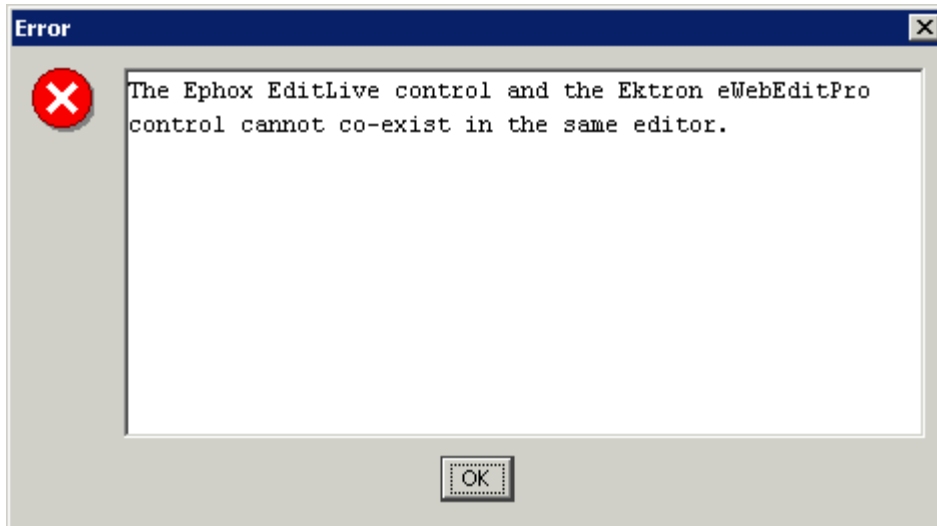


Figure 52: Error when mixing controls

Click **[OK]**, and change the fields to use the same controls.

sys_WebImageFX and the WebImageFX Editor

Your Rhythmyx license may include Ektron's WebImageFX graphics editor which includes a variety of tools for creating and editing graphics files. With the WebImageFX editor, Rhythmyx includes the WebImageFX control. The control uploads a graphics file and displays it in a Content Editor using the WebImageFX editor.

An XML configuration file (ImageEditConfig.xml) defines the WebImageFX controls and styles available to the end user. You can customize this configuration file to add new functionality or to remove existing functionality. By default, the WebImageFX editor lets you upload, create, or paste (from Windows clipboard) images to edit in its window.

During installation, Rhythmyx installs a copy of WebImageFX to `Rhythmyxroot/sys_resources/webimagefx` **and checks the version of WebImageFx in** `Rhythmyxroot/rx_resources/webimagefx`. If the version in `rx_resources` is earlier than the current version (or there is no version file), Rhythmyx backs up the copy of WebImageFX in `rx_resources` (by adding a time stamp to the directory name, for example, `webimagefx__0301_1538`), and installs the current version into it.

The following Content Editor uses the `sys_WebImageFX` control to upload and display images.

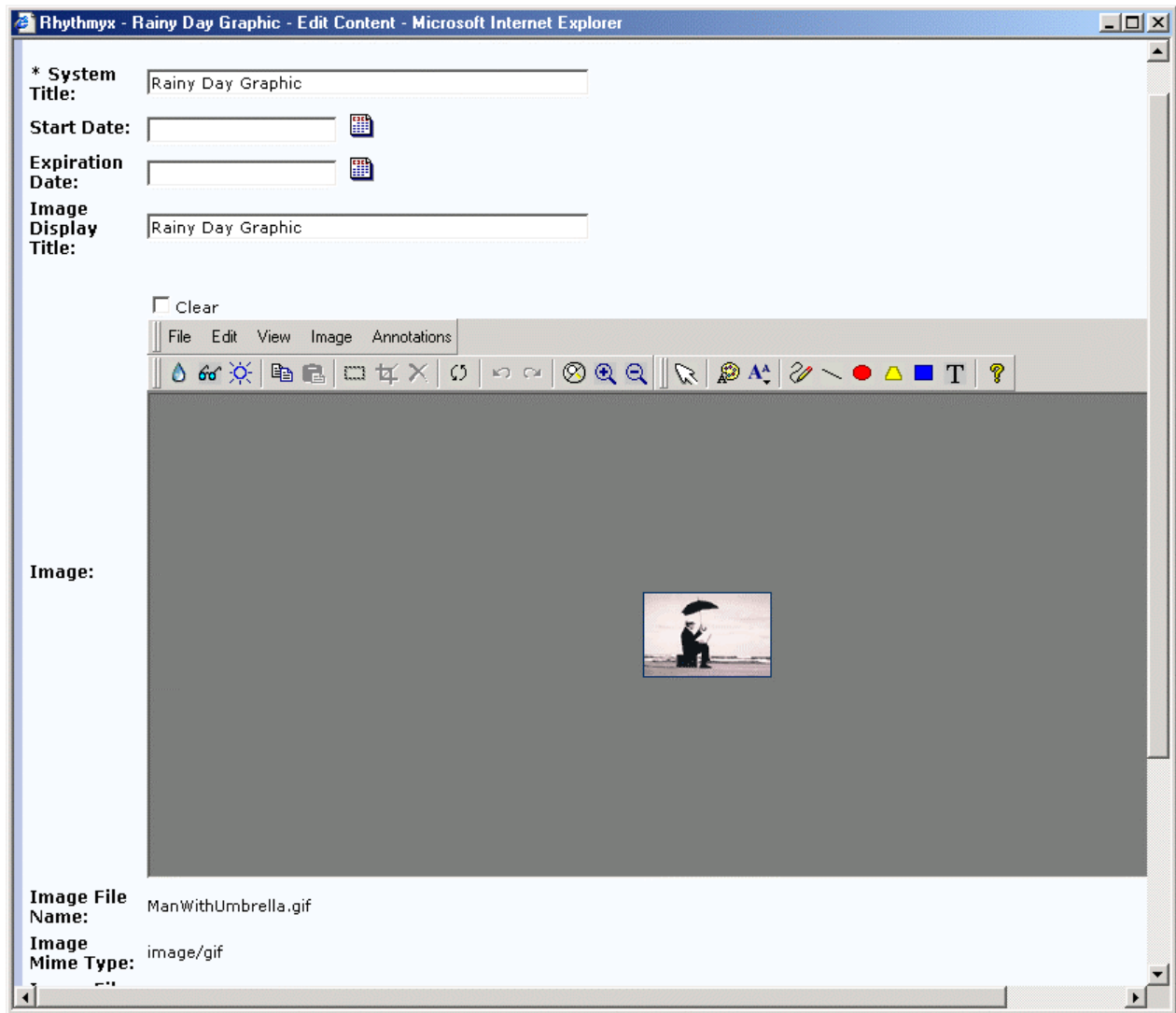


Figure 53: Content Editor with `sys_WebImageFX` control

sys_WebImageFX Control

The `sys_WebImageFX` control functions almost identically to the `sys_File control` (see "[sys_File](#)" on page 183). It includes most of the same properties as the `sys_File` control, and like the `sys_file` control, it is a file upload element that allows the user to supply a file as the input, and it corresponds to a single, one-dimensional field. The main difference between the `sys_WebImageFX` control and the `sys_File` control is that the `sys_WebImageFX` control appears in a Content Editor with the `WebImageFX` image editor.

When hand-coding a Content Editor that includes a `sys_WebImageFX` control, include `sys_FileInfo` as a pre-exit. When you create a Content Editor using the Content Editor Properties dialog, Rhythmyx adds this exit for you automatically. The `sys_FileInfo` exit searches for attached files in a content item's HTML and returns values for file name, MIME type, character length and file encoding. The exit returns the values to field names formed by combining the filename (the `<FieldRef>` value) with descriptive suffixes. If you want to return information about a file, such as the filename or size, refer to the `sys_FileInfo` exit documentation in the *Workbench Online Help* for the correct syntax.

The sys_WebImageFX control displays a WebImageFX editor that allows a user to modify an image. For details about the standard features of WebImageFX, see the developer's guide at <http://www.ektron.com/webimagefx.aspx> (<http://www.ektron.com/webimagefx.aspx>). Note: This control has been tested with Internet Explorer.

Parameters:

Parameter	Data Type	Parameter Type	Description	Default
id	String	Generic	This parameter assigns a name to an element. This name must be unique in a document.	None
class	String	Generic	This parameter assigns a class name or set of class names to an element. Any number of elements may be assigned the same class name or names. Multiple class names must be separated by white space characters.	datadisplay
style	String	Generic	This parameter specifies style information for the current element. The syntax of the value of the style attribute is determined by the default style sheet language.	None
width	Number	Generic	This parameter tells the user agent the initial width of the control. The width is given in pixels.	800
height	Number	Generic	This parameter tells the user agent the initial width of the control. The width is given in pixels.	400
config_src_url	String	Generic	This parameter specifies the location of the config.xml that will the control will use for configuration.	../sys_resources/webimagefx/ImageEditConfig.xml
cleartext	String	custom	This parameter determines the text that will be displayed along with a checkbox when the field supports being cleared.	Clear

Example Field Definition

```
<PSXField clearBinaryParam="yes" forceBinary="yes"
modificationType="user" name="uploadfilephoto" showInPreview="yes"
```

```
showInSummary="no" type="local" userCustomizable="yes"
userSearchable="yes">
    <DataLocator>
        <PSXBackEndColumn id="0">
            <tableAlias>RXWEBIMAGEFX</tableAlias>
            <column>IMGDATA</column>
            <columnAlias/>
        </PSXBackEndColumn>
    </DataLocator>
    <DataType>binary</DataType>
    <DataFormat>max</DataFormat>
    <OccurrenceSettings delimiter=";"
dimension="optional" multiValuedType="delimited"/>
    <PSXPropertySet>
        <PSXProperty locked="no"
name="mayHaveInlineLinks">
            <Value type="Boolean">no</Value>
        </PSXProperty>
        <PSXProperty locked="no"
name="cleanupBrokenInlineLinks">
            <Value type="Boolean">no</Value>
        </PSXProperty>
    </PSXPropertySet>
</PSXField>
```

Example UI Definition

```
<PSXDisplayMapping>
    <FieldRef>uploadfilephoto</FieldRef>
    <PSXUISet>
        <Label>
            <PSXDisplayText>Image:</PSXDisplayText>
        </Label>
        <PSXControlRef id="1477" name="sys_webImageFx"/>
        <ErrorLabel>
            <PSXDisplayText>Uploadfilephoto:</PSXDisplayText>
        </ErrorLabel>
    </PSXUISet>
</PSXDisplayMapping>
```

Adding the sys_WebImageFX Control to a Content Editor

To create a Content Editor that uses WebImageFX:

- 1 Follow the procedure in the document *Implementing Content Editors* for creating a new Content Editor.
- 2 Include a field with the **Field Name** *uploadfilephoto* and the **Control Name** *sys_WebImageFX*.
- 3 When you choose *sys_WebImageFX* as the **Control Name**, Rhythmyx automatically includes the *sys_FileInfo* exit, which fills in the uploaded file's name, mime type, extension, and size into the proper Content Editor fields if you provide them. Add any of these fields to the Content Editor. See *sys_FileInfo* for required naming conventions for these fields.
- 4 Complete the standard procedure for creating the Content Editor.

To add the WebImageFX editor to a content editor:

- 1 In the Rhythmyx Workbench, access the Content Editor Properties dialog for the Content Editor.
- 2 Select the field that you want to associate with WebImageFX and click [**Edit**].
Rhythmyx displays the Field Properties dialog. (If you are defining a new field, Rhythmyx displays the New Field Properties dialog.)
- 3 Change the **Field Name** to *uploadfilephoto*. If you do not use this name, the WebImageFX control cannot upload the file.
- 4 Check **Treat Data as Binary**.
- 5 Select *sys_WebImageFX* as the **Control**.

Figure 54: Field Properties Dialog for a field that uses *sys_WebImageFX* control

- 6 On the Field Properties dialog, click [**OK**].
- 7 When you choose *sys_WebImageFX* as the **Control Name**, Rhythmyx automatically includes the *sys_FileInfo* exit, which fills in the uploaded file's name, mime type, extension, and size into the proper Content Editor fields if you provide them. Add any of these fields to the Content Editor. See *sys_FileInfo* for required naming conventions for these fields.

- 8 On the Content Editor Properties dialog, click **[OK]**.

The changes will take effect the next time you start your application. To see your changes, stop and restart the application, log in to Rhythmyx, and activate the editor.

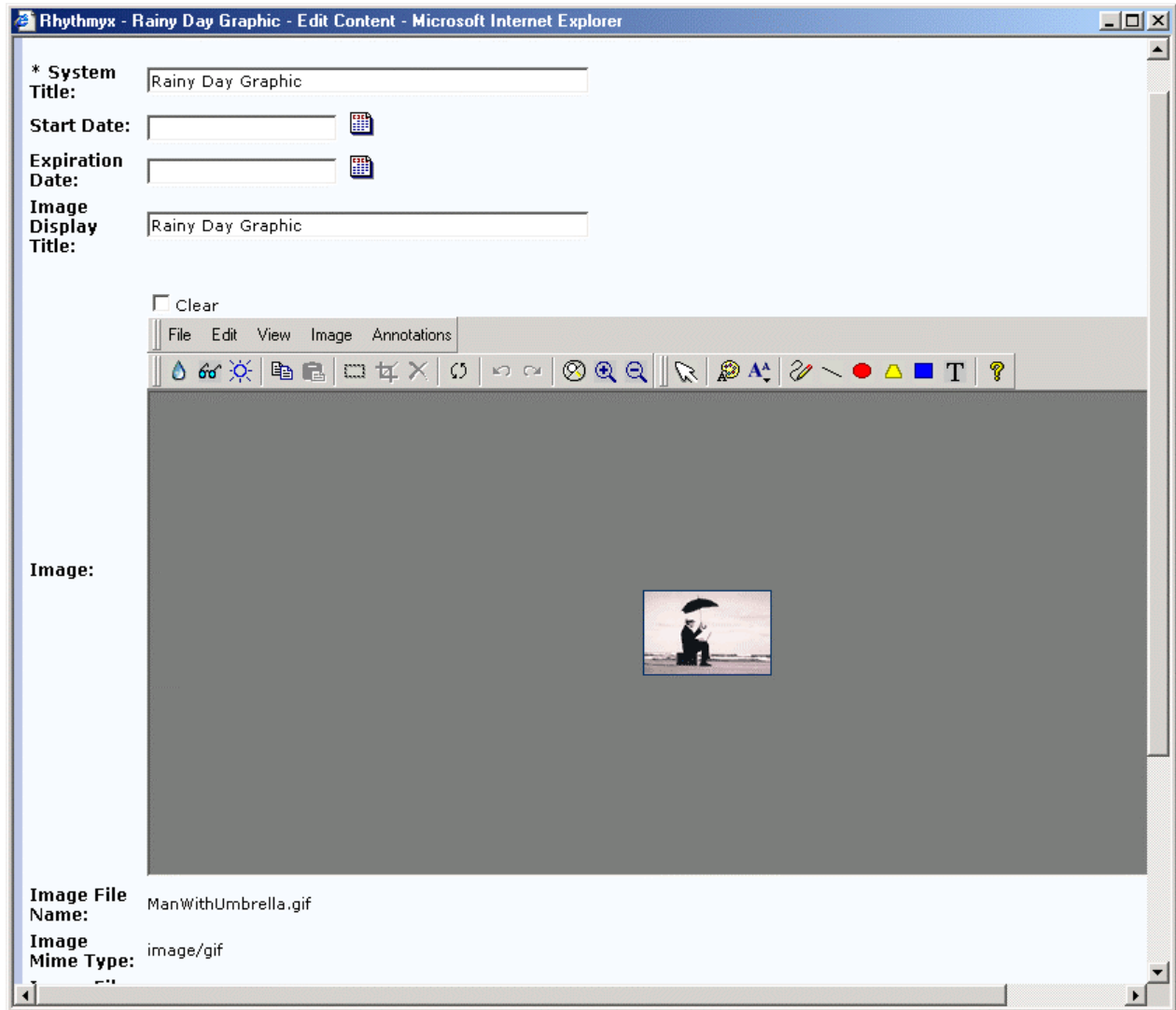


Figure 55: Content Editor with `sys_WebImageFX` control

The following limitations apply to all Content Editors that use this control:

- The name of the field containing the `sys_WebImageFX` control must be *uploadfilephoto*.
- Because the name of a field containing the `sys_WebImageFX` control must be *uploadfilephoto*, a Content Editor cannot have more than one `sys_WebImageFX` control. If it does, the additional controls will not be able to upload images.
- The names of fields in the Content Editor that `sys_FileInfo` updates (filename, type, size, and extension) must be prefixed with *uploadfilephoto*. For example, *uploadfilephoto_filename*, *uploadfilephoto_type*, *uploadfilephoto_size*, *uploadfilephoto_ext*. A Content Editor that contains a `sys_WebImageFX` control cannot also contain a `sys_File` control; if it does the `sys_File` control will not be able to upload a file.

NOTE: The first time you open a Content Editor that uses the `sys_WebImageFX` control in your Web browser, a dialog will prompt you to install WebImageFX. Follow the installation instructions in the dialog. After you initially install WebImageFX, you will not have to install it again.

Implementing the `sys_WebImageFX` Control Manually

When editing a content editor XML file by hand, specify **`sys_WebImageFX`** in the `<PSXControlRef>` element, specifying the control parameters normally. See the example in the `sys_WebImageFX` control documentation.

You can customize both the parameters of the `sys_WebImageFX` control and the configuration files of the WebImageFX editor itself.

Customizing the `sys_WebImageFX` Control

The parameters of the `sys_WebImageFX` control define the height and width of the display of the editor, the path to configuration file (`ImageEditConfig.xml`) and other characteristics. You can customize these parameters in the control definition (either the Display Control Properties for `<field>` dialog or in the `PSXParam` child nodes of the `PSXControlRef` node). If you customize the configuration file for the WebImageFX editor, update the `SRC` parameter of each instance of the `sys_WebImageFX` control to point to the correct configuration file.

For guidance on customizing (and localizing) the WebImageFX editor, consult the **WebImageFX Developer's Reference Guide**, at <http://www.ektron.com/webimagefx.aspx> (<http://www.ektron.com/webimagefx.aspx>).

Most customizations of the WebImageFX editor involve modifications to the configuration file (`ImageEditConfig.xml`). Do not modify the default configuration file, which is located in the **`Rhythmyxroot/rx_resources/WebImageFX`** directory. Instead, customize shared or local definition files. If you only use one customized configuration file, best practice is to use a shared configuration file.

In the default `ImageEditConfig.xml` used in Rhythmyx, the upload and exit options are disabled because these actions cannot function in Rhythmyx; do not enable these options when you edit copies of the `ImageEditConfig.xml` file.

Several instances of the control can use the same configuration XML file (shared configuration file), or you can use a local configuration file for each instance of the editor; you can also use a shared configuration file for some instances and a local configuration file for other instances. The files must be stored in the following manner:

- The default configuration file is stored in the directory `rx_resources/WebImageFX`. This configuration file should not be modified.
- Shared configuration files should be stored in a directory with the path `rx_resources/[path]/WebImageFX`, where `[path]` is the path to a subdirectory that logically categorizes the file. For example, you might want to use the name of your project as part of the path; for a project with the name *sample*, the path would be `rx_resources/sample/WebImageFX`.
- Local configuration files should be stored in a subdirectory of the Content Editor application. For example, if you have a local configuration file for a Press Release content editor, the configuration file would be stored in the subdirectory `Rhythmyxroot/pressrelease/WebImageFX`.

To define an instance of the `sys_WebImageFX` control to use a customized configuration file:

- 1 Open the Content Editor in the Rhythmyx Workbench and access the Content Editor Properties dialog.
- 2 Select the field that uses the WebImageFX editor and click **[Edit]** to open the Field Properties dialog.
- 3 Click the browse button (...) next to the **Control** field.
Rhythmyx displays the Display Control Properties for <field> dialog.
- 4 Click in the **Param name** column and choose `config_src_url`.
- 5 Click in the **Value** column of the same row and enter the relative URL of the configuration file you want to use for this instance of the control as a literal value.
- 6 On the Display Control Properties for <field> dialog, click **[OK]**.
- 7 On the Field Properties dialog, click **[OK]**.
- 8 On the Content Editor Properties dialog, click **[OK]**.

The changes will take effect the next time you start your application. To see your changes, stop and restart the application, log in to Rhythmyx, and activate the editor.

Best Practices: sys_WebImageFX

To simplify maintenance and promote effective technical support, observe the following Best Practices when working with the WebImageFX editor and the sys_WebImageFX control:

- Keep shared configuration files (configuration files used by more than one instance of the control) in directories with the name **Rhythmyxroot/rx_resources/[path]/webimagefx**, where **[path]** defines a category (such as the name of a project or customer). For example, if you are working on a project named **sample**, the directory should be **Rhythmyxroot/rx_resources/sample/webimagefx**.
- If only one editor is going to use a configuration file, store the file in a subdirectory of the editor application directory. If you decide to use this configuration file for other editors, move it to a shared directory and update the **SRC** parameters of the instances of the control that use that configuration file.
- When disabling a command or parameters of a command (such as lists of fonts or font sizes), hide the disabled elements first by commenting them out (**<!-- text --!>**), then test and refine your development. Remove the disabled commands and parameters when testing is complete to minimize clutter in the files and simplify future modification.

sys_File

The sys_File control is a file upload element that allows the user to supply a file as the input. This control corresponds to a single, one-dimensional field.

When hand-coding a Content Editor that includes one or more sys_File controls, include sys_FileInfo as a pre-exit. When you create a Content Editor using the Content Editor Properties dialog, Rhythmyx adds this exit for you automatically. The sys_FileInfo exit searches for attached files in a content item's HTML and returns values for file name, MIME type, character length and file encoding. The exit returns the values to field names formed by combining the filename (the <FieldRef> value) with descriptive suffixes. If you want to return information about a file, such as the filename or size, refer to the sys_FileInfo exit documentation for the correct syntax.

sys_File control Parameters

Parameter	Data Type	Parameter Type	Description	Default
id	String	Generic	XHTML 1.0 attribute	None
class	String	Generic	XHTML 1.0 attribute	None
style	String	Generic	XHTML 1.0 attribute	None
size	String	Generic	XHTML 1.0 attribute	50
maxlength	Number	Generic	XHTML 1.0 attribute	None
tabindex	Number	Generic	XHTML 1.0 attribute	None

Example Field Definition

```
<PSXField name="imagebody" showInSummary="no" showInPreview="no"
forceBinary="yes">
  <DataLocator>
    <PSXBackEndColumn id="0">
      <tableAlias>TESTIMAGE</tableAlias>
      <column>IMAGEDATA</column>
    </PSXBackEndColumn>
  </DataLocator>
  <DataType/>
  <OccurrenceSettings dimension="optional" multiValuedType="delimited"
delimiter=";" />
</PSXField>
```

Example UI Definition

```
<PSXDisplayMapping>
  <FieldRef>imagebody</FieldRef>
  <PSXUISet>
    <Label>
      <PSXDisplayText>Image Upload:</PSXDisplayText>
    </Label>
    <PSXControlRef name="sys_File"/>
  </PSXUISet>
</PSXDisplayMapping>
```

Controlling Processing of XML files

When uploading XML files, you have the option of specifying that the server process them normally (checking that the document is well-formed and that it conforms to a DTD), that it performs no validation (only checking that the document is well-formed), or that it treats the file as text. The `psxmldoc` HTML parameter controls this processing.

To use the `psxmldoc` parameter, include a hidden field to store the `psxmldoc` parameter (typically the field is named "psxmldoc"), which is stored in a backend column (also typically called "PSXMLDOC"). This field must occur before the field where the file is stored.

The `psxmldoc` parameter is typically mapped to a literal value. Acceptable values are:

Value	Processing
useValidating (default)	Server validates the document according to the DTD specified in the document.
useNonValidating	Server confirms that the document is well-formed, but does not validate it against a DTD.
treatAsText	Server does not parse the document. Document can be mapped as a single parameter to a CLOB or text column.

If the MIME type of the request is `text/xml` or `application/xml`, the body content must be an XML document. In this case, if the parameter value is `treatAsText`, the server ignores it and uses the default value. If the request MIME type is `multipart/form-data`, the parameter can store multiple values, each separated by a semicolon (";"). Only one of these values can specify parsing; the remaining values must be `treatAsText`. If multiple parser values are specified, only the last is used.

sys_HiddenInput

Rhythmyx does not display a field that uses the `sys_HiddenInput` control to the user, but it does include the content of the field with the data submitted to the database. The value in the field can be set to a literal value defined by the control itself, or a UDF or exit might populate it. Use this control to store information that the system needs, but is unnecessary for the user to see, such as a file extension. The dimension is *single*.

Parameters

Parameter	Data Type	Parameter Type	Description	Default
id	String	Generic	XHTML 1.0 attribute	None
class	String	Generic	XHTML 1.0 attribute	None
style	String	Generic	XHTML 1.0 attribute	None

Example Field Definition

```
<PSXField name="sys_pathname" showInSummary="yes" showInPreview="yes"
forceBinary="no">
  <DataType/>
  <DefaultValue>
    <DataLocator>
      <PSXTextLiteral id="0">
        <text>article/art</text>
      </PSXTextLiteral>
    </DataLocator>
  </DefaultValue>
  <OccurrenceSettings dimension="optional"
multiValuedType="delimited" delimiter=";" />
  <FieldRules>
    <PSXVisibilityRules dataHiding="xsl">
      <PSXRule boolean="and">
        <PSXConditional id="8">
          <variable>
            <PSXTextLiteral id="9">
              <text>1</text>
            </PSXTextLiteral>
          </variable>
          <operator>=</operator>
          <value>
            <PSXTextLiteral id="10">
              <text>2</text>
            </PSXTextLiteral>
          </value>
          <boolean>AND</boolean>
        </PSXConditional>
      </PSXRule>
    </PSXVisibilityRules>
  </FieldRules>
</PSXField>
```

```

        </PSXVisibilityRules>
    </FieldRules>
</PSXField>

```

Example UI Definition

```

<PSXDisplayMapping>
    <FieldRef>sys_pathname</FieldRef>
    <PSXUISet>
        <PSXControlRef name="sys_HiddenInput"/>
    </PSXUISet>
</PSXDisplayMapping>

```

Note: In this example default value is used to set up value for sys_hidden control. In addition, a visibility rule is implemented to make data invisible for preview mode.

sys_RadioButtons

The sys_RadioButtons control displays a set of radio buttons that allow the user to select one a set of values. A set of radio buttons must be multidimensional, so the values for the group should always be stored in a child table. The child table should consist of at least three columns: one for contentid, one for revisionid and one for the value to be stored. You should only define the value column in the field definition. The server will populate the contentid and revisionid fields automatically. The dimension is array.

Parameters

Parameter	Data Type	Parameter Type	Description	Default
Class	String	Generic	This parameter assigns a class name or set of class names to an element. Any number of elements may be assigned the same class name or names. Multiple class names must be separated by white space characters.	datadisplay
Style	String	Generic	This parameter specifies style information for the current element. The syntax of the value of the style attribute is determined by the default style sheet language.	None
Tabindex	Number	Generic	This parameter specifies the position of the current element in the tabbing order for the current document. This value must be a number between 0 and 32767.	None
Disabled	String	Generic	If set, this boolean attribute disables the control for user input.	None

Example Field Definition

```

<PSXField name="options" showInSummary="yes" showInPreview="yes"
forceBinary="no" type="local">
    <DataLocator>

```

```

    <PSXBackEndColumn id="0">
      <tableAlias>ARTICLE</tableAlias>
      <column>OPTIONS</column>
      <columnAlias></columnAlias>
    </PSXBackEndColumn>
  </DataLocator>
  <DataType>text</DataType>
  <OccurrenceSettings dimension="optional" multiValuedType="delimited"
  delimiter=";" />
</PSXField>

```

Example UI Definitions

Use the type attribute of the PSXChoices element in the UI definition to specify how you are storing the radio button values so that Rhythmyx can retrieve them.

To store radio button values in the lookup table (RXLOOKUP), use the value `global` for the type attribute. Rhythmyx gets the values when it builds the document by using the value in the Key element to determine the lookup key for the table. Use the Rhythmyx System Administrator to maintain the Keywords in this table. The key is a system-generated value, so manual addition of values to this table is not recommended.

Example UI Definition for global:

```

<PSXDisplayMapping>
  <FieldRef>global</FieldRef>
  <PSXUISet>
    <Label>
      <PSXDisplayText>Global choices:</PSXDisplayText>
    </Label>
    <PSXControlRef name="sys_RadioButtons"/>
    <PSXChoices type="global" sortOrder="ascending">
      <Key>3</Key>
      <PSXNullEntry sortOrder="first" includeWhen="onlyIfNull">
        <PSXEntry sequence="0" default="no">
          <PSXDisplayText> -- choose -- </PSXDisplayText>
          <Value>0</Value>
        </PSXEntry>
      </PSXNullEntry>
    </PSXChoices>
  </PSXUISet>
</PSXDisplayMapping>

```

To store radio button values in the content editor definition as a list of PSXEntry elements, use the value `local` for the type attribute.

Example UI Definition for local:

```

<PSXDisplayMapping>
  <FieldRef>local</FieldRef>
  <PSXUISet>
    <Label>
      <PSXDisplayText>Local choices:</PSXDisplayText>
    </Label>
    <PSXControlRef name="sys_RadioButtons"/>
    <PSXChoices type="local" sortOrder="user">
      <PSXEntry sequence="0" default="no">

```

```
        <PSXDisplayText>Computing Machinery</PSXDisplayText>
        <Value>acm</Value>
    </PSXEntry>
    <PSXEntry sequence="1" default="no">
        <PSXDisplayText> Society of Electrical
Engineers</PSXDisplayText>
        <Value>ieee</Value>
    </PSXEntry>
    <PSXEntry sequence="3" default="no">
        <PSXDisplayText>Society for Prevention</PSXDisplayText>
        <Value>spca</Value>
    </PSXEntry>
    <PSXNullEntry sortOrder="last" includeWhen="always">
        <PSXEntry sequence="0" default="no">
            <PSXDisplayText> -- choose -- </PSXDisplayText>
            <Value>0</Value>
        </PSXEntry>
    </PSXNullEntry>
    <DefaultSelected>
        <PSXDefaultSelected type="nullEntry"/>
    </DefaultSelected>
</PSXChoices>
</PSXUISet>
</PSXDisplayMapping>
```

To store or generate radio button values in another control or stylesheet, use the value lookup for the type attribute. Define a URL with the control or stylesheet's address in a `PSXUrlRequest` child element.

Example UI Definition for lookup:

```
<PSXDisplayMapping>
  <FieldRef>lookup</FieldRef>
  <PSXUISet>
    <Label>
      <PSXDisplayText>Lookup choices:</PSXDisplayText>
    </Label>
    <PSXControlRef name="sys_RadioButton"/>
    <PSXChoices type="lookup " sortOrder="user">
      <PSXUrlRequest>
        <Href>http://38.164.160.56:9992/Rhythmyx/sys_wfLookups/extroles.html</Href>
        <PSXParam name="workflowid">
          <DataLocator>
            <PSXTextLiteral id="0">
              <text>1</text>
            </PSXTextLiteral>
          </DataLocator>
        </PSXParam>
      </PSXUrlRequest>
    </PSXChoices>
  </PSXUISet>
</PSXDisplayMapping>
```


To store radio button values in a Rhythmyx table other than the RXLOOKUP table, use the value `internalLookup` for the type attribute. Define a URL in a `PSXURLRequest` child element so that Rhythmyx can get the entries through an internal request to the specified URL. The lookup query must conform to the `sys_Lookup.dtd`. You can add it to the content editor application or place it in a separate application. See *Creating an Internal Lookup Query* (on page 195) for a procedure for adding the lookup query.

Example UI Definition for internal lookup:

```
<PSXDisplayMapping>
  <FieldRef>lookup</FieldRef>
  <PSXUISet>
    <Label>
      <PSXDisplayText>Lookup choices:</PSXDisplayText>
    </Label>
    <PSXControlRef name="sys_RadioButtons"/>
    <PSXChoices type=" internalLookup " sortOrder="user">
      <PSXURLRequest>
        <Href>http://38.164.160.56:9992/Rhythmyx/
          sys_wfLookups/extroles.html</Href>
        <PSXParam name="workflowid">
          <DataLocator>
            <PSXTextLiteral id="0">
              <text>1</text>
            </PSXTextLiteral>
          </DataLocator>
        </PSXParam>
      </PSXURLRequest>
    </PSXChoices>
  </PSXUISet>
</PSXDisplayMapping>
```

sys_Table

Image Name:	Image Title:	Image Ext:
Spring	Season	gif

Edit table

Figure 56: Example sys_Table

The sys_Table control creates a table to display multiple fields from a related database table. It is multidimensional and may contain multiple fields. The graphic shows a table with three text fields and one file upload control. Because this is a complex child, the user edits data on a different page, which they access by clicking a button labeled 'Edit table' on the page. The content editor displays a summary view of all rows in the table. The showInSummary attribute of each child element within the table controls the visibility of these values. Note that the PSXFieldSet has a name, and each PSXField has its own name. The dimension is *table*.

Parameters

Parameter	Data Type	Parameter Type	Description	Default
id	String	Generic	XHTML 1.0 attribute	None
class	String	Generic	XHTML 1.0 attribute	None
style	String	Generic	XHTML 1.0 attribute	None
summary	String	Generic	XHTML 1.0 attribute	None
width	String	Generic	XHTML 1.0 attribute width	100%
cellspacing	String	Generic	XHTML 1.0 attribute cellpadding	0
cellpadding	String	Generic	XHTML 1.0 attribute cellpadding	5
border	Number	Generic	XHTML 1.0 attribute tabindex	1

Example Fieldset Definition

```
<PSXFieldSet type="complexChild" name="image"
repeatability="zeroOrMore">
  <PSXField name="imgname" showInSummary="yes" showInPreview="yes"
forceBinary="no">
    <DataLocator>
      <PSXBackEndColumn id="1">
        <tableAlias>TESTIMAGE</tableAlias>
        <column>FILENAME</column>
        <columnAlias/>
      </PSXBackEndColumn>
    </DataLocator>
  </DataField/>
  <OccurrenceSettings dimension="optional"
multiValuedType="delimited" delimiter=";" />
</PSXFieldSet>
```

```

</PSXField>
<PSXField name="imgtitle" showInSummary="yes" showInPreview="yes"
forceBinary="no">
  <DataLocator>
    <PSXBackEndColumn id="2">
      <tableAlias>TESTIMAGE</tableAlias>
      <column>DISPLAYTITLE</column>
      <columnAlias/>
    </PSXBackEndColumn>
  </DataLocator>
  <DataType/>
  <OccurrenceSettings dimension="optional" multiValuedType="delimited"
delimiter=";" />
</PSXField>
<PSXField name="imgext" showInSummary="yes" showInPreview="yes"
forceBinary="no">
  <DataLocator>
    <PSXBackEndColumn id="3">
      <tableAlias>TESTIMAGE</tableAlias>
      <column>EXTENSION</column>
      <columnAlias/>
    </PSXBackEndColumn>
  </DataLocator>
  <DataType/>
  <OccurrenceSettings dimension="optional" multiValuedType="delimited"
delimiter=";" />
</PSXField>
<PSXField name="imagebody" showInSummary="no" showInPreview="no"
forceBinary="yes">
  <DataLocator>
    <PSXBackEndColumn id="0">
      <tableAlias>TESTIMAGE</tableAlias>
      <column>IMAGEDATA</column>
    </PSXBackEndColumn>
  </DataLocator>
  <DataType/>
  <OccurrenceSettings dimension="optional" multiValuedType="delimited"
delimiter=";" />
</PSXField>
</PSXFieldSet>

```

Example UI Definition

```

<PSXDisplayMapping>
  <FieldRef>image</FieldRef>
  <PSXUISet>
    <Label>
      <PSXDisplayText>Image:</PSXDisplayText>
    </Label>
    <PSXControlRef name="sys_Table"/>
  </PSXUISet>
  <PSXDisplayMapper id="2" fieldSetRef="image">
    <PSXDisplayMapping>
      <FieldRef>imgname</FieldRef>
      <PSXUISet name="" defaultSet="">

```

```

        <Label>
          <PSXDisplayText>Image Name:</PSXDisplayText>
        </Label>
        <PSXControlRef name="sys_EditBox"/>
      </PSXUISet>
    </PSXDisplayMapping>
  <PSXDisplayMapping>
    <FieldRef>imgtitle</FieldRef>
    <PSXUISet name="" defaultSet="">
      <Label>
        <PSXDisplayText>Image Title:</PSXDisplayText>
      </Label>
      <PSXControlRef name="sys_EditBox"/>
    </PSXUISet>
  </PSXDisplayMapping>
</PSXDisplayMapping>
<FieldRef>imgext</FieldRef>
  <PSXUISet>
    <Label>
      <PSXDisplayText>Image Ext:</PSXDisplayText>
    </Label>
    <PSXControlRef name="sys_EditBox"/>
  </PSXUISet>
</PSXDisplayMapping>
<FieldRef>imagebody</FieldRef>
  <PSXUISet>
    <Label>
      <PSXDisplayText>Image Upload:</PSXDisplayText>
    </Label>
    <PSXControlRef name="sys_File"/>
  </PSXUISet>
</PSXDisplayMapping>
</PSXDisplayMapper>
</PSXDisplayMapping>

```

sys_TextArea

The `sys_TextArea` control is used to give the user that ability to enter multiple lines of plain text. The dimension of this control is *single*.



Figure 57: Example `sys_TextArea`

Parameters

Parameter	Data Type	Parameter Type	Description	Default
id	String	Generic	XHTML 1.0 Attribute	None

class	String	Generic	XHTML 1.0 Attribute	None
style	String	Generic	XHTML 1.0 Attribute	None
rows	Number	Generic	XHTML 1.0 Attribute	4
cols	Number	Generic	XHTML 1.0 Attribute	80
tabindex	Number	Generic	XHTML 1.0 Attribute	None

Example Field Definition

```
<PSXField name=" abstract " showInSummary="yes" showInPreview="yes"
forceBinary="no">
  <DataLocator>
    <PSXBackEndColumn id="0">
      <tableAlias>RXARTICLE</tableAlias>
      <column> ABSTRACT </column>
      <columnAlias/>
    </PSXBackEndColumn>
  </DataLocator>
  <DataType/>
  <OccurrenceSettings dimension="optional" multiValuedType="delimited"
delimiter=";" />
</PSXField>
```

Example UI Definition

```
<PSXDisplayMapping>
  <FieldRef>abstract </FieldRef>
  <PSXUISet>
    <Label>
      <PSXDisplayText> Abstract:</PSXDisplayText>
    </Label>
    <PSXControlRef name="sys_TextArea" />
  </PSXUISet>
</PSXDisplayMapping>
```

sys_HTMLEditor

NOTE: The sys_HTMLEditor control is now deprecated. Use the sys_eWebEditPro control to provide dynamic HTML editing in content editors.

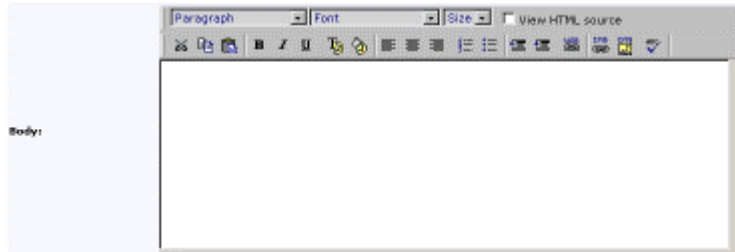


Figure 58: Sample sys_HTMLEditor

The sys_HtmlEditor is a multiple-line text entry control in which the user can type and edit text. It displays a DHTML editor that allows a user to enter text and apply standard formatting, such as changing the font or the alignment. The editor has a built in support for inserting inline links and images and an optional spell check feature. The editor only works in IE 5x and 6 browsers. In Netscape browsers, a standard textarea box will be displayed. The dimension is *single*.

Parameters

Parameter	Data Type	Parameter Type	Description	Default
id	String	Generic	XHTML 1.0 attribute cols	dynamsg
class	String	Generic	XHTML 1.0 attribute	None
width	String	Generic	XHTML 1.0 attribute cols	100%
style	String	Generic	XHTML 1.0 attribute cols	position:relative
height	Number	Generic	XHTML 1.0 attribute cols	250
type	String	Generic	XHTML 1.0 attribute cols	text/x-scriptlet
data	String	Generic	XHTML 1.0 attribute cols	../sys_resources/texteditor/deditor.html
viewastext	String	Generic	XHTML 1.0 attribute cols	viewastext
Formname	String	JavaScript	name of the form that contains this control	EditForm

Example Field Definition

```
<PSXField name="bodycontent" showInSummary="yes" showInPreview="no"
forceBinary="no">
  <DataLocator>
    <PSXBackEndColumn id="0">
      <tableAlias>RXARTICLE</tableAlias>
      <column>BODYCONTENT</column>
      <columnAlias/>
    </PSXBackEndColumn>
  </DataLocator>
  <DataType/>
  <OccurrenceSettings dimension="optional" multiValuedType="delimited"
delimiter=";" />
</PSXField>
```

Example UI Definition

```
<PSXDisplayMapping>
  <FieldRef>bodycontent</FieldRef>
  <PSXUISet>
    <Label>
      <PSXDisplayText>Body:</PSXDisplayText>
    </Label>
    <PSXControlRef name="sys_HtmlEditor">
      <PSXParam name="NAME">
        <DataLocator>
          <PSXTextLiteral id="0">
            <text>bodycontent</text>
          </PSXTextLiteral>
        </DataLocator>
      </PSXParam>
    </PSXControlRef>
  </PSXUISet>
</PSXDisplayMapping>
```

Creating an Internal Lookup Query

When you use *sys_DropDownSingle* (on page 159), *sys_CheckBoxGroup* (on page 152), and *sys_RadioButtons* (on page 186) or any custom controls that require a list of entries, you may choose to set the <PSXChoices> type to `internalLookup` and retrieve the entries from an existing Rhythmyx table using a query. You can add the query to the content editor application or to a separate application.

To create an internal lookup query:

- 1 Open the content editor application or a new application.
- 2 Drag the `Rhythmyx/DTD/sys_Lookup.dtd` file onto the application.
- 3 Select *Query*.
- 4 Right-click on the `sys_Lookup` XML and select *Properties* to open the Resource Editor.
- 5 Add the table(s) containing the content that you want as list values.

- 6 Open the mapper and map table values to the sys_Lookup Value and PSXDisplayText elements.

Back-end	XML
CONTENTSTATUS.CONTENTID	PSXxmlField/sys_Lookup/PSXEntry/Value
RXARTICLE.DISPLAYTITLE	PSXxmlField/sys_Lookup/PSXEntry/PSXDisplayText

- 7 Optionally, add a Result Pager to sort the list results.
- 8 Right-click the sys_Lookup XML and choose *Request Properties*.
- 9 Change the name of the sys_Lookup.XML and click [OK].
- 10 Save and close the application.
- 11 Open the content editor application in an XML editor.
- 12 In the UI definition of the content editor, set the PSXChoices type to "internalLookup" and add a <PSXUrlRequest> element that references the sys_Lookup XML. In the following example UI definition, the developer named the lookup query articlechoice.xml and stored it in the articlelookup folder within the article content editor folder.

```

<PSXDisplayMapping>
  <FieldRef>category</FieldRef>
  <PSXUISet>
    <Label>
      <PSXDisplayText>*Category:</PSXDisplayText>
    </Label>
    <PSXControlRef name="sys_CheckBoxGroup"/>
    <PSXChoices type="internalLookup" sortOrder="ascending">
      <PSXUrlRequest>
        <Href>../articlelookup/articlechoice.xml</Href>
      </PSXUrlRequest>
    </PSXChoices>
  </PSXUISet>
  <PSXDisplayMapper id="9" fieldSetRef="category">
    <PSXDisplayMapping>
      <FieldRef>categoryvalue</FieldRef>
    </PSXDisplayMapping>
  </PSXDisplayMapper>
</PSXDisplayMapping>

```

NOTE: If you retrieve entries from the RXLOOKUP table, the <PSXChoices> type is global. See the individual controls for information about using global and the RXLOOKUP table. See the online help for a description of the RXLOOKUP table.

APPENDIX III

Content Editor XML Reference

Basic Objects

A collection of named data items that compose an editor or portion of an editor. Corresponds roughly to a row in a table.

Used in : Local, Shared

Contained by: *PSXContentEditorMapper* (on page [253](#)), *PSXSharedFieldGroup* (on page [255](#))

Attributes:

Name	Required?	Value	Effect	Default
name	Yes	CDATA	Defines the name of the fieldset. Must be unique among all fields in this object, including shared and system fields.	None
type	No	parent simpleChild complexChild multiPropertySimpleChild	Root fieldset Pre-defined options; edited within parent editor 1 or more columns, arbitrary number of rows; shown in summary view with parent; edited in separate editor 1 or more columns, but only one row; edited in parent editor	parent
Repeatability (not applicable if type = parent)	No	zeroOrMore oneOrMore	Field set may appear zero or more times Field must appear at least once	zeroOrMore

supportsSequencing (Only applicable if type = complexChild)	No	true false	Order can be specified; child table must include SORTRANK column Order cannot be specified	true
--	----	---------------	---	------

Elements:

Name	Appearance	Description
PSXField	Zero or more (At least one is required to start an application.)	Defines specific fields in the field set
<i>PSXFieldSet</i> (on page 197)	Zero or more	Defines (child) field sets

PSXContainerLocator

Collective container for multiple table sets. (Currently we only support one database and schema, but in the future we may expand to allow multiple databases and schemas.)

Used in : Local, Shared

Contained by: *PSXContentEditorPipe* (on page 252), *PSXSharedFieldGroup* (on page 255)

Attributes : None

Elements:

Name	Appearance	Description
<i>PSXTableSet</i> (on page 199)	One or more	Defines the location of the tables where data for the content items are stored.

PSXTableSet

Collective container for tables that define an editor or portion of an editor. (Currently we only support one database and schema, but in the future we may expand to allow multiple databases and schemas.)

Used in : Local, Shared

Contained by: *PSXContainerLocator* (on page 198)

Attributes:

Name	Required?	Value	Effect
href (not currently supported)	No	URI	Specifies the location of a document containing table definitions.

Elements:

Name	Appearance	Description
<i>PSXTableLocator</i> (on page 199)	Once	Defines the specific location of the tables where the data for the content items are stored.
<i>PSXTableRef</i> (on page 201)	One or more	Used with PSX Table Locator

PSXTableLocator

Specifies the location of the database where data for the content item is stored. If an Alias is used anywhere in the file, at least one PSXTableLocator must be defined to specify the location of the database.

Used in : Local, Shared

Contained by: *PSXTableSet* (on page 199)

Attributes:

Name	Required?	Value	Effect
alias	No	CDATA	Allows other definitions to share this locator. If this attribute is not used, the table locator cannot be shared with other definitions. Must be unique among the PSXTableLocators in the system, shared, and local definitions.

Elements:

Specifies the owner or the schema.	Appearance	Description
<i>PSXBackendCredential</i> (on page 233)	Once	Defines backend database credential.
<i>Database</i> (on page 200)	Once	Defines the name of the backend database where the data for this editor or portion of this editor is stored.
<i>Origin</i> (on page 201)	Once	Specifies the owner or the schema.

Database

Defines the name of the database where the data for the editor or portion of the editor is stored.

Used in: Local, Shared

Contained by: *PSXTableLocator* (on page [199](#))

Attributes: None

Elements: None

TableLocatorAlias

Used to refer to an existing database table alias. Not currently supported.

Used in: Local, Shared

Contained by: *PSXTableLocator* (on page [199](#))

Attributes: None

Elements: None

Alias (uppercase)

The name of an existing object. Where it must exist (for example, within this object, or somewhere on the server) depends on the context in which it is used.

Used in: Local, Shared

Contained by: *PSXTableLocator* (on page [199](#)), *PSXBackendCredential* (on page [233](#))

Attributes: None

Elements: None

Origin

Owner of the database or schema.

Used in: Local, Shared

Contained by: *PSXTableLocator* (on page 199)

Attributes: None

Elements: None

PSXTableRef

Used with the *PSXTableLocator* tag to fully specify the location of a table.

Used in: Local, Shared

Contained by: *PSXTableLocator* (on page 199)

Attributes:

Name	Required?	Value	Effect
name	Yes	CDATA	Defines the base name of the table.
alias	No	CDATA	Used to identify the table if the same table exists in multiple Table Sets. Must be unique among the aliases of all tables in all Table Set definitions in all system, shared, and local definition files. If not supplied, Rhythmyx uses the name attribute. Must be unique among all the <i>PSXTableRef</i> in the system, shared, and local definitions.

Elements: None

PSXField

A PSXField element contains all of the non-UI data required to define a field, including business rules. All DataLocator types except FieldRef are supported. If the DataLocator of a field specifies anything other than PSXBackEndColumn, this field is for query only; it cannot be updated.

Used in: Local, Shared

Contained by: *PSXFieldSet* (on page [197](#))

Attributes:

Name	Required?	Value	Effect	Default
name	Yes	CDATA	Defines the name of the specific field. Must be unique among all fields in this object, including shared and system fields.	None
showInSummary (Applies only to fields not in the "parent" fieldset. Does not apply if forceBinary = yes.)	No	no yes	Field will not be displayed in summary views in parent editor. Typically only used for large text fields that would not make sense in summary view. Field will be displayed in summary view in parent editor.	no
showInPreview (Does not apply if forceBinary = yes. Preview occurs only when the "Preview" command is supplied to the content editor)	No	no yes	Field will not be displayed in previews Field will be displayed in previews.	yes
forceBinary Binary fields have an asymmetrical life. They are saved (and uploaded if a file) using the standard editor, but to retrieve them, you must use the binary command of the command handler. The data is not returned when editing the data in a row editor.	No	no yes	Fields is not treated as binary Field is treated as binary.	no

Name	Required?	Value	Effect	Default
modificationType	Yes	system systemCreate userCreate user	<p>Only the server can modify this field. If a user submits a value, the system ignores it.</p> <p>The server sets the field when the Content Item is created. It does not allow subsequent modification of the field.</p> <p>The server sets the field when the Content Item is created. It does not allow subsequent modification of the field.</p> <p>The user can update the field at any time</p>	user
userSearchable	No	Yes No	<p>Field is can be included in user searches. Whether it is available to Business Users or only to implementers is controlled by the userCustomizable attribute.</p> <p>Field cannot be included in user searches.</p> <p>NOTE: This attribute is deprecated. It is included for backward compatibility with servers prior with servers prior to version 5.5. This attribute is superceded by the PSXSearchProperties element.</p>	yes
userCustomizable	No	Yes No	<p>Field is available to Business Users for use in Searches.</p> <p>Field is only available in implementer-defined searches</p> <p>NOTE: This attribute is deprecated. It is included for backward compatibility with servers prior with servers prior to version 5.5. This attribute is superceded by the PSXSearchProperties element..</p>	yes

Name	Required?	Value	Effect	Default
defaultSearchLabel	No	CDATA	<p>If a field does not have a UIDef, or if the UIDef has been overridden, Rhythmyx uses the value of this attribute as the field label in any search dialog it presents to the user. If both the UIDef and this attribute are missing, Rhythmyx uses the field name as the field label in search dialogs.</p> <p>NOTE: This attribute is deprecated. It is included for backward compatibility with servers prior to version 5.5. This attribute is superseded by the PSXSearchProperties element.</p>	None
clearBinaryParam Name of the HTML parameter to check to determine whether or not to clear this field.	No	Yes No	<p>If this attribute is included and has a value of "yes", then the backend database column corresponding to this field will be cleared when the content item is updated. The attribute only applies if the column specified contains binary data, or if the value of forceBinary equals "yes".</p> <p>The parameter specified must be defined in a PSXParam child element of the PSXControlRef for the field in the format <code><fieldname>_clear</code> and given a value of "yes."</p>	None
type	No	local shared system	Indicates which Content Editor definition file the field belongs to.	None
mimetype	No	CDATA	Defines the MIME type of the field	None
fieldvaluetype	No	content meta	The field stores content data. The field stores metadata.	None

Elements:

Name	Appearance	Description
<i>DataLocator</i> (on page 207)	Once	Specifies the location of the data for this field.
<i>DataType</i> (on page 206)	Zero or one	Defines the type of data (character data versus binary). For future use.
<i>DefaultValue</i> (on page 209)	Zero or one	Defines the default value of the field. If a value is specified for this element, that value will be included in the output document when the content editor is displayed for a new item.
<i>OccurrenceSettings</i> (on page 205)	Zero or one	Specifies how many times the field should occur (in other words the business rules).
<i>FieldRules</i> (on page 222)	Zero or one	Defines set of rules regarding validation, translation, and visibility for field.
<i>PSXSearchProperties</i> (on page 248)	Zero or one	Contains the attributes that define the look and feel of the search engine and its field-level configuration.

OccurrenceSettings

Defines how many times the field should occur.

Used in: Local, Shared

Contained by: PSXField

Attributes:

Name	Required ?	Value	Effect	Default
dimension	No	optional required oneOrMore zeroOrMore count	Field may not appear Field appears once. Field appears at least once. Requires child table. Field may not appear, or may appear one or more times. Typically requires child table but this is not mandatory if all of the data is stored in one column. Field appears the specific number of times specified. Requires a numeric value for the content.	Optional

Name	Required ?	Value	Effect	Default
multiValuedType	No	delimited separate	How to display and send to server fields with multiple values. delimited: Concatenate all values into a single field and separate by the character indicated in delimiter. separate: Add multiple values to display field when result document is generated.	delimited
delimiter	No	CDATA	Delimiter to use between multiple values when stored in database. Must be a single character. If this character appears in a value, use delimiter twice to escape.	;
transitionId	No	CDATA	If specified, only use these occurrence settings when the transition being performed matches this transitionId. No two OccurrenceSettings can have the same transitionId. Only one OccurrenceSetting can have no transitionId. The default OccurrenceSetting has no transitionId.	None

Elements: None

Datatype

Defines the type of data in the field.

Used in: Local, Shared

Contained by: PSXField

Attributes: None

Elements: None

DataLocator

This tag specifies how to find data. Not all children are supported in all contexts. The context must specify which children are allowed.

Used in: Local, Shared

Contained by: PSXField, *DefaultValue* (on page 209), *PSXParam* (on page 221)

Attributes: None

Elements:

Name	Appearance	Description
<i>FieldRef</i> (on page 208)	Choice	Uses the definition in another field to find the data.
<i>PSXBackendColumn</i> (on page 236)	Choice	Specifies the column in the backend database table where the data is located.
<i>PSXCookie</i> (on page 243)	Choice	Uses the cookie specified to find the data.
<i>PSXNumericLiteral</i> (on page 241)	Choice	Uses the defined numeric literal value.
<i>PSXDateLiteral</i> (on page 241)	Choice	Uses the defined date value.
<i>PSXLiteralSet</i> (on page 240)	Choice	Specifies a set of literal options.
<i>PSXHtmlParameter</i> (on page 243)	Choice	Uses the value of the HTML parameter specified.
<i>PSXHtmlSingleParameter</i>	Choice	Like <i>PSXHtmlParameter</i> except that if the parameter is multi-valued, it returns only the first value.
<i>PSXXmlField</i> (on page 244)	Choice	Uses the value of the XML field specified.
<i>PSXCgiVariable</i> (on page 245)	Choice	Uses the value of the CGI variable specified.

Name	Appearance	Description
<i>PSXUserContext</i> (on page 245)	Choice	Uses the value of the user-context variable specified.
<i>PSXExtensionCall</i> (on page 246)	Choice	Defines a UDF that will calculate the data.
<i>PSXUserContext</i> (on page 245)	Choice	Uses the value of the user context variable specified.
<i>PSXExtensionCallSet</i> (on page 246)	Choice	Defines a set of UDFs that will calculate the data. Not supported until nested UDFs are implemented.
<i>PSXTextLiteral</i> (on page 240)	Choice	Uses the literal text specified.

FieldRef

Names an existing field in the definition. In the local XML definition, this element could refer to a field defined in the local, shared, or system definition XML files.

Used in: Local, Shared

Contained by: *DataLocator* (on page 207), *PSXDisplayMapping* (on page 209)

Attributes: None

Elements: None

PSXDisplayMapper

Contains all of the mappings of fields to their display control.

Used in: Local, Shared

Contained by: *PSXDisplayMapping* (on page 209), *PSXUIDefinition* (on page 210)

Attributes:

Name	Required?	Value	Effect
id	Yes	CDATA	A unique numeric identifier for the mapper. This value is used as the sys_childid during updates.
fieldSetRef	Yes	CDATA	Names the field set for which the mappings are defined.

Elements:

Name	Appearance	Description
<i>PSXDisplayMapping</i> (on page 209)	Zero or more (Must be at least one before starting the application.)	Defines the individual display mappings.

PSXDisplayMapping

Contains the display mapping for a specific field. A mapping defines a row in the output document and links a PSXField to it. Nearly all operations are based on the mappings, not the field sets. Extra fields in a field set are ignored.

Used in: Local, Shared

Contained by: *PSXDisplayMapper* (on page [208](#))

Attributes: None

Elements:

Name	Appearance	Description
<i>FieldRef</i> (on page 208)	Once	Specifies the field for which the display mapping is being defined. Must refer to an existing field.
<i>PSXUISet</i> (on page 210)	Once	Defines the user interface specification for this field.
<i>PSXDisplayMapper</i> (on page 208)	Zero or one	Defines display mappings for child fields. Only present if the FieldRef element refers to a field set.

DefaultValue

Specifies the default value of a field.

The FieldRef, PSXBackendColumn, and PSXXmlField child elements of the DataLocator element are not allowed, but any other child element can be used.

Used in: Local, Shared

Contained by: PSXField

Attributes: None

Elements:

Name	Appearance	Description
<i>DataLocator</i> (on page 207)	Zero or more	Specifies a location where the default value can be found or generated.

PSXUIDefinition

Contains the user interface definition for the editor.

Used in: Local, Shared

Contained by: *PSXContentEditorMapper* (on page 253), *PSXSharedFieldGroup* (on page 255)

Attributes: None

Elements:

Name	Appearance	Description
<i>PSXDefaultUI</i> (on page 210)	Zero or one	Defines the default user interface for the editor or portion of the editor.
<i>PSXDisplayMapper</i> (on page 208)	Once	Contains all mappings of fields to their display controls.

PSXDefaultUI

Contains the user interface definition for a portion of the editor. If a PSXUISet refers to an existing default UI, then every property in the default set that is not set in the UI set is merged into the UI set. These can be used to make creating a consistent user interface easier.

Used in: Local, Shared

Contained by: *PSXUIDefinition* (on page 210)

Attributes: None

Elements:

Name	Appearance	Description
<i>PSXUISet</i> (on page 210)	One or more	Defines default properties for a particular kind of presentation.

PSXUISet

Used in: Local, Shared

Contained by: *PSXDefaultUI* (on page 210)

Attributes:

Name	Required?	Value	Effect
name (Either name or defaultSet can be present, but not both.)	No	CDATA	Defines a name for this user interface set. Other objects can use this name to refer to the user interface set. Only used if this object is being defined as a default UI set.

defaultSet (Either name or defaultSet can be present, but not both.)	No	CDATA	Uses the properties of the specified user interface set as the properties of the user interface set for this field, unless overridden for this field.
---	----	-------	---

Elements:

Name	Appearance	Description
<i>Label</i> (on page 212)	Zero or one	Defines the label for the field in the user interface. This text is visible the user of the editor.
<i>PSXControlRef</i> (on page 217)	Zero or one	Refers to a control used when displaying the data in this field. Controls are defined in an XSL field, so no validation is performed on this name until a request is processed.
<i>ErrorLabel</i> (on page 212)	Zero or one	Defines the label displayed when the field fails a validation test. Replaces the standard label in the output document in this case.
<i>PSXChoices</i> (on page 213)	Zero or one	Defines a choice list for the field. Only used for SDMP field sets.
<i>ReadOnlyRules</i> (on page 218)	Zero or one	Defines the rules that determine whether a field should be displayed as read-only.
<i>PSXCustomActionGroup</i> (on page 231)	Zero or one	Defines overrides to the default editor behavior, removing existing buttons and adding new buttons.

PSXDisplayText

Specifies text to be displayed to the end user.

Used in: Local, Shared

Contained by: *Label* (on page 212), *ErrorLabel* (on page 212), *PSXEntry* (on page 216), *ErrorInfo* (on page 218), *PSXActionLink* (on page 230)

Attributes: None

Elements: None

Label

Used In: Local, Shared

Contained By: *PSXUISet* (on page [210](#))

Attributes: None

Elements:

Name	Appearance	Description
<i>PSXDisplayText</i> (on page 211)	Once	Specifies the text to be displayed for the label of the field.

ErrorLabel

Used in: Local, Shared

Contained by: *PSXUISet* (on page [210](#)), *PSXFieldTranslation* (on page [223](#)), *PSXFieldValidationRules* (on page [224](#)), *PSXRule* (on page [225](#))

Attributes: None

Elements: None

PSXChoices

Used to define a choice list of options for the field. Options can come from a table (`type = global`; requires the child element `Key`) or from another URL (`type = lookup` or `type = internalLookup`; requires the child element `PSXUrlRequest`), or they can be defined locally (`type = local`; requires one or more `PSXEntry` children).

Used in: Local, Shared

Contained by: *PSXUISet* (on page 210)

Attributes:

Name	Required?	Value	Effect	Default
type	No	global local lookup internalLookup	<p>Entries are stored in a lookup table; requires the child element <code>Key</code>.</p> <p>Entries are defined locally. Requires one or more <code>PSXEntry</code> child elements.</p> <p>Entries are stored in a remote URL. Requires the child element <code>PSXUrlRequest</code>. The associated control is responsible for making the request and retrieving all entries from the result document.</p> <p>Entries are stored in a remote URL that is internal to Rhythmyx. Requires the child element <code>PSXUrlRequest</code>. The document returned from this request must conform to the <code>sys_Lookup.dtd</code>. The request is made internally and the entries are provided to the control as if it had been a local lookup request.</p>	global
sortOrder	No	ascending descending user	<p>Entries are sorted in ascending dictionary order (case insensitive).</p> <p>Entries are sorted in descending dictionary order (case insensitive).</p> <p>If <code>type = local</code>, uses the sequence attribute of the <code>PSXEntry</code> children to determine the order. If <code>type = global</code>, uses the sequence defined in the lookup table to determine the order. Otherwise is ignored.</p>	ascending

Elements:

Name	Appearance	Description
<i>Key</i> (on page 243)	Choice	Specifies the lookup key in the RXLOOKUP table under which the options are stored.
<i>PSXEntry</i> (on page 216)	Choice; One or more	Defines options locally, for this editor only.
<i>PSXUrlRequest</i> (on page 219)	Choice	Specifies a URL where the options can be found. The URL is passed to the control.
<i>PSXNullEntry</i> (on page 215)	Zero or one	Specifies the entry in the field if the value of the field is null.
<i>DefaultSelected</i> (on page 214)	Zero or one	Defines the option(s) selected when creating a new document.

DefaultSelected

Defines the option(s) in a choice list selected by default.

Used in: Local, Shared

Contained by: *PSXChoices* (on page [213](#))

Attributes: None

Elements:

Name	Appearance	Description
<i>PSXDefaultSelected</i>	Zero or more. The number allowed is determined by the type of attribute.	Defines the default option selected.

PSXDefaultSelected

Defines the specific method of determining the default selection. The selected entry is indicated in the XML document generated by setting the `selected` attribute of the `DisplayEntry` element to `yes` in the output XML document generated by the `sys_ContentEditor.dtd`.

Used in: Local, Shared

Contained by: *DefaultSelected* (on page [214](#))

Attributes:

Name	Required?	Value	Effect	Default
type	No	nullEntry	Uses the option specified in the PSXNullEntry as the default entry.	nullEntry
		sequence	Uses the option at the specified sequence number (for example, <code>sequence = 2</code> would use the second entry) as the default option.	
		text	Uses the first entry found containing a string that matches the text string specified.	

Elements: None

PSXNullentry

Defines the entry if the current value of the field is null or the field is empty.

Used in: Local, Shared

Contained by: *PSXChoices* (on page [213](#))

Attributes:

Name	Required?	Value	Effect	Default
includeWhen	No	always onlyIfNull	Always include the null entry in the list of available choices Only include the null entry when the current value of the field is null.	onlyIfNull

sortOrder	No	first last sorted	The null entry will always be the first entry in the list. The null entry will always be the last option in the list. The null entry will be sorted in the list like all other entries.	first
-----------	----	-------------------------	---	-------

Elements:

Name	Appearance	Description
<i>PSXEntry</i> (on page 216)	Once	Specifies the entry to use as the null entry.

PSXEntry

Used in: Local, Shared

Contained by: *PSXChoices* (on page [213](#)), *PSXNullEntry* (on page [215](#))**Attributes:**

Name	Required ?	Value	Effect	Default
sequence	No	CDATA	Determines the place of this entry relative to other entries in the choice list.	None
default	No	no yes	Option is not the default option for the field. Option is the default option for the field.	no

Elements:

Name	Appearance	Description
<i>PSXDisplayText</i> (on page 211)	Once	Specifies the text to display in the user interface.
<i>Value</i> (see " Value (uppercase) " on page 217)	Once	Specifies the value of the option for processing. When the user selects this option, the content of this element must be returned as the value of the submitted HTML parameter.

Value (uppercase)

A generic tag used to avoid a mixed content model. Use this tag to define the value of an element that contains other child elements.

Used in: Local, Shared

Contained by: *PSXEntry* (on page 216)

Attributes: None

Elements: None

PSXControlRef

Defines a link to a control used when displaying field data.

Used in: Local, Shared

Contained by: *PSXUISet* (on page 210), *ReadOnlyRules* (on page 218)

Attributes:

Name	Required?	Value	Effect
name	Yes	CDATA	Specifies the name of the control. Typically it is the name of a template (XSL stylesheet).

Elements:

Name	Appearance	Description
<i>PSXParam</i> (on page 221)	Zero or more	Defines the parameters used in the control. These parameters are passed through to the output document without interpretation.

ReadOnlyRules

Defines a set of rules that determine whether the field should be displayed as read-only. If a `PSXControlRef` child is specified, and the rules evaluate to `true`, Rhythmyx will use the control defined in the `PSXControlRef` tag to display the field rather than the control specified in the `PSXUISet`. If the read-only rules evaluate to `true`, Rhythmyx will set the `readOnly` attribute of the `DisplayField` tag in the output XML document generated by the `sys_ContentEditor.dtd` to `yes`.

Used in: Local, Shared

Contained by: *PSXUISet* (on page 210)

Attributes: None

Elements:

Name	Appearance	Description
<i>PSXRule</i> (on page 225)	Zero or more	Defines a rule.
<i>PSXControlRef</i> (on page 217)	Zero or one	Specifies a control to use when displaying the field if the rules evaluate as true.

ErrorInfo

Defines the text to display when a validation error occurs. The text specified in the `PSXDisplayText` tag is combined with the text from all other field or item validation errors, and is added as the first child node of the results document.

Used in: Shared

Contained by: *PSXContentEditor* (on page 250)

Attributes: None

Elements:

Name	Appearance	Description
<i>PSXDisplayText</i> (on page 211)	Once	Defines the text to display.

PSXStylesheet

Defines an XSL stylesheet reference.

Used in: Local, Shared

Contained by: *PSXConditionalStylesheet* (on page 219), *CommandHandler* (on page 227)

Attributes: None

Elements:

Name	Appearance	Description
<i>PSXUrlRequest</i> (on page 219)	Once	Defines the URL to the stylesheet.

PSXConditionalStyleSheet

Defines a set of conditions and the stylesheet to use if the conditions evaluate as true.

Used in: Local, Shared

Contained by: *CommandHandler* (on page 227)

Attributes: None

Elements:

Name	Appearance	Description
<i>PSXStyleSheet</i> (on page 218)	Once	Specifies the stylesheet to use.
<i>Conditions</i> (on page 221)	Once	Defines the set of conditions to evaluate.

PSXURLRequest

Defines the pieces required to build a URL, either directly or by calling a UDF. If the components are supplied directly, the server generates a URL in the following form (with missing components handled properly):

href?param1=value1¶m2=value2#anchor

Used in: Local, Shared

Contained by: *PSXChoices* (on page 213), *PSXStylesheet* (on page 218), *PSXConditionalRequest* (on page 220), *CommandHandler* (on page 227)

Attributes:

Name	Required?	Value	Effect
name	No	CDATA	Identifier for the request. Should be unique within the level definition.

Elements:

Name	Appearance	Description
<i>href</i> (on page 220)	Once (if building URL locally)	Defines the base of the URL, including the query string.
<i>PSXParam</i> (on page 221)	Zero or more times (if building URL locally)	Specifies the parameters for the URL.
<i>Anchor</i> (on page 220)	Zero or one time (if building URL locally)	Defines the anchor of the URL.
OR		

Name	Appearance	Description
<i>PSXExtensionCall</i> (on page 246)	Choice	Calls the a UDF that builds the URL.

Href

The location portion of a URL.

Used in: Local, Shared

Contained by: *PSXUrlRequest* (on page [219](#))

Attributes: None

Elements: None

Anchor

The anchor portion of a URL.

Used in: Local, Shared

Contained by: *PSXUrlRequest* (on page [219](#))

Attributes: None

Elements: None

PSXConditionalRequest

Defines a set of conditions and the URL to use if those conditions evaluate as true.

Used in: Local, Shared

Contained by: *CommandHandler* (on page [227](#))

Attributes: None

Elements:

Name	Appearance	Description
<i>PSXUrlRequest</i> (on page 219)	Once	Defines the URL request.
<i>Conditions</i> (on page 221)	Once	Defines the set of conditions to evaluate.

Conditions

Defines the set of rules to evaluate for a conditional stylesheet or conditional URL request.

Used in: Local, Shared

Contained by: *PSXConditionalStylesheet* (on page 219), *PSXConditionalRequest* (on page 220)

Attributes: None

Elements:

Name	Appearance	Description
<i>PSXRule</i> (on page 225)	One or more	Defines the rule to evaluate.

PSXParam

Defines a name-value pairing to pass parameters to a URL request.

Used in: Local, Shared

Contained by: *PSXControlRef* (on page 217), *PSXUrlRequest* (on page 219), *PSXActionLink* (on page 230)

Attributes:

Name	Required?	Value	Effect
type	No	CDATA	Allowed values are defined by the context in which the PSXParam tag is used.
name	Yes	CDATA	Name of the parameter.

Elements:

Name	Appearance	Description
<i>DataLocator</i> (on page 207)	Once	Specifies where to find the data for the tag.

FieldRules

Defines the set of rules for validating, translating, or determining the visibility of a field.

Used in: Local, Shared

Contained by: PSXField

Attributes: None

Elements:

Name	Appearance	Description
<i>PSXVisibilityRules</i> (see " PSXVisibility Rules " on page 222)	Zero or one	Defines rules that determine whether the field will be visible.
<i>PSXFieldValidationRules</i> (on page 224)	Zero or one	Defines the rules for validating the data in the field.
<i>FieldInputTranslation</i> (on page 223)	Zero or one	Converts data from one form to another when updating the database
<i>FieldOutputTranslation</i> (on page 223)	Zero or one	Converts data from one form to another when requesting data from the database.

PSXVisibility Rules

Determines the setting of the visibility flag when a data is requested.

Used in: Local, Shared

Contained by: *FieldRules* (on page [222](#))

Attributes:

Name	Required?	Value	Effect	Default
dataHiding	No	xsl xml	Hidden data is included in the output document. Hidden data is not included in the results document.	xsl

Elements:

Name	Appearance	Description
<i>PSXRule</i> (on page 225)	One or more	Defines the rules.

FieldInputTranslation

Specifies the UDF used to convert data from one form to another when it is updated to the database. Typically, this process "undoes" the transformation by the FieldOutputTranslation.

Used in: Local, Shared

Contained by: *FieldRules* (on page [222](#))

Attributes: None

Elements:

Name	Appearance	Description
<i>PSXFieldTranslation</i> (on page 223)	Once	Defines the UDF that performs the data conversion.

FieldOutputTranslation

Specifies the UDF used to convert data from one form to another when it is requested from the database.

Used in: Local, Shared

Contained by: *FieldRules* (on page [222](#))

Attributes: None

Elements:

Name	Appearance	Description
<i>PSXFieldTranslation</i> (on page 223)	Once	Defines the UDF that performs the data conversion.

PSXFieldTranslation

Specifies the call to the UDF that perform the data translation for the field.

Used in: Local, Shared

Contained by: *FieldInputTranslation* (on page [223](#)), *FieldOutputTranslation* (on page [223](#))

Attributes: None

Elements:

Name	Appearance	Description
<i>PSXExtensionCallSet</i> (on page 246)	Once	Specifies the call to the UDF that performs the data translation.
<i>ErrorLabel</i> (on page 212)	Zero or once	Defines a label to use if the data translation fails.

PSXFieldValidationRules

Defines the set of rules to validate the data in the field after it is transformed.

Used in: Local, Shared

Contained by: *FieldRules* (on page [222](#))

Attributes:

Name	Required?	Value	Effect	Default
name	No	CDATA	Defines a name that other fields in the editor can use to share this validation set.	None
maxErrorsToStop	No	CDATA	Defines the maximum number of errors generated before validation is terminated and errors are reported to the user.	10

Elements:

Name	Appearance	Description
<i>PSXRule</i> (on page 225)	Choice (Zero or more)	Defines validation rule.
<i>FieldValidationRuleRef</i> (on page 224)	Choice(Zero or more)	Defines existing validation set to use when validating this field.
<i>PSXApplyWhen</i> (on page 225)	Zero or one	Defines a set of rules that determine whether or not to perform validation on the field.
ErrorMessage	Zero or one	Defines text to use for the field if the data fails validation. This text will be returned to the user in the error document.

FieldValidationRuleRef

Specifies an existing set of field validation rules to use when validating the data for this field.

Used in: Local, Shared

Contained by: *PSXFieldValidationRules* (on page [224](#))

Attributes: None

Elements: None

PSXApplyWhen

Defines rules that determine whether validation will be performed.

Used in: Local, Shared

Contained by: *PSXFieldValidationRules* (on page 224)

Attributes:

Name	Required?	Value	Effect	Default
ifFieldEmpty	No	yes	yes: Data in field will be validated, even if field is empty.	no
		no	no: Data in field will not be validated if the field is empty.	

Elements:

Name	Appearance	Description
<i>PSXRule</i> (on page 225)	Zero or more	Defines rules that determine whether or not to apply field validation rules. If the rules evaluate to true, the validations are performed.

PSXRule

A set of conditions or a UDF that returns a Boolean value. When processing more than one rule joined using the `boolean` attribute, `and` takes precedence over `or`. Rules use short-circuit programming, meaning that once the answer is determined, no further rules are evaluated. For example, if two rules are combined by OR, and the first rule evaluates to true, the second rule is not evaluated.

Used in: Local, Shared

Contained by: *ReadOnlyRules* (on page 218), *Conditions* (on page 221), *PSXVisibilityRules* (see "PSXVisibility Rules" on page 222), *PSXFieldValidationRules* (on page 224), *PSXApplyWhen* (on page 225)

Attributes:

Name	Required ?	Value	Effect	Default
boolean	No	and or	If multiple rules are present, both rules must evaluate as true for the pair to evaluate as true. If either rule evaluates as true, the rule evaluates as true.	and

Elements:

Name	Appearance	Description
<i>PSXConditional</i> (on page 237)	Choice (Zero or more)	Defines rules locally as a set of conditions.
<i>PSXExtensionCallSet</i> (on page 246)	Choice	Calls a Java extension that defines and processes the rules. Not supported until nested UDFs are implemented.
<i>ErrorLabel</i> (on page 212)	Zero or one	Defines a label for the field if the rule evaluates to a negative value. The meaning of negative is interpreted by the user.

PSXApplicationFlow

Determines where the server should go after a non-query request. The default application flow is specified in the system definition. It can be overridden in the shared definition or the local definition.

Used in: Local, Shared

Contained by: *PSXContentEditor* (on page [250](#)), *PSXContentEditorSharedDef* (on page [255](#))

Attributes: None

Elements:

Name	Appearance	Description
<i>CommandHandler</i> (on page 227)	One or more	Contains a set of conditional URL requests one of which indicates where the server will redirect after the non-query operation has completed.

PSXCommandHandlerStylesheet

Defines the set of stylesheets used in the set of content editors. A stylesheet is required for each query handler. The stylesheet will be used to process an output document that conforms to the `sys_ContentEditor.dtd`. The default stylesheets are specified in the system definition. They can be overridden in the shared definition or the local definition.

Used in: Local, Shared

Contained by: *PSXContentEditor* (on page [250](#)), *PSXContentEditorSharedDef* (on page [255](#))

Attributes: None

Elements:

Name	Appearance	Description
<i>CommandHandler</i> (on page 227)	One or more	Defines a set of stylesheets used by the editor to render the output document.

CommandHandler

Defines the set of stylesheets or requests used within a specific content editor. Which children are allowed is determined by the parent of this element.

Used in: Local, Shared

Contained by: *PSXApplicationFlow* (on page 226), *PSXCommandHandlerStylesheets* (see "*PSXCommandHandlerStylesheet*" on page 226)

Attributes:

Name	Required?	Value	Effect
Name	Yes	CDATA	The internal name of the command handler that uses this stylesheet. Must use the associated object.

Elements:

Name	Appearance	Description
<i>PSXConditionalStylesheet</i> (on page 219)	Zero or more	Defines a set of conditions and the stylesheet to use if the conditions evaluate as true.
<i>PSXStylesheet</i> (on page 218)	Once	Defines the stylesheet to use for the command handler. If any <i>PSXConditionalStylesheet</i> elements appear, this stylesheet is used if all <i>PSXConditionalStylesheets</i> evaluate as false.
OR		
<i>PSXConditionalRequest</i> (on page 220)	Zero or more	Defines a set of conditions and the URL request to use if the conditions evaluate as true.
<i>PSXUrlRequest</i> (on page 219)	Once	Defines the URL request to use for the command handler. If any <i>PSXConditionalRequest</i> elements appear, this URL request is used if all <i>PSXConditionalRequests</i> evaluate as false.

PSXInputTranslations

Converts data from one form to another when data is updated to the database from two or more fields. Can only use IPSRequestPreProcessor exits.

Used in: Shared

Contained by: *PSXSharedFieldGroup* (on page [255](#)), ContentEditorSystemDef

Attributes: None

Elements:

Name	Appearance	Description
<i>PSXConditionalExit</i> (on page 229)	Zero or one	Defines the exit to use to perform the translation.

PSXOutputTranslations

Converts data from one form to another when data is requested from two or more fields. Can only use IPSResultDocumentProcessor exits. Typically, when an output translation is created, an corresponding PSXInputTranslation is required to "undo" the translation before the data is stored.

Used in : Shared

Contained by: *PSXSharedFieldGroup* (on page [255](#)), ContentEditorSystemDef

Attributes: None

Elements:

Name	Appearance	Description
<i>PSXConditionalExit</i> (on page 229)	Zero or one	Defines the exit to use to perform the translation.

PSXValidationRules

Defines the rules used to validate data from two or more fields before it is transformed. Can only use exits. If failures occur, the DisplayLabel of the field is replaced by the ErrorLabel, and text defined in the ErrorInfo element is also displayed.

Used in : Shared

Contained by: *PSXSharedFieldGroup* (on page 255), ContentEditorSystemDef

Attributes:

Name	Required?	Value	Effect
MaxErrorsToStop	No	CDATA	Defines the maximum number of errors generated before validation is terminated and errors are reported to the user. Must be a numeric value greater than 1.

Elements:

Name	Appearance	Description
<i>PSXConditionalExit</i> (on page 229)	Zero or one	Defines the exit to use to perform the validation.

PSXConditionalExit

Used in : Local, Shared

Contained by: *PSXInputTranslations* (on page 228), *PSXOutputTranslations* (on page 228), *PSXValidationRules* (on page 229)

Attributes:

Name	Required?	Value	Effect
MaxErrorsToStop	No	CDATA	Defines the maximum number of errors generated before validation is terminated and errors are reported to the user. Must be a numeric value greater than 1.

Elements:

Name	Appearance	Description
<i>PSXExtensionCallSet</i> (on page 246)	Once	Defines the call to the exit(s) that performs the translation or validation.

Name	Appearance	Description
<i>PSXApplyWhen</i> (on page 225)	Zero or one	Defines rules that determine whether the exit will be called. If not present, the exits are always called.

PSXActionLink

Defines an action that the end user can take. Meant to support objects such as buttons on HTML forms. The object must have an action that appends the name/value pairs defined in the PSXParam children of this element onto the resulting action

If the first parameter of this element is named submitHref, then the value of that parameter will be used as the target of the action. The rest of the parameters will be appended to this action as described in the previous paragraph.

Addressing actions in this manner separates the text on the object from the value submitted when that object is activated. It also allows a single object to support multiple actions that require different values for more than one parameter.

If the object is disabled, the stylesheet determines whether it is hidden or "grayed out". The stylesheet can use any user-interface control available as long as it conforms to the semantics of the type of object to which this element is assigned.

Used in : Local, Shared

Contained by: *PSXActionLinkList* (on page [231](#))

Attributes:

Name	Required?	Value	Effect	Default
name	No	CDATA	Defines a name for the link.	None
isDisabled	No	no yes	Link is not disabled; is the default, which conforms to HTML semantics. Link is disabled.	no
isTransition	No	no yes	Link is not a workflow Transition; is the default, which conforms to HTML semantics. Link is a Transition.	no

Elements:

Name	Appearance	Description
<i>PSXDisplayText</i> (on page 211)	Once	Defines the text displayed on the object.
<i>PSXParam</i> (on page 221)	Zero or more	Defines a name/value pair for the object.

PSXActionLinkList

Defines a set of PSXActionLinks for use in a single content editor.

Used in : Local

Contained by: *PSXCustomActionGroup* (on page 231)

Attributes: None

Elements:

Name	Appearance	Description
<i>PSXActionLink</i> (on page 230)	One or more	Defines an action available in the content editor.

PSXCustomActionGroup

A custom action group lets the user overwrite the default form action, remove default buttons and add new buttons.

Used in : Local, Shared

Contained by: DefaultUI

Attributes: None

Elements:

Name	Appearance	Description
<i>PSXLocation</i> (on page 232)	Once	Defines the location of the custom action group.
<i>RemoveAction</i> (on page 231)	Zero or one	Specifies actions to be removed from the user interface.
<i>PSXActionLinkList</i> (on page 231)	Once	Defines the group of actions to overwrite.

RemoveAction

Defines a list of ActionLink objects to remove from the user interface.

Used in : Local, Shared

Contained by: PSXCustomActionGroup

Attributes: None

Elements:

Name	Appearance	Description
<i>ActionLinkRef</i>	One or more	Specifies an ActionLink object to remove from the user interface.

ActionLinkRef

Reference to an ActionLink element through its name attribute.

Used in: Local, Shared

Contained by: *RemoveAction* (on page [231](#))

Attributes: None

Elements: None

PSXLocation

Defines the location of a custom action group on the page. The interpretation depends on the values assigned to the attributes.

Rhythmyx ignores the content of this element except in the following circumstances:

- The value of the page attribute is `summaryView`.
- The value of the page attribute is `rowEdit` and the value of the type attribute is `field` or `form`.

To specify additional parameters needed by the target, use the `PSXParam` child element of the `PSXActionLink`. For example, to replace the **[Add New]** in the summary editor, include a parameter (`PSXParam` child) in the `PSXActionLink` element called `targeturl`, and use the `sys_MakeAbsoluteLink` UDF to create a link to the modifyhandler. The customized page can use this URL to submit the new children into the modify handler.

Used in : Local, Shared

Contained by: *PSXCustomActionGroup* (on page [231](#))

Attributes:

Name	Required?	Value	Effect	Default
page	No	<code>summaryView</code> <code>rowEdit</code>	The objects will appear on one or more summary editors. The objects will appear on all row editors.	<code>summaryView</code>

type	No	form row field wfAction wfTransition	Objects appear for submission of main form. Only applicable to summary view; objects will appear on each row of the summary table. Only applicable to row editors; objects appear with the specified child summary views. Objects appear in the Workflow Action box. The page attribute is ignored; this object will appear on all pages in the content editor. Objects appear in the Workflow Transition box. The page attribute is ignored; this object will appear on all pages in the content editor.	form
sequence	No	CDATA	Defines the position of the object relative to the other objects. In other words, if the value is 1, this object will appear first. If this attribute has no value, the object is added to the end of the list.	None

Elements:

Name	Appearance	Description
<i>FieldRef</i> (on page 208)	Zero or more	Defines the (complex child) fields where the objects will appear.

PSXBackendCredential

Specifies the credential information needed to connect to a backend database.

Used in : Local, Shared

Contained by: *PSXTableLocator* (on page 199)

Attributes:

Name	Required?	Value	Effect
id	Yes	CDATA	Unique numeric identifier

Elements:

Name	Appearance	Description
<i>alias</i> (see " alias (lowercase) " on page 234)	Once	Identifies the backend credential so it can be reused.
<i>comment</i> (on page 234)	Zero or one	Free-form comment on the back end credential
<i>driver</i> (on page 235)	Once	Defines the JDBC driver. Must be one of the driver names specified when the driver was configured.
<i>server</i> (on page 235)	Zero or one	Specifies the server where the backend database resides.
<i>userid</i> (on page 235)	Zero or one	Specifies the user ID to use to access the backend database.
<i>password</i> (on page 235)	Zero or one	Specifies the password to use to access the backend database.

alias (lowercase)

Defines an alias to identify the backend database credential. Note that this element is different from the Alias element (defined using title case) that appears elsewhere.

Used in : Local, Shared

Contained by: *PSXBackEndCredential* (on page [233](#))

Attributes: None

Elements: None

comment

Allows the addition of free-form comments to the PSXBackEndCredential.

Used in : Local, Shared

Contained by: *PSXBackEndCredential* (on page [233](#))

Attributes: None

Elements: None

driver

Defines the connectivity to the backend database.

Used in : Local, Shared

Contained by: *PSXBackEndCredential* (on page [233](#))

Attributes: None

Elements: None

server

Defines the name of the machine where the backend database resides.

Used in : Local, Shared

Contained by: *PSXBackEndCredential* (on page [233](#))

Attributes: None

Elements: None

userID

Specifies the user ID to access the backend database.

Used in : Local, Shared

Contained by: *PSXBackEndCredential* (on page [233](#))

Attributes: None

Elements: None

password

Specifies the password to use to access the backend database.

Used in : Local, Shared

Contained by: *PSXBackEndCredential* (on page [233](#))

Attributes:

Name	Required?	Value	Effect	Default
encrypted	No	yes	Password is encrypted	no
		no	Password is not encrypted.	

Elements: None

PSXBackEndColumn

Defines a column in the backend database table.

Used in : Local, Shared

Contained by: *DataLocator* (on page 207), *variable* (on page 238), *value* (see "[value \(lowercase\)](#)" on page 239)

Attributes:

Name	Required?	Value	Effect
id	Yes	CDATA	Unique numeric identifier

Elements:

Name	Appearance	Description
<i>tableAlias</i> (on page 236)	Once	Specifies the backend database table.
<i>column</i> (on page 236)	Once	Specifies the name of the column.
<i>columnAlias</i> (on page 237)	Zero or one	Optional alias for this column. Useful if more than one column of the same name exists in a set of joined tables.

tableAlias

Specifies an existing database table definition.

Used in : Local, Shared

Contained by: *PSXBackEndColumn* (on page 236)

Attributes: None

Elements: None

column

Specifies a column in the backend database.

Used in : Local, Shared

Contained by: *PSXBackEndColumn* (on page 236)

Attributes: None

Elements: None

columnAlias

Specifies an existing column definition.

Used in : Local, Shared

Contained by: *PSXBackendColumn* (on page [236](#))

Attributes: None

Elements: None

PSXConditional

Used to define conditional statements. The format is {variable} {operator} {value} [{boolean} {cond}]....

Used in : Local, Shared

Contained by: *PSXRule* (on page [225](#))

Attributes:

Name	Required?	Value	Effect
id	Yes	CDATA	Defines an identifier for the conditional statement.

Elements:

Name	Appearance	Description
<i>variable</i> (on page 238)	Once	Defines the variable to be tested.
<i>operator</i> (on page 238)	Once	Defines the operation to use to test the variable.
<i>value</i> (see " value (lowercase) " on page 239)	Once	Defines the value against which to test the variable.
<i>boolean</i> (on page 239)	Zero or one	Defines Boolean operator between multiple conditional statements.

variable

The name of the variable to test in a PSXConditional statement.

Used in: Local, Shared

Contained by: *PSXConditional* (on page [237](#))

Attributes: None

Elements:

Name	Appearance	Description
<i>PSXBackendColumn</i> (on page 236)	Choice	Defines a backend database column where the data to be tested can be found.
<i>PSXTextLiteral</i> (on page 240)	Choice	Defines literal text to test.
<i>PSXCgiVariable</i> (on page 245)	Choice	Defines the CGI variable whose value to test.
<i>PSXHtmlParameter</i> (on page 243)	Choice	Defines the HTML parameter whose value to test.
<i>PSXCookie</i> (on page 243)	Choice	Defines the cookie whose value to test.
<i>PSXUserContext</i> (on page 245)	Choice	Defines the user-context variable whose value to test.
<i>PSXXmlField</i> (on page 244)	Choice	Defines the XML field whose value to test.

operator

Defines the relational operator to use in the comparison in a PSXConditional statement. Acceptable values are <>, <, <=, >, >=, IS NULL, IS NOT NULL, BETWEEN, NOT BETWEEN, IN, NOT IN, LIKE, NOT LIKE. (NOTE: Remember that some of the characters used above are reserved characters in XML, such as "<"; you must use the escape characters for these characters in the XML content.)

Used in : Local, Shared

Contained by: *PSXConditional* (on page [237](#))

Attributes: None

Elements: None

value (lowercase)

Defines the value for a `PSXConditional` or `PSXExtensionParam`.

Used in: Local, Shared

Contained by: *PSXConditional* (on page [237](#))

Attributes: None

Elements:

Name	Appearance	Description
<i>PSXBackEndColumn</i> (on page 236)	Choice	Defines the backend database column whose value the tested variable must match.
<i>PSXDateLiteral</i> (on page 241)	Choice	Defines a literal text date whose value the tested variable must match.
<i>PSXNumericLiteral</i> (on page 241)	Choice	Defines a literal numeric value that the tested variable must match.
<i>PSXTextLiteral</i> (on page 240)	Choice	Defines a literal text string that the tested variable must match.
<i>PSXCgiVariable</i> (on page 245)	Choice	Defines the CGI variable whose value the tested variable must match.
<i>PSXHtmlParameter</i> (on page 243)	Choice	Defines the HTML parameter whose value the tested variable must match.
<i>PSXCookie</i> (on page 243)	Choice	Defines the cookie whose value the tested variable must match.
<i>PSXUserContext</i> (on page 245)	Choice	Defines the user-context variable whose value the tested variable must match.
<i>PSXXmlField</i> (on page 244)	Choice	Defines the XML field whose value the tested variable must match.

boolean

The Boolean operator to use when joining multiple conditionals. Value can be either AND (default) or OR. Statements joined by AND take precedence over statements joined by OR in processing.

Used in : Local, Shared

Contained by: *PSXConditional* (on page [237](#))

Attributes: None

Elements: None

PSXLiteralSet

Defines a set of literal objects.

Used in : Local, Shared

Contained by: *DataLocator* (on page 207)

Attributes: None

Elements:

Name	Appearance	Description
<i>PSXDateLiteral</i> (on page 241)	Choice; Zero or more	Defines a specific date.
<i>PSXNumericLiteral</i> (on page 241)	Choice; Zero or more	Defines a specific numeric value.
<i>PSXTextLiteral</i> (on page 240)	Choice; Zero or more	Defines a specific text string.

PSXTextLiteral

Defines a literal text string.

Used in: Local, Shared

Contained by: *DataLocator* (on page 207), *variable* (on page 238), *value* (see "[value \(lowercase\)](#)" on page 239), *PSXLiteralSet* (on page 240)

Attributes:

Name	Required?	Value	Effect
id	Yes	CDATA	Defines an identifier for the string.

Elements:

Name	Appearance	Description
<i>text</i> (on page 242)	Once	Defines the text string.

PSXDateLiteral

Defines a literal date string.

Used in : Local, Shared

Contained by: *value* (see "[value \(lowercase\)](#)" on page 239), *PSXLiteralSet* (on page 240)

Attributes:

Name	Required?	Value	Effect
id	Yes	CDATA	Defines an identifier for the string.

Elements:

Name	Appearance	Description
<i>date</i> (on page 242)	Once	Specifies the date string
<i>format</i> (on page 242)	Once	Defines the format for the data. Must conform to the requirements of <code>java.text.SimpleDateFormat</code> . For more information, see the following URL: <i>http://java.sun.com/j2se/1.3/docs/api/java/text/SimpleDateFormat.html</i> (http://java.sun.com/j2se/1.3/docs/api/java/text/SimpleDateFormat.html).

PSXNumericLiteral

Defines a literal numeric value.

Used in: Local, Shared

Contained by: *value* (see "[value \(lowercase\)](#)" on page 239), *PSXLiteralSet* (on page 240)

Attributes:

Name	Required?	Value	Effect
id	Yes	CDATA	Defines an identifier for this numeric literal.

Elements:

Name	Appearance	Description
<i>number</i> (on page 242)	Once	Specifies the numeric value.

Name	Appearance	Description
<i>format</i> (on page 242)	Once	Defines the format for the value. Must conform to the requirements of <code>java.text.DecimalFormat</code> . For more information, see the following URL: http://java.sun.com/j2se/1.3/docs/api/java/text/DecimalFormat.html (http://java.sun.com/j2se/1.3/docs/api/java/text/DecimalFormat.html)

date

Defines the date string.

Used in : Local, Shared

Contained by: *PSXDateLiteral* (on page 241)

Attributes: None

Elements: None

number

Defines the number value.

Used in : Local, Shared

Contained by: *PSXNumericLiteral* (on page 241)

Attributes: None

Elements: None

text

Defines the text string.

Used in : Local, Shared

Contained by: *PSXTextLiteral* (on page 240)

Attributes: None

Elements: None

format

Defines the format for a the date in a *PSXDateLiteral* or the number in a *PSXNumericLiteral*.

Used in: Local, Shared

Contained by: *PSXDateLiteral* (on page 241), *PSXNumericLiteral* (on page 241)

Attributes: None

Elements: None

Key

Specifies the lookup key in the RXLOOKUP table under which the options are stored for a PSXChoices element.

Used in: Local, Shared

Contained by: *PSXChoices* (on page 213)

Attributes: None

Elements: None

PSXCookie

Specifies a cookie.

Used in : Local, Shared

Contained by: *DataLocator* (on page 207), *variable* (on page 238), *value* (see "[value \(lowercase\)](#)" on page 239)

Attributes:

Name	Required?	Value	Effect
id	Yes	CDATA	Defines an identifier for the cookie element.

Elements:

Name	Appearance	Description
name	Once	Specifies the name of the cookie.

PSXHTMLParameter

Specifies an HTML parameter.

Used in : Local, Shared

Contained by: *DataLocator* (on page 207), *variable* (on page 238), *value* (see "[value \(lowercase\)](#)" on page 239)

Attributes:

Name	Required?	Value	Effect
id	Yes	CDATA	Defines an identifier for the HTML parameter element.

Elements:

Name	Appearance	Description
<i>name</i> (on page 247)	Once	Specifies the name of the HTML parameter.

PSXHTMLSingleParameter

Specifies an HTML parameter. If the parameter has more than one value, only the first value will be returned.

Used in: Local, Shared

Contained by: *DataLocator* (on page 207), *variable* (on page 238), *value* (see "[value \(lowercase\)](#)" on page 239)

Attributes:

Name	Required?	Value	Effect
id	Yes	CDATA	Defines an identifier for the HTML parameter element.

Elements:

Name	Appearance	Description
<i>name</i> (on page 247)	Once	Specifies the name of the HTML parameter.

PSXXmIField

Specifies an XML field.

Used in: Local, Shared,

Contained by: *DataLocator* (on page 207), *variable* (on page 238), *value* (see "[value \(lowercase\)](#)" on page 239)

Attributes:

Name	Required?	Value	Effect
id	Yes	CDATA	Defines an identifier for the XML field element.

Elements:

Name	Appearance	Description
<i>name</i> (on page 247)	Once	Specifies the name of the XML field.

PSXCGIVariable

Specifies a CGI variable.

Used in : Local, Shared.

Contained by: *DataLocator* (on page 207), *variable* (on page 238), *value* (see "[value \(lowercase\)](#)" on page 239)

Attributes:

Name	Required?	Value	Effect
id	Yes	CDATA	Defines an identifier for the CGI variable element.

Elements:

Name	Appearance	Description
<i>name</i> (on page 247)	Once	Specifies the name of the CGI variable.

PSXUserContext

Specifies a user-context variable.

Used in : Local, Shared

Contained by: *DataLocator* (on page 207), *variable* (on page 238), *value* (see "[value \(lowercase\)](#)" on page 239)

Elements:

Name	Required?	Value	Effect
id	Yes	CDATA	Defines an identifier for the user-context variable.

Elements:

Name	Appearance	Description
<i>name</i> (on page 247)	Once	Specifies the name of the user-context variable.

PSXExtensionCallSet

Defines a set of calls to a group of Java extensions.

Used in : Local, Shared

Contained by: *DataLocator* (on page 207), *PSXFieldTranslation* (on page 223), *PSXRule* (on page 225), *PSXConditionalExit* (on page 229)

Attributes:

Name	Required?	Value	Effect
id	Yes	CDATA	Defines an identifier for the set of calls to the Java extensions.

Elements:

Name	Appearance	Description
<i>PSXExtensionCall</i> (on page 246)	Zero or more	Specifies the Java extension to call.

PSXExtensionCall

Defines the call to a Java extension.

Used in: Local, Shared

Contained by: *DataLocator* (on page 207), *PSXUrlRequest* (on page 219)

Attributes:

Name	Required?	Value	Effect
id	Yes	CDATA	Defines an identifier for the call to the Java extension.

Elements:

Name	Appearance	Description
<i>name</i> (on page 247)	Once	Specifies the fully qualified name of the Java extension to call in the format <handler>/<context>/<name>.
<i>PSXExtensionParamValue</i> (on page 247)	Zero or more	Specifies the parameters to provide to the Java extension.

PSXExtensionParamValue

Specifies a parameter of the Java extension called and defines a value for the parameter.

Used in : Local, Shared

Contained by: *PSXExtensionCall* (on page 246)

Attributes:

Name	Required?	Value	Effect
id	Yes	CDATA	Defines an identifier for the PSXExtensionParamValue element.

Elements:

Name	Appearance	Description
<i>value</i> (see " value (lowercase) " on page 239)	Once	Specifies the value to pass to the Java extension.

name

Specifies the name of an object called.

Used in: Local, Shared

Contained by: *PSXCookie* (on page 243), *PSXHtmlParameter* (on page 243), *PSXXmlField* (on page 244), *PSXCgiVariable* (on page 245), *PSXUserContext* (on page 245), *PSXExtensionCall* (on page 246)

Attributes: None

Elements: None

InputDataExits

Specifies a set of Java exits to apply as each request is made to the content editor, before the server begins its main processing steps.

Used in: Local

Contained by: *PSXContentEditorPipe* (on page 252)

Attributes: None

Elements:

Name	Appearance	Description
<i>PSXExtensionCallSet</i> (on page 246)	Zero or one	Defines the set of exits to call to process the data.

ResultDataExits

Specifies a set of Java exits to apply after the server has finished processing but before the document is returned to the requester. Each exit gets an opportunity to modify the results document.

Used in : Local

Contained by: *PSXContentEditorPipe* (on page [252](#))

Attributes: None

Elements:

Name	Appearance	Description
<i>PSXExtensionCallSet</i> (on page 246)	Zero or one	Defines the set of exits to call to process the data.

PSXSearchProperties

The PSXSearchProperties element serves as a container for all of the attributes that define the look and feel of the search interface and that define its field-level configuration.

NOTE: For backwards-compatibility, if this element is not present, the server refers to the PSXField element for search attributes that existed prior to version 5.5.

Used in: Local, Shared, System

Contained by: PSXField

Attributes

Name	Required?	Value	Effect	Default
userSearchable	Yes	yes no	Field is available to implementers and to Content Contributors for use in searches. Field is not available for search	yes
enableTransformation	Yes	yes no	Field data is transformed into simple text data during indexing (in other words, HTML, documents, rich-text formats, .pdf files, etc., are converted into raw text. Field data is not transformed to simple text format during indexing. (This value is preferred for fields that store simple text data in the first place.)	Depends on the datatype of the field. For example, the default for BLOB data types is yes (and cannot be changed).

userCustomizable	Yes	yes no	The field is available to be added to custom searches by the user. The field is not available to be added to custom searches by the user.	yes
defaultSearchLabel	No	CDATA	If a field does not have a UIDefinition, or if the UI definition has been overridden, the value of this attribute is used in search dialogs presented to the user. If both the UIDef and this attribute are missing, the name of the field is used.	None
visibleToGlobalQuery	Yes	yes no	The fields is available to be searched using the Search for field on the Search dialog in Content Explorer. The field is not available to be searched using the Search for field on the Search dialog in Content Explorer. The user can search on the field if customizing the search.	yes
searchToken	Yes	yes no	Punctuation marks are included in search strings, rather than being used as word separators. This option should be selected if the text in the field includes file names, product IDs, etc. Punctuation marks are used as word separators and are not included in search strings.	no

Elements: None

Content Editor Local Def

PSXContentEditor

The root element of the content editor proper.

Used in: Local

Contained by: Root

Attributes:

Name	Required?	Value	Effect
contentType	Yes	Numeric	Specifies the content type associated with the editor. This value is a key into the CONTENTTYPES table. The supplied value must exist in the table before the application is started.
workflowID	Yes	Numeric	Specifies the workflow associated with the editor. This value is a key into the WORKFLOWAPPS table. The supplied value must exist in the table before the application is started.

Elements:

Name	Appearance	Description
<i>PSXDataSet</i> (on page 251)	Once	Defines a set of fields for the content editor.
<i>PSXCommandHandlerStylesheets</i> (see "PSXCommandHandlerStylesheet" on page 226)	Zero or one	Defines the set of stylesheets used in the content editor. Any entry defined overrides the values in the system definition.
<i>PSXApplicationFlow</i> (on page 226)	Zero or one	Determines the redirection of the server after each query. Any entry defined overrides the values in the system definition.
<i>SectionLinkList</i> (on page 252)	Zero or one	Any URL provided in this list is passed directly to the output document, after being processed. The editor does not make use of these elements.
<i>PSXValidationRules</i> (on page 229)	Zero or one	Defines validation rules that require multiple fields to execute. They are implemented as exits.
<i>PSXInputTranslations</i> (on page 228)	Zero or one	Defines translation rules for the submission of data in multiple fields to the database.

<i>PSXOutputTranslations</i> (on page 228)	Zero or one	Defines translation rules for the retrieval of data in multiple fields from the database.
<i>PSXCustomActionGroup</i> (on page 231)	Zero or more	Defines the behavior of buttons and other HTML objects in the editor.

PSXDataSet

Defines a set of fields for the content editor.

Used in: Local

Contained by: *PSXContentEditor* (on page 250)

Attributes:

Name	Required?	Value	Effect
id	Yes	Numeric	Defines an identifier for this dataset.

Elements:

Name	Appearance	Description
<i>name</i> (on page 247)	Once	Defines a name for this dataset.
<i>description</i> (on page 251)	Once	Free-form description of the dataset.
<i>transactionType</i> (on page 251)	Once	Not used by content editors.
<i>PSXContentEditorPipe</i> (on page 252)	Once	Defines the set of fields for the dataset, their location, and how they are mapped to the output document.

PSXRequestor

Only needed if building the application containing the content editor without using the workbench.

description

Free-form description of the parent element.

Used in: Local

Contained by: *PSXDataSet* (on page 251), *PSXContentEditorPipe* (on page 252)

Attributes: None

Elements: None

transactionType

Obsolete.

SectionLinkList

Defines a set of named URLs to other Rhythmyx components.

Used in: Local

Contained by: *PSXContentEditor* (on page 250),

Attributes: None

Elements:

Name	Appearance	Description
<i>PSXUriRequest</i> (on page 219)	One or more	Defines a URL.

PSXContentEditorPipe

Defines a set of fields and the location of the values for those fields.

Used in: Local

Contained by: *PSXDataset* (on page 251)

Attributes:

Name	Required?	Value	Effect
id	Yes	Numeric	Defines an identifier for this pipe.

Elements:

Name	Appearance	Description
<i>name</i> (on page 247)	Once	Defines the name of the pipe.
<i>InputDataExits</i> (on page 247)	Once	Defines a set of Java exits to transform data when updating the database
<i>ResultDataExits</i> (on page 248)	Once	Defines a set of Java exits to transform data when querying data from the database.
<i>description</i> (on page 251)	Once	Free-form description of the pipe.
<i>PSXContainerLocator</i> (on page 198)	Once	Defines the set of backend database tables where the data for the fields in the pipe are stored.
<i>PSXContentEditorMapper</i> (on page 253)	Once	Defines the user interface of the fields in the content editor.

PSXContentEditorMapper

Defines the set of fields and any business rules that affect one field.

Used in : Local

Contained by: *PSXContentEditorPipe* (on page [252](#))

Attributes: None

Elements:

Name	Appearance	Description
<i>PSXFieldSet</i> (on page 197)	Once	Defines the set of fields in the content editor.
<i>PSXUIDefinition</i> (on page 210)	Once	Defines the user interface display of the fields in the content editor.
<i>SystemFieldExcludes</i> (on page 253)	Zero or one	Defines the system fields not used in this content editor.
<i>SharedFieldIncludes</i> (on page 254)	Zero or one	Defines the shared field groups included in the content editor.

SystemFieldExcludes

By default, all system fields are included in each content editor. Any field specified in this tag will be excluded from the content editor.

Used in: Local

Contained by: *PSXContentEditorMapper* (on page [253](#))

Attributes: None

Elements:

Name	Appearance	Description
<i>FieldRef</i> (on page 208)	One or more	Specifies a field to be excluded from the content editor. If the field does not exist, the system returns an error.

SharedFieldIncludes

Specifies the sets of shared fields that will be included in the content editor.

Used in: Local

Contained by: *PSXContentEditorMapper* (on page [253](#))

Attributes: None

Elements:

Name	Appearance	Description
<i>SharedFieldGroupName</i> (on page 254)	One or more	Specifies a shared field group to include in the content editor. If the group does not exist, the system returns an error.
<i>SharedFieldExcludes</i> (on page 254)	One or more	Specifies a shared field to exclude from the content editor.

SharedFieldExcludes

Specifies a field in an included shared field group that will be excluded from the content editor.

Used in: Local

Contained by: *SharedFieldIncludes* (on page [254](#))

Attributes: None

Elements:

Name	Appearance	Description
<i>FieldRef</i> (on page 208)	One or more	Specifies a field to be excluded from the content editor. If the field does not exist, the system returns an error.

Content Editor Shared Def

SharedFieldGroupName

Specifies the name of a shared field group to include in the content editor.

Used in: Local

Contained by: *SharedFieldIncludes* (on page [254](#))

Attributes: None

Elements: None

PSXContentEditorSharedDef

Defines a set of shared field groups

Used in: Shared

Contained by: Root

Attributes: None

Elements:

Name	Appearance	Description
<i>PSXSharedFieldGroup</i> (on page 255)	One or more	Defines a set of fields shared by two or more content editors.
<i>PSXApplicationFlow</i> (on page 226)	Zero or one	Determines the redirection of the server after each query. Overrides the definition in the system definition for the specified handlers.
<i>PSXCommandHandlerStylesheets</i> (see "PSXCommandHandlerStylesheet" on page 226)	Zero or one	Defines a set of stylesheets used by the shared field group. Overrides the definition in the system definition for the specified handlers.

PSXSharedFieldGroup

Defines a collection of fields, their location, business rules associated with the fields, and optional default values and default display properties.

Used in: Shared

Contained by: *PSXContentEditorSharedDef* (on page [255](#))

Attributes:

Name	Required?	Value	Effect
name	Yes	CDATA	Defines a name for this group of shared fields.

Elements:

Name	Appearance	Description
<i>PSXContainerLocator</i> (on page 198)	Once	Defines the set of backend database tables where the data for the fields in the shared field group are stored.
<i>PSXFieldSet</i> (on page 197)	Once	Defines the set of fields in the shared field group.
<i>PSXUIDefinition</i> (on page 210)	Once	Defines the user interface display of the fields in the shared field group.
<i>PSXValidationRules</i> (on page 229)	Zero or one	Defines validation rules for multiple fields.
<i>PSXInputTranslations</i> (on page 228)	Zero or one	Defines translation rules for the submission of data in multiple fields to the database.

Name	Appearance	Description
<i>PSXOutputTranslations</i> (on page 228)	Zero or one	Defines translation rules for the retrieval of data in multiple fields from the database.

Index

A

ActionLinkRef • 231, 232
 Activating a Content Editor Resource for Edit • 21, 23
 Adding Choices to a Control • 49, 52
 Adding Components to the Content Editor • 132, 142
 Adding Custom Menu and Toolbar Actions • 170, 171
 Adding Data Validation • 75
 Adding Parameters to a Control • 47
 Adding Related Content Links to a Content Editor • 132, 139
 Adding the sys_EditLive Control to a Content Editor • 169
 Adding the sys_WebImageFX Control to a Content Editor • 178
 alias (lowercase) • 234
 Alias (uppercase) • 200
 Anchor • 219, 220
 Appendices • 129
 Attaching Tables to a Content Editor • 21, 23

B

Basic Objects • 197
 Best Practices
 sys_EditLive • 172
 sys_WebImageFX • 183
 boolean • 237, 239

C

Child Editor Field Properties Dialog • 35
 Child Editors • 38
 Choices Tab • 67
 Clearing Binary Data from a Field • 60
 column • 236
 columnAlias • 236, 237
 CommandHandler • 218, 219, 220, 226, 227
 comment • 234
 Conditions • 219, 220, 221, 225
 Configuring a Content Editor Control • 72

Content Assemblers and the ELJ HTML Editor • 169
 Content Editor Control Dialogs • 64
 Content Editor Control Reference • 47, 72, 147
 Content Editor Definition Files • 6
 Content Editor Field Controls • 47, 63
 Content Editor Local Def • 250
 Content Editor Maintenance • 21
 Content Editor Maintenance Dialogs • 14
 Content Editor Properties Dialog • 14
 Content Editor Settings Content Item Input and Output Translations Tabs • 18
 Content Editor Settings Dialog • 15
 Content Editor Settings General Tab • 16
 Content Editor Settings Item Validation Tab • 19, 80
 Content Editor Settings Workflow Tab • 17
 Content Editor Shared Def • 254
 Content Editor XML Reference • 197
 Content Type • 21, 23
 Continuous Conversion Example • 100, 108, 109
 Control • 47
 Control Events • 148
 Control Header • 147
 Control Tab • 65
 Control Template Standards • 148
 Controlling Processing of XML files • 184
 Copying the Template File to the Client Word Application • 119, 126
 Create Choice Lookup Request Dialog • 54, 71
 Creating a Content Editor File Based on an Existing Definition File • 132
 Creating a Content Editor from a Database Table • 22
 Creating a Content Editor from an XML File • 22
 Creating a Content Editor from Scratch • 21, 114
 Creating a Content Editor Resource • 21
 Creating a Content Editor that Extracts Text • 92, 94
 Creating a New Field • 36
 Creating an Internal Lookup Query • 157, 158, 161, 189, 195
 Creating the Content Editor Application • 144
 Creating the Content Editor Definition • 131
 Creating the Word Template File • 115, 116
 Customizing ELJ Configuration • 170
 Customizing the ArticleWord Content Editor • 111
 Customizing the ELJ Editor • 169

Customizing the sys_EditLive control • 169
Customizing the sys_WebImageFX Control • 181

D

Data Type • 55
Database • 200
DataLocator • 205, 207, 208, 209, 221, 236, 240, 243, 244, 245, 246
Datatype • 205, 206
date • 241, 242
Default Value of Field • 58
DefaultSelected • 214, 215
DefaultValue • 205, 207, 209
Defining Conditions for Exits, Effects, and Cloning Processes • 25
Defining Fields • 136
Defining the Content Editor Mapper • 132, 134
Defining the Database Definition • 132, 133
Defining the Interface • 138
Deleting a Field • 38
description • 251, 252
Display Control Properties Dialog • 64
Displaying Extracted Text in a Content Editor • 95, 97
driver • 234, 235

E

Editing a Field • 38
EditLive for Java Editor • 164
Error Label • 46
ErrorInfo • 211, 218
ErrorLabel • 211, 212, 223, 226
Example
 Hiding the Workflow Field in the Content Editor System Definition • 86
Excluding System Fields • 137

F

Field Data • 33, 39
Field Maintenance • 38
Field Maintenance Dialogs • 32
Field Properties Dialog • 34
Field Sets • 134
FieldInputTranslation • 222, 223
Field-Level Validation • 19, 76
FieldOutputTranslation • 222, 223
FieldRef • 207, 208, 209, 233, 253, 254
FieldRules • 205, 222, 223, 224
FieldValidationRuleRef • 224

format • 241, 242
Format of Field • 57

H

How to Create a Word-based Content Editor • 114
How Word-based Content Editors Work • 113
Href • 219, 220

I

Implementing a Content Editor Manually • 22, 31, 131
Implementing Text Extraction in Rhythmyx • 92
Implementing the sys_WebImageFX Control Manually • 181
Including Shared Fields • 137
InputDataExits • 247, 252
Installing New Features of Rhythmyx Word Connector • 119, 122, 124
Introduction to Content Editors • 5
Item-Level Validation • 80

K

Key • 214, 243

L

Label • 211, 212
Label of Field • 44
Local Definition Structure • 6

M

Maintaining Content Editor Fields • 31
Maintaining Content Editor Settings • 21, 24, 80
Maintaining Content Editors • 13
Managing Dependencies in a Control • 51
Migration Example • 108
Mnemonic • 45
Modifying the Content Assembler to Display Custom Word-based Content Editor Fields • 115, 118
Modifying the Style Sheet for Parsing Fields • 115, 116
Moving Rhythmyx Word Connector Files to the Correct Directory • 119, 120

N

name • 243, 244, 245, 246, 247, 251, 252
Name of Field • 43
New Field Properties Dialog • 33, 114
number • 241, 242

O

Occurrence of Field • 56
 OccurrenceSettings • 205
 operator • 237, 238
 Origin • 200, 201

P

password • 234, 235
 Processing Related Links • 124
 PSXActionLink • 211, 221, 230, 231
 PSXActionLinkList • 230, 231
 PSXApplicationFlow • 226, 227, 250, 255
 PSXApplyWhen • 224, 225, 230
 PSXBackEndColumn • 207, 236, 237, 238, 239
 PSXBackEndCredential • 200, 233, 234, 235
 PSXCGIVariable • 207, 238, 239, 245, 247
 PSXChoices • 211, 213, 214, 215, 216, 219, 243
 PSXCommandHandlerStylesheet • 226, 227, 250, 255
 PSXConditional • 226, 237, 238, 239
 PSXConditionalExit • 228, 229, 246
 PSXConditionalRequest • 219, 220, 221, 227
 PSXConditionalStyleSheet • 218, 219, 221, 227
 PSXContainerLocator • 198, 199, 252, 255
 PSXContentEditor • 218, 226, 250, 251, 252
 PSXContentEditorMapper • 197, 210, 252, 253, 254
 PSXContentEditorPipe • 198, 247, 248, 251, 252, 253
 PSXContentEditorSharedDef • 226, 255
 PSXControlRef • 211, 217, 218, 221
 PSXCookie • 207, 238, 239, 243, 247
 PSXCustomActionGroup • 211, 231, 232, 251
 PSXDataSet • 250, 251, 252
 PSXDateLiteral • 207, 239, 240, 241, 242
 PSXDefaultSelected • 214, 215
 PSXDefaultUI • 210
 PSXDisplayMapper • 208, 209, 210
 PSXDisplayMapping • 208, 209
 PSXDisplayText • 211, 212, 216, 218, 230
 PSXEntry • 211, 214, 216, 217
 PSXExtensionCall • 208, 220, 246, 247
 PSXExtensionCallSet • 208, 223, 226, 229, 246, 247, 248
 PSXExtensionParamValue • 246, 247
 PSXField • 202
 PSXFieldTranslation • 212, 223, 246
 PSXFieldValidationRules • 212, 222, 224, 225
 PSXHTMLParameter • 207, 238, 239, 243, 247

PSXHTMLSingleParameter • 207, 244
 PSXInputTranslations • 228, 229, 250, 255
 PSXLiteralSet • 207, 240, 241
 PSXLocation • 231, 232
 PSXNullentry • 214, 215, 216
 PSXNumericLiteral • 207, 239, 240, 241, 242
 PSXOutputTranslations • 228, 229, 251, 256
 PSXParam • 207, 217, 219, 221, 230
 PSXRequestor • 251
 PSXRule • 212, 218, 221, 222, 224, 225, 237, 246
 PSXSearchProperties • 205, 248
 PSXSharedFieldGroup • 197, 198, 210, 228, 229, 255
 PSXStylesheet • 218, 219, 227
 PSXTableLocator • 199, 200, 201, 233
 PSXTableRef • 199, 201
 PSXTableSet • 198, 199
 PSXTextLiteral • 208, 238, 239, 240, 242
 PSXUIDefinition • 208, 210, 253, 255
 PSXUISet • 209, 210, 212, 213, 217, 218
 PSXURLRequest • 214, 218, 219, 220, 221, 227, 246, 252
 PSXUserContext • 208, 238, 239, 245, 247
 PSXValidationRules • 229, 250, 255
 PSXVisibility Rules • 222, 225
 PSXXmlField • 207, 238, 239, 244, 247

Q

Quick Field Creation • 37

R

Read-only Rules • 89
 ReadOnlyRules • 211, 217, 218, 225
 Registering a New Content Type • 131
 RemoveAction • 231, 232
 ResultDataExits • 248, 252
 Rule Editor • 20

S

Sample Error Page • 83
 Sample Item Validation Exit • 81
 Search Properties • 62
 SectionLinkList • 250, 252
 Selecting a Content Editor Definition Template • 131, 132
 server • 234, 235
 Setting a Field in a Content Editor to Read-Only • 90

- Setting the Address in the Word Template Files • 119, 122
- Shared Definition Structure • 10
- SharedFieldExcludes • 254
- SharedFieldGroupName • 254
- SharedFieldIncludes • 253, 254
- Show in Preview • 42
- Show in Summary • 61
- Source of Field • 40
- Specifying a URL for a Control • 49
- Specifying an External Request • 49
- Specifying an Internal Request • 50
- Specifying the URL of a Choices Application • 53
- Standard Rhythmyx Controls • 149
 - sys_CalendarSimple • 150
 - sys_CheckBoxGroup • 152, 195
 - sys_DropDownSingle • 159, 195
 - sys_EditBox • 163
 - sys_EditLive Control • 165, 173
 - sys_File • 93, 176, 183
 - sys_HiddenInput • 185
 - sys_HTMLEditor • 194
 - sys_RadioButton • 186, 195
 - sys_Table • 190
 - sys_TextArea • 192
 - sys_TextExtraction • 96, 98
 - sys_WebImageFX and the WebImageFX Editor • 175
 - sys_WebImageFX Control • 176
- SystemFieldExcludes • 253

T

- tableAlias • 236
- TableLocatorAlias • 200
- text • 240, 242
- Text Extraction • 91, 98
- transactionType • 251
- Transition-Dependent Field-Level Validation • 78
- Treating Text as Binary • 59
- Type of Field • 41

U

- Understanding the System Definition • 12
- Updating the Content Type Registration • 143
- Updating the sys_FileWord Content Editor Control • 119, 127
- Upgrading from sys_eWebEditPro to sys_EditLive • 173

- Uploading External Binary Files into Rhythmyx • 92, 93
- URL Request Properties Dialog • 68
- URL Request Properties Dialog for External Requests • 69
- URL Request Properties Dialog for Internal Requests • 70
- userID • 234, 235
- Using the Value Selector • 26, 54, 70, 71, 72, 73

V

- Validating the Content Editor Definition • 145
- value (lowercase) • 236, 237, 239, 240, 241, 243, 244, 245, 247
- Value (uppercase) • 216, 217
- variable • 236, 237, 238, 240, 243, 244, 245
- Visibility and Read-only Rules • 85
- Visibility Rules • 86