Rhythmyx

Technical Reference Manual

Version 6.7

Printed on 8 June, 2009

Copyright and Licensing Statement

Licenses and Source Code

Rhythmyx uses Mozilla's JavaScript C API. See ***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

Percussion Software
600 Unicorn Park Drive
Woburn, MA 01801 U.S.A.
781.438.9900
Internet E-Mail: technical_support@percussion.com
Website: http://www.percussion.com

Contents

CHAPTER 1

About the Rhythmyx Technical Reference Manual

The Rhythmyx Technical Reference Manual provides detailed technical information about the system for advanced implementers performing advanced implementation and customization of Rhythmyx, such as developing custom extensions or specialized web applications and JSP pages.

Users of this manual should have attended Developer's Training for the Rhythmyx Content Management System and should have significant hands-on experience implementing the system.

Users of this manual should already be familiar with the *Rhythmyx Concepts Guide* and *Rhythmyx Implementation Guide*.

CHAPTER 2

Content Reference

The Content engine is the means by which users enter and maintain content.

The basic content unit in Rhythmyx is the Content Item. A Content Item is a portion of a page, such as an image, banner, footer, or block of text; or a collection of other Content Items, such as a sidebar, or even a complete page. By defining a web page in terms of Content Items and collections of Content Items, Rhythmyx provides the maximum flexibility to modify only the portions of a page that actually change, leaving the remainder undisturbed, and to reformat individual Content Items for multiple uses.

The first section of this chapter outlines the logical architecture and processing of the Content engine. The next section is a reference to the extensions used in this engine. The last section documents the controls available for use on Content Editors.

Logical Architecture and Processing

This section consists of the following topics:

Logical Architecture

The Content engine is comprised of two distinct but related engines:

- the Content Editor engine, which allows users to interact with Content Items, and which interacts with the Repository to add and retrieve Content Item data; and
- the query engine, queries the Repository when users submit searches for Content Items.

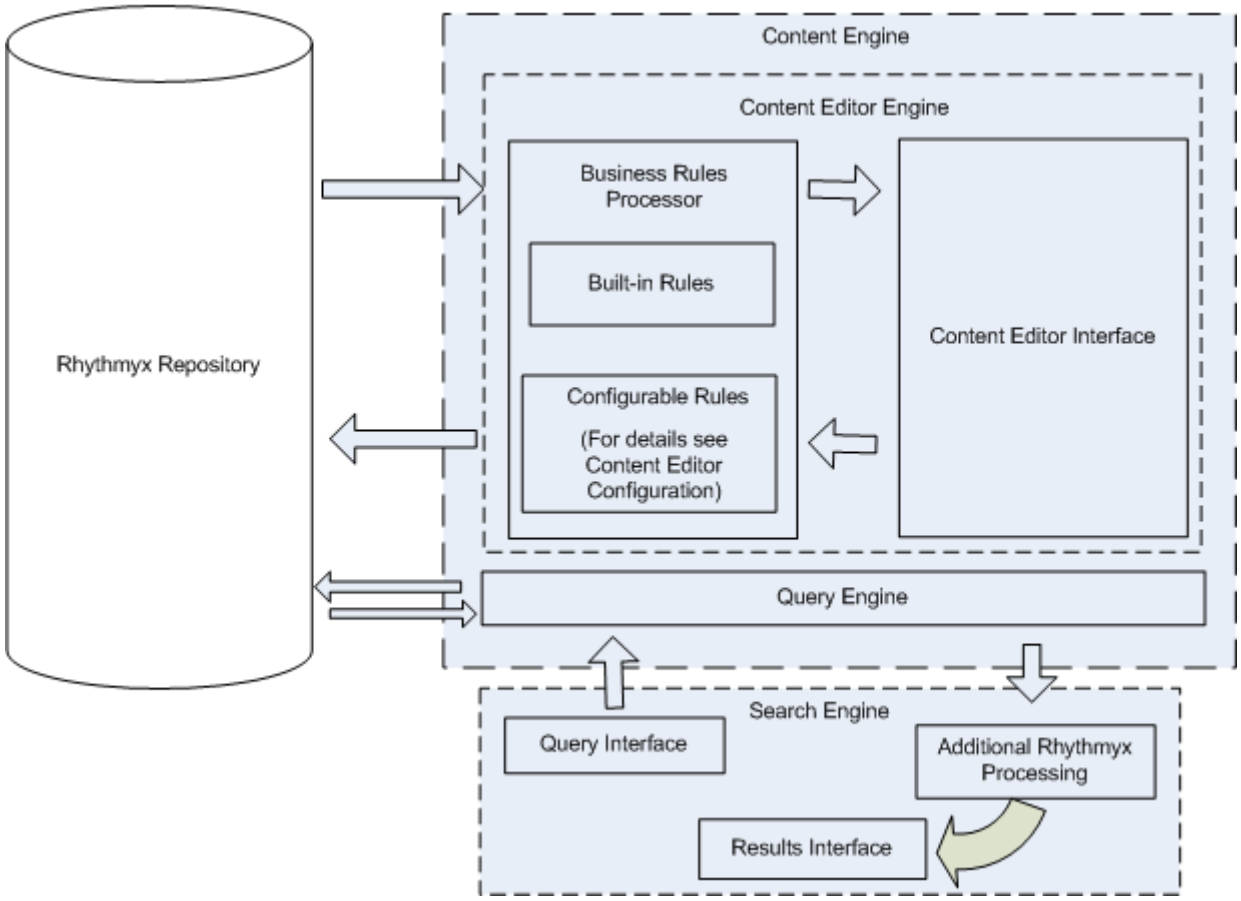


Figure 1: Logical architecture of the Content engine

The logical structure of the Content Editor engine consists of a set of business rules and a user interface. The set of business rules includes a small set of built-in rules (for example, each Content Item must have a title) and a larger set of configurable business rules. The business rules define such behavior as:

- the set of fields that comprise a Content Item;
- default values for fields when a new Content Item is created;
- processing of the Content Item when it is retrieved from the Repository or updated in the Repository.

The query engine interacts with the Search engine, which provides a user interface for searches and which performs additional processing on search results.

Content Editor Configuration

Content Editor configuration defines the business rules for processing Content Items of a specific Content Type. Configurations are defined at three levels:

- System definition

  The system definition define a set of fields that are always shared across revisions. Some of these fields are required on all Content Items (such as sys_title, and sys_contentid), but others are optional. Note that the interface behavior of these fields can be overridden at the local level, although the field properties cannot be modified at the local level.

- Shared definitions

  Each shared definition configuration defines one or more sets of fields (shared field groups) that are used by more than one Content Editor. An implementation may include any number of shared definition configurations, or may contain no shared definition configurations. The interface behavior of fields defined in a shared definition configuration can be overridden at the local level, although the field properties cannot be modified at the local level.

- Local definition

  The local definition defines the set of fields specific to a particular Content Type (each Content Type must include at least one local field, even if that field is a hidden dummy field). A specific subset of system fields are always included in a local definition, including:

  - System title (sys_title)
  - Community ID (sys_communityid)
  - Locale (sys_lang)
  - Current View (sys_currentview; hidden)
  - Workflow ID (sys_workflowid)
  - Hibernate Version (sys_hibernateVersion; hidden)

  Other systems fields can also be included.

  The local definition also specifies which shared fields are included in the Content Type. Note that in practice, all the fields in a specific shared field group are included in the local definition, and fields other than those specified are then excluded. In the Repository, however, when a Content Item is created of a Content Type that includes a shared field, the space allocated in the Repository for that Content Item includes space for all fields in the Shared field group. For example, suppose a shared field group included ten fields, but a Content Type included only one of those fields. Whenever a Content Item of that Content Type is created, space will be allocated for all ten fields even though only one of them is being used. Thus shared field groups should be defined as compactly as possible to minimize the impact of unused fields.

The local definition defines the Workflows available for Content Items of the Content Type, as well as any item-level processing, such as item-level validation or data transformations.

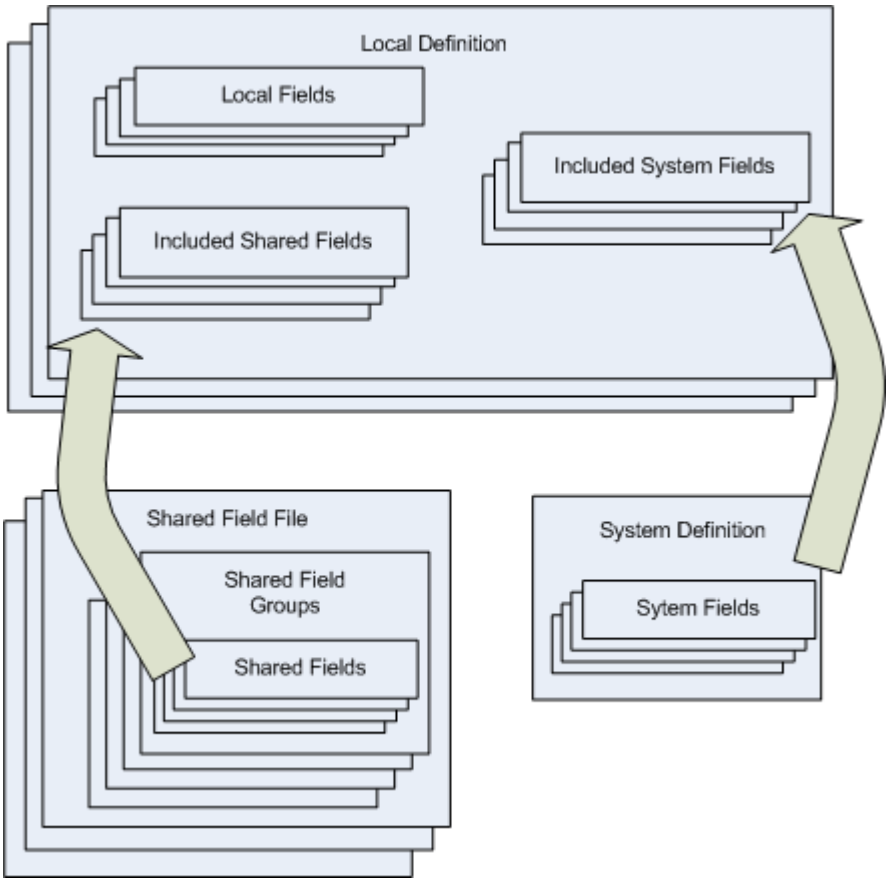


Figure 2: Content Editor configuration

Regardless of whether a field is defined in the system, shared, or local definition, the same field configuration options are available. For each field, a set of basic field properties must be defined, such as the name of the field, the type of data stored in the field, and its size. Special processing, such as validation and visibility rules, can also be defined for each field. Each field also requires a user-interface (UI) definition. The UI definition must at least specify the control used to render the field (including the sys_hidden control for fields that are never visible), but usually also specifies the label displayed for the field and may define a label to display if the field contains errors.

Content Processing

Content processing begins when a user submits a request for a Content Item to the server. If a new Content Item is requested, a new Content Item instance is created with default values. If an existing Content Item is requested, the Content Item data is retrieved from the Repository.

Once the Content Item data is available, any output Transforms or other pre-processing extensions are run on the Content Item. The Content Item is then displayed to the user in a Content Editor interface.

When the user has made all changes, they submit the Content Item to the server. At this point, the server runs any field validation processing for the Content Item. (Item-level validations are run when a user performs a Workflow Transition on the Content Item.) If the Content Item fails any validations, it is returned to the user in the Content Editor interface with error messages displayed.

If the Content Item passes all validations, any input transforms or post-processing extensions are run on it. The server then sends the Content Item data to the Repository. The updated Content Item data is then retrieved from the Repository and displayed to the user again.

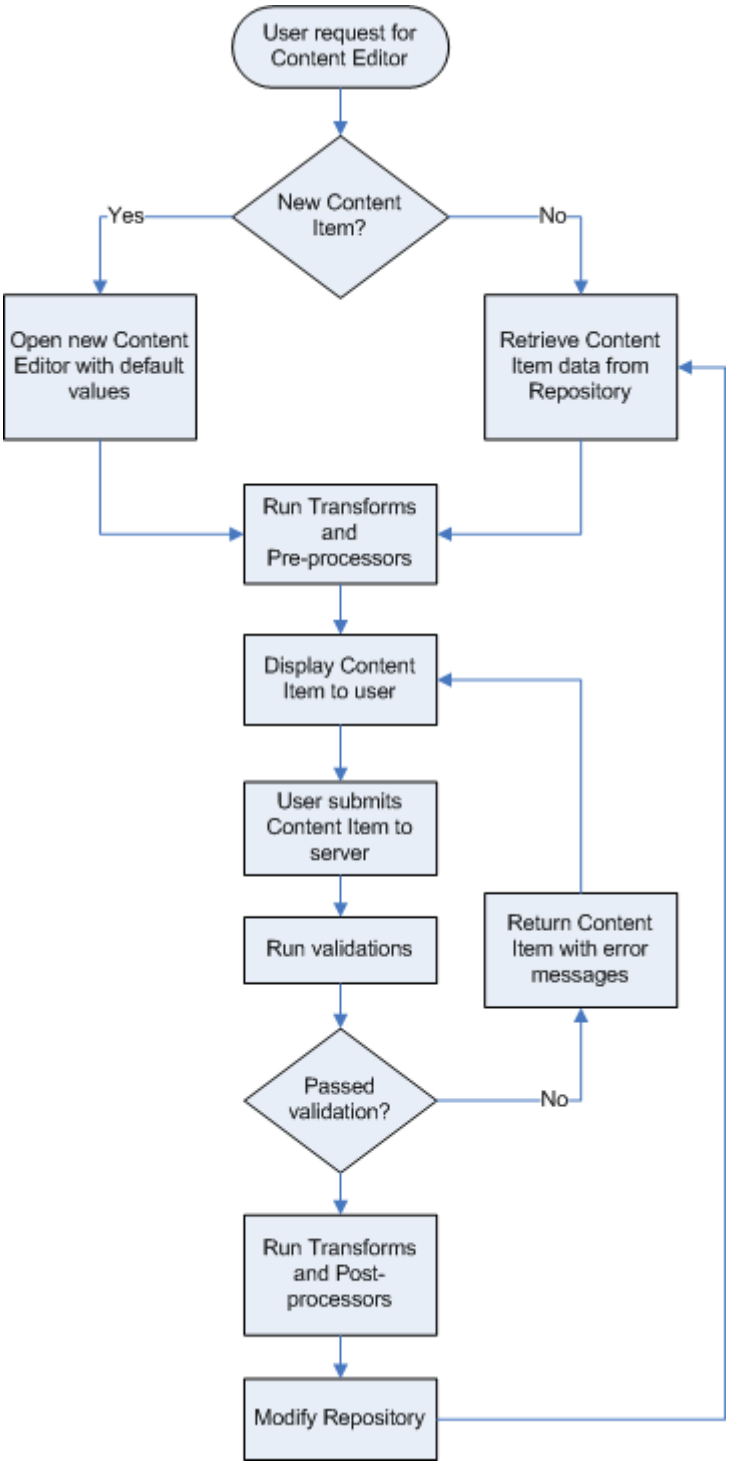


Figure 3: Content Engine Processing

When closing a Content Item, the user is given the option of saving the Content Item. If they choose to save it, it is submitted to the server and the processing described above is executed from that point. If they choose not to save it, the Content Editor is dismissed and any changes are lost.

Search Processing

Search processing begins when a search request is issued, such as a user search for a Content Item or a related content search. If the user issues the search request, a search interface is returned, in which the user can enter the criteria for the search query. When the user has defined the criteria for the query, they submit the query to the search engine. The search engine processes the query and returns a set of results. Rhythmyx may perform some additional filtering on this results set (such as filtering out Content Items not in the user's currently logged Community) before returning the final results set to the user.

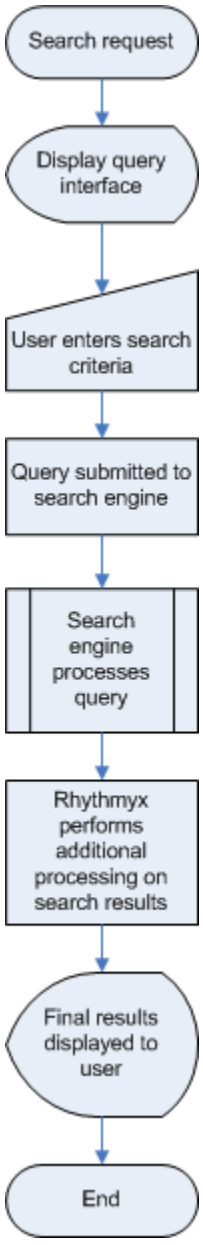


Figure 4: Search processing

Content Editor Extensions

This section documents extensions used in Content Editor processing, including:

- ***Item Validations*** (see below)
- ***Field Validations*** (see below)
- ***Field Transformers*** (see page 23)
- ***Item Transformers*** (see page 33)
- ***Document Pre-processors*** (see page 33)
- ***Result Document Processors*** (see page 33)

Item Validations

Item Validations are run when a Content Item is Transitioned from one Workflow State to another. Item Validations operate on multiple fields in a Content Item. In most cases, Field Validations provide adequate validation functionality, but Item Validations may provide improved performance in some circumstances. For example, if a field value must be validated against the value of more than one other field, multiple Field Validations could be implemented. Each of these validations would be run individually, and could result in diminished performance. Using a single Item Validation instead could alleviate the performance impact of the validation processing.

Item Validations must implement the interface IPSItemValidator. (NOTE: The implementation must be thread safe; for details see ***General Requirements of Extensions*** on page 180.)

Field Validations

Field validations are the first set of extensions processed when a Content Item is submitted to the Repository. Field validations process the data in a field to check whether it conforms to specified validation parameters. If the value in the field does not match the specified validation parameters, an error is returned. If the field passes the validation, processing continues to the next field. Once all Field validations have been processed, processing continues to Field transforms.

A field validation validates the data only in the associated field, although it can validate against data in other fields in the Content Item.

Field validations must implement the interface IPSFieldValidator. (NOTE: The implementation must be thread safe; for details see ***General Requirements of Extensions*** on page 180.)

Note that the Rhythmyx Workbench includes a number of built-in Field Validation functions. In the server, these functions are implemented as field validation extensions.

sys_ValidateDateRange

Validates that the date in the field falls within the specified range.

Class Name

com.percussion.validate.PSValidateDate

Interface

com.percussion.extension.IPSFieldValidator

Context

global/percussion/content/

Parameters

Name	Data Type	Description
value	Object	The value to be validated. If the value is a string (java.lang.String), it is converted to a date (java.util.Date); a value that is a date is evaluated directly.
min	Object	The minimum date value. Can be either a java.util.Date or a java.lang.String (which is converted to a date). This value is not included in the range unless the includemin flag is set to *true*.
includemin	Object	If the value is true, the range includes the minimum value. Otherwise the range does not include the minimum value. Can be either a java.lang.Boolean or a java.lang.String (which will be converted to a Boolean value; "true", "yes", or "t" [case-insensitive] convert to true, all others convert to false).
max	Object	The maximum date value. Can be either a java.util.Date or a java.lang.String (which is converted to a date). This value is not included in the range unless the includemax flag is set to *true*.
includemax	Object	If the value is true, the range includes the maximum value. Otherwise the range does not include the maximum value. Can be either a java.lang.Boolean or a java.lang.String (which is converted to a Boolean value; "true", "yes", or "t" [case-insensitive] convert to true, all others convert to false).

sys_ValidateJexlFieldExpression

Validates that the value of the field falls within the specified range.

Class Name

com.percussion.validate.PSValidateJexlExpression

Interface

com.percussion.extension.IPSFieldValidator

Context

global/percussion/content/

Parameters

Name	Data Type	Description
value	Object	The value to be validated, used without conversion, but extracted if a replacement value.
expression	String	The Java Expression Language (JEXL) expression to use to validate the value. The value must be bound as $value.

sys_ValidateNumberRange

Validates that the numeric value in the field falls within the specified range.

Class Name

com.percussion.validate.PSValidateNumber

Interface

com.percussion.extension.IPSFieldValidator

Context

global/percussion/content/

Parameters

Name	Data Type	Description
value	Object	The value to be validated. The value can be a java.lang.Number, a replacement value, or a java.lang.String. Values of the type java.lang.Number or replacement values are evaluated directly. Values of the type java.lang.String are converted to Double before evaluation.
min	Object	The minimum date value. Can be either a java.lang.Number or a java.lang.String (which will be converted to a Double). This value is not included in the range unless the includemin flag is set to *true*.
includemin	Object	If the value is true, the range includes the minimum value. Otherwise the range does not include the minimum value. Can be either a java.lang.Boolean or a java.lang.String (which is converted to a Boolean value; "true", "yes", or "t" [case-insensitive] converts to true, all others convert to false).
max	Object	The maximum date value. Can be either a java.lang.Number or a java.lang.String (which is converted to a Double). This value is not included in the range unless the includemax flag is set to *true*.

Name	Data Type	Description
includemax	Object	If the value is true, the range includes the maximum value. Otherwise the range does not include the maximum value. Can be either a java.lang.Boolean or a java.lang.String (which is converted to a Boolean value; "true", "yes", or "t" [case-insensitive] will convert to true, all others convert to false).

sys_ValidateRequiredField

Validates that field contains a value.

Class Name

com.percussion.validate.PSValidateRequired

Interface

com.percussion.extension.IPSFieldValidator

Context

global/percussion/content/

Parameters

Name	Data Type	Description
value	Object	The value to be validated; used without conversion, but is extracted if a replacement value

sys_ValidateStringLength

Validates that the length of the string value in the field falls within the specified range.

Class Name

com.percussion.validate.PSValidateStringLength

Interface

com.percussion.extension.IPSFieldValidator

Context

global/percussion/content/

Parameters

Name	Data Type	Description
value	Object	The value to be validated; is extracted if a replacement value

Name	Data Type	Description
min	Object	The minimum length for the string. Can be either a java.lang.Number or a java.lang.String (which is converted to a Double).
max	Object	The maximum length for the string. Can be either a java.lang.Number or a java.lang.String (which is converted to a Double).

sys_ValidateStringPattern

Validates that the value of the field matches the regular expression.

Class Name

com.percussion.validate.PSValidateStringPattern

Interface

com.percussion.extension.IPSFieldValidator

Context

global/percussion/content/

Parameters

Name	Data Type	Description
value	String	The value to be validated; should be a string or a replacement value.
regex	String	The string pattern to be compared with the value.

Field Transformers

When a Content Item is submitted to the Repository, Field Transformers are processed after Field Validations and before Item Transforms. Field Transformers transform either the format or the content of data in a Content Item field. Field Transformers fall into two categories:

- Field Input Transforms change the format or content of data before it is entered into the Rhythmyx Repository.
- Field Output Transforms change the format or content of data after it is retrieved from the Rhythmyx Repository before it is rendered in a Content Editor or assembled into a Content Item.

Field transformers operate only on a specific field. If you need to manipulate multiple fields (for example concatenating the values in multiple fields to generate the value updated to the Repository), you must use an Item transformer.

Once all field transformers have been run, Item transformers are run.

Field input transformers must implement the interface IPSFieldInputTransformer; field output transformers must implement the interface IPSFieldOutputTransformer. (NOTE: The implementation must be thread safe; for details see ***General Requirements of Extensions*** on page 180.)

Note that the Rhythmyx Workbench includes a number of built-in Field Transformer functions. In the server, these functions are implemented as Field Transformer extensions.

Input Transformers

sys_MapInputValue

Maps an input value to a specified set of keys and values. Note that this extension has a defined interface in the Workbench (when the Map option is selected for an input transformer.

Class Name

com.percussion.extensions.translations.PSMapInputValue

Interface

com.percussion.extension.IPSFieldInputTransformer

Context

global/percussion/content/

Parameters

Name	Data Type	Description
value	String	The value to be transformed.
map	Object	A map of values encoded as a URL query string. The name/value pairs are separated by ampersands; within each pair, the name is separated from the value by an equal sign. The actual name and value are URL encoded.

sys_NormalizeDate

Normalizes input date to ISO standard format.

Class Name

com.percussion.validate.PSNormalizeDate

Interface

com.percussion.extension.IPSFieldInputTransformer

Context
global/percussion/content/

Parameters

Name	Data Type	Description
value	String	The value to be transformed.
format	String	Simple date format template of the input value.

sys_OverrideLiteral

Context
Java/global/percussion/generic/

Description
This UDF converts the supplied 'default' parameter to a String and returns either that string or the value of the overrideParameterName HTML parameter. If the HTML request includes this parameter, it is removed from the request after it is used.

Class Name
com.percussion.extensions.general.PSSimpleJavaUdf_overrideLiteral

Interface
com.percussion.extension.IPSUdfProcessor

Parameters

Name	Data Type	Description
default	java.lang.String	The default object, which is returned as a string. Required.
overrideParameterName	java.lang.String	The name of the HTML parameter that stores the value which which to override the default value. Optional.

sys_Replace

Context:
Java/global/percussion/generic/

Description:
Replaces each occurrence of search in source with replacement.

Class name:

com.percussion.extensions.general.PSSimpleJavaUdf_replace

Interface:

com.percussion.extension.IPSUdfProcessor

Parameters:

Name	Data Type	Description
source	java.lang.String	The original string.
search	java.lang.String	The substring for which the exit searches.
replacement	java.lang.String	The replacement value for the search string.

sys_ToHash

Context:

Java/global/percussion/generic/

Description:

Converts supplied parameters to a hashcode by concatenating them with a delimiter.

Class name:

com.percussion.extensions.general.PSSimpleJavaUdf_toHash

Interface:

com.percussion.extension.IPSUdfProcessor, com.percussion.extension.IPSFieldInputTransformer, com.percussion.extension.IPSFieldOuputTransformer

Parameters:

Name	Data Type	Description
source1	java.lang.String	First string to include in the hash code.
source2	java.lang.String	Second string to include in the hash code
source3	java.lang.String	Third string to include in the hash code
source4	java.lang.String	Fourth string to include in the hash code

sys_ToLowerCase

Context:
Java/global/percussion/generic/

Description:
This exit converts a UDF-supplied string to lower case.

Class name:
com.percussion.extensions.general.PSSimpleJavaUdf_toLowerCase

Interface:
com.percussion.extension.IPSUdfProcessor

Parameters:

Name	Data Type	Description
source	java.lang.String	The string to convert.

sys_ToProperCase

Context:
Java/global/percussion/generic/

Description:
This exit capitalizes the first character of every word in the UDF-supplied string.

Class name:
com.percussion.extensions.general.PSSimpleJavaUdf_toProperCase

Interface:
com.percussion.extension.IPSUdfProcessor

Parameters:

Name	Data Type	Description
source	java.lang.String	The string to convert.

sys_ToUpperCase

Context:
Java/global/percussion/generic/

Description:
This exit converts each character in the UDF-supplied string to upper case.

Class name:
com.percussion.extensions.general.PSSimpleJavaUdf_toUpperCase

Interface:
com.percussion.extension.IPSUdfProcessor

Parameters

Name	Data Type	Description
source	java.lang.String	The string to convert.

sys_TranslateJexlExpressionValue
Evaluates a Java Expression Language (JEXL) expression and outputs the result for update to the Repository. To use the input value of the field, use the variable $value, which is bound to the value of the value parameter.

Class Name
com.percussion.extensions.translations.PSJexlInputTranslation

Interface
com.percussion.extension.IPSFieldInputTransformer

Context
global/percussion/content/

Parameters

Name	Data Type	Description
value	Object	The value to be transformed.
expression	String	The JEXL expression to evaluate. To use the input value of the field, use the variable $value, which is bound to the value of the value parameter.

sys_Trim

Context:

Java/global/percussion/generic/

Description:

This exit strips leading and trailing white space from the supplied string. It does this by calling toString() on the supplied object.

Class name:

com.percussion.extensions.general.PSStringTrimmerUdf

Interface:

com.percussion.extension.IPSUdfProcessor

Parameters:

Name	Data Type	Description
source	java.lang.String	The string to trim.

sys_TrimString

Trims whitespace in the input value. Whitespace can be trimmed before the input value, after, or both before and after.

Class Name

com.percussion.extensions.translations.PSTrimStringValue

Interface

com.percussion.extension.IPSFieldInputTransformer

Context

global/percussion/content/

Category String

translation

Parameters

Name	Data Type	Description
value	String	The value to be transformed.

Name	Data Type	Description
trim	String	Specifies how to trim the input value. Options are *start* (trims whitespace at the start of the input string), *end* (trims whitespace at the end of the input string, or *both* (default; trims whitespace both at the start and at the end of the input string).

Output Transformers

sys_DateFormat

Context:

Java/global/percussion/generic/

Description:

This exit formats the supplied date using the UDF-supplied pattern. Any Java SimpleDateFormat patterns (***http://java.sun.com/j2se/1.3/docs/api/java/text/SimpleDateFormat.html***) are acceptable. Before the exit runs, the user must define two objects through the GUI.

The first object is a string representing a desired format. The second object is a string representing a reference input date. The date string should be in a format recognizable by the Rhythmyx server's PSDataConverter, otherwise the exit throws an exception that terminates the format procedure.

Class name:

com.percussion.extensions.general.PSSimpleJavaUdf_dateFormat

Interface:

com.percussion.extension.IPSUdfProcessor

Parameters

Name	Data Type	Description
pattern	java.lang.String	The format pattern
date	java.util.Date	The date to format
returnNullForEmpty	java.lang.String	Defines the behavior if the value of the column is empty. Valid values are *true* and *false*; default is *false*. If the value of this parameter is *true*, a null is inserted into the XML document if the database column is empty. If the value of this parameter is *false*, the current date is inserted into the XML document if the database column is empty. If this parameter has any other value, or if the value is not specified, the system assumes the default value of *false*.

sys_DateFormatEx

Context:

Java/global/percussion/generic/

Description:

This exit formats the supplied date according to a user-supplied input pattern and saves it as a string using the supplied output pattern. Any Java ***SimpleDateFormat patterns*** (http://java.sun.com/j2se/1.3/docs/api/java/text/SimpleDateFormat.html) are acceptable for the input and output patterns.

Class name:

com.percussion.extensions.general.PSSimpleJavaUdf_dateFormatEx

Interface:

com.percussion.extension.IPSUdfProcessor

Parameters:

Name	Data Type	Description
outputPattern	java.lang.Object	Optional. The output format pattern. If not provided, the exit uses the default: yyyy/mm/dd hh:mm:ss
date	java.lang.Object	Optional. The date to format. The function accepts java.util.Date and java.lang.String types. If not provided, the exit uses the default of current date and time.
inputPattern	java.lang.Object	Optional. The input format pattern of the provided date. If not provided, the exit tries to find the input pattern.
returnNullForEmpty	java.lang.String	Defines the behavior if the value of the column is empty. Valid values are *true* and *false*; default is *false*. If the value of this parameter is *true*, a null is inserted into the XML document if the database column is empty. If the value of this parameter is *false*, the current date is inserted into the XML document if the database column is empty. If this parameter has any other value, or if the value is not specified, the system assumes the default value of *false*.

sys_FormatDate

Converts a date field value output from the Repository from ISO standard to another format.

Class Name

com.percussion.extensions.translations.PSFormatDate

Interface

com.percussion.extension.IPSFieldOutputTransformer

Context

global/percussion/content/

Parameters

Name	Data Type	Description
value	String	The value to be transformed.
format	String	Simple date format template to which to convert the date extracted from the Repository.

sys_MapOutputValue

Maps an input value to a specified set of keys and values. Note that this extension has a defined interface in the Workbench (when the Map option is selected for an input transformer.

Class Name

com.percussion.extensions.translations.PSMapOutputValue

Interface

com.percussion.extension.IPSFieldOutputTransformer

Context

global/percussion/content/

Parameters

Name	Data Type	Description
value	String	The value to be transformed.
map	Object	A map of values encoded as a URL query string. The name/value pairs are separated by ampersands; within each pair, the name is separated from the value by an equal sign. The actual name and value are URL encoded.

Item Transformers

When a Content Item is submitted to the Repository, Item Transformers are processed after Field Transformers but before generic post-processors. Item Transformers transform data in multiple fields, such as combining the values in two or more fields to generate the value updated to the Repository. Item transformers fall into two categories:

- Item Input Transformers change the format or content of data before it is entered into the Rhythmyx Repository.
- Item Output Transformers change the format or content of data after it is retrieved from the Rhythmyx Repository and before it is rendered in a Content Editor or assembled into a Content Item.

Item input transformers must implement the interface IPSItemInputTransformer; Item output transformers must implement the interface IPSItemOutputTransformer. (NOTE: The implementation must be thread safe; for details see ***General Requirements of Extensions*** on page 180.)

Note that no default Item output transformers are shipped with Rhythmyx.

Document Pre-processors

Document pre-processors are run when a Content Item is created or retrieved by Rhythmyx or before a lookup document is generated. Document pre-processors can be used to modify any parameter submitted with the request or to generate a default value.

NOTE: In common usage, document pre-processors are often referred to as "pre-exits", or collectively with result document processors simply as "exits".

Document pre-processors must implement the interface IPSRequestPreprocessor. (NOTE: The implementation must be thread safe; for details see ***General Requirements of Extensions*** on page 180.)

Rhythmyx is shipped with a number of legacy document preprocessor extensions for backwards-compatibility. For details, see ***Legacy Extension Reference*** on page 190.

Result Document Processors

Result document processors are run when a Content Item is submitted to the Repository or after a lookup document has been generated.

NOTE: In common usage, result document processors are often referred to as "post-exits", or collectively with document pre-processors simply as "exits".

Result document processors must implement the interface IPSResultDocumentProcessor. (NOTE: The implementation must be thread safe; for details see ***General Requirements of Extensions*** on page 180.)

Rhythmyx is shipped with a number of legacy result document processor extensions for backwards-compatibility. For details, see ***Legacy Extension Reference*** on page 190.

Writing Content Editor Extensions

Most of the manipulations performed on Content Item data by a Content Editor extension are equivalent to standard actions performed in a Content Editor. This section describes the code that you can use to perform these actions.

Complex child content must be handled separately from the parent Content Item. Therefore, this section has two sub-sections, one devoted to manipulation of Content Items and one devoted to manipulation of child content.

In the interest of clarity, error checking has been omitted from the code examples in this section. In most cases, an exception will be returned if the object you request does not exist, but in some cases null values or empty sets are returned instead. Review the JavaDoc of the cited methods for details about which methods throw exceptions and which return null or empty values.

In some cases, default or hard-coded values have been provided for parameters. The JavaDoc describes these parameters in detail.

Basic Editing Operations

This section illustrates simple examples of common operations on Content Items. Note that all methods illustrated operate on lists of Content Items, while the simple examples in the code operate on one example at a time. It is more efficient to build a list of Content Items and perform the operation on the whole list rather than operating individually on each Content Item.

Creating New Content Items

Use `IPSContentWs.createItems()` to create a new Content Item:

```
public static PSCoreItem createBrief(String session, String user)
      throws PSUnknownContentTypeException, PSErrorException
      {
         initServices();
         String typeName = "rffBrief";
         List<PSCoreItem> items = cws.createItems(typeName, 1, session,
user);
         return items.get(0);
      }
```

New Content Items do not have a Content ID until they are saved by calling the `saveItems()` method. Thus, you cannot create complex child items or Relationships of any kind until the new Content Items have been saved.

A new Content Item always has a Revision of "1".

When a new Content Item is created, it contains all local and shared fields defined for the Content Type. If a default value is specified for any fields, the field will contain that default value.

Loading Existing Content Items

To load existing Content Items, use `IPSContentWs.loadItems()`. In most cases, before loading Content Items, you will want to call `IPSContentWs.prepareForEdit()` on those Content Items first.

```
public static PSCoreItem loadItem(String contentId, String session,
String user)
      throws PSErrorResultsException
```

```
{
      initServices();
      IPSGuid cid = gmgr.makeGuid(new PSLocator(contentId));
      List<IPSGuid> glist = Collections.<IPSGuid>singletonList(cid);
      List<PSItemStatus> statusList = cws.prepareForEdit(glist, user);
      List<PSCoreItem> items = cws.loadItems(glist, true, false, false,
false, session, user);
      return items.get(0);
}
```

Note that in this fragment, the binary fields are loaded, but child content, related Content Items, and Folder paths are not loaded.

Managing Revisions

In the example code for loading existing Content Items, they were loaded without specifying a Revision. Once the Content Items are loaded, the Revision can be obtained and a new GUID created for other purposes:

```
int contentid = item.getContentId();
int revision = item.getRevision();
IPSGuid guid = gmgr.makeGuid(new PSLocator(contentid, revision));
```

On occasion, you may need to know the current or edit locator of a Content Item you do not have loaded. Use `PSComponentSummary` to access this data. The PSO toolkit provides a convenient static method for accessing PSComponentSummary.

```
PSComponentSummary summ = PSOItemSummaryFinder.getSummary(contentid);
PSLocator loc = PSOItemSummaryFinder.getCurrentOrEditLocator(contentid);
```

Use the `getSummary()` method to accessing information about the Content Item (such as who has it checked out). The `getCurrentOrEditLocator()` method returns the edit locator if it exists; otherwise, it returns the current locator.

Manipulating Fields

When Content Items are loaded or created, all of the defined fields are populated, except for binary fields, which are optional.

Fields can be loaded using the method `PSCoreItem.getFieldByName`

```
PSItemField title = item.getFieldByName("sys_title");
String label = RxItemUtils.getFieldValue(item, "display_title");
Date someDate = RxItemUtils.getFieldDate(item, "some_date");
```

The PSO toolkit class `RxItemUtils` provides convenient static methods for retrieving and setting fields by name:

```
RxItemUtils.setFieldValue(item, "myfield", "some value");
RxItemUtils.setFieldValue(item, "some_date", new Date());
```

Binary fields can handled in a similar fashion:

```
InputStream istream = new ByteArrayInputStream(buf);
RxItemUtils.setFieldValue(item, "binary_field", istream);
```

Saving Content Items

To save Content Items, use `IPSContentWs.saveItem`:

```
public static IPSGuid simpleSave(PSCoreItem item, String session, String
user) throws PSErrorResultsException
{
```

```
      List<PSCoreItem> items =
Collections.<PSCoreItem>singletonList(item);
      List<IPSGuid> guids = cws.saveItems(items, false, false,
session,user);
      return guids.get(0);
}
```

If any Content Items were opened using a sequence of `prepareForEdit()` followed by `loadItems()` as described earlier, they must be released . If the Content Item is new, simply checking it in is usually enough to release from edit.

```
public static void releaseItem(IPSGuid guid, PSItemStatus itemStatus,
String user) throws PSErrorsException
{
      if(itemStatus != null)
      { //we got it from prepareForEdit
         List<PSItemStatus> stats =
         Collections.<PSItemStatus>singletonList(itemStatus);
         cws.releaseFromEdit(stats, false);
      }
      else
      { //a new item, just check it in.
         List<IPSGuid> glist = Collections.<IPSGuid>singletonList(guid);
         cws.checkinItems(glist, "no comment", user);
      }
}
```

Note, however, that releasing a Content Item from edit can occur much later than the save of the Content Item.

Editing Complex Child Data

Complex child data is manipulated separately from the parent Content Item. Complex children require a valid parent Content ID, so child entries cannot be created until the parent Content Item has been saved (which creates the Content ID for the parent). A Content Type may have several different complex child field sets. Each of these field sets has a unique name defined in the parent Content Type. The name is used to retrieve child content records.

Rhythmyx includes two different APIs for working with complex children: an "inner API" on the PSCoreItem itself and an "outer API on PSContentWs. You cannot mix the two APIs; for example you cannot call loadItems with the includeChildren flag set to true, then use IPSContentWs.saveChildEntries to save the children.

The example code provided in the following topics uses the "outer API". There is no substantial performance difference between the two APIs, but the outer API is generally simpler for programmers new to Rhythmyx.

Creating New Child Entries

Use the method `IPSContentWs.createChildEntries()` to create new child entries.

```
if(newChild)
{
      List<PSItemChildEntry> newEntries =cws.createChildEntries(item,
childName, 1, session, user);
      // update the child fields here
      RxItemUtils.setFieldValue(newEntries.get(0), "child_field","some
textValue");
```

```
        toBeSaved.addAll(newEntries);
    }
```

As with Content Items, the child entries are not persisted to the Repository at this time; the entries must be saved. The newly created entries can be modified just as other fields.

Loading Existing Child Entries

To load existing child entries, use the method `IPSContetnWs.loadChildEntries()`.

```
List<PSItemChildEntry> children = cws.loadChildEntries(item,
childName,false, session, user);
```

These entries can be modified in the same fashion as field son the parent Content Item.

Modifying Child Fields

Child fields are read and modified in the same way that fields are read and manipulated on the parent Content Item. The fields have values and you can use the same convenience methods on child fields as are used on fields in the parent Content Item.

```
PSItemField field = child.getFieldByName("child_field");
RxItemUtils.setFieldValue(child, "child_field", "some new value");
```

Saving Child Entries

After child entries have been modified, changes can be saved using the method `IPSContentWs.saveChildEntries`

```
PSItemField field = child.getFieldByName("child_field");
RxItemUtils.setFieldValue(child, "child_field", "some new value");
```

Some child field sets support ordering (the field set is defined with a sortrank field). The `saveChildEntries` method saves the entries, but does not set the order. The entries must be re-ordered after being saved.

```
List<IPSGuid> toBeOrdered = new ArrayList<IPSGuid>();
for(PSItemChildEntry childEntry : children)
{
      toBeOrdered.add(childEntry.getGUID());
}
cws.reorderChildEntries(item, childName, toBeOrdered, session, user);
```

Note that the re-order list is a list of GUIDs, not a list of `PSItemChildEntry`.

Removing Child Entries

To remove child entries, use IPSContentWs.deleteChildEntries. Note that the list of entries to delete is a list of GUIDs, not a list of PSChildEntry objects.

```
List<IPSGuid> toBeDeleted = new ArrayList<IPSGuid>();
toBeDeleted.add(child.getGUID());
cws.deleteChildEntries(item, childName, toBeDeleted, session, user);
```

Content Editor Control Reference

As with any interface, Content Editors require controls that allow the user to interact with the data. Rhythmyx Content Editors use a set of controls similar to those commonly found on HTML pages. In Rhythmyx, these controls are defined using XSL and JavaScript.

Writing Custom Controls

Content Editors use a set of controls defined in XSL stylesheets.

Standard controls are defined in `<CMServerroot>/ sys_resources /Stylesheets/sys_Templates.xsl`.Customers should not modify controls defined in this file as it is overwritten on upgrade and modifications will be lost. Customers should not modify controls defined in this file as it is overwritten on upgrade and modifications will be lost. All controls provided by Percussion Software begin with the string "`sys_`"; for example, `sys_DatePicker`. User-developed controls should not begin with this prefix.

Custom controls can be defined in two ways.

Custom controls that can be included in a package must be defined in XSL files in the directory `<CMServerroot>/rx_resources/stylesheets/controls`. Each custom control must have a unique XSL file, and the XSL file should have the same name as the control; for example, if you wanted to make a new calendar control named rff_newCalendar, the XSL file containing the control would be rff_newCalendar.xsl.

Custom controls that will be included in a package can be defined in the file `<CMServerroot>/rx_resources /Stylesheets/rx_Templates.xsl`. If you want to package a control that is already defined in this file, you must re-implement it as described in the previous paragraph.

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.xsl` or `rx_template.xsl` 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 describe the file and specify its location.

Any extensions 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 mode "psxcontrol". 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 `datacell1` and `datacell2` 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 be an 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
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>
```

Customizing Controls

Certain Rhythmyx controls allow customization short of implementing a new control. Controls that allow customization include:

- sys_EditLive
- sys_WebImageFX

Customizing the EditLive! for Java Editor

You can customize both the parameters of the sys_EditLive control and the configuration files of the EditLive! for Java 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 its configuration file (`elj_config.xml`) as well as other properties. You can customize these parameters in the control definition (in the Control Properties dialog accessible from the Rhythmyx Workbench). 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.

See ***sys_EditLive Control*** (on page 52) for a list of the parameters that you can customize and instructions on how to change them.

Customizing EditLive! for Java Configuration

The EditLive! for Java (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 for the sys_EditLive control 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 42) for instructions on adding custom javascript functions.

Several instances of the sys_EditLive 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 `rx_resources/ephox` or a subdirectory created under this directory. For example, if you have a local configuration file for a Press Release content editor, you might want to store the configuration file in the subdirectory `rx_resources/ephox/pressrelease`.

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

1. In the Rhythmyx Workbench, open the Content Type editor for the Content Type in which you want to use the ELJ editor.
2. Select the field in which you want to use the ELJ editor or add a new field.
3. In the Control field, choose *sys_EditLive*.
4. Click the browse button (**…**) next to the Control field.

   Rhythmyx displays the Control Properties dialog.
5. Click in the Name column and choose config_src_url.
6. Click in the Value column of the same row and enter the URL (relative to the Rhythmyx root) of the configuration file you want to use for this instance of the control as a literal value.
7. On the Control Properties dialog, click [**OK**].
8. Save the changes to the Content Type.

   To see your changes, log in to Rhythmyx, and activate the editor.

For guidance on customizing (and localizing) the ELJ editor, consult EditLive documentation at ***http://liveworks.ephox.com/documentation/editlive/v60/***.

Customizing the sys_EditLiveDynamic control

Note: This control is deprecated. Customers who installed Rhythmyx prior to Version 6.5.2 may have fields that use it.

The parameters of the sys_EditLiveDynamic control define the control for any field that uses it in a Content Editor. You can customize these parameters in the control definition (in the Control Properties dialog). We recommend first changing the control to the sys_EditLive control in the Rhythmyx Workbench, and then following the instructions in ***sys_EditLive Control*** (on page 52) to customize the parameters.

You can customize the configuration file for the sys_EditLiveDynamic control if you want to implement your own toolbars and menus. We recommend first changing the control to the sys_EditLive control in the Rhythmyx Workbench, and then following the instructions in ***Customizing EditLive! for Java Configuration*** (see page 40) and ***Adding Custom Menu and Toolbar Actions*** (on page 42) for help customizing your configuration file.

Adding Custom Menu and Toolbar Actions

Rhythmyx provides you with xml code that you can use to create custom actions for your sys_EditLive controls. 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.xsl, 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="raiseEvent"
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="raiseEvent"
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 in <Rhythmyx root>/sys_resources/ephox/ephox_developerguide.pdf**.**

Customizing the WebImageFX Editor

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***.

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/sys_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 initially stored in the directories `sys_resources/WebImageFX` `rx_resources/WebImageFX`. Do not modify the configuration file in the directory `sys_resources/WebImageFX`.
- 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 a 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.

Standard Rhythmyx Controls

Several 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 jscript. 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 the use of the parameter. A parameter may or may not include a default value.

sys_CalendarSimple

Figure 5: sys_CalendarSimple control

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 date 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

Parameter	Data Type	Parameter Type	Description	Default
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

sys_CheckBoxGroup

The sys_CheckBoxGroup control 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 6: 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
id	String	Generic	XHTML 1.0 attribute	None
class	String	Generic	XHTML 1.0 attribute	None
style	String	Generic	XHTML 1.0 attribute	None
columncount	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
columnwidth	String	Generic	Specifies the width of the column in pixels or percentage.	100%

sys_CheckBoxTree

This control renders a set of options as a "tree" of checkboxes that allows multiple boxes to be checked. This control does not include any validation to control which options the user can and cannot check.

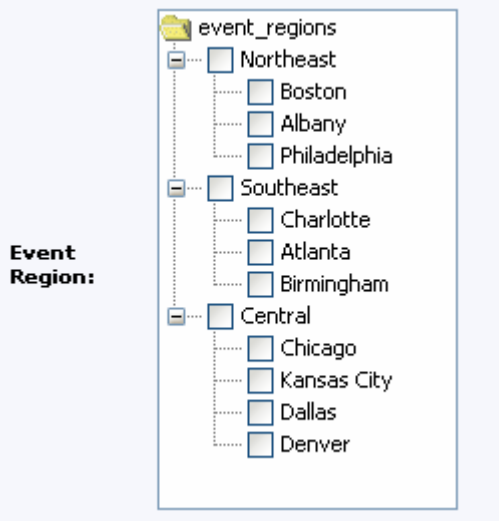


Figure 7: sys_CheckBoxTree control

The control is rendered using two XML files. The first file defines the structure of the tree and defines the choices. The following code illustrates a simple example of the "tree" XML:

```
<tree label="products">
```

```
        <node id="s1" label="generic products" selectable="no">
           <node id="prodx" label="product x" selectable="yes"/>
           <node id="prody" label="product y" selectable="yes"/>
        </node>
        <node id="s2" label="special products" selectable="no">
           <node id="s2a" label="extra special products" selectable="yes">
              <node id="prodq" label="product q" selectable="yes"/>
           </node>
           <node id="prodz" label="product z" selectable="yes"/>
        </node>
   </tree>
```

The `<node>` element has three attributes:

- `id` (required): specifies the value that will be stored in the Repository if the checkbox is checked.
- `label` (required): specifies the value that will be displayed when the Content Editor is rendered.
- `selectable` (optional): indicates whether the specified node can be checked or selected. The default value of "no" indicates that the node cannot be selected; a value of "yes" specifies that the node can be selected.

The second XML file is a dynamically-generated lookup used to support Content Editor validation. This file has the same structure as the "tree" XML. The value of the id attributes of the <node> elements must match. If these values do not match, validation will fail for nodes that do not match and no value will be saved for those nodes. The lookup file can be generated either from a keyword or dynamically using an internal lookup. For details on creating keywords, see "Creating Keywords" in the Rhythmyx Workbench Help. For details on creating an internal lookup, see ***Creating an Internal Lookup Query*** (on page 68).

Parameters

Parameter	Data Type	Parameter Type	Description	Default
width	String	XHTML 1.0 attribute	Specifies the width of the control, in either pixels or a percentage of available horizontal space.	400 (pixels)
height	String	XHTML 1.0 attribute	Specifies the height of the control, in either pixels or a percentage of available vertical space.	300 (pixels)
tree_src_url	String	XHTML 1.0 attribute	Specifies the relative location of the xml that defines the tree.	../rx_resources/treedef.xml
formname	String	XHTML 1.0 attribute	Internal parameter. Do not modify.	EditForm

sys_DropDownMultiple

The sys_DropDownMultiple is a combo box control that allows users to select multiple options from the list of potential values. Hold the <CTRL> key while clicking on values to select multiple values; hold the <SHIFT> key while selecting values to select the range between the selected values.

Figure 8: sys_DropDownMultiple 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	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

sys_DropDownSingle

The sys_DropDownSingle is a basic drop down 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. The dimension is *single*.

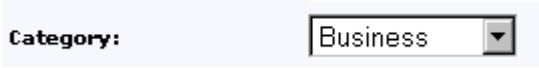


Figure 9: Example sys_DropDownSingle

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

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 10: 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

EditLive for Java Editor

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

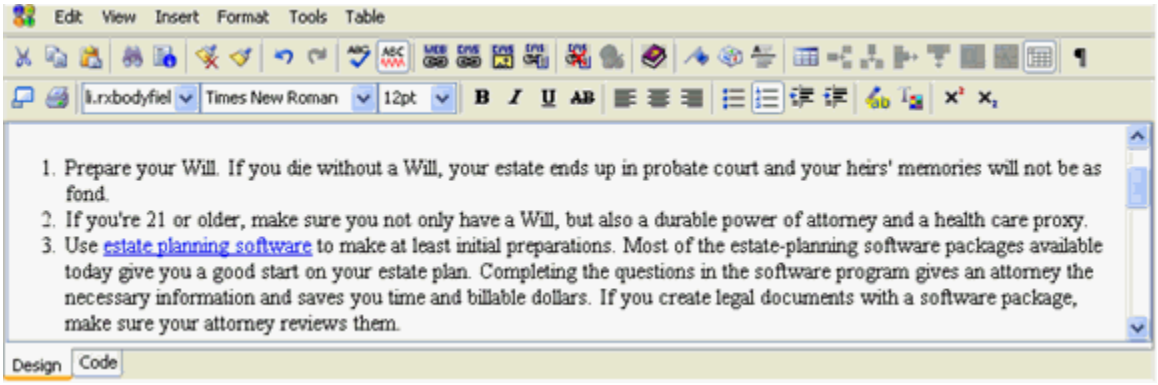


Figure 11: sys_EditLive Cotnrol

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

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.

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

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.

When more than one sys_EditLive controls are used in a Content Type, only one field in the content editor can access the control at a time. When a user accesses the control for one field, the control closes in the last field that displayed it.

The following graphic shows how the control works. When a user clicks the placeholder box for a field, the control is visible, and the user can enter data. When the placeholder box is not clicked for a field, the box displays the field's formatted text and graphics but they cannot be edited.

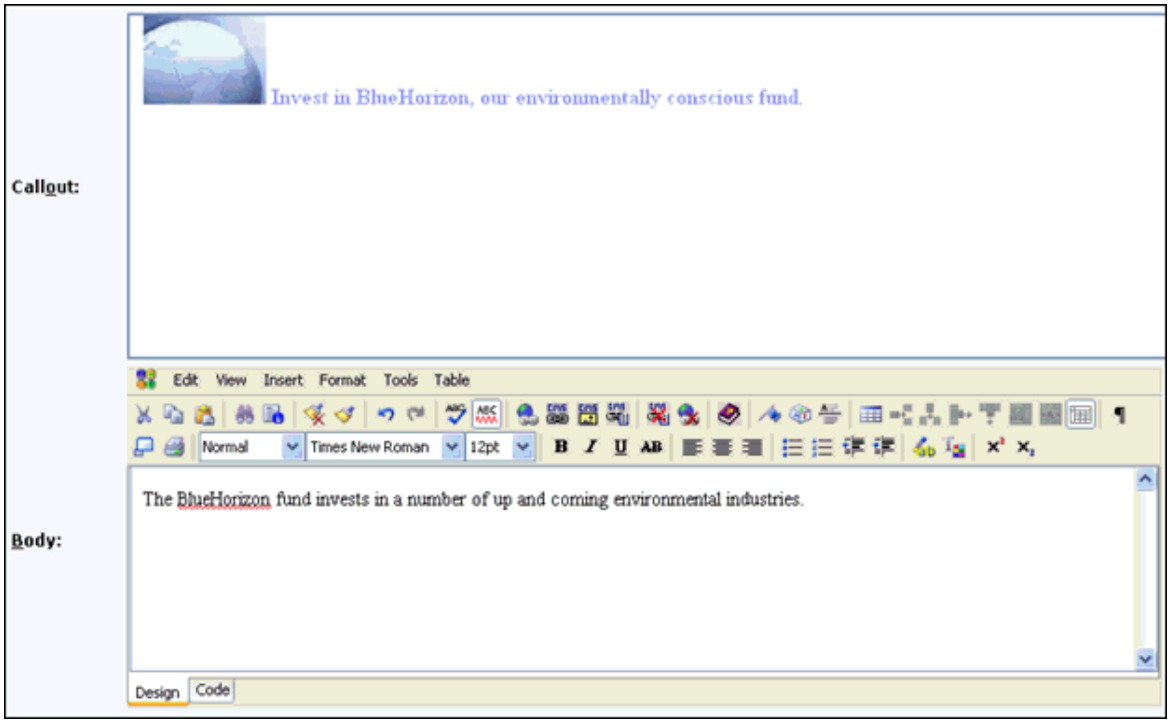


Figure 12: sys_EditLive control

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.xsl.

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 Template slot. The search dialog for the inline Template slot shows the content types that have a Template associated with the inline Template 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 Type field.

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

1. In the Rhythmyx Workbench, open the Content Type editor for the Content Type containing the field.
2. In the field, double-click on sys_EditLive to display the browse button (**…**) next to it.
3. Click the browse button (**…**).

   Rhythmyx displays the Control Properties dialog. Parameters that take their default values from the sys_templates.xsl file are not shown in the Parameters table, so the default table is empty.

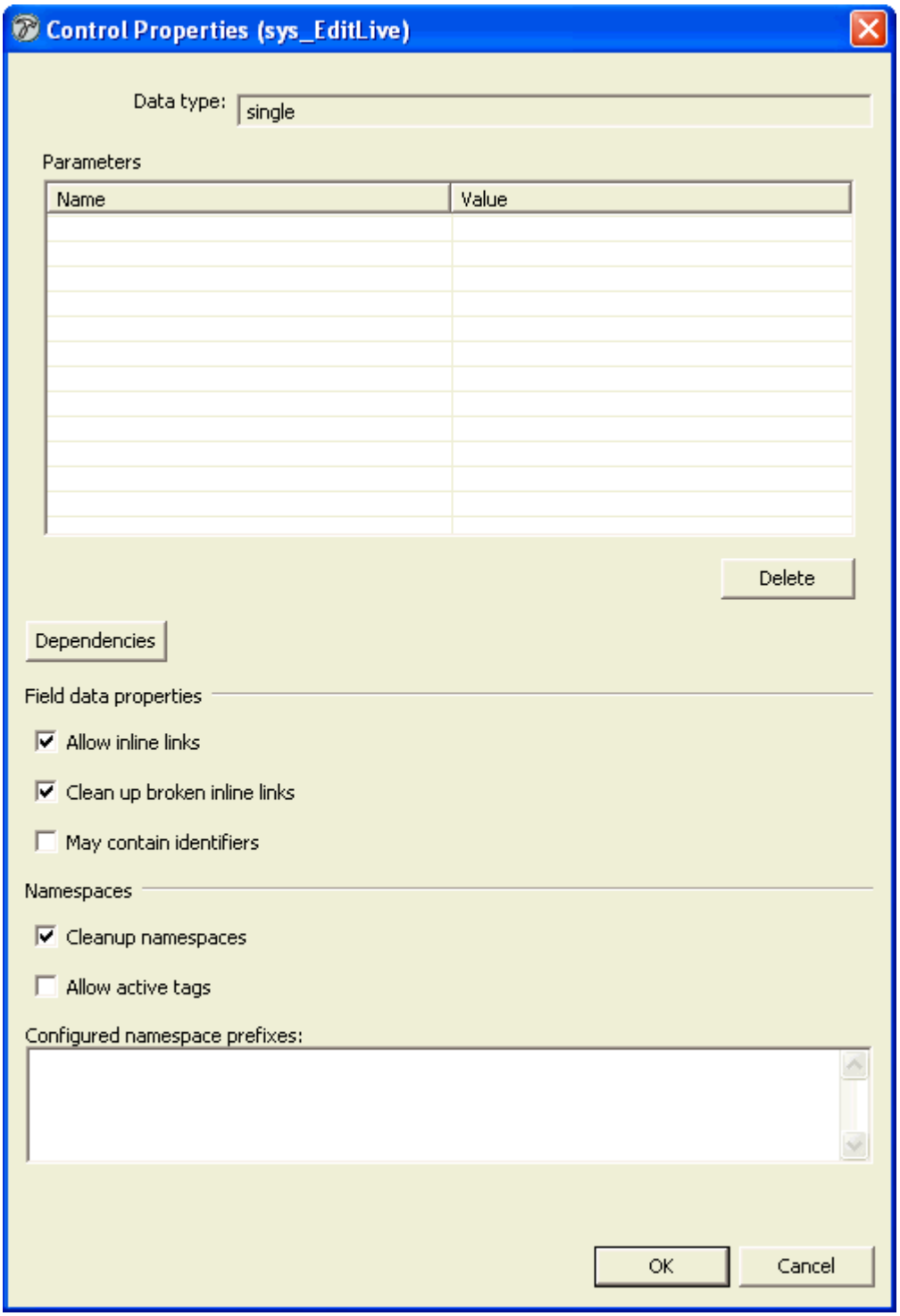


Figure 13: Default Control Properties dialog for sys_EditLive

4 Click in the Name column and choose the parameter whose value you want to change from the drop list.

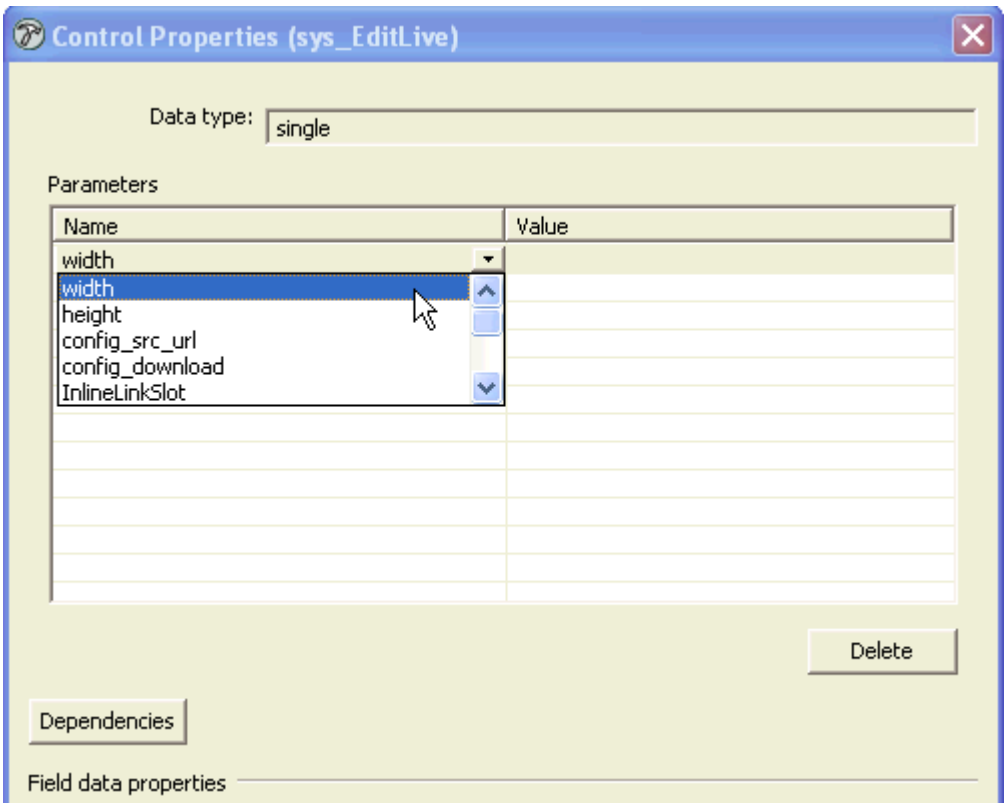


Figure 14: Changing a parameter value in the Control Properties dialog

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

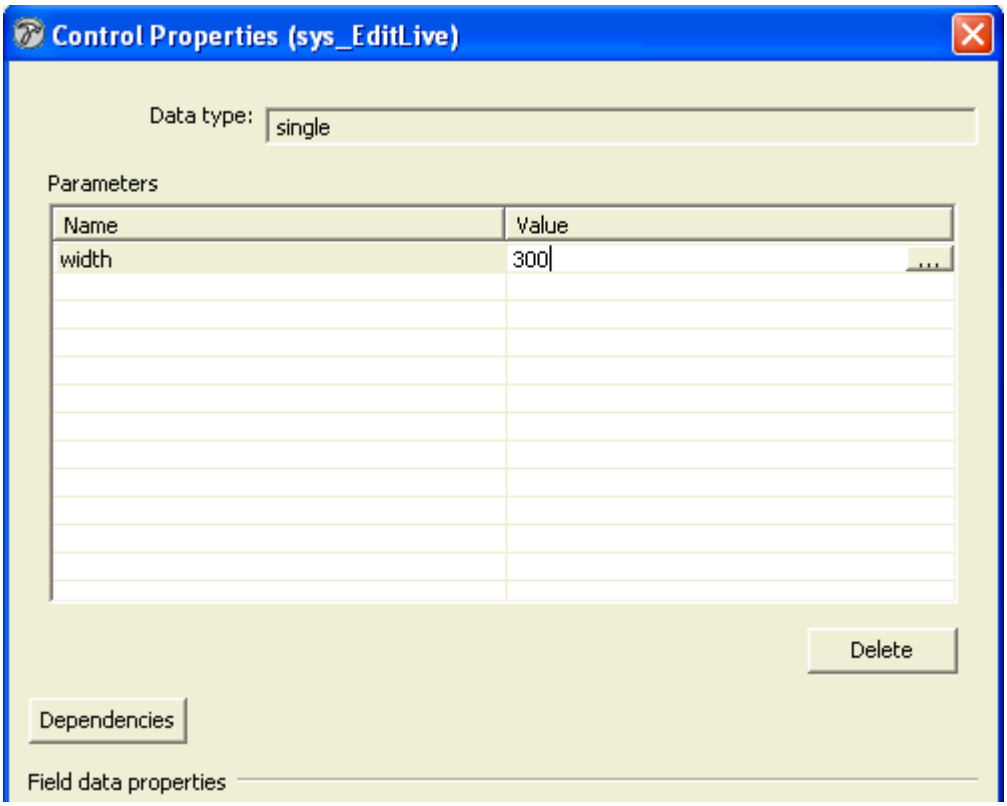


Figure 15: Parameter value changed in Control Properties dialog

6 On the Control Properties dialog, click [**OK**].

7 Save the changes to the Content Type.

8 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.

sys_EditLiveDynamic Control

Note: This control is deprecated. Customers who installed Rhythmyx prior to Version 6.5.2 may have fields that use it.

The sys_EditLiveDynamic control functions identically to the sys_EditLive control, except it offers fewer features. Originally, it was intended for Content Editors that experienced slow load times because they used multiple EditLive editors, but now the sys_EditLive control offers the same faster load time. Therefore, the sys_EditLiveDynamic control is now deprecated.

You can mix the two controls within one Content Editor. Some fields can use sys_EditLive and others can use sys_EditLiveDynamic.

The sys_EditLiveDynamic control includes the same parameters as those in the ***sys_EditLive control*** (see page 52). The default values, which are the same as those for sys_EditLive, are set in the file <Rhythmyx root>/sys_resources/stylesheets/sys_templates.xsl.

To change parameters for a Content Editor field that uses the sys_EditLiveDynamic control, we recommend first changing the control to the sys_EditLive control in the Rhythmyx Workbench and then following the ***instructions for changing any of the parameters in a sys_EditLive control*** (see "sys_EditLive Control" on page 52).

Adding the sys_EditLive Control to a Content Editor

To add the ***sys_EditLive control*** (see page 52) 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. Rhythmyx automatically adds the `sys_xdTextCleanup` exit as a dependency of the control, and automatically configures its required parameters.

Adding Form and Script Support to a sys_EditLive Control

If the field you plan to maintain using the sys_EditLive control will include form or script tags, you must add special processing to the control.

- Add the sys_EditLiveFormDecode Input Translation extension. The value of the Field name parameter should be PSXSingleHtmlParameter/fieldname, where fieldname is the name of the field to which the control is assigned.
- Add the sys_EditLiveFormEncode Output Translation extension. The value of the Field name parameter should be PSXSingleHtmlParameter/fieldname, where fieldname is the name of the field to which the control is assigned.

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 a 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 it in `Rhythmyxroot/rx_resources/ephox` or a subdirectory created under this directory. For example, if you have a local configuration file for a Press Release content editor, you might want to store the configuration file in the subdirectory `Rhythmyxroot/rx_resources/ephox/pressrelease`.

  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 from Rhythmyx 5.x to Rhythmyx 6.0 or higher, you receive the sys_EditLive control in addition to the sys_eWebEditPro control. Both options appear in the Control drop list in the Content Type 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 52).

To manually replace eWebEditPro with ELJ in a Content Type:

1. In the Rhythmyx Workbench, open the editor for the Content Type that you want to change.

   In the row for the Content Type field whose control you want to change, double-click in the control field to access the drop-list.

2. 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.

3. If you want to use a customized configuration file, or modify other parameters:

   c) Click the browse button next to the Control field.

   Rhythmyx displays the Control Properties dialog.

   d) Enter the parameters and associated values you want to assign to the control.

   e) Click [**OK**] to save your edits.

4. Save the changes to the Content Type.

   To see your changes, 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 them, when you attempt to save the Content Editor, the following error dialog appears:

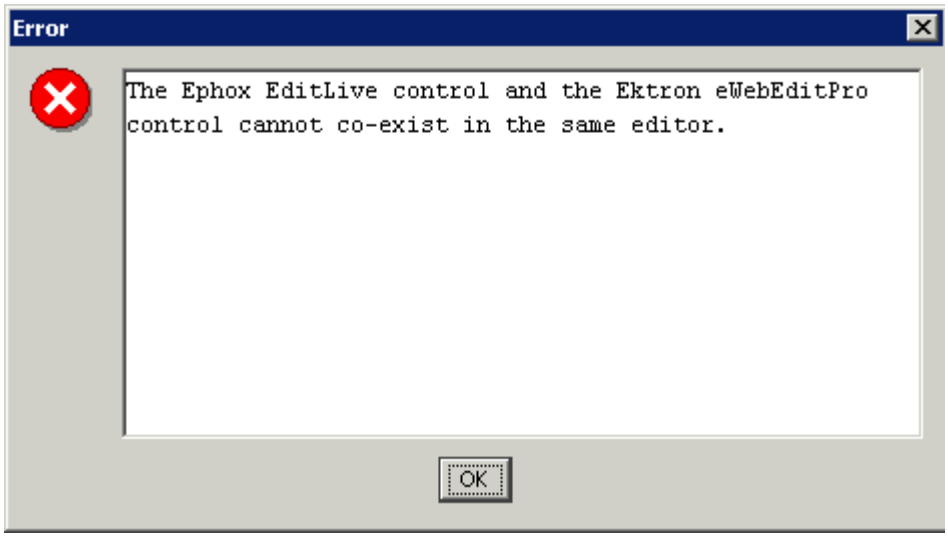


Figure 16: Warning when mixing controls

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

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 you add a sys_file control to a field in a content editor, Rhythmyx adds ***sys_FileInfo*** (on page 238) as a dependency of the Content Editor 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.

In order for the sys_File control to correctly upload a file to a content editor field, it must have access to the file's extension and mime type. The ***sys_FileInfo*** (on page 238) exit automatically extracts this information and stores it in fields that you have created in the content editor. To enable the sys_File control to use these fields, you must name them with the proper syntax. See ***sys_FileInfo*** (on page 238) for information about naming these fields.

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

Parameter	Data Type	Parameter Type	Description	Default
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

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	Sever 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

sys_RadioButtons

The sys_RadioButtons control displays a set of radio buttons that allow the user to select one of 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

Parameter	Data Type	Parameter Type	Description	Default
Disabled	String	Generic	If set, this boolean attribute disables the control for user input.	None

sys_SingleCheckBox

A single checkbox, used to denote boolean (true/false) values.

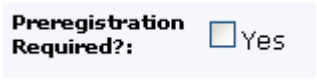


Figure 17: sys_SingleCheckBox control

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

sys_Table

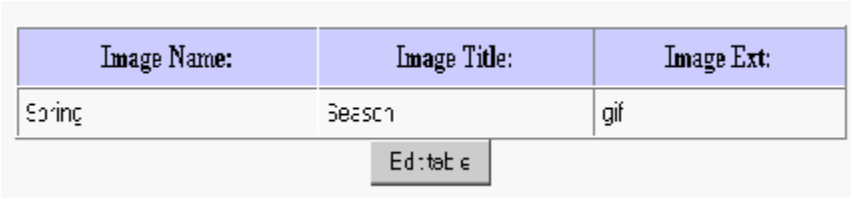


Figure 18: Example sys_Table

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

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

sys_TextArea

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

Figure 19: 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

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.

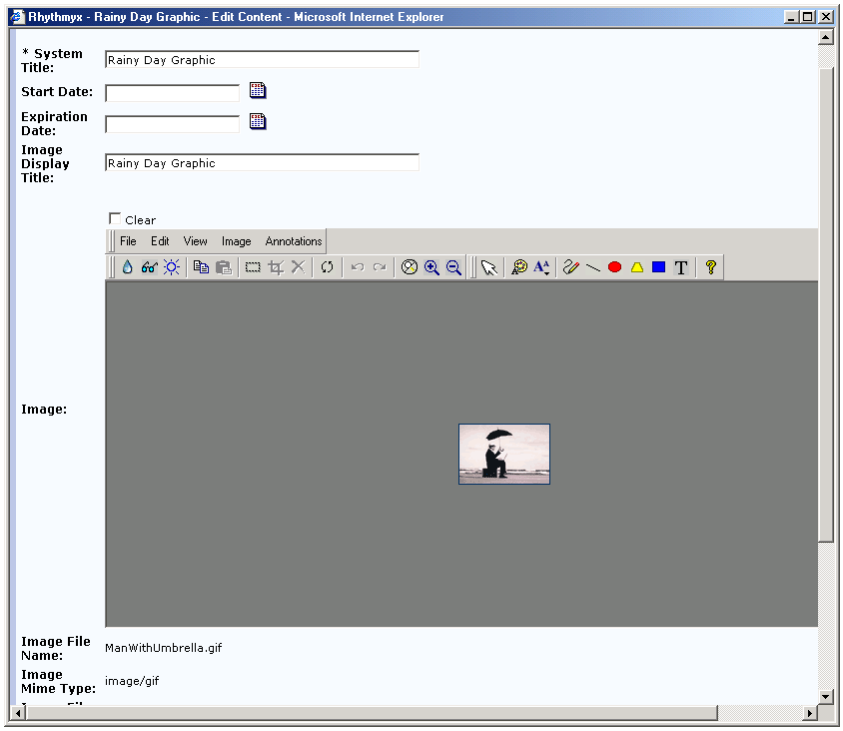


Figure 20: Content Editor with sys_WebImageFX control

sys_WebImageFX Control

The sys_WebImageFX control functions almost identically to the ***sys_File control*** (see "sys_File" on page 60). 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 you add a field that uses a sys_WebImageFX control to a Content Type, Rhythmyx adds the sys_FileInfo exit as a dependency 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.

In order for the sys_WebImageFX control to correctly upload a file to a content editor field, the field that holds the file must be named *uploadfilephoto*, and the control must have access to the file's mime type and filename. The ***sys_FileInfo*** (on page 238) exit automatically extracts this information and stores it in fields that you have created in the content editor. To enable the sys_File control to use these fields, you must name them with the proper syntax. See ***sys_FileInfo*** (on page 238) for information about naming these fields.

The sys_WebImageFX control displays a WebImageFX editor that allows a user to not only upload an image, but also to create or modify an image. For details about the standard features of WebImageFX, see the developer's guide.

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

Adding the sys_WebImageFX Control to a Content Editor

To create a Content Editor that uses WebImageFX:

1. Follow the procedure in the document *Rhythmyx Implementation Guide* 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 the following fields for storing the filename and mime type. You must use the exact names specified.
   - *uploadfilephoto_filename*
   - *uploadfilephoto_type*

   For each of these fields, do the following:

   a) Click [**All Properties**].

      The Field Properties dialog opens.

   b) Click [**Read Only**].

      The Field Editability dialog opens.

   c) In Rule Type, choose *Always*.

   d) Click [**Add**].

      The rule is added to the Rules Table.

   e) Click [**OK**].

      The Field Editability dialog closes.

   f) Click [**OK**].

      The Field Properties dialog closes.
4. In Mime type mode, choose From *Mime Type Field*.
5. In Mime type value, choose *uploadfilephoto_type*.
6. Add any of the other fields that sys_FileInfo extracts to the Content Editor. Always use the prefix *uploadfilephoto*. See ***sys_FileInfo*** (on page 238) for other required naming conventions for these fields.
7. Complete the standard procedure for creating the Content Editor.

The following limitations apply to all Content Types 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 Type 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 Type 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.

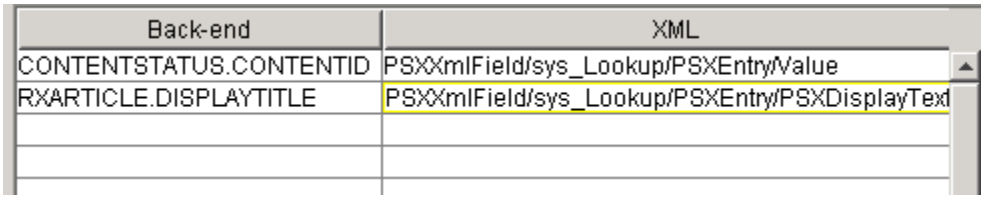
NOTE: The first time you open a Content Editor that uses the sys_WebImageFX control in your Web browser, a dialog will prompt you to install WebImageFX. Follow the installation instructions in the dialog. After you initially install WebImageFX, you will not have to install it again.

Creating an Internal Lookup Query

When you use the sys_DropDownSingle, sys_CheckBoxGroup, and sys_RadioButtons controls, or any custom controls that require a list of entries, you may choose to derive the choices for the control from an existing Rhythmyx table using a query. You should create a separate Rhythmyx application for these internal lookup queries.

To create an internal lookup query:

1. In the Rhythmyx Workbench, go to the XML Server view and create a new application. For details about creating a new application, see the topic "Creating a New Application" in the Rhythmyx Workbench Help.
2. Drag the `Rhythmyx/DTD/sys_Lookup.dtd` file onto the application.
3. From the popup menu, choose *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.



7. Optionally, add a Result Pager to sort the list results.
8. Right-click the sys_Lookup XML resource and choose *Request Properties*.
9. Change the name of the sys_Lookup.XML resource to the custom name you want to use to a and click [**OK**].
10. Save and close the application.

11 Go to the Content view and open the Content Type where you want to use this lookup.

12 Open the Control Properties dialog and on the Choices tab, specify the lookup resource you just created.

13 Click the [**OK**] button to close the Control Properties dialog.

14 Save the Content Type.

Content Editor System Definition Reference

The following table describes the fields defined in the Content Editor System Definition that are eligible to be included in Content Editors. By default, all of these fields are defined with the following property values:

Treat data as binary: No

Show in Preview: Yes

Allow this field to be searched: Yes

Name	Label	Mandatory	Comments
sys_communityid	Community Id	Yes	Defined when the Content Item is created and never modified afterwards. By default, value is derived from the currently logged Community of the user that creates the Content Item. Hidden by default. If visible, options include all Communities defined in the system.
sys_contentexpirydate	Content expiration date	No	
sys_contentstartdate	Content start date	Yes	Date format is yyyy-MM-dd
sys_currentview	(None)	Yes	Hidden Input.
sys_hibernateVersion	(None)	Yes	Hidden Input. Field only used internally, but must be included on all Content Editors.
sys_lang	Locale ID	Yes	Defined when the Content Item is created and never modified afterwards. By default, value is derived from the currently logged Locale of the user that creates the Content Item. Hidden by default. If visible, options include all Locales defined in the system.
sys_pathname	Path name	No	
sys_pubdate	Publication date	No	
sys_reminderdate	Reminder date	No	
sys_suffix	Suffix	No	Defaults to ".html". Hidden by default.
sys_title	System title	Yes	This field cannot be empty and must be unique within the folder.

Name	Label	Mandatory	Comments
sys_workflowid	Workflow	Yes	Hidden by default.

The next table describes fields defined in the Content Editor System Definition that are not eligible to be included in Content Editors. These fields are used mostly for processing of Content Items or to provide human-readable information for ID fields defined in the system definition. The value of some of these fields is computed at runtime. Those fields are not eligible to be searched, but, like all fields in the system definition, can be included in Display Formats.

Name	Label	Searchable	Comments
sys_assignees	Assignees	No	Computed.
sys_assignmenttype	Assignment type	No	Computed. Valid values include: ▪ None ▪ Reader ▪ Assignee ▪ Admin
sys_assignmenttypeid	Assignment type ID	No	Computed.
sys_checkoutstatus	Checkout status	No	Computed.
sys_communityname	Community Name	Yes	
sys_contentcreatedby	Created by	Yes	Defined when the Content Item is created and never modified afterwards
sys_contentcreateddate	Created on	Yes	Defined when the Content Item is created and never modified afterwards.
sys_contentcheckoutusername	Checked out user name	Yes	
sys_contentid	Content id	Yes	Defined when the Content Item is created and never modified afterwards.
sys_contentlastmodifieddate	Last modified date	Yes	
sys_contentlastmodifier	Last modified by	Yes	
sys_contentstateid	Workflow State ID	Yes	
sys_contenttypeid	Content Type	Yes	Defined when the Content Item is created and never modified afterwards.
sys_contenttypename	Content Type Name	Yes	
sys_folderid	Folder Path	Yes	
sys_localename	Locale Name	Yes	

Name	Label	Searchable	Comments
sys_objecttype	Object type	No	Defined when the Content Item is created and never modified afterwards.
sys_publishabletype	Publishable status	No	Computed
sys_relevancy	Rank	Yes	This field is used to provide the relevancy ranking returned by the external search engine. The field value is overwritten by the search engine at the time the search results are processed. If no rank is available, or if the search was performed against the internal engine, the value is left at -1.
sys_siteid	Site	Yes	
sys_statename	Workflow State Name	Yes	
sys_thumbnail	Thumbnail	Yes	
sys_variantid	Variant	Yes	
sys_variantname	Variant Name	Yes	
sys_workflowname	Workflow Name	Yes	

Search Reference

This section explains what ***search indexing*** (see page 73) is, and how Rhythmyx performs indexing, as well as defining some of the specialized plugins that Rhythmyx uses to index search terms.

Rhythmyx uses text extractors and text analyzers to perform search engine indexing. This section explains the purpose of each type of java plugin, describes what Rhythmyx provides out of the box, and discusses how administrators can override the out of the box plugins.

- ***Text Extractors*** (see below)
- ***Text Analyzers*** (see page 74)

Search Indexing

Search indexing in Rhythmyx is the process of extracting text from fields in content items, parsing the text for search terms, and storing the search terms in files. Search indexing occurs when a content item is created or when an administrator enters a console command to perform indexing.

The process of indexing involves a text extractor copying strings of text from content fields and a text analyzer parsing the text to find words and phrases that Rhythmyx stores as search terms.

Text Extractors

A text extractor runs when Rhythmyx indexes content items for searching. Rhythmyx first identifies each field's mime type, and then chooses the text extractor associated with that mime type.

Out of the box text extractors support the Mime types in the following table. The table lists the file types or text formats associated with the supported Mime types.

Format or File Type	Mime Types
HTML	text/html
Microsoft Excel	application/vnd.ms-excel application/vnd.ms-excel.sheet.macroEnabled.12 application/vnd.openxmlformats-officedocument.spreadsheetml.sheet
Microsoft Power Point	application/mspowerpoint application/vnd.ms-powerpoint.presentation.macroEnabled.12 application/vnd.openxmlformats-officedocument.presentationml.presentation
Microsoft Word	application/msword application/vnd.ms-word.document.macroEnabled.12 application/vnd.openxmlformats-officedocument.wordprocessingml.document
PDF	application/pdf
Plain Text	text/plain

Format or File Type	Mime Types
RTF	application/rtf application/x-rtf text/richtext
XML	text/xml

Administrators can write custom text extractors using the IPSLuceneTextConverter interface (see the JavaDoc for help). Custom text extractors override out-of-the-box text extractors for the mime types that the administrator specifies when adding them to the Server Administrator's Full-Text Search tab. For more information, see the topic *How to Override the Default Text Extractor* in the Server Administrator online help.

Text Analyzers

A text analyzer runs after text is extracted from content items. The text analyzer parses the extracted text strings into search terms for indexing.

Rhythmyx determines which text analyzer to use by determining the language of the text from the content item's Locale and choosing the text analyzer associated with that language.

Out of the box text analyzers support the following languages:

- English
- French
- German
- Italian
- Portuguese
- Spanish
- Danish
- Dutch
- Finnish
- Norwegian
- Russian
- Swedish
- Chinese
- Japanese
- Korean

Administrators can write custom text analyzers using the IPSLuceneAnalyzer interface (see the JavaDoc for help). A custom text analyzer can be associated with one or more Locales, and the administrator must register it separately for each Locale in the Rhythmyx Server Administrator. Once registered with a Locale, a custom text analyzer overrides the out-of-the-box text analyzer for that Locale. For more information, see the topic *How to Override the Default Text Analyzer* in the Server Administrator online help.

CHAPTER 3

Assembly Reference

The assembly process transforms the Content Items managed by the system into the published outputs: page elements and pages. The assembly process is recursive, allowing any number of formatting and merge transformations to take place before completing the final assembled output. The assembly process can produce either a complete HTML page or partially-assembled page elements published to application servers and databases.

The first section of this chapter outlines the logical architecture and processing of the Assembly engine. The next section details how Rhythmyx uses the Velocity templating technology to produce text (i.e., HTML) outputs. A third section is reference to the extensions provided for assembly, while the fourth a reference to the assembly API.

Logical Architecture and Processing: Assembly

This section is comprised of four subsections. This first details the logical architecture of the assembly engine. A second section describes the assembly process for a specific Content Item. The recursive nature of assembly is examined in the third part, while the final part describes how Managed Navigation is assembled.

Logical Architecture: Assembly

The centerpiece of the assembly engine is the Assembly Service, which receives requests for assembled output and produces a complete set of data for assembly. The actual assembly is performed by an assembly plugin (see below for details).

The following graphic illustrates the logical architecture of the Assembly engine:

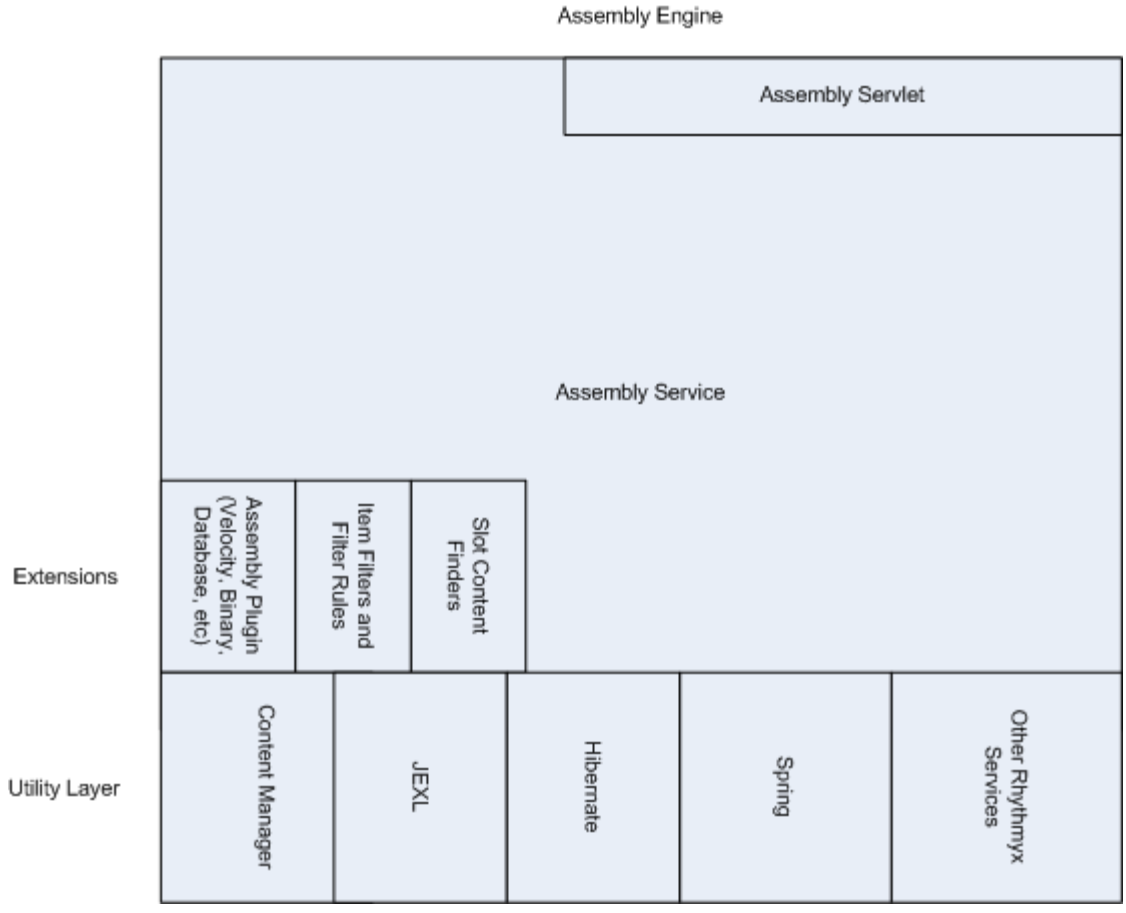


Figure 21: Logical architecture of the Assembly engine

The Assembly services rests on a utility layer composed of:

- The Content Manager
- JEXL (Java Expression Language; for additional details, see ***Java Expression Language (JEXL)*** on page 132).
- Hibernate
- Spring
- Other Rhythmyx services

In general, the interface to the Assembly Service is the Assembly Servlet, which receives requests and passes them to the Assembly Service for processing. The Assembly Servlet is the preferred interface, but it is possible to access the Assembly Service directly if necessary.

The Assembly Service also interacts with the following extensions:

- Assembly plugins

  Assembly plugins receive the dataset produced by the Assembly Service and process it to produce an assembled output. Rhythmyx ships with a complete set of Assembly plugins that handle standard assembly cases. The following standard Assembly plugins are shipped with Rhythmyx:

  - Velocity

    The Velocity Assembly plugin is the standard text assembly plugin shipped with Rhythmyx. It produces arbitrary text outputs, including HTML and XML, by merging Content Item data passed by the Assembly Services with formatting defined in a Velocity Template.

  - Legacy

    The Legacy Assembly plugin is a wrapper that invokes a legacy XSLT assembly application. This Assembly plugin is provided to ensure backwards-compatibility with earlier versions of Rhythmyx for Rhythmyx Version 6.0 and later.

  - Binary

    The Binary Assembly plugin is the standard plugin used to produce binary outputs. It produces the bound binary value (defined in the $sys.binary binding) as output. The MIME type is output from the bound value of $sys.mimetype as well.

  - Dispatch

    The Dispatch Assembly plugin provides conditional Template processing using a conditional binding. The Template select by the conditional processing produces an output as if it had been called directly.

  - Debug

    The Debug Assembly plugin is used for debugging Templates. It is invoked by adding the HTTP parameter `sys_debug="true"` to the assembly URL. This plugin returns the results of all bindings and bound Slots. (NOTE: Debug works differently in the Legacy Assembly plugin. Debug output of the Legacy Assembly plugin is the plain text XML document produced by the assembly request handler.)

- Database Publishing

  The Database Publishing Assembly plugin generates XML output used to publish content to databases.

- Slot Content Finders

  Slot Content Finders are extensions that determine the list of Content Items that can potentially be included in a Slot when assembling a Content Item. Rhythmyx ships with a set of four standard Slot Content Finders:

  - sys_RelationshipContentFinder

    This Content Finder is the standard Content Finder used to retrieve a list of Content Items manually assigned to a Slot.

  - sys_AutoSlotContentFinder

    This Content Finder automatically generates a list of related Content Items based on a query defined when assigning the Content Finder to a Slot.

  - sys_LegacyAutoSlotContentFinder

    This Content Finder automatically generates a list of related Content Items using a legacy XML query resource.

  - sys_ManagedNavContentFinder

    This Content Finder generates a Managed Navigation tree for us in Managed Navigation Slots. For additional details about Managed Navigation, see "Managed Navigation" in the *Rhythmyx Implementation Guide*.

- Item Filters and Filter Rules

  While Slot Content Finders define a potential list of related Content Items to assemble, the final list of related Content Items actually assembled is determined by the Item Filter that is run on the list of related Content before executing the assembly. An Item Filter defines a set of Filter Rules that will be applied to the list of related Content Items. Filter Rules are extensions that define the rules to use to filter the list of related Content Items to determine the final list of Content Items that will be processed. The following standard Filter Rules are shipped with Rhythmyx:

  - sys_filterByPublishableFlag

    Filters based on value of the Publishable Flag of the State of the Content Item.

  - sys_filterByFolderPaths

    Filters based on the path of the Content Item.

  - sys_filterBySitefolder

    Used for cross-site linking.

Assembly Processing

Assembly processing begins when an assembly request is submitted to the Assembly servlet. The assembly request may originate in a variety of ways:

- a user requesting preview of a Content Item;
- a user requesting Active Assembly of a Content Item;
- a publishing request for an assembled page;
- a request to assemble Slot contents.

The Assembly Servlet creates an Assembly Item. At this point, the Assembly Item consists of the following data:

- the Content ID and Folder ID;
- Context variables;
- the ID or Name (if derived from sys_template) of the Template to use when assembling the Content Item;
- any HTTP parameters submitted with the request.

Next, the Template is loaded into memory.

Following loading of the Template, the Content Item data is loaded into the the Assembly Item as a Java Content Repository (JCR) Node and Property Structure. The Content Item is loaded as a JSR-170 Node containing one Property object per Content Item field. Simple children (children that are edited directly within the parent Content Item) are loaded as multi-value Property objects of the JCR Node. Complex children (children that are edited in a popup Detail Editor) are loaded as child Node objects of the parent Node. Each child node is comprised of a set of Property objects containing the child field data.

After the Content Item data has been loaded, Variable bindings are calculated to produce final binding values.

At this point, the Assembly plugin is invoked. An Assembly plugin takes one parameter (item), whose value is the Assembly Item created by the earlier processing.

The resulting assembled Content Item is then returned to the requestor.

The following flowchart illustrates the overall process:

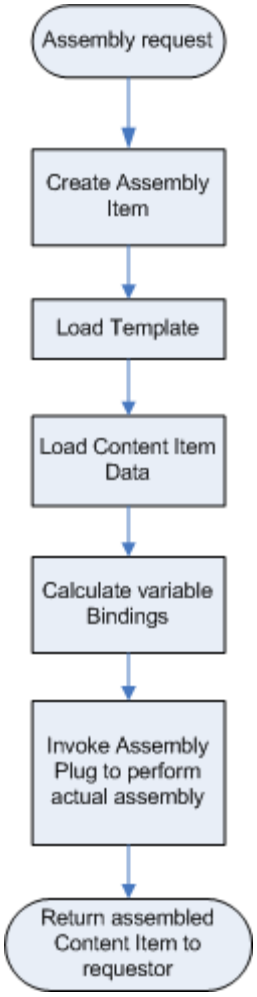


Figure 22: Assembly Processing

Assembly Plugin Processing

In most cases, Assembly plugin processing is simple. The Binary plugin retrieves the binary data supplied by the value of the $sys.binary binding. The Dispatch plugin calls the Template specified by the conditional processing in the bindings. The Debug Assembler returns all content Item node properties and the results of all binding calculations.

The Velocity plugin, which is used to assemble text content, is more complicated. When the Velocity plugin receives an an Assembly Item, it invokes the Velocity engine to assemble the dynamic content into the Template. When it encounters a Slot, the plugin invokes the Slot Content Finder extension specified for that Slot to retrieve the list of related Content Items to add to that Slot. Slot Content Finder extensions take the following parameters:

- the ID of the Content Item that owns the Slot;
- the Slot for which to find the related Content Items (as IPSTemplateSlot); and

- a map of parameters (the specific parameters are defined by the individual Content Finder extension).

The Content Finder uses these parameters to define a preliminary list of related Content Items to include in the Slot. The Content Finder then invokes an Item Filter to filter the list. An Item Filter consists of a set of Filter Rule extensions. Input parameters to Filter Rules include the list of Content Items to filter and a map or parameters (the specific parameters are defined by individual Filter Rule extensions). Each Filter Rule returns a list of Content Items that have passed the Filter Rule. Filter Rules are processed in the order in which they are specified in the Item Filter. Thus Filter Rules specified later in the Item Filter are only applied to the list Content Items that have been returned by the Filter Rules specified earlier.

The final list of related Content Items is then submitted to the Assembly engine to be assembled.

The following flowchart illustrates the processing of the Velocity plugin:

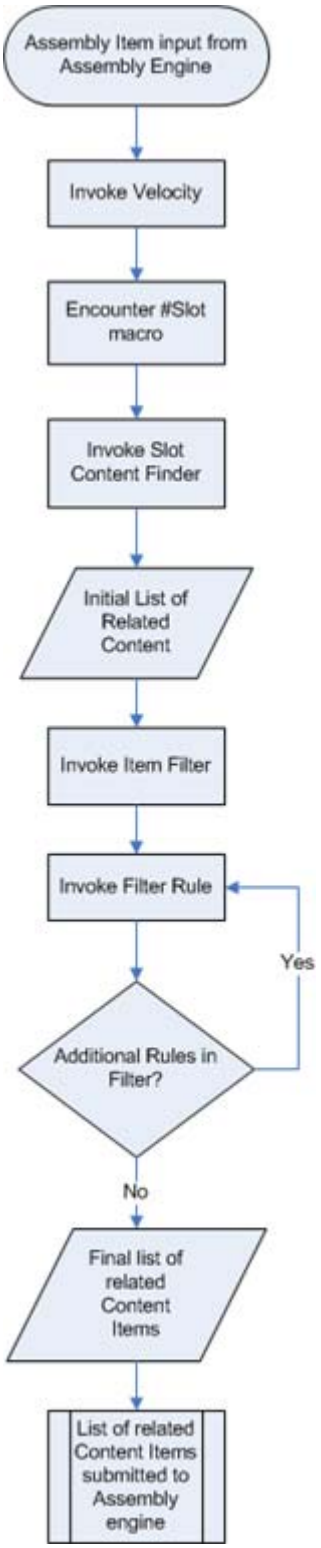


Figure 23: Velocity assembly processing

Recursive Content Roll-up

Rhythmyx assembles content into an output page by applying the fixed formatting to the local content of the content item, then recursively rolling up any snippets into the slots. The snippets are in turn assembled by applying the fixed formatting to the local content of the snippet, then recursively rolling up any child snippets into their slots, and so on. Each level of recursion includes only its own local content and templates and the data to identify the Content Items one level down. No level needs information about any level deeper than the next one down.

When Rhythmyx assembles a page:

1. An assembly request is submitted to the Assembly engine.
2. The engine processes the request and invokes the Assembly plugin. Recursive rollup takes place in either the Velocity plugin or the Database Publishing plugin.
3. When the plugin encounters a Slot, the Content Finder assigned to that Slot is invoked to determine a list of related Content Items to assemble. The Content Finder invokes an Item Filter to filter that list and output a final list of related Content Items.
4. Each related Content Item is itself submitted to the Assembly engine. Assembly processing starts from the beginning for each submitted Content Item. Each of these Content Items will be returned as a Snippet.
5. If any of the related Content Items themselves contain Slots, Step 3 is repeated for each Slot. As assembly of each related Content Item is completed, the assembled Content Item is added to the page.
6. When the recursive assembly of the Snippets is complete and all local content has been formatted, Rhythmyx returns the Snippet or Page.

The following graphic illustrates the process:

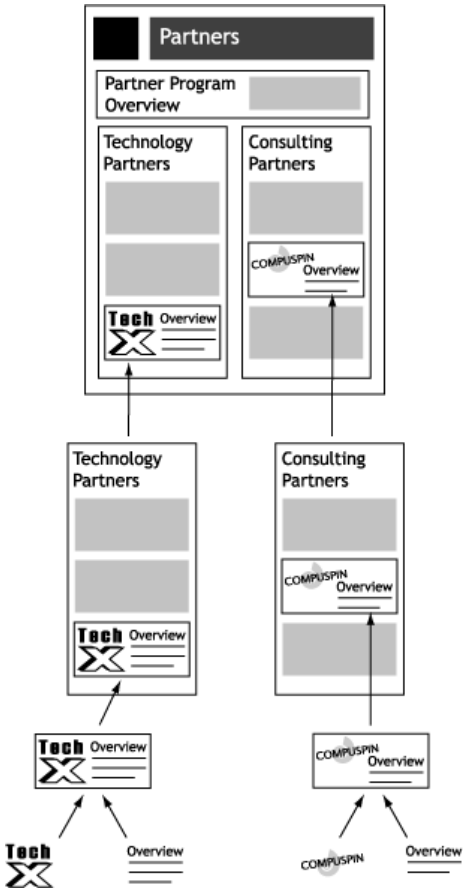


Figure 24: Recursive Rollup

Velocity in Rhythmyx

Velocity is the standard text templating languages used in Rhythmyx to produce text output. Rhythmyx supports all Velocity functionality.

Rhythmyx is shipped with a complete set of Velocity macros to handle standard assembly tasks, but you can also define your own Velocity macros. Standard macros are defined in the file `<Rhythmyxroot>/sys_resources/vm/sys_assembly.vm`. No customer macros should be added to this file, as it will be overwritten on upgrade. Custom macros should be defined in the file `<Rhythmyxroot>/rx_resources/vm/rx_assembly.vm`.

Note that there is no system of precedence in Velocity, so you cannot override a system macro with a custom macro. Instead, you must define a custom macro and use that instead.

Best practice for defining a custom macro is to copy the system macro that most closely matches the functionality you want from the `sys_assembly.vm` to `rx_assembly.vm`, change the name, then modify it.

If you want to allow Active Assembly in a macro, you must include the `#startAA<object>` and `#endAA<object>` macros appropriate to that type of object:

- Page
  ```
  #startAAPage
  #endAAPage
  ```
- Field
  ```
  #startAAField
  #endAAField
  ```
- Slot
  ```
  #startAASlot
  #endAASlot
  ```
- Snippet
  ```
  #startAASnippet
  #endAASnippet
  ```

Whenever you modify a macro, preview a Content Item, then add the HTTP parameter `sys_reinit=true` to the URL and resubmit the URL. This parameter re-initializes the Velocity engine, which reloads the macros. If you do not submit a request with this parameter, the cached macros will be used, which will not include the changes you have made

For details about Velocity and implementing Velocity macros, consult one of the following references:

- Joseph D. Gradecki and Jim Cole, *Mastering Apache Velocity*
- Rob Harrop, *Pro Jakarta Velocity*

Embedding Velocity Code in Templates

In addition to using macros, you can embed Velocity code directly into Template Markup. Use this option when you want to produce a specific Velocity result in a single Template rather that across several Templates. (If you use the same code in more than one Template, writing a macro instead makes more sense.

When embedding Velocity code in a Template, follow the same rules as when writing a macro. Specifically, when defining a Rhythmyx object in your Velocity code (a Field, a Slot, or a Snippet), ***you must use the #startAA<object> and $endAA<object> macros within your Velocity markup*** (see "Velocity in Rhythmyx" on page 85).

Standard Velocity Macros

Standard macros shipped with Rhythmyx can be found in the Snippet Drawer. (NOTE: You can also add custom macros to the Snippet Drawer; for details see ***Adding Macros to the Snippet Drawer*** on page 94.) Standard macros are classified into three categories:

- Field macros

  The macros in this category are used to add Content Item field data to the Template.

- Slot macros

  The macros in this category are used to add Slots to the Template.

- Slot Miscellaneous macros and prebuilds.

  Miscellaneous macros do not fit in the other two categories. Prebuilds are prebuild examples of pages and common Managed Navigation Templates. (NOTE: Only the miscellaneous macros will be documented below. Prebuilds are provided as examples for the development of your own Templates.)

NOTE: Macros that begin with a prefix of two underscores ("__macroname") are internal system macros and are not documented.

Field Macros

Field macros are used to add Content Item field data to a Template. They are found in the Rx Field Macros section of the Snippet Drawer.

#field

```
#field(fieldname)
```

This is the standard macro used to add field data to a Template. When Active Assembly is invoked, fields that are added to the Template using this macro will be displayed with Active Assembly decorations (meaning the data in the field can be edited. If this field does not have a value, an error will be generated when assembling the Template.

Parameters

Parameter	Description
fieldname	Name of the field whose data will be added to the Template during assembly. If the specified field does not have a value, an error will be returned when assembling the Template.

#field_if_set

```
#field_if_set(before,field,after)
```

This macro is used to add field data to the Template if the field is optional in the Content Editor (meaning it may not contain data). If the field contains no data, when assembling the Template, it is simply omitted from the assembled output.

The beforetext and aftertext parameters can be used to add formatting that will only be included in the assembled output if the field is included.

If the field is included in the assembled output, it will include Active Assembly decorations in Active Assembly mode.

Parameters

Parameter	Description
before	Text that will be included in the assembled output before the field value. Generally used to add HTML formatting that will be included in the assembled output only if the field is included.
fieldname	Name of the field whose data will be added to the Template during assembly. If the specified field does not have a value, it will be omitted from the assembled output.
after	Text that will be included in the assembled output after the field value. Generally used to add HTML formatting that will be included in the assembled output only if the field is included.

#fieldLink

```
#fieldlink(fieldname,$pagelink)
```

This macro is used to add field data to the Template when:

- the field data will be the contents of an anchor tag (<a>) ; and
- you want users to be able to follow the link (users can follow the link by hold down the ALT key while clicking on it; double-clicking on the link opens the field for editing).

If you use any other field macro for the contents of an anchor tag, users will not be able to follow the link; clicking on the link will open the field for editing.

Parameter	Description
fieldname	Name of the field whose data will be added to the Template during assembly. If the specified field does not have a value, an error will be returned when assembling the Template.
$pagelink	Required binding parameter. The value of this parameter must be *$pagelink*.

#displayfield

```
#displayfield(fieldname)
```

This macro is used to add field data to the Template when the field data is not intended to be eligible for Active Assembly, such as when adding the title to the HTML header. When Active Assembly is invoked, fields that are added to the Template using this macro will not be displayed with Active Assembly decorations (in other words, the field is not eligible to be edited in Active Assembly). If this field does not have a value, an error will be returned when assembling the Template.

Parameters

Parameter	Description
fieldname	Name of the field whose data will be added to the Template during assembly. If the specified field does not have a value, an error will be returned when assembling the Template.

#datefield

```
#datefield(fieldname,fieldformat)
```

This macro is used to add date fields to the Template. When assembled, the field data will be formatted using the pattern specified in the format parameter. When Active Assembly is invoked, fields that are added to the Template using this macro will be displayed with Active Assembly decorations (meaning the data in the field can be edited). If this field does not have a value, an error will be generated when assembling the Template.

Parameters

Parameter	Description
fieldname	Name of the field whose data will be added to the Template during assembly. If the specified field does not have a value, an error will be returned when assembling the Template.
fieldformat	The format that will be applied to the field data when assembled into the output.

#displaydatefield

```
#displaydatefield(fieldname,fieldformat)
```

This macro is used to add date fields to the Template, when the field data is not intended to be eligible for Active Assembly. When assembled, the field data will be formatted using the pattern specified in the format parameter. When Active Assembly is invoked, fields that are added to the Template using this macro will not be displayed with Active Assembly decorations (in other words, the field is not eligible to be edited in Active Assembly). If this field does not have a value, an error will be returned when assembling the Template.

Parameters

Parameter	Description
fieldname	Name of the field whose data will be added to the Template during assembly. If the specified field does not have a value, an error will be returned when assembling the Template.
fieldformat	The format that will be applied to the field data when assembled into the output.

#datefield_if_set

```
#datefield_if_set(before,field,format,after)
```

This macro is used to add date field data to the Template if the field is optional in the Content Editor (meaning it may not contain data). If the field contains no data, when assembling the Template, it is simply omitted from the assembled output.

The beforetext and aftertext parameters can be used to add formatting that will only be included in the assembled output if the field is included.

When assembled, the field data will be formatted using the pattern specified in the format parameter.

If the field is included in the assembled output, it will include Active Assembly decorations in Active Assembly mode.

Parameters

Parameter	Description
before	Text that will be included in the assembled output before the field value. Generally used to add HTML formatting that will be included in the assembled output only if the field is included.
field	Name of the field whose data will be added to the Template during assembly. If the specified field does not have a value, it will be omitted from the assembled output.
format	The format that will be applied to the field data when assembled into the output.
aftter	Text that will be included in the assembled output after the field value. Generally used to add HTML formatting that will be included in the assembled output only if the field is included.

Slot Macros

Slot macros are used to add Slots and their Contents to a Template. They are found in the Rx Slot Macros section of the Snippet Drawer.

#slot_simple

```
#slot_simple(slotname)
```

The simplest Slot macro, #slot_simple inserts the Slot Contents with no additional formatting. If the Slot contains no related Content Items, it is omitted from the assembled output.

Parameters

Parameter	Description
slotname	Name of the Slot to add to the Template.

#slot_wrapped

```
#slot_wrapped(slotname,startslottext,endslottext)
```

This macro adds the contents of the Slot, with each related Content Item wrapped in the in the text specified by the beforetext and aftertext parameters. If the Slot contains no related Content Items, it is omitted from the assembled output.

Parameters

Parameter	Description
slotname	Name of the Slot to add to the Template .
startslottext	Text that will be included in the assembled output before each related Content Item in the Slot. Generally used to add HTML formatting for each individual Content Item. (For users familiar with earlier versions of Rhythmyx, the beforetext and aftertext parameters are equivalent to the Snippet Wrapper.)
endslottext	Text that will be included in the assembled output after each related Content Item in the Slot. Generally used to add HTML formatting for each individual Content Item. (For users familiar with earlier versions of Rhythmyx, the beforetext and aftertext parameters are equivalent to the Snippet Wrapper.)

#slot

```
#slot(slotname,header,before,after,footer,params)
```

This macro adds the contents of the Slot to the Template, wrapped in HTML formatting. Each related Content Item wrapped in the in the text specified by the beforetext and aftertext parameters. The Slot contents as a whole are wrapped in the text specified by the header and footer parameters. If the Slot contains no related Content Items, it is omitted from the assembled output.

Parameters

Parameter	Description
slotname	Name of the Slot to add to the Template .

Parameter	Description
header	Text that will be included in the assembled output before the contents of the Slot. Generally used to add HTML formatting to wrap the Slot as a whole. For users familiar with earlier versions of Rhythmyx, the header and footer are equivalent to the Slot Wrapper.
before	Text that will be included in the assembled output before each related Content Item in the Slot. Generally used to add HTML formatting for each individual Content Item. (For users familiar with earlier versions of Rhythmyx, the beforetext and aftertext parameters are equivalent to the Snippet Wrapper.)
after	Text that will be included in the assembled output after each related Content Item in the Slot. Generally used to add HTML formatting for each individual Content Item. (For users familiar with earlier versions of Rhythmyx, the beforetext and aftertext parameters are equivalent to the Snippet Wrapper.)
footer	Text that will be included in the assembled output after the contents of the Slot. Generally used to add HTML formatting to wrap the Slot as a whole. For users familiar with earlier versions of Rhythmyx, the header and footer are equivalent to the Slot Wrapper.
params	The extra parameters to pass to the slot content finder. The parameters can either be a Java java.util.Map object, or a string that is encoded as a URL query, i.e. name=value&name2=value2&... Generally, using a Java Map object is preferable; it is required if the values of the parameters you want to pass include the ampersand ["&"] or equals sign ("=") characters.

#node_slot

```
#node_slot(node slotname, header, before, after, footer, params)
```

This macro is used when both Velocity Templates and XSL Variants co-exist on the same system, generally to implement Managed Navigation. Works like the full slot macro, but uses the Content Item specified in the node parameter to expand the list of related Content Items in the Slot. Contents of `#node_slot` are not eligible to be modified using Active Assembly.

Parameters

Parameter	Description
node	The Content Item used to expand the list of related Content Items in the Slot.
slotname	Name of the Slot to add to the Template .
header	Text that will be included in the assembled output before the contents of the Slot. Generally used to add HTML formatting to wrap the Slot as a whole. For users familiar with earlier versions of Rhythmyx, the header and footer are equivalent to the Slot Wrapper.
before	Text that will be included in the assembled output before each related Content Item in the Slot. Generally used to add HTML formatting for each individual Content Item. (For users familiar with earlier versions of Rhythmyx, the beforetext and aftertext parameters are equivalent to the Snippet Wrapper.)

Parameter	Description
after	Text that will be included in the assembled output after each related Content Item in the Slot. Generally used to add HTML formatting for each individual Content Item. (For users familiar with earlier versions of Rhythmyx, the beforetext and aftertext parameters are equivalent to the Snippet Wrapper.)
footer	Text that will be included in the assembled output after the contents of the Slot. Generally used to add HTML formatting to wrap the Slot as a whole. For users familiar with earlier versions of Rhythmyx, the header and footer are equivalent to the Slot Wrapper
params	The extra parameters to pass to the slot content finder. The parameters can either be a Java java.util.Map object, or a string that is encoded as a URL query, i.e. name=value&name2=value2&...

#slot_page

```
#slot_page(slotname,header,before,after,footer,params)
```

This macro adds the contents of the Slot to the Template, wrapped in HTML formatting. Each related Content Item wrapped in the in the text specified by the beforetext and aftertext parameters. The Slot contents as a whole are wrapped in the text specified by the header and footer parameters. If the Slot contains no related Content Items, it is omitted from the assembled output.

Parameters

Parameter	Description
slotname	Name of the Slot to add to the Template .
header	Text that will be included in the assembled output before the contents of the Slot. Generally used to add HTML formatting to wrap the Slot as a whole. For users familiar with earlier versions of Rhythmyx, the header and footer are equivalent to the Slot Wrapper.
before	Text that will be included in the assembled output before each related Content Item in the Slot. Generally used to add HTML formatting for each individual Content Item. (For users familiar with earlier versions of Rhythmyx, the beforetext and aftertext parameters are equivalent to the Snippet Wrapper.)
after	Text that will be included in the assembled output after each related Content Item in the Slot. Generally used to add HTML formatting for each individual Content Item. (For users familiar with earlier versions of Rhythmyx, the beforetext and aftertext parameters are equivalent to the Snippet Wrapper.)
footer	Text that will be included in the assembled output after the contents of the Slot. Generally used to add HTML formatting to wrap the Slot as a whole. For users familiar with earlier versions of Rhythmyx, the header and footer are equivalent to the Slot Wrapper.
params	The extra parameters to pass to the slot content finder. The parameters can either be a Java java.util.Map object, or a string that is encoded as a URL query, i.e. name=value&name2=value2&...
itemsPerPage	The number of Slot Content Items to include on each output HTML page
pageNumber	The current page being rendered; generally $sys.page. For example, $pageNumber=if ($sys.page != null) {$sys.page;} else {1;}

Miscellaneous Macros

The macros in this category do not fit in either of the other categories. The are located in the Miscellaneous and Prebuild section of the Snippet Drawer.

#inner

```
#inner()
```

This macro is used with Global Templates to add the page content top the Global Template output. This macro has no parameters.

#children

```
#children(childname,template,beforetext,aftertext,header,footer)
```

This macro is used on page Templates to add child editor data to the Page Template output. If the Content Item has no child Content Items, the formatted results of this macro are omitted from the assembled output.

Parameters

Parameter	Description
childname	Name of the child Field Set to add to the Template .
template	The Template used to format the content from the child editor.
beforetext	Text that will be included in the assembled output before each child Content Item in the Slot. Generally used to add HTML formatting for each individual child Content Item.
aftertext	Text that will be included in the assembled output after each child Content Item in the Slot. Generally used to add HTML formatting for each individual child Content Item.
header	Text that will be included in the assembled output before the contents of the Slot. Generally used to add HTML formatting to wrap the Slot as a whole.
footer	Text that will be included in the assembled output after the contents of the Slot. Generally used to add HTML formatting to wrap the Slot as a whole.

#pager

```
#pager($pagecount $pagenumber $previous_markup $pagetext
$next_markup)
```

This macro is used on page Templates to add child editor data to the Page Template output. If the Content Item has no child Content Items, the formatted results of this macro are omitted from the assembled output.

Parameters

Parameter	Description
$pagecount	The total number of pages to be generated. Usually $sys.pagecount

Parameter	Description
$pagenumber	The current page being generated. Usually $sys.page
$previous_markup	The markup to render in the previous link. The previous link is rendered if the page number is greater than one. Anything valid for an anchor tag is allowed in this parameter.
$pagetext	Text to render that could, for example, indicate where the user is in the sequence of pages. For example, “page 3” or “page 2 of 5”. The string is usually defined in the bindings; for example: `$pagetext = “Page “ + $sys.page + “of” + $sys.pagecount`.
$next_markup	The markup to render in the next link. The next link is rendered if the page number is less than the total count of pages. Anything valid for an anchor tag is allowed in this parameter.

#linkback_head

```
#linkback_head()
```

This macro is used to add linkback functionality to HTML pages generated by Percussion CM Server. Linkback allows a user to go directly from a published HTML page to the Percussion CM System Content Item from which the page is generated. The macro adds the following linkback meta tag code to the Template:

```
<meta name="perc_linkback" id="perc_linkback"
content="$rx.linkback.encode($sys.params)"/>
```

Percussion CM System uses this code to process linkback.

The macro should be added to the header of Global Templates or to the header of Page Templates that do not use Global Templates.

This macro has no parameters

Adding Macros to the Snippet Drawer

If you define a custom macro, you may want to add it to the Snippet Drawer of your Rhythmyx Workbench to make it easy to use. You can share the Snippet Drawer entry with other Rhythmyx implementers in your organization.

Custom macros should be added to a new Category rather than to one of the standard Categories shipped with Rhythmyx. To add a custom Category:

1. In the Rhythmyx Workbench, right-click in the Snippet Drawer and from the popup menu, choose *Customize*. (Note: do not click on the "Snippets" tab. Clicking on the tab displays a different popup menu that does not include the Customize option.

The Rhythmyx Workbench displays the Customize Palette dialog.

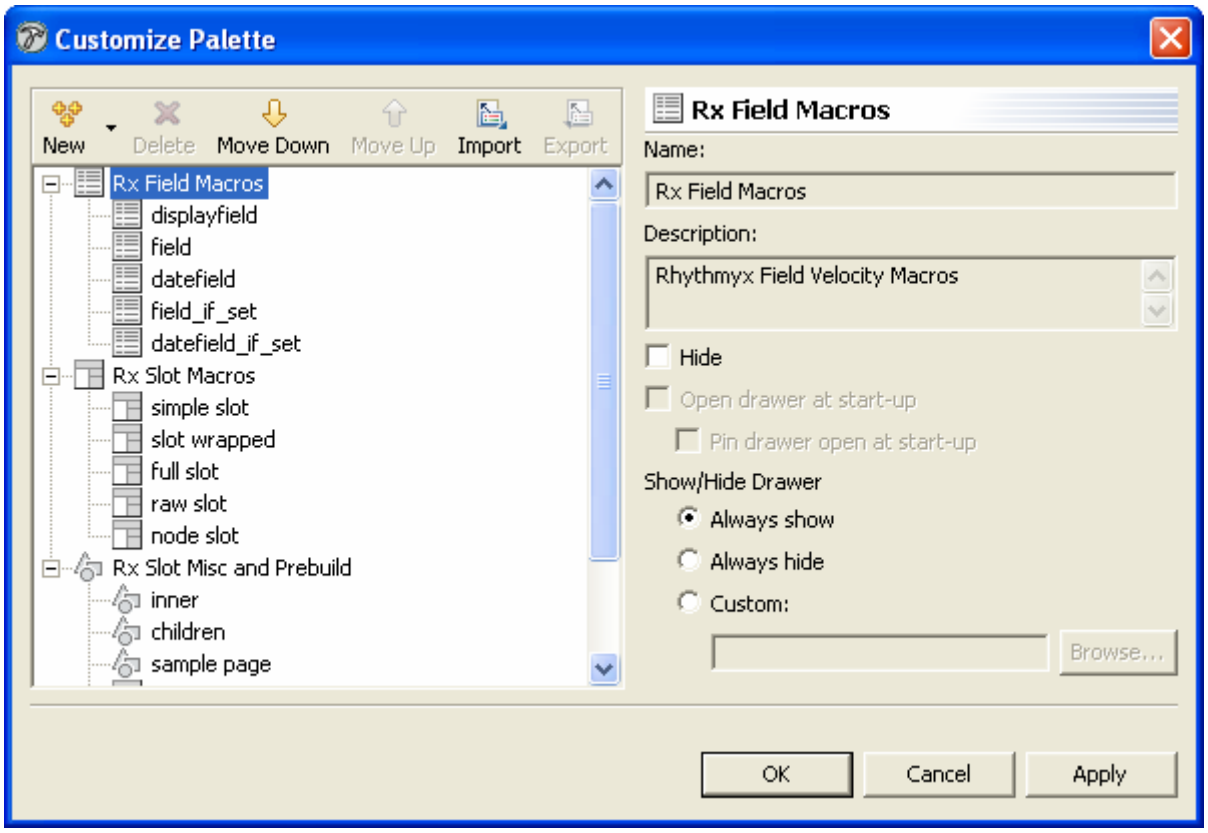


Figure 25: Customize Palette dialog

2 In the Button bar, click the [**New**] button and choose *New Category*. (Note: You must add custom Snippets to a unique category. They cannot be added to the categories shipped with Rhythmyx.)

The Rhythmyx Workbench adds a new Snippet category with the default name Unnamed Category.

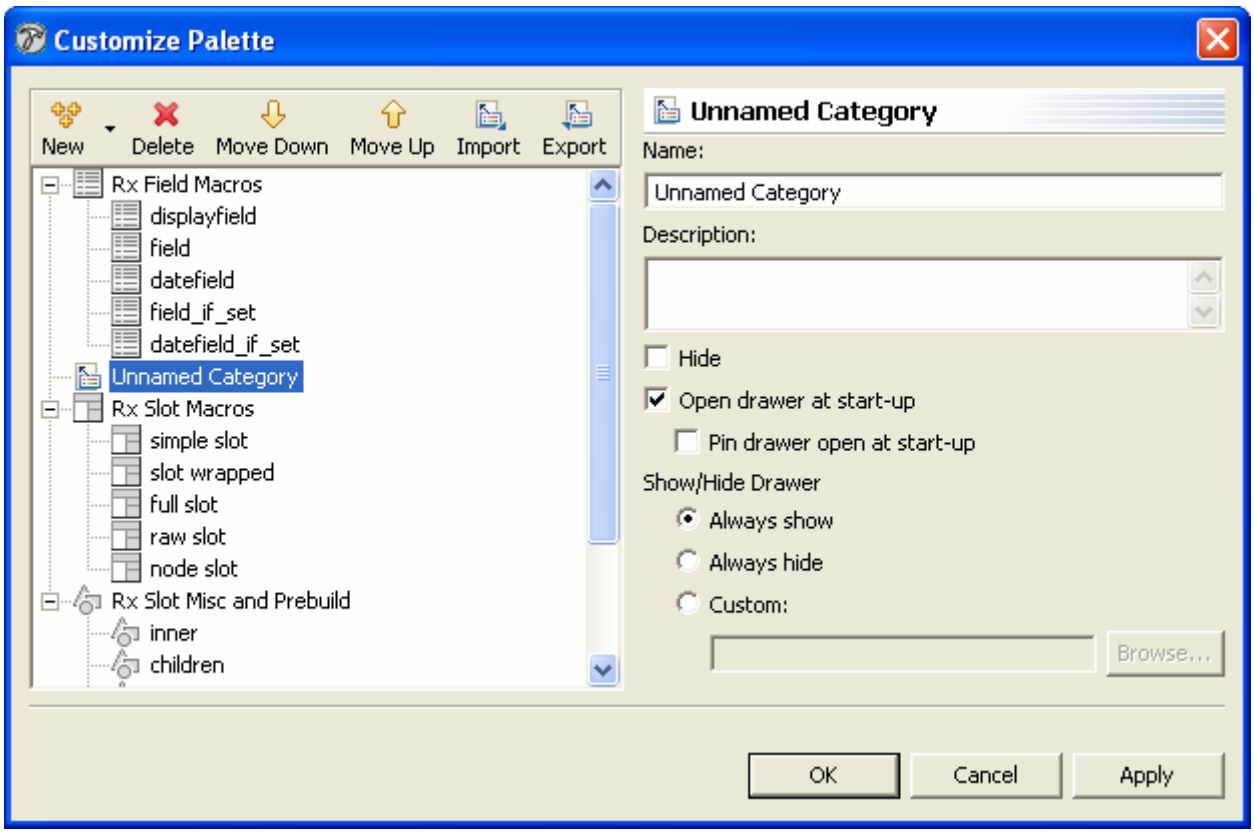


Figure 26: Customize Palette dialog with Unnamed Category

3 Enter a new Name for the Category.

4 Click the [**OK**] button to save your changes.

To add a a new macro

1 Open the Customize Palette dialog as described in Step 1 of adding a custom Category.

2 Select the custom Category to which you want to add the macro. Macros should not be added to the standard Categories shipped with Rhythmyx.

3 In the Button bar, click the [**New**] button and choose *New Item*.

4 The Customize Palette dialog displays the Template panel with the default name *Unnamed Template*.

5 Enter the Name of the macro and an optional Description.

6 To add macro parameters,

a) Click the [**New**] button next to the Variables table.

The Custom Palette dialog adds opens a new row with the value *name_1* in the Name column.

b) Change the default value of the Name to the name of the first parameter in your macro.

c) Optionally, enter a Description.

d) Optionally, specify a Default value for the parameter. This value will be used when processing the Template if no value is specified for the parameter in the Template markup.

e) Repeat Steps a-d for each parameter in the macro.

7 In the Template Pattern field, enter the macro as you want it added to the Template HTML markup. You can use the [Insert Variable Placeholder] button to add macro parameters or add them manually.

8 Click the [**OK**] button to save the macro entry.

To copy and modify an existing macro entry:

1 In the Snippet Drawer, right-click on the macro you want to copy and from the popup menu, choose *Copy*.

2 Open the custom macro category to which you want to add the macro. Right-click and from the popup menu, choose *Paste*.

3 Open the Customize Palette dialog as described in Step 1 of adding a custom Category.

4 Modify the macro data to match your custom macro.

5 Click the [**OK**] button to save the macro.

Assembly Extensions

This section documents extension types used only in content assembly:

- ***Assembly Plugins*** (see below)
- ***Content Finders*** (see "Slot Content Finders" on page 102)

The following extension types, which are used in both assembly and publishing, are documented elsewhere:

- ***JEXL functions*** (see "JEXL Extensions" on page 132)
- ***JSR-170 queries*** (see "Java Content Repository" on page 143)
- ***Item Filter rules*** (see "Item Filters and Filter Rules" on page 144)
- ***Location Scheme Generators*** (see "Location Scheme Generator Extensions" on page 147)

Assembly Plugins

Assembly plugins perform the actual assembly of content output, either directly by invoking an underlying template engine such as Velocity.

The output produced by an assembly plugin depend on the configuration of the plugin and the parameters defined for the Assembly Item input to the plugin.

Assembly plugins must implement the interface IPSAssembler They must also explicitly implement the interface IPSExtension. (NOTE: The implementation must be thread safe; for details see ***General Requirements of Extensions*** on page 180.)

binaryAssembler

Passes binary Content Item data directory to the output.

Class Name

com.percussion.services.assembly.impl.plugin.PSBinaryAssembler

Interface

com.percussion.extension, com.percussion.services.assembly.IPSAssembler

Context

global/percussion/assembly/

Category String

assembly

Parameters

None

databaseAssembler

Generates an XML document to match the requirements of the database publisher handler.

Class Name

com.percussion.services.assembly.impl.plugin.PSDatabaseAssembler

Interface

com.percussion.extension, com.percussion.services.assembly.IPSAssembler

Context

global/percussion/assembly/

Category String

assembly

Parameters

None

debugAssembler

Generates debug output regardless of the specified Template.

Class Name

com.percussion.services.assembly.impl.plugin.PSDebugAssembler

Interface

com.percussion.extension, com.percussion.services.assembly.IPSAssembler

Context

global/percussion/assembly/

Category String

assembly

Parameters

None

dispatchAssembler

Chooses a Template based on the defined bindings and invokes assembly of the result.

Class Name

com.percussion.services.assembly.impl.plugin.PSDispatchAssembler

Interface

com.percussion.extension, com.percussion.services.assembly.IPSAssembler

Context

global/percussion/assembly/

Category String

assembly

Parameters

None

legacyAssembler

Assembles content using a legacy query application and stylesheet.

Class Name

com.percussion.services.assembly.impl.plugin.PSLegacyAssembler

Interface

com.percussion.extension, com.percussion.services.assembly.IPSAssembler

Context

global/percussion/assembly/

Category String

assembly

Parameters

None

velocityAssembler

Assembles the submitted Content Item using the using the Velocity engine and the submitted Template.

Class Name

com.percussion.services.assembly.impl.plugin.PSVelocityAssembler

Interface

com.percussion.extension, com.percussion.services.assembly.IPSAssembler

Context

global/percussion/assembly/

Category String

assembly

Parameters

None

Slot Content Finders

Slot Content Finders generate a list of related Content Items to be added to a Slot during assembly. Content Finders must also be able to invoke Item Filters to filter the initial list to a final list.

Slot Content Finders must implement the interface IPSSlotContentFinder. (NOTE: The implementation must be thread safe; for details see ***General Requirements of Extensions*** on page 180.)

sys_AutoSlotContentFinder

Automatically generates a list of related Content Items for the associated Slot based on the specified Java Content Repository query. This list if filtered by an Item Filter then submitted to be assembled using the Template specified in the template parameter.

Class Name

com.percussion.services.assembly.impl.finder.PSAutoSlotContentFinder

Interface

com.percussion.services.assembly.IPSSlotContentFinder

Context

global/percussion/slotcontentfinder/

Parameters

Name	Data Type	Description
query	String	(Required) The JSR-170 "SQL" query to use to generate the base list of Content Items for the slot.
type	String	The type of query. Options include `sql` and `xpath` (NOTE: Only `sql` is currently supported). Defaults to `sql` if not specified.
template	String	(Required) The Template to use to format the Content Items returned. Either the name or the ID may be specified.
max_results	String	The maximum number of items to return for the slot. Defaults to unlimited.

sys_LegacyAutoSlotContentFinder

Uses a legacy query resource to automatically generate a list of related Content Items for the associated Slot. When invoked, this Content Finder builds an internal request to the Rhythmyx resource specified in the resource parameter. This request includes any parameters passed from the calling Template, as well as the parameters of the ContentFinder itself. Note that if the calling Template specifies values for any parameters of the Content Finder, the parameters passed from the Template override the parameters of the Content Finder's association with the Slot.

The returned XML document must conform to the sys_AssemblerInfo DTD. The document must consist of a set of linkurl elements. Each linkurl element must include the attributes contentid and variantid. Rhythmyx does not return an error if these attributes do not have a value, but the Slot will contain no content. If a Slot using this Content Finder does not include any Content Items, check to be sure that the resource is returning an XML document that meets the requirements.

Class Name

com.percussion.services.assembly.impl.finder.PSLegacyAutoSlotContentFinder

Interface

com.percussion.services.assembly.IPSSlotContentFinder

Context

global/percussion/slotcontentfinder/

Parameters

Name	Data Type	Description
resource	String	Specifies the Rhythmyx query resource to run to generate the list of related Content Items. The query resource must conform to the `sys_AssemblerInfo` DTD.
template	String	(Required) The Template to use to format the Content Items returned. Either the name or the ID may be specified.
max_results	String	The maximum number of items to return for the slot. Defaults to unlimited.

sys_ManagedNavContentFinder

Returns the list of Content Items assigned to the Slot by users.

Class Name

com.percussion.services.assembly.impl.finder.PSNavSlotContentFinder

Interface

com.percussion.services.assembly.IPSSlotContentFinder

Context

global/percussion/slotcontentfinder/

Parameters

Name	Data Type	Description
node_to_return	String	If the value of this parameter is "self", the Navon associated with the current Content Item is returned. If the value of this parameter is "root" ot is not specified, then the Navtree is returned.
template	String	(Required) The Template to use to format the Content Items returned. Either the name or the ID may be specified.

sys_RelationshipContentFinder

Returns the list of Content Items assigned to the Slot by users.

Class Name

com.percussion.services.assembly.impl.finder.PSRelationshipContentFinder

Interface

com.percussion.services.assembly.IPSSlotContentFinder

Context

global/percussion/slotcontentfinder/

Parameters

Name	Data Type	Description
template	String	(Required) The Template to use to format the Content Items returned. Either the name or the ID may be specified.
max_results	String	The maximum number of items to return for the slot. Defaults to unlimited.

Name	Data Type	Description
order_by	String	Comma-separated list of fields to use to sort the list of related Content Items. For each field specified, you can add either DESC to sort in descending order or ASC to sort in ascending order (defaults to DESC if neither is specified).

sys_TranslationContentFinder

Returns the list of Content Items associated in a Translation Relationship with the Content Item being assembled.

Class Name

com.percussion.services.assembly.impl.finder.PSTranslationContentFinder

Interface

com.percussion.services.assembly.IPSSlotContentFinder

Context

global/percussion/slotcontentfinder/

Parameters

Name	Data Type	Description
template	String	(Required) The Template to use to format the Content Items returned. Either the name or the ID may be specified.
max_results	String	The maximum number of items to return for the slot. Defaults to unlimited.
order_by	String	Comma-separated list of fields to use to sort the list of related Content Items. For each field specified, you can add either DESC to sort in descending order or ASC to sort in ascending order (defaults to DESC if neither is specified).

Writing Assembly Extensions

Use methods in the assembly service when performing assembly processing.

When writing a method that retrieves assembly design elements, the method should always call the assembly service itself before calling any of its methods.

```
IPSAssemblyService asm = PSAssemblyServiceLocator.getAssemblyService();
```

Obtaining Slots

To obtain a single Slot, use the findSlotByName method of the assembly service. Be sure you have called the assembly service before attempting to use this method.

```
IPSTemplateSlot slot = asm.findSlotByName(slotName);
```

To obtain multiple Slots, use the findSlotsByNames method.

```
IPSTemplateSlot slots = asm.findSlotsByNames(slotNames);
```

Generating a List of Slot Contents

Once you have loaded a Slot, you can generate a list of the Content Items in that Slot:

```
String findername = slot.getFinderName();
IPSSlotContentFinder finder = asm.loadFinder(findername);
List<IPSAssemblyItem> relitems = finder.find(item, slot, params);
return relitems;
```

CHAPTER 4

Workflow Reference

A Workflow is a business process that defines a sequence of processing stages in the content management system. Workflows organize the content development and management process by defining the process, controlling the progress of Content Items through the process, and controlling user access to Content Items at particular points in the process. Each Content Item must exist in a Workflow, although a particular Content Editor may provide a choice of Workflows.

Workflows exist separately from other elements of the content management system, but are fully integrated into the system as whole. Content Editors require Workflows to function and the Publisher must know the Workflow State of the Content Items to extract for publishing.

Logical Architecture and Processing

This section is comprised of two subsections. The first details the logical architecture of the Workflow engine. The second outlines how Content Items are processed by the Workflow engine.

Logical Architecture

The central architectural feature of the Workflow engine is the Workflow object itself, as illustrated in the following graphic:

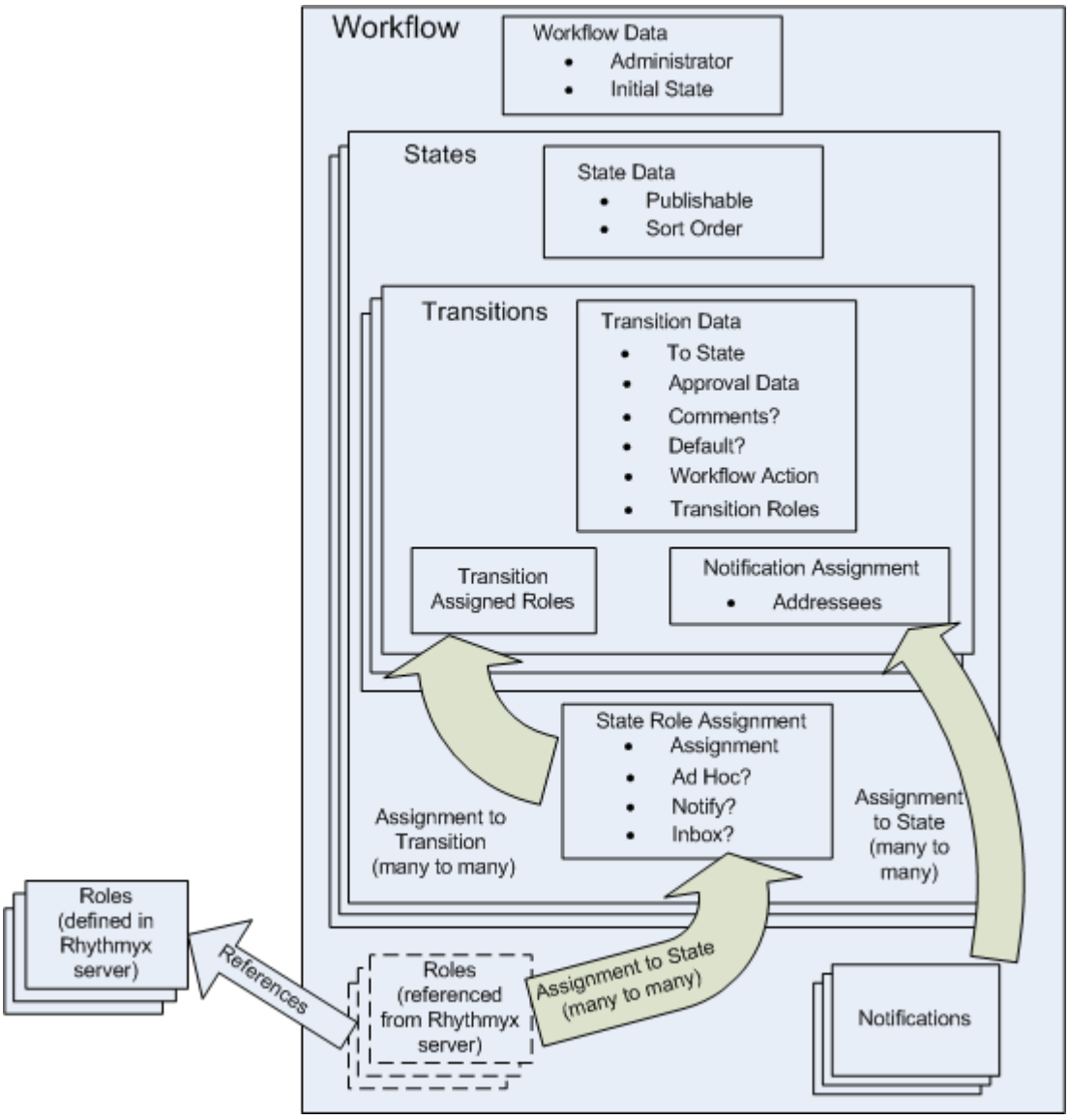


Figure 27: Logical architecture of the Workflow engine

While the Workflow object itself does have some properties (namely the name of the Role assigned as the Workflow Administrator and the name of the Initial State of the Workflow), it serves primarily as a container for the other elements of the Workflow.

Three elements are defined at the Workflow level:

- A set of references to Roles defined on the Rhythmyx server.

  Roles do not exist within Workflow. Roles exist on the Rhythmyx server, where they are defined and their properties and list of Members are maintained. Each Workflow includes a set of references to these Roles. A Workflow can only use Roles that have been associated with the Workflow.

- A set of zero or more Notifications.

  A Notification is an object that defines the subject and message contents for e-mail messages that will be sent to users based on the occurrence of certain events within the Workflow.

- A set of one or more States

States define the stages in the Workflow. States must be ordered to provide sequence in the Workflow (although Content Items can generally move from one State to another without reference to the sequence of the Workflow). Each State includes the following data:

- The Sort Order of the State

  The Sort Order defines the position of the State in relation to the other States in the Workflow.

- The Publishable flag

  The Publishable flag is used by the Publishing engine to determine whether a Content Item is eligible to be published.

- A set of assigned Roles

  A Role must be assigned to a State before Members of that Role can act on Content in that State. (Note that a Role must be associated with the Workflow before it can be assigned to a State). Each Role associated with a Workflow can be assigned to multiple States in the Workflow. The actions available to the Role depend on the data defined for its assignment to the State:

  - Assignment type

    The Assignment type defines the visibility of the Content Item to Members of the Role and the actions users in the Role can take on those Content Items. If the Assignment type is *Assignee*, Members of the Role have full access to Content Items in the State. If the Assignment type is *Reader*, Members of the Role can see Content Items in the State and can view their content and properties, but cannot act on them (such as to edit them or to Transition them to another State). If the Assignment type is *None*, Members of the Role can neither see Content Items in that State nor act on them.

  - Ad Hoc Assignment enabled

    If Ad Hoc Assignment is enabled, when a user Transitions a Content Item into the State, they can assign the Content Item to a specific Member of a State-assigned Role for action. Otherwise, the Content Item is available generally to any Member of any State-assigned Role with an Assignment Type of *Assignee*.

- Notification enabled

  If Notification is enabled, Members of the Role can receive e-mail Notifications when a Transition occurs.

- Show in Inbox

  If Show in Inbox is enabled, Content Items in the State are listed in the Inbox of users in State-assigned Roles.

A State also serves as a container for a set of Transitions. A Transition is a mechanism that moves a Content Item from one State to another. Each Transition specifies the Target State, to which the Content Item will move. A Transition also specifies:

- Approval data, such as whether a specific number of approvals is required or a specific set of Roles is required to approve the Content Item before the Transition is actually executed. If specific Roles are required to approve the Transition, these Roles must be associated with the Transition.
- Whether comments are required when executing the Transition.
- Whether a Transition is the default Transition out of the State
- A Workflow Action extension to execute when making the Transition.
- A Notification to send when executing the Transition, including the set of users to whom to send the Notification e-mail message.

Workflow Processing

When a Content Item is created, it is assigned to a Workflow and moves into the Initial State of that Workflow. A user must be in a Role assigned to the current State of a Content Item to potentially have access to that Content Item, depending on the Assignment type. Users in Roles with an Assignment type of *Assignee* potentially have full access to Content Items in the State. Users in Roles with an Assignment type of *Reader* can see the Content Item and can view its properties, but cannot act on Content Items in the State (such as to modify its data or to Transition it to another State). Users in Roles with an Assignment type of *None* can neither see nor act on Content Items in the State.

Note that the Assignment of a Role to a State is only one factor that determines whether a user has access to a Content Item. Other factors affecting access are the Community of the Content Item and whether the Content Item is checked out and to whom. If a user is in a Role that has access to a Content Item based on its current State, but the user is logged in to a different Community than that of the Content Item, the user will not be able to access the Content Item (although they may be able to see the Content Item for Active Assembly). If a user is in a Role that has access to a Content Item based on its current State, and the Content Item is in the user's logged Community, but the Content Item is checked out to another user, the first user will be able to see the Content Item and view its properties and content, but will not be able to modify it or Transition it.

The current State of a Content Item also determines how it will be processed for publishing. Item Filter Rules can be defined to select Content Items for publishing based on the Publishable flag. For additional details, see ***Item Filters and Filter Rules*** (on page 144).

When the Content Item is ready to move to another State, a user executes a Transition on it. The approval configuration of the Transition determines whether the Content Item actually makes the Transition to another State. A Transition can be configured to require a specific number of approvals or to require approvals from a specific set of Roles. If the Transition is configured to require a specific number of approvals, the Content Item remains in its current State until the specified number of approvals have occurred, at which point the Content Item Transitions to the State specified by that Transition. (Thus if one approval is required for the Transition, the Transition is executed immediately.) Note that if another Transition with fewer approvals is executed in the meantime, the Content Item will be Transitioned once the lower number is achieved. For example, suppose the following Transition configurations have been defined:

- An Approve to Public Transition that requires three approvals.
- A Return to Draft Transition that requires only one approval.

If the Approve to Public Transition has two approvals when another user executes the Return to Draft Transition, the Return to Draft Transition is executed immediately. The previous approvals for the Approve to Public Transition are deleted. Those users will have to approve the Content Item again after it re-enters the State.

If a Transition is configured to require approvals from specified Roles, it remains in its current State until each of the required Roles has approved the Transition. Again, if another Transition that has lower requirements is executed in the meantime, that Transition occurs and all existing approvals are deleted.

If comments are required, the system displays a Comment dialog where the user must enter text before the Transition will actually take place.

At this point, any Workflow Actions associated with the Transition are also triggered.

Finally, if any Notifications are associated with the Transition, e-mails are generated to the specified recipients.

Extending Publishable States

The values of the Publishable property for States are maintained in the Rhythmyx Keyword Editor of the Rhythmyx Workbench. You can thus add more values to the Publishable property to extend it and make it more flexible.

WARNING! Do not delete the default values of this Keyword. If you delete any of these default values, Publishing will no longer work correctly.

These values are used in the Item Filters that filter Content Items during Publishing. Use the sys_filterByPublishableFlag Item Filter and set the value of the sys_flagValue to the Keyword value for your Publishable State.

For example, suppose you wanted to implement a staging area where you could evaluate content before publishing it to your live web site. You could add a new value to the Publishable property, s (for staging). You would also create a State in the Workflow (perhaps also called Staging) and would assign S as the value of Publishable for this State. Finally, you would define a new Item Filter including the Filter Rule sys_filterByPublishableFlag with the value of the sys_flagValues parameter set to *s*.

Set the Publishable value for a State using the Edit State Page. The default Publishable values are:

Value	Processing
N	Default. Content in this State is not published, or will be unpublished the next time the Publisher runs.
Y	Content in this State is published when the Publisher runs.
I	Publish the Last Public Revision of the Content Item.

Workflow Actions

Only one extension type is associated with the Workflow engine, Workflow Actions. Workflow actions process Content Items when triggered by a Transition. The specific processing is defined by the extension.

Workflow Actions must implement the interface IPSWorkflowAction. (NOTE: The implementation must be thread safe; for details see ***General Requirements of Extensions*** on page 180.)

sys_createTranslations

Name:

sys_createTranslations

Context:

global/percussion/workflow/

Description:

This action creates a Translation Content Item of the original Content Item in each Locale in which the original Content Item does not already have a corresponding Translation Content Item. The action uses a configuration file, sys_createTranslations.properties, which is located in the directory <Rhythmyxroot>/rxconfig/i18n. This file defines the type of Translation Relationship to create between the original Content Item and the Translation Content Item for each Locale. It also defines a list of Locales for which Translation Content Items will not be generated.

Class Name;

com.percussion.workflow.PSCreateTranslations

Resource File:

rxconfig/I18n/sys_createTranslations.properties

Interface:

com.percussion.extension.IPSWorkflowAction

Parameters:

None

sys_PublishContent

Name:

sys_PublishContent

Context:

global/percussion/workflow/

Description:

This extensions triggers the publication of a Edition when a Transition is executed. The action requires an XML file (`rxconfig/Workflow/publish.xml`), which defines the following data. The XML should conform to the following DTD:

```
<?xml encoding="UTF-8"?>
      <!ELEMENT PSXConfig (PSXPublish+)>
      <!ATTLIST PSXConfig
   polling-time CDATA #IMPLIED>
      <!ELEMENT PSXPublish (PSXWorkflowId, PSXTransitionId, PSXEdition)>
      <!ELEMENT PSXWorkflowId PCDATA>
      <!ELEMENT PSXTransitionId PCDATA>
      <!ELEMENT PSXEdition PCDATA>
```

The root element of the document can have any name; we use `PSXConfig` for convenience. The optional `polling-time` attribute of this element specifies the time interval (in milliseconds) between successive attempts to publish a specific Edition when that Edition is already being published. A longer interval results in fewer requests to the server but a longer lag between the Transition of the Content Item and its publication.

The root element contains one or more `PSXPublish` elements. The `PSXPublish` element is a container for a configuration defining

- a Workflow

  The Workflow is specified by the `PSXWorkflowId` child element of `PSXPublish`. The value of this element is the ID of the Workflow.

- a Workflow Transition

  The Transition is specified by the `PSXTransitionId` child element of `PSXPublish`. The value of this element is the ID of the Transition.

- an Edition

  The Edition to run when a Content Item is Transitioned in the Workflow specified by the `PSXWorkflowId` element using the Transition specified by the `PSXTransitionId` element. The Edition is specified in the `PSXEdition` child element of `PSXPublish`. Any Edition can be specified, but typically an Incremental Edition is used.

  (specified by

For example:

```
<PSXConfig polling-time="1500">
      <PSXPublish>
```

```
          <PSXWorkflowId>1</PSXWorkflowId>
          <PSXTransitionId>5</PSXTransitionId>
          <PSXEdition>301</PSXEdition>
         </PSXPublish>
         <PSXPublish>
          <PSXWorkflowId>1</PSXWorkflowId>
          <PSXTransitionId>9</PSXTransitionId>
          <PSXEdition>301</PSXEdition>
         </PSXPublish>
   </PSXConfig>
```

This XML defines two configurations. The first configuration runs the Edition with the ID "301" when the Transition with the ID "5" is performed in the Workflow with the ID "1". The second configuration runs the Edition with the ID "301" when the Transition with the ID "9" is performed in the Workflow with the ID "1". The system will attempt to publish the Editions every 1500 milliseconds (1.5 seconds).

Class Name;

com.percussion.workflow.PSPublishContent

Resource File:

rxconfig/Workflow/publish.properties

Interface:

com.percussion.extension.IPSWorkflowAction

Parameters:

None.

sys_TouchParentItems

Name:

sys_TouchParentItems

Context:

Java/global/percussion/extensions/general/

Description:

This action touches all "parent" (Owner) items of the current item in Relationships whose Category is Active Assembly. It finds all Ancestors of the Content Item in Active Assembly Relationships and updates them by putting the current date/time and current user name in the CONTENTLASTMODIFIEDDATE and CONTENTLASTMODIFIER columns of the CONTENTSTATUS table.

This exit uses the following resources in the sys_ceDependency application:

- parents.xml query - this resource must have a "pipe name" of parents.
- touchitem.xml - an update resource (with a pipe name of touchitem. this resource updates the CONTENTSTATUS table. The only parameter of touchitem.xml is sys_contentid. This parameter specifies a list of content IDs as a {link java.util.ArrayList ArrayList} object.

Class name:

com.percussion.extensions.general.PSTouchParentItems

Resource file:

classes

Interface:

com.percussion.extension.IPSWorkflowAction

Parameters:

None

CHAPTER 5

Publishing Reference

Publishing is the final phase of the Content Management process. Publishing extracts Content Item data from the Repository, merges it with formatting to produce a final output, and saves the final output to its delivery location.

The first section of this chapter outlines the logical architecture and processing of the Publishing engine. The second section is a reference to the extensions used in Publishing.

Logical Architecture and Processing

This section is comprised of two topics. The first describes the logical architecture of the Publishing engine. The second describes publishing processing.

Logical Architecture

At the highest level, the logical architecture of publishing consists of the publishing engine, which resides within the Rhythmyx server; and a set of configurations that determine what content will be output and the target location for the output content.

The publishing engine consists of publishing jobs and one or more instances of the Publishing Handler. The publishing job communicates with the Publishing Handler via a publishing queue. The Publishing handler returns results to the publishing job via a status queue.

The configurations include:

- A set of Site registrations.

  A Site registration defines a location where output will be published when publishing to a file system. The output location may be a directory location or an FTP site. (In Database Publishing, the output location is defined in the Templates.) The Site registration also defines the Delivery Handler that deliver the published output to the target location.

- A set of Content Lists

  A Content List is a named configuration that is submitted to a servlet that generates the list of Content Items to publish. The key data of the Content List are:

  - The Content List Generator

    A Content List Generator is an extension that actually generates the list of Content Items to publish. In most cases, the sys_SearchGenerator is used. This generator uses a Java Content Repository (JCR) query to generate the list of Content Items to publish. The generator sys_SelectedItemsGenerator is used in Content Lists for on-demand publishing.

    Note that you can write your own Content List Generator extensions. For details, see ***Content List Generators*** (on page 126).

  - The Item Filter

    An Item Filter is a set of Filter Rule extensions that filter the list of Content Items generated by the Content List Generator to produce a final list of Content Items to be published. Rhythmyx is shipped with a number of standard Item Filters and Filter Rules.

    Note that you can write your own Filter Rule extensions. For details, see ***Item Filters and Filter Rules*** (on page 144).

- The Template Expander

  A Template Expander is an extension that generates the list of Templates to publish. The sys_SiteTemplateExpander publishes all Templates associated with the Site being published. The sys_ListTemplateExpander published only the Templates specified.

  Note that you can write your own Template Expander extensions. For details, see ***Template Expanders*** (on page 127).

- A set of Editions

  An Edition specifies a set of one or more Content Lists and the Site to which they will be published.

- A set of Delivery Handlers

  A Delivery Handler is an extension that delivers assembled content to an output location. Rhythmyx includes a set of standard Delivery Handlers for common output targets (file system, database, FTP, and secure FTP), but you can also write your own Delivery Handlers. For details, see ***Delivery Types*** (see "Delivery Handlers" on page 129).

The publishing process also invokes the Assembly engine.

The following graphic illustrates the logical architecture of the Publishing engine:

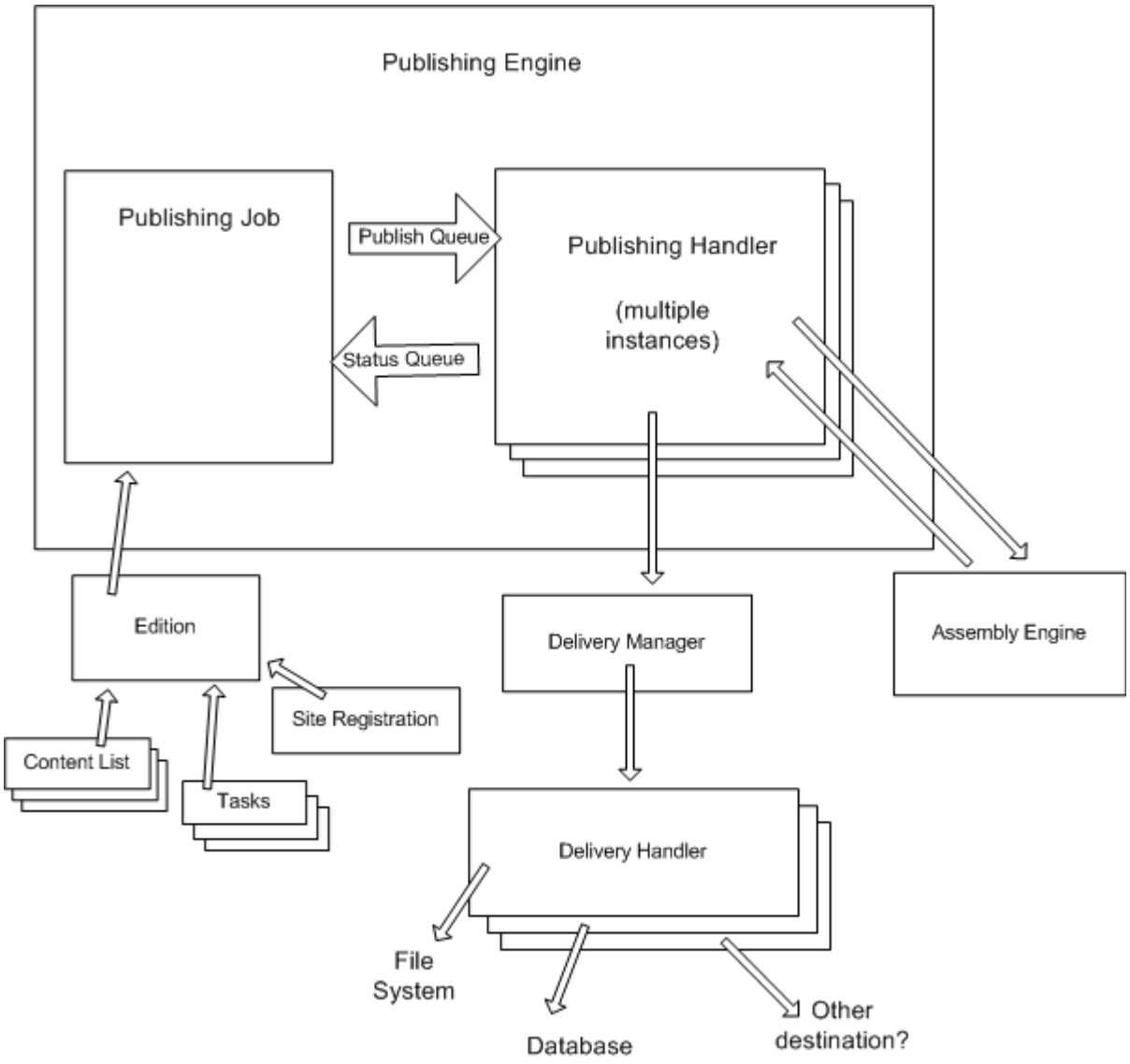


Figure 28: Publishing Architecture

Publishing Processing

Publishing processing begins when an Edition is submitted to be published. The request may be submitted manually or it may be generated by the server as a scheduled task. When the Rhythmyx server receives the publish Edition request, it generates a publishing job, which manages the remaining publishing processing. Each publishing job is initiated with a priority. Publishing jobs with a higher priority will interrupt jobs with a lower priority. Once processing of the higher-priority job is complete, the lower priority job resumes.

The publishing job first runs any Editions Tasks that are defined as pre-Publishing tasks in the order in which they are specified. If any task fails, processing stops unless the task is flagged as "Continue on failure", in which case processing continues. If processing stops at this point, no content is delivered.

The publishing job next invokes the Content Lists associated with the Edition to generate a list of Content Items to publish. In most cases, the sys_SearchGenerator is used. This Content List generator uses a JCR query to select the Content Items to publish. Rhythmyx also includes another Content List generator, sys_SelectedItemsGenerator, which is used for on-demand publishing. If these Content Lists generators do not provide the required functionality, implementers can write their own Content List generators. For details, see ***Content List Generators*** (on page 126).

The initial list of Content Items is then submitted to an Item Filter. An Item Filter is an ordered set of Filter Rule Extensions. The list of Content Items is submitted to each rule in turn, and the filtered set of Content Items is then submitted to the next Filter Rule. Rhythmyx is shipped with a set of standard Filter Rules, but you can also write your own Filter Rule extensions if you need different functionality. For details, see ***Item Filters and Filter Rules*** (on page 144).

The final filtered list of Content Items is then submitted to the Template Expander. A Template Expander is an extension that generates a list of Templates to publish for each Content Item. The result may include one, several or even zero Templates for each Content Item. Two standard Template Expanders are shipped with Rhythmyx:

- sys_SiteTemplateExpander (publishes all Templates associated with the Site)
- sys_ListTemplateExpander (Publishes only the listed Templates)

Note that you can also write you own Template Expanders if the default Template Expanders do not provide the desired functionality. For details, see ***Template Expanders*** (on page 127).

The Content List generator then formats the final list of Content Items into an XML document and sends it to the publishing job. The publishing job then sets up a queue of Content Items to be published by the Publishing Handler. Several instances of the Publishing Handler are generated. The exact number of instances depends on the number of CPUs Rhythmyx can use. If only one CPU is available, a handful of Publishing Handler instances may be created; a more powerful system with eight or sixteen CPUs may process dozens of instances of the Publishing Handler.

When the Publishing Handler receives a Content Item, it first submits the assembly URL to the assembly engine. After receiving the assembled Content Item returned from the assembly engine, the Publishing Handler sends the assembled Content Item to the Delivery Manager. The Delivery Manager sends the Content Item to the correct Delivery Handler. A Delivery Handler is a Rhythmyx extension that delivers assembled Content Item to the final output location. Rhythmyx is shipped with four standard Delivery Handlers:

- File System
- FTP
- SFTP
- Database

If the standard Delivery Handles do not provide the needed functionality, you can write your own Delivery Handler. For details, see ***Delivery Handlers*** (on page 129)..

The Publishing Handler returns the results of the processing to the publishing job through the status queue. At the highest level, three results are possible for any specific Content Item:

- The Content Item is published successfully (it is successfully assembled and delivered to the specified output location).
- Assembly of the Content Item may fail.
- Delivery of the Content Item may fail.

Publishing jobs can be cancelled. A cancellation takes priority over all other processing. When a job is cancelled, all outstanding processing for that job is halted. All successfully published Content Items are held in memory until processing of the last Content Item in the job is complete, at which point all Content Items are delivered. Thus, a cancelled job does not result in a partially published output.

Once processing of the last Content Item is complete and the published output is delivered to the target location, the publishing job runs any Edition Tasks that are defined as post-Publishing Tasks. Like pre-Publishing tasks, these tasks are run in the order specified, and if a task fails processing stops unless the task is flagged as "Continue on Failure".

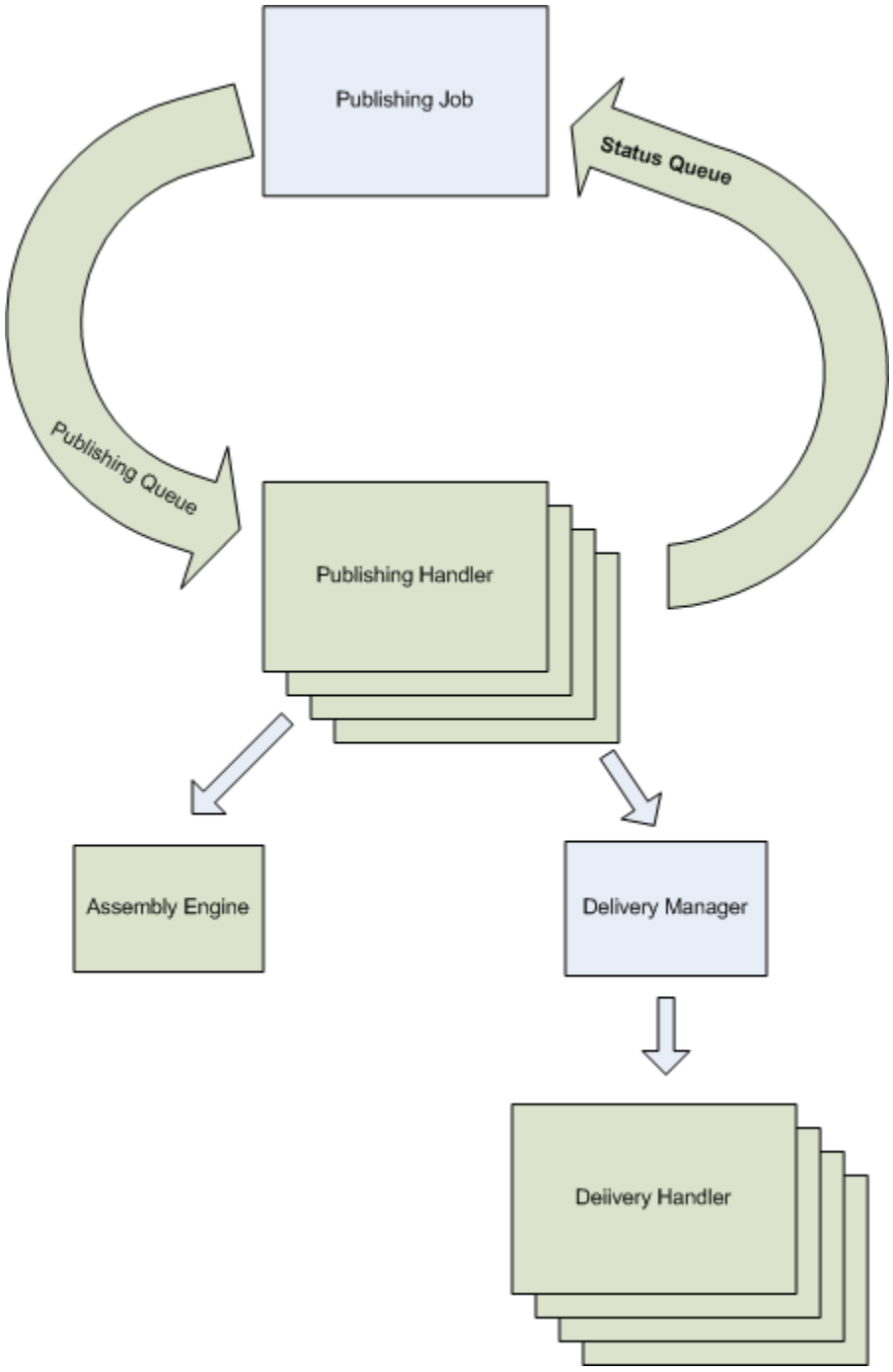


Figure 29: Publishing Processing

Demand Publishing

Demand Publishing processing is performed by a servlet designed specifically for that purpose. The Publish Now Menu option sends its request to this servlet. The request must include either a Site ID or an Edition ID. If an Edition ID is included, the specified Edition is run. If a Site ID is included, the servlet searches the Editions associated with the specified Site to find an Edition that includes only one Content List, which uses the sys_SelectedItemsGenerator. If multiple Editions are found that meet these criteria, a warning is written to the log and an Edition is selected arbitrarily.

As installed, the Publish Now Menu Entry is configured to include the Site ID; the command configuration includes the sys_siteid, with the value derived from the binding variable $sys_siteid.

If the sys_SelectedItemsGenerator does not provide functionality you want in your implementation, you can implement and use a different Content List Generator for Demand Publishing. To use your custom Content List Generator, add the HTML parameter sys_DemandPublishingGenerator to the Publish Now Menu Entry. The value of this parameter should be the fully-qualified extension name of your custom Content List Generator.

Configuring Unpublish Flags

Unpublish flags specify the value of the Publishable field of a Workflow State that indicates the Content Items in that State should be unpublished. These values are case-insensitive alphabetical characters that match the Value of a Publishable Keyword Choice, as illustrated below:

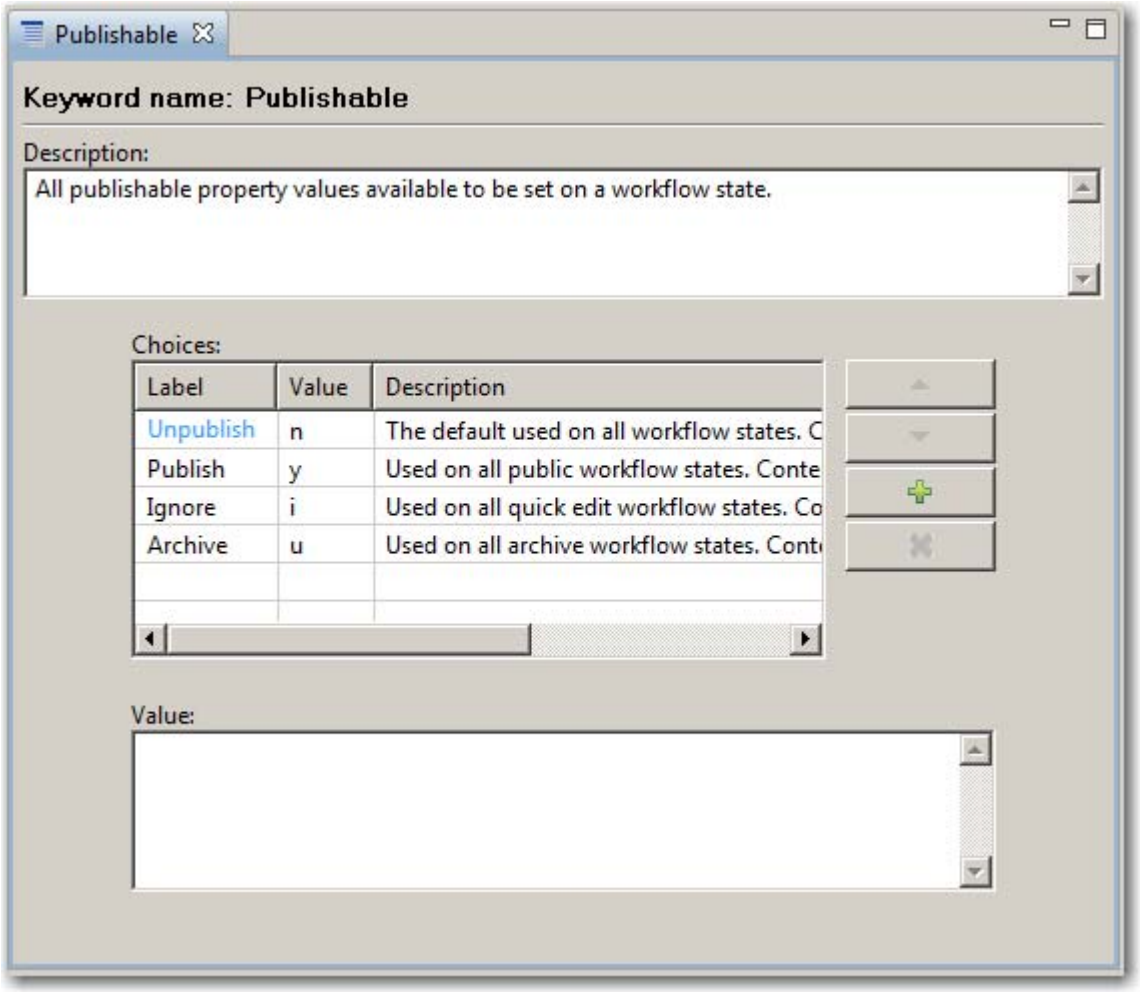


Figure 30: Publishable Keyword

The default unpublish flag is *u*. which is the value of the Archive Publishable Keyword Choice.

To specify multiple flags, enter multiple characters, separated by commas.

When you add a new unpublishable flag, you should also add a new Choice to the Publishable Keyword, whose value matches the character you specified. You should also define a Workflow State that uses that Keyword Choice in the Publishable field.

Publishing Extensions

This section documents extension types used only in publishing:

- Content List Generators
- Template Expanders
- Publishing plugins

The following extension types, which are used in both assembly and publishing, are documented elsewhere:

- JEXL functions
- Item Filter rules
- Location Scheme Generators

Content List Generators

Content List Generators generate a list of IDs (GUIDs) of Content Items to publish.

Content List Generators must implement the interface IPSContentListGenerator. (NOTE: The implementation must be thread safe; for details see ***General Requirements of Extensions*** on page 180.)

sys_PublishedSiteItems

This Content List Generator generates a list of all Content Items published on a Site. Must be used in conjunction with the sys_SiteTemplateExpander. Usually used for unpublishing.

Class Name

com.percussion.services.publisher.impl.PSSiteItemsGenerator

Interface

com.percussion.services.publishing.IPSContentListGenerator

Context

global/percussion/system

Parameters

None

sys_SelectedItemsGenerator

This Content List Generator generates a list of Content Items to publish based on a set of selected Content Items. It is used in on-demand publishing.

Class Name

com.percussion.services.publisher.impl.PSSelectedItemsGenerator

Interface

com.percussion.services.publishing.IPSContentListGenerator

Context

global/percussion/system

Parameters

None

Template Expanders

Template Expanders generate a list of Templates to publish for each Content Item ID (GUID) input.

Template Expanders must implement the interface IPSTemplateExpander. (NOTE: The implementation must be thread safe; for details see ***General Requirements of Extensions*** on page 180.)

NOTE: Percussion Software strongly recommends that you seek assistance from Percussion Professional Services Organization before implementing a custom Template Expander.

sys_ListTemplateExpander

This Template Expander assigns the Templates specified in the template parameter to the Content Items input. The specified Templates are assigned regardless of the Publish options specified for the Template or of the visibility of the Template to the Site being published. If none of the specified Templates is valid for an input Content Item ID, then no assembled output will be published for that Content Item.

Class Name

com.percussion.services.publisher.impl.PSListTemplateExpander

Interface

com.percussion.services.publishing.IPSTemplateExpander

Context
global/percussion/system

Parameters

Name	Data Type	Description
template	String	(Required) Comma-separated list of Templates to publish.

sys_SearchGenerator

This Content List Generator generates a list of Content Items to publish based on a JCR query.

Class Name
com.percussion.services.publisher.impl.PSQueryContentListGenerator

Interface
com.percussion.services.publishing.IPSContentListGenerator

Context
global/percussion/system

Parameters

Name	Data Type	Description
query	String	(Required) The JSR-170 "SQL" query to use to generate the base list of Content Items to be published.

sys_SiteTemplateExpander

This Template Expander assigns the Page Templates of the target Site to the Content Items input.

Class Name

com.percussion.services.publisher.impl.PSSiteTemplateExpander

Interface

com.percussion.services.publishing.IPSTemplateExpander

Context

global/percussion/system

Parameters

Name	Data Type	Description
default_template	String	If the value is *all* or is unspecified, all default Templates of the specified Site will be published. If the value is *dispatch*, only default Templates that use the Dispatch Assembly plugin will be published. If the value is *none*, no default Templates will be published.

Delivery Handlers

A Delivery Handler delivers assembled Content Items to the final output location. Implementation of a Delivery Handler requires three steps:

1. Write the Delivery Handler code.

   Delivery Handlers must implement the interface IPSDelivery Handler. (NOTE: The implementation must be thread safe; for details see ***General Requirements of Extensions*** on page 180.)

2. Register the Delivery Handler as a Spring bean.

   For details about registering a Spring bean in Rhythmys, see ***Spring Configurations*** (on page 171).

3. Create a Delivery Type registration for the Delivery Handler.

   A Delivery Type exposes the Delivery Handler extension to the Publishing engine. Multiple Delivery Types can registered for the Delivery Handler. To register a Delivery type:

   a) In Content Explorer, choose the Publishing Design tab.

   b) Click the Delivery Types Link.

      Content Explorer displays the Delivery Types list.

   c) In the Menu bar, choose *Action* > *Create Delivery Type*.

      Content Explorer display the Delivery Type editor in the View and Edit pane.

d) Enter a Name for the Delivery Type. The name must begin with a letter, and can contain any alphanumeric characters, underscores, hyphens, or dots (periods). Optionally, enter a free-form Description of the Delivery Type.

e) Enter the Spring Bean Name of the bean that configures the Delivery Handler used by the Delivery Type. This value will be validated when the Delivery Type registration is saved; therefore, the bean must be configured before you create the Delivery Type registration.

f) If you want to assemble Content Items when unpublishing them, check the Assemble Item for Unpublish checkbox.

g) Click the [**Save**] button to save the Delivery Type registration.

CHAPTER 6

Shared Features

This chapter documents several Rhythmyx features that are shared by multiple Rhythmyx engines:

- Java Expression Language (JEXL)
- Java Content Repository (JCR) queries
- Item Filters and Filter Rules
- Location Schemes and Location Scheme Generators
- Scheduled Tasks

Java Expression Language (JEXL)

Rhythmyx embeds the Java Expression Language (JEXL) engine to provide scripting functionality.

- In the Assembly engine, JEXL is used to define and process Bindings. For details, see "Bindings" in the *Rhythmyx Implementation Guide*.
- In the Publishing Engine, JEXL is used to define locations for Location Scheme Generators. For details, see ***Link Generation and Context*** (on page 147) and "Defining Contexts and Location Schemes" in the *Rhythmyx Implementation Guide*.

For details about JEXL, see ***http://jakarta.apache.org/commons/jexl/***.

JEXL Extensions

Standard JEXL functions are available in Rhythmyx. For details about these functions, see the Velocity tools documentation (***http://jakarta.apache.org/velocity/tools/index.htm***). Rhythmyx also includes a number of JEXL function extensions to support both assembly and location scheme generation.

If you need additional functionality, you can implement custom JEXL function extensions. JEXL function extensions must implement the interface IPSJexlExpression. (NOTE: The implementation must be thread safe; for details see ***General Requirements of Extensions*** on page 180.) They usually also extend the PSJexlUtilBase class:

```
public class PSJexlExample extends PSJexlUtilBase implements
IPSJexlExpression
```

In addition to importing these two classes, you must also import IPSJexlMethod and IPSJexlParameter.

All JEXL extensions use Java 5 Annotation with the @IPSJexlMethod and @IPSJexlParam annotations:

```
@IPSJexlMethod(description="my method does something useful",
      params={@IPSJexlParam(name="part1",description="the first part"),
      @IPSJexlParam(name="part2", description="the second part")})
      public String myMethod(String part1, String part)
```

Note that the @IPSJexlParam annotations are an Array; they must be enclosed in curly braces and separated by commas.

Assembly Utilities

The methods of this function provide data for use in assembly. The following methods are available:

- $rx.asmhelper.assemble
- $rx.asmhelper.isAASlot (slot)
- $rx.asmhelper.getPopupMenu
- $rx.asmhelper.getSingleParamValue
- $rx.asmhelper.getTidiedContent
- $rx.asmhelper.getTitle ($sys.item.guid)
- $rx.asmhelper.combine
- $rx.asmhelper.childValues

For details see "$rx.asmhelper" in the *Rhythmyx Implementation Guide*.

Class Name

com.percussion.services.assembly.jexl.PSAssemblerUtils

Interface

com.percussion.extension.IPSJexlExpression

Context

global/percussion/system/

Parameters

None

Code and Decode Utilities

The methods of this function encode and decode data.

- $rx.codec.base64Decoder
- $rx.codec.base64Encoder
- $rx.codec.escapeForXml
- $rx.codec.decodeFromXml

For details see "$rx.codec" in the *Rhythmyx Implementation Guide.*

Class Name

com.percussion.services.assembly.jexl.PSCodecUtils

Interface

com.percussion.extension.IPSJexlExpression

Context

global/percussion/system/

Parameters

None

Keyword Utilities

The methods of this function provide access to Keyword data.

- $rx.keyword.keywordSelectChoices
- $rx.keyword.keywordChoices
- $rx.keyword.getLabel

For details see "$rx.keyword" in the *Rhythmyx Implementation Guide*.

Class Name

com.percussion.services.assembly.jexl.PSKeywordUtils

Interface

com.percussion.extension.IPSJexlExpression

Context

global/percussion/system/

Parameters

None

Conditional Processing Utilities

NOTE: This function is deprecated. JEXL expressions that use this binding should be rewritten to use the JEXL `if...else` conditional function instead.

The method of this function is used to evaluate conditional statements.

For details see "$rx.cond" in the *Rhythmyx Implementation Guide*.

Class Name

com.percussion.services.assembly.jexl.PSCondUtils

Interface

com.percussion.extension.IPSJexlExpression

Context

global/percussion/system/

Parameters

None

Database Utilities

The method of this function is used in database publishing.

For details see "$rx.db" in the *Rhythmyx Implementation Guide*.

Class Name

com.percussion.services.assembly.jexl.PSDbUtils

Interface

com.percussion.extension.IPSJexlExpression

Context

global/percussion/system/

Parameters

None

Document Utilities

The methods of this function process XML and HTML documents.

- $rx.doc.getDocument(url)
- $rx.doc.getDocument(url,user,password)
- $rx.doc.extractBody

For details see "$rx.doc" in the *Rhythmyx Implementation Guide*.

Class Name

com.percussion.services.assembly.jexl.PSDocumentUtils

Interface

com.percussion.extension.IPSJexlExpression

Context

global/percussion/system/

Parameters

None

Extension Utilities

The method of this function allows you to call an extension.

For details see "$rx.ext" in the *Rhythmyx Implementation Guide*.

Class Name

com.percussion.services.assembly.jexl.PSExtensionUtils

Interface

com.percussion.extension.IPSJexlExpression

Context

global/percussion/system/

Parameters

None

GUID Utilities

The method of this function allows you to retrieve GUIDs.

For details see "$rx.guid" in the *Rhythmyx Implementation Guide*.

Class Name

com.percussion.services.assembly.jexl.PSGuidUtils

Interface

com.percussion.extension.IPSJexlExpression

Context

global/percussion/system/

Parameters

None

Internationalization Utilities

The method of this function is used to retrieve internationalized and localized data.

For details see "$rx.i18n" in the *Rhythmyx Implementation Guide*.

Class Name

com.percussion.services.assembly.jexl.PSI18nUtils

Interface

com.percussion.extension.IPSJexlExpression

Context

global/percussion/system/

Parameters

None

Link Utilities

The methods of this function allow you to manipulate links.

- $rx.link.addParams
- $rx.link.getAbsUrl
- $rx.link.getRelUrl

For details see "$rx.link" in the *Rhythmyx Implementation Guide*.

Class Name

com.percussion.services.assembly.jexl.PSLinkUtils

Interface

com.percussion.extension.IPSJexlExpression

Context

global/percussion/system/

Parameters

None

Location Utilities

The methods of this function allow you to generate hypertext links.

- $rx.location.generate
- $rx.location.generateToPage
- $rx.location.getFirstDefined
- $rx.location.siteBase($sys.site)

For details see "$rx.location" in the *Rhythmyx Implementation Guide*.

Class Name

com.percussion.services.assembly.jexl.PSLocationUtils

Interface

com.percussion.extension.IPSJexlExpression

Context

global/percussion/system/

Parameters

None

Navigation Utilities

The methods of this function are used in processing Managed Navigation. They are only valid when applied to nodes returned from the Managed Navigation Slot Content Finder.

- $rx.nav.findProperty
- $rx.nav.findNode

For details see "$rx.nav" in the *Rhythmyx Implementation Guide.*

Class Name

com.percussion.services.assembly.jexl.PSManagedNavUtils

Interface

com.percussion.extension.IPSJexlExpression

Context

global/percussion/system/

Parameters

None

Pagination Utilities

The methods of this function are used when paginating assembled Content Items. The following methods are available:

- $rx.paginate.fieldContentPageCount
- $rx.paginate.getFieldPage
- $rx.paginate.getSlotPage
- $rx.paginate.slotContentPageCount

For details see "$rx.pagination" in the *Rhythmyx Implementation Guide*.

Class Name

com.percussion.services.assembly.jexl.PSPaginateUtils

Interface

com.percussion.extension.IPSJexlExpression

Context

global/percussion/system/

Parameters

None

String Utilities

The methods of this function return session IDs that can be returned to Rhythmyx when calling Rhythmyx applications or other URLs via HTTP.

- $rx.session.getJSessionID
- $rx.session.getSessionID

For details see "$rx.session" in the *Rhythmyx Implementation Guide*.

Class Name

com.percussion.services.assembly.jexl.PSSessionUtils

Interface

com.percussion.extension.IPSJexlExpression

Context

global/percussion/system/

Parameters

None

String Utilities

The methods of this function allow you to allow you to manipulate string values.

- $rx.string.stringTo Map
- $rx.string.equalNumbers
- $rx.string.extractNumber

For details see "$rx.string" in the *Rhythmyx Implementation Guide*.

Class Name

com.percussion.services.assembly.jexl.PSStringUtils

Interface

com.percussion.extension.IPSJexlExpression

Context

global/percussion/system/

Parameters

None

Java Content Repository

Rhythmyx uses the Java Content Repository (JCR) to retrieve Content Item data from the Repository and to represent it for assembly.

In Assembly, the Content Item data is submitted to the Assembly engine as a JCR Node and Property object; for details see ***Assembly Processing*** (on page 79). JCR queries are also used to generate lists of Content Items automatically when assembling Automated Slots. The JCR query generates the list of Content Items to include in the Slot. For details, see the topics "Creating an Automated Slot" and "Writing Automated Slot Queries" in the *Rhythmyx Implementation Guide*.

In publishing, JCR queries are used by the Query Content List Generator, which is the standard Content List Generator.

Rhythmyx only supports the JCR functionality required to support assembly. Only Content Items can be accessed as nodes; Folders cannot be accessed as JCR nodes. JCR data methods are supported to provide read-only access to Nodes and properties. Behavioral and set methods are not supported. If called, these methods throw either an UnsupportedOperationException or a JCR-specific exception, such as a LockException.

A Content Item is represented as a JCR node. Use node methods to access or operate on a Content Item as a whole., The fields in the Content Item are represented as properties of the Content Item node. Use property methods to access and operate on Content Item fields. Simple child content (child content stored in a separate table but edited within the Content Editor) are represented as multi-valued properties of the Content Item node. Use multivalued Property methods to access and operate on these fields. Complex child content (child content edited in a popup Detail Editor) are represented as child nodes of the parent Content Item. Use standard node methods to access and operate on these fields.

Read access is available for NodeDefinition, Node Type, and Property Definition. UUIDs of nodes are not globally unique.

Since a Rhythmyx Content Item can have multiple parents (in Rhythmyx terms, it can be the Dependent in multiple Relationships), the methods getParent and getDepth cannot be supported. The only exception is Managed Navigation nodes, which do support these methods.

Field and Content Type names are transformed by adding "rx:" as the namespace. Content Type names are case-insensitive, but field names are case-sensitive. Space characters in field and Content Type names are replaced with underscore characters ("_") because spaces are invalid in a JCR query.

The JCR Query Manager is partially implemented. Row, Query, and QueryResult are implemented, but you cannot store queries. Only the SQL syntax is supported. The XPath syntax is implemented but not supported. Full-text search queries are not implemented. To query all Content Types, use nt:base.

For complete details about the Java Content Repository, see the JSR-170 spec at ***http://www.jcp.org/en/jsr/detail?id=170***.

NOTE: JCR queries cannot be extended.

Item Filters and Filter Rules

Item Filters filter a list of Content Items to be published. An Item Filter is a named set of Filter Rules. The Filter Rules are extensions that perform the actual filter processing. For example, the following standard Filter Rules are shipped with Rhythmyx:

- Filter by Folder Paths

  Filters based on the path of the Content Item.

- Filter by Publishable Flag

  Filters based on value of the Publishable Flag of the State of the Content Item.

- Filter by Site Folder

  Validates that the target Content Item for a link exists in the location specified.

Filter Rules are ordered within an Item Filter, and Filter Rules defined with a higher precedence are run before those with a lower precedence. Each Filter Rule in an Item Filter operates only on Content Items that have passed previous Filter Rules.

Item Filters perform two functions:

- Link filtering

  When assembling a page, link filtering prevents the broken links that point from a Public Content Item to related Content Items that are not public by preventing the assembly of these links.

  Link filtering also prevents Snippets of Content Items that are not public from being assembled into a Public Content Item when assembling a published Page Template.

NOTE: In Rhythmyx Version 5.7 and earlier, this functionality was known as Authorization Type, or Auth Type. Auth Type functionality has been subsumed into Item Filter functionality in Rhythmyx Version 6.0.

- Content List filtering

  During Publishing, after a Content List Generator generates the initial list of Content Items to publish, an Item Filter filters the initial list to generate a final list of Content Items to publish. For example, a standard Item Filter used during publishing is the Filter by Publishable Flag. This filters the initial list of Content Items (usually all Content Items on a Site) to return only those Content Items that are in a State flagged as Publishable.

Filter Rule Extensions

Filter Rule extensions perform the actual filtering processing. A Filter Rule takes a list of Content Items as its input, applies the rules to that list and returns a list of Content Items that meet the specified criteria.

Filter Rule extensions must implement the interface IPSItemFilterRule. (NOTE: The implementation must be thread safe; for details see ***General Requirements of Extensions*** on page 180.)

sys_filterByFolderPaths

Filters the submitted list of Content Items based on the Folders specified. Only Content Items that exist in the specified Folder pass the filter.

Class Name

com.percussion.services.filter.impl.PSFolderPathFilter

Interface

com.percussion.services.filter.IPSItemFilterRule

Context

global/percussion/itemfilter/

Parameters

Name	Data Type	Description
sys_folderPaths	String	One or more Folder paths (use semicolons to separate multiple Folder paths). The paths may contain one or more wildcards (%) which have the same semantics as in SQL. If the path does not contain wildcards, the path must match exactly. To return the contents of a Folder and all of its descendants, you must specify the path twice, first without the wildcard, then with the wildcard. For example, to return all of the contents of the EnterpriseInvestments Site, you would have to specify "//Sites/EnterpriseInvestments/,//Sites/EnterpriseInvestments/%".

sys_filterByPublishableFlag

Filters the submitted list of Content Items based on value of the Publishable Flag of the State of the Content Item.

Class Name

com.percussion.services.filter.impl.PSPublishableStateFilter

Interface

com.percussion.services.filter.IPSItemFilterRule

Context

global/percussion/itemfilter/

Parameters

Name	Data Type	Description
sys_flagValues	String	Content Valid values of the Content Item State to use when filtering the list of Content Items. Use commas to separate multiple values.

sys_filterBySiteFolder

Used for cross-site linking. Checks the Folder specified to ensure that the target Content Item for the link being published exists in that Folder, then checks to confirm that the Path to that Folder includes the Site Root for the specified Site. If this test fails, checks whether the Content Item exists on the specified Site at all. If the Content Item passes any of these tests, the matching path will be published. Otherwise, the link is not published.

Class Name

com.percussion.services.filter.impl.PSSiteFolderFilter

Interface

com.percussion.services.filter.IPSItemFilterRule

Context

global/percussion/itemfilter/

Parameters

None

Link Generation and Context

Since Rhythmyx decouples Content Management from delivery, a mechanism is required to generate links (such as hypertext links between HTML pages, or links to image files and CSS files). Links that are valid when previewing a Content Item on the Rhythmyx server will not be valid when viewing a final published page from a Web server.

Location Schemes build these links. In Rhythmyx Version 6.0, links are generated based on Java Expression Language (JEXL) expressions. (In earlier version, UDFs provided link generation functionality. In both cases, a Location Scheme builds a path to use in a link.) The output of the JEXL expression defines the path to the location.

Location Schemes are defined for different output environments, known in Rhythmyx as Contexts. Each Context, which is identified using an integer, defines an output environment that has a different path for links. For example, at its simplest, a Rhythmyx implementation will include two Contexts. The first, Context 0, is the default Preview Context for Rhythmyx; it defines all links used when previewing Rhythmyx Content Items in relation to the Rhythmyx installation Root. The second Context (in this case, we will call it Context 1) is the Publish Context. It defines all links in relation to the root of the Web application to which Rhythmyx page output is published.

In many cases, the links within published output use a different path than that to which the output is published. In that case, you will need two Contexts, one to define the delivery location and one to define the links within the published output. The FastForward implementation uses this technique.

Using Context also allows you to define different formatting for your outputs. You can choose to link to different stylesheets to produce different output renderings in different output locations, or define Context Variables that allow you to change the rendering of the output depending on Context.

Location Schemes are used during assembly to define the paths for hypertext links and links to images and static site files such as CSS files. Location Schemes are used during publishing to define the location to which the published output will be delivered.

Location Scheme Generator Extensions

The default Location Scheme Generator in Rhythmyx Version 6.0 is sys_JexlAssemblyLocation. This extension builds a location path by evaluating a Java Expression Language (JEXL) expression. (NOTE: Several legacy Location Scheme Generation Extensions are also shipped with Rhythmyx. For details about these extensions, see ***Legacy Extension Reference*** on page 190.

If none of these extensions meet your needs, you can implement a new Location Scheme Generator Extension. Location Scheme Generator extensions must implement the interface IPSAssemblyLocation. (NOTE: The implementation must be thread safe; for details see ***General Requirements of Extensions*** on page 180.)

sys_JexlAssemblyLocation

Builds a delivery location by evaluating a Java Expression Language (JEXL) expression

Class Name

com.percussion.services.publisher.impl.PSJexlLocationGenerator

Interface

com.percussion.extension.IPSAssemblyLocation

Context

global/percussion/contentassembler/

Parameters

Name	Data Type	Description
expression	String	Required. The JEXL expression to be evaluated to create the delivery location

Scheduled Tasks

A scheduled task is processing run by Rhythmyx automatically according to a pre-defined schedule.

Rhythmyx uses the Quartz Enterprise Job Scheduler to provide scheduling processing. See ***http://www.opensymphony.com/quartz/*** for more details.

Scheduled Task extensions provide the task processing. The processing is defined by the extension, and need not include only Rhythmyx processing. The sys_runCommand Scheduled Task extension, for example, runs a native system command. Scheduled Task extensions use the interface IPSTask. (NOTE: The implementation must be thread safe; for details see ***General Requirements of Extensions*** on page 180.) These extensions must return an IPSTaskResult object, which includes the following properties:

- wasCompleted

  A boolean property that indicates whether task processing was completed successfully or failed.

- getProblemDescription

  A string property that describes the result of processing. The text should provide a description of the result that will be meaningful to a non-technical user. May be null if extension processing was completed successfully.

- getNotificationVariables

  A map of the binding variables returned by the extension. The key for each map property should be the name of the binding variable in the correct format ($variablename; for example, $sys.editionName). The value of the property is the value of the variable.

sys_purgePublishingLog

Purges publishing logs created more than the specified number of days in the past. The extension can be configured to archive logs before purging.

Class Name

com.percussion.services.schedule.impl.PSPurgePublishingLog

Interface

com.percussion.services.filter.IPSTask

Context

global/percussion/system/

Parameters

Name	Data Type	Description
numberOfDays	String	The number of days for which to preserve logs. Logs created more than the specified number of days in the past will be purged. Defaults to 30 if not specified. (Logs created more than thirty days in the past will be purged.)
enableArchive	String	If the value is *true*, logs will be archived before being purged. If the parameter is null or contains any other value, logs will not be archived before being purged.

sys_purgeScheduledTaskLog

Purges scheduled task logs created more than the specified number of days in the past.

Class Name

com.percussion.services.schedule.impl.PSPurgeScheduledTaskLog

Interface

com.percussion.services.filter.IPSTask

Context

global/percussion/system/

Parameters

Name	Data Type	Description
numberOfDays	String	The number of days for which to preserve logs. Logs created more than the specified number of days in the past will be purged. Defaults to 30 if not specified. (Logs created more than thirty days in the past will be purged.)

sys_runCommand

Runs the specified native system command.

Class Name

com.percussion.services.schedule.impl.PSRunCommand

Interface

com.percussion.services.filter.IPSTask

Context

global/percussion/system/

Parameters

Name	Data Type	Description
command	String	The command to run.

sys_runEdition

Publishes the specified Rhythmyx Edition.

Class Name

com.percussion.services.schedule.impl.PSRunEdition

Interface

com.percussion.services.filter.IPSTask

Context

global/percussion/system/

Parameters

Name	Data Type	Description
editionName	String	The name of the Edition to publish.

CHAPTER 7

System Issues

This chapter addresses technical issues involving the underlying system (Rhythmyx server, JBoss container, etc) rather than a specific engine of the Content Management System.

Custom Implementations

Rhythmyx can be extended by adding custom JSPs or servlets.

Before attempting any custom implementation, familiarize yourself with the Rhythmyx container, JBoss. A variety of resources are available for learning about JBoss, including the jboss.org Web site.

Implementing Custom Java Server Pages and Servlets

Customizations can be implemented as a separate web application from Rhythmyx, as a Rhythmyx web application, or as a Web Services client application.

Implementing a separate web application should be reserved for applications that cannot be implemented as a Rhythmyx web application, such as a third-party product. These web applications cannot use Rhythmyx server APIs; they must use Rhythmyx Web Services to access Rhythmyx functionality.

If your customization needs a user interface and needs to authenticate in Rhythmyx, implement it as a custom Rhythmyx Web application. These applications use Rhythmyx authentication and can access Rhythmyx functionality and data (such as user session data) directly. If you do not need a user interface, consider using Web Services to implement your customization as a Web Services client. For details about implementing Web Services, see the Rhythmyx Services Development Kit.

Custom Java Server Pages (JSPs) should be added to the directory `<Rhythmyxroot/AppServer/server/rx/deploy/rxapp.ear/rx-app.war/user/pages`. Any subdirectory structure below this directory is allowed. None of the contents of this directory are touched when the system is upgraded. JSPs in this directory require authentication unless their security configuration specifically allows anonymous access. (See below for details about security configuration.) These JSPs have access to session data and Rhythmyx server APIs.

Custom servlets should be implemented as dispatched Spring MVC Controllers. These MVC Controllers are configured in the file `<Rhythmyxroot/AppServer/server/rx/deploy/rxapp.ear/rx-app.war/WEB-INF/config/user/spring/user-Dispatcher-servlet.xml`. Specify any controller classes and URL mappings. Optionally, you can also specify any initialization parameters for the servlet as well. The configuration file includes example configurations. While you can update the mappings in the file directly, recommended practice is to create a separate XML file in the Dispatch directory (one level below the /rx-app.war/WEB-INF/config/spring/ directory), and add <include> tags to user-Dispatcher-servlet.xml. When running several "applications" on the Rhythmyx server, this practice isolates them from one another somewhat.

Recommended practice for adding Spring beans to the user dispatcher configuration is to define a bean configuration file for each MVC Controller bean in a subdirectory of `<Rhythmyxroot/AppServer/server/rx/deploy/rxapp.ear/rx-app.war/WEB-INF/config/user/spring/` and import these beans into the user dispatcher configuration. This practice averts conflicts between beans defined in the user dispatcher configuration. If the name of the file ends with the string *-servlet.xml*, it will be loaded into the dispatcher servlet application context and will inherit the Spring application context constructed from all of the other files in that directory. Files that do not end with the string *-servlet.xml* are loaded as user beans.

NOTE: You should not deploy a servlet by simply adding it to `<Rhythmyxroot/AppServer/server/rx/deploy/rxapp.ear/rx-app.war/WEB-INF/web.xml` because this file is a system resource that may be overwritten on upgrade and your deployed servlet will be lost.

When implementing form-based applications, recommended practice is to extend Spring's SimpleFormController. The form controller automates much of the routine processing, including creating a model bean for your form and binding all submitted values to this bean. A variety of view technologies is available, including Velocity templates. Recommended practice for most forms is to use simple JSP pages (with or without JSTL) and placing these pages in the WEB-INF/pages directory where they cannot be browsed outside of the form servlet.

Use the class PSDatasourceSessionFactoryBean to configure a Hibernate session factory based on a datasource configured in the Rhythmyx Server Administrator. For details, see the Javadoc for this method. Note that Hibernate mappings cannot be specified with wildcards. The Javadoc for PSDatasourceSessionFactoryBean describes this issue, or see the jboss.org Web site for details.

Obtaining the User and Session

Custom applications often need the Rhythmyx session ID and the authenticated user name.

The authenticated user name can be obtained from the standard getRemoteUser() method.

The Rhythmyx request is stored as an attribute on the ServletRequest. The Rhythmyx request contains the session ID, which is the only data needed for access to the Rhythmyx API. The PSO Toolkit provides a convenient method for obtaining the session ID:

```
public PSSessionUser(HttpServletRequest request)
{
      String user = request.getRemoteUser();
      String session = RxRequestUtils.getSessionId(request);
}
```

Handling PSItemStatus

A common issue encountered when writing Web applications in Rhythmyx is how to handle the call to prepareForEdit().

In most form-based Web applications, the client performs a GET to retrieve the content information for editing. Later, the user will submit the form to the server, usually with a POST.

If prepareForEdit() is not called until the form-submission request (the POST of modified data), it may be too late. Another user may have already started to modify the same Content Item (or Content Items). If you call prepareForEdit() when the user first opens the edit page (on the GET), the Content Item is checked out for the duration of the edit session.

In most applications, implementers opt to check out the Content Item on the GET rather than waiting for the POST. This practice requires, however, that you store the PSItemStatus information that was returned from prepareForEdit() until the the subsequent POST request. Two options are available for storing this data:

- The simplest solution is to pass the PSItemStatus object to the Web layer and store it as an attribute on the HttpSession. This approach works so long as session timeouts do not occur while the form window is open. If a session timeout occurs, the form will be renewed, but the checkout state will be lost.
- An alternative is to use EHCache, which is installed as part of the Hibernate Stack. Spring has an adapter for EHCache. See the documentation for `org,springframework.cache.ehcache.EhCacheFactoryBean`.

Implementing Transactional Services

In Rhythmyx, binary fields are always loaded when they are referenced, not when the Content Item is opened. Programs that need to manipulate binary fields should do so inside the Rhythmyx transaction. Rhythmyx uses Spring 2.0 and Hibernate 3.2 to maintain transactions, and you can take advantage of the support they provide. Creating a new transactional service allows you to extend the Rhythmyx transaction to include your own methods, which is the safest way to manipulate binary fields and other data that might be lazily loaded.

To add a new service:

1. Define a new interface that contains your service methods.
2. Build a new class that extends HibernateDaoSupport and implements your interface.
3. Create an XML file that defines your implementation as a Spring bean.
4. Build a locator class that extends PSBaseServiceLocator.

An example code package illustrating this process is available from the Percussion Forum (***http://forum.percussion.com/***).

Service Interface

The service interface methods should encapsulate all of the code that needs to be transactional. Processing contained in objects returned by your service will not be inside the transaction. The only restriction on the interface is that none of the methods can have the same name as any of the methods in HibernateDaoSupport.

Service Implementation

The services class must:

- implement the service interface you created; and
- extend org.springframework.orm.hibernate3.support.HibernateDaoSupport (which contains the support for the transaction manager.)

```
@Transactional
public class SampleServiceImpl extends HibernateDaoSupport
      implements SampleService
```

Note the `@Transactional` annotation, which tells Spring to execute the class inside the transaction.

Implementation class methods must be thread safe (they must contain appropriate synchronization methods for access to shared data.

Spring Bean

The Spring bean is defined in an XML file that is added to the user configuration directory (<Rhythmyxroot>/AppServer/server/rx/deploy/rxapp.ear/rxapp.war/WEB-INF/config/user/spring). The Spring bean must contains a reference to a Hibernate session factory *even if your service implementation class does not use Hibernate*. The session factory allows DAO support code to reference the transaction manager.

```
<bean id="mySampleServiceBean"
      class="com.percussion.pso.service.xact.impl.SampleServiceImpl">
      <property name="sessionFactory" ref="mySessionFactory" />
</bean>
```

You can include any additional properties required by your implementation class.

The beans file must also contain a Hibernate session factory. The standard way to include a Hibernate session factory is to use PSDatasourceSessionFactoryBean. This class automatically references the Rhythmyx internal session factory.

```
<bean id="mySessionFactory"

   class="com.percussion.services.datasource.PSDatasourceSessionFactoryB
ean">
      <!-- if your service uses any hibernate mappings, you can add them
here-->
</bean>
```

If your implementation class uses Hibernate, you many need to add further mappings to the session factory. Not that the ID used for this bean is not important, but it must match the reference in the implementation bean. Bean IDs can follow any naming convention, except that they must not being with the string "sys_"; that string is reserved for IDs of system objects.

Service Locator

The service locator allows classes outside of the server's Spring infrastructure (including JSP pages and servlets defined in the User Dispatcher Servlet context) to find your service. Create a services locator by extending PSBaseServiceLocator:

```
public class SampleServiceLocator extends PSBaseServiceLocator
      {
         public static SampleService getSampleService()
      {
         return (SampleService) getBean(SAMPLE_SERVICE_BEAN_
      }
//this must match the ID of the bean in the xml file.
public static final String SAMPLE_SERVICE_BEAN_NAME
="mySampleServiceBean";
}
```

The bean name constant can be any value, but it must match the ID of the bean in the XML file used to configure the Spring bean.

The name of the static "get" method should match the name of the service.

Deploying a Transaction Service

The recommended method of deploying code in Rhythmyx is to build a JAR file and add it to the directory `<Rhythmyxroot/AppServer/server/rx/deploy/rxapp.ear/rx-app.war/WEB-INF/lib`. Beans files, as noted earlier, should be added to the directory `<Rhythmyxroot/AppServer/server/rx/deploy/rxapp.ear/rx-app.war/WEB-INF/config/user/spring/`.

When you restart the server, the server will automatically load these files. The first time you restart, review the server log carefully for any references to the server bean and server implementation class. Major errors may cause exceptions; if severe enough, these exceptions may prevent the server from starting. More subtle errors may result in the service not being loaded.

To call the services, call the services locator:

```
SampleService sample = SampleServiceLocator.getSampleService();
```

The example code package includes a test JSP that illustrates a call to a transaction service.

Extending Java Server Faces Page Flows

Rhythmyx uses Apache MyFaces.

To add a new JSF page flow to Rhythmyx, add a new JSF configuration to the file `<Rhythmyxroot>/AppServer/server/rx/deploy/rxapp.ear.rxapp.war/WEB-INF/config/user/faces/faces-config.xml`. The file `<Rhythmyxroot>/AppServer/server/rx/deploy/rxapp.ear.rxapp.war/WEB-INF/faces-config.xml` should not be modified.

The names of system JSF beans begin with the prefix "sys_". This prefix should not be used when naming custom JSF beans. However, best practice is to use a consistent naming convention for your project beans. For information about naming conventions, see "Design Object Naming Conventions" in the *Rhythmyx Implementation Guide*.

File Locations

Any classes required by a custom implementation should be added to the directory `<Rhythmyxroot>/AppServer/server/rx/deploy/rxapp.ear.rxapp.war/WEB-INF/classes`.

Any .jar files required by a custom implementation should added to the directory `<Rhythmyxroot>/AppServer/server/rx/deploy/rxapp.ear.rxapp.war/WEB-INF/lib`.

The working directory of the Rhythmyx server installation is the root directory, but the use of a specific working directory in future versions of Rhythmyx is not guaranteed. Use the service IPSRhythmyxInfo to determine the location of the root directory if your custom implementation needs to refer to files not located in the directory `rxapp.war`.

NOTE for upgraded systems: Custom implementations in systems upgraded from Rhythmyx Version 5.7 and earlier should be updated to use the service IPSRhythmyxInfo to refer to files. Reliance on a working directory is deprecated.

Rhythmyx Request Context

To access data in the Rhythmyx request context, use the methods of the interface IPSRequestContext. In custom JSPs and servlets, you can access the request context by calling `ServletRequest.getAttribute("RX_REQUEST_CONTEXT")`:

- for JSPs, ServletRequest is available from the implicit `pageContext` object;
- for servlets, the `HttpServletRequest` subclass is provided to the `handleRequest()` method of the `Controller` interface.

The body of the request is not parsed.

If you need to access an XML document or HTML parameters within Rhythmyx (such as for a Rhythmyx XML application), use the method `IPSRequestContext.parseBody()`. If the body of the request is an XML document, it will then be available by calling `IPSRequestContext.getInputDocument`. If the body of the request is a multi-part form, each field is converted to an HTML parameter in the request context.

Query string parameters and non-multipart form parameters are automatically parsed by JBoss. You can access them by calling either `ServletRequest.getParameterMap()` or `IPSRequestContext.getParameter()`.

Rhythmyx Server Information

Information about the Rhythmyx server can be accessed using the method `IPSRhythmyxInfo.getProperty(Key)`. The following keys can be specified:

Key Value	Returned Data
ROOT_DIRECTORY	The root directory of the Rhythmyx server.
LISTENER_PORT	The HTTP listener port of the Rhythmyx server (9992 by default).
LISTENER_SSL_PORT	The HTTPS port of the Rhythmyx server.
VERSION	The installed version of Rhythmyx, as a string.

Integrating Content Explorer Action Menu Entries

Action Menu entries define HTML requests that initiate processing. These HTML requests specify the processing the server should perform (such as retrieving a Content Item, updating edits to it, or performing a Workflow Transition on it). HTML requests are processed by servlets, JSPs, or Rhythmyx applications.

JSPs and servlets should be used to add custom user interfaces. Requests to servlets and JSPs use standard URLs for those implementations, for example:

```
/Rhythmyx/user/apps/sampleapplication/sampleurl
/Rhythmyx/user/pages/samplepage.jsp
```

Rhythmyx Content Editors are accessed as applications. Otherwise, Rhythmyx applications should be used when specific Rhythmyx functionality is needed.

Requests to Applications

The general format of requests to Rhythmyx applications is:

```
http://server:port/rxroot/approot/resource.xml?parameters
```

Where:

`server` is the hostname or IP address of the machine where the Rhythmyx server is running.

`port` is the port number in which the Rhythmyx server listens for HTTP requests.

`rxroot` is the URL root of the Rhythmyx server; for example, the default root is `/Rhythmyx`.

`approot` is the name of the Rhythmyx application you want to use to process the request.

`resource` is the name of the resource you want to use to process the request (or a file in the application directory

`xml` is the extension used for the request. In the majority of cases, these requests are internal and should use the extension "xml".

`parameters` is a list of parameters used by the specified resource. The set of parameters used in a request are determined by the parameters required by the resource, which falls into one of four categories:

- query
- update
- Content Editor
- non-text

Query Request Parameters

The optional and required parameters of a query resource are determined by the parameters specified when defining the resource itself.

Update Request Parameters

<table>
<tr><th>Parameter Name</th><th>Description</th><th>Allowed Values</th></tr>
<tr><td>DBActionType</td><td>Specifies the database action.

If all entries in the submitted XML document use the same action, you can supply this parameter as an HTML parameter. Otherwise, you must provide an ActionType for each row.</td><td>
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>INSERT</td><td>Tells the database server to create a new row</td></tr>
<tr><td>UPDATE</td><td>Tells the database server to update an existing row; a key must be provided. Depending on the settings of the resource, this command may also insert a row if it does not already exist.</td></tr>
<tr><td>DELETE</td><td>Remove the row specified by the supplied key</td></tr>
<tr><td>QUERY</td><td>Not generally useful for Update resources.</td></tr>
</table>
</td></tr>
<tr><td>Others</td><td>Other parameters as determined by resource design.</td><td>Determined by resource design.</td></tr>
</table>

Content Editor Request Parameters

The only parameter available for Content Editor resources is `sys_command`, which can take the following values:

Value	Result
edit (see "Edit Command" on page 162)	Opens the specified Content Item so a user can change it.
preview (see "Preview Command" on page 164)	Assembles the specified Content Item using the specified Template.

Value	Result
modify (see "Modify Command" on page 167)	Adds a new Content Item to the Repository or updates an existing Content Item in the Repository.
workflow (see "Workflow Command" on page 169)	Changes the current State of the specified Content Item.
binary (see "Binary Command" on page 169)	Retrieves a non-text Content Item.
clone (see "Clone Command" on page 170)	Creates a duplicate of the specified Content Item.
relate (see "Clone Command" on page 170)	Creates or modifies a Relationship.

The value of the `sys_command` parameter determines the additional parameters required. Note that some values of `sys_command` require subcommands in the format:

```
http://server:port/rxroot/approot/resource.xml?sys_command/subcommand?parameters
```

Edit Command

The `edit` command opens the specified Content Item so a user can change the Item. It takes the following parameters:

Parameter Name	Description	Allowed Values
sys_contentid	The ID of the Content Item to be edited. If this value is not specified, the server returns an empty results set.	Integers
sys_revision	The ID of the Revision of the Content Item to return.	Integers

<table>
<tr><th>Parameter Name</th><th>Description</th><th>Allowed Values</th></tr>
<tr><td>sys_pageid</td><td>Specifies whether to retrieve data for all fields on the Content Item or for a specific child table. This parameter is optional. If not included, the server returns all fields (as if the value of the parameter were 0).</td><td>
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>0</td><td>Return all Content Item fields, including all child table entries.</td></tr>
<tr><td>1</td><td>Return the summary of the first child table in the Content Item (in the order the child tables are specified in the Content Item definition); or all child entries for the first child table.</td></tr>
<tr><td>2</td><td>Return the editing information for a specific row in the first child table in the Content Item (in the order the child tables are specified in the Content Item definition). The row to return is specified by the sys_childrowid parameter.</td></tr>
<tr><td>3</td><td>Return the summary of the second child table in the Content Item (in the order the child tables are specified in the Content Item definition); or all child entries for the second child table.</td></tr>
<tr><td>4</td><td>Return the editing information for a specific row in the second child table in the Content Item (in the order the child tables are specified in the Content Item definition). The row to return is specified by the sys_childrowid parameter.</td></tr>
<tr><td colspan="2">And so on. Odd values for this parameter specify to return the summary or all child entries of a table (1 is the first table, 3 is the second table, 5 is the third table, etc.). Even values specify to return a specific row from a child table (2 is the first table, 4 is the second table, 6 is the third table). The rule essentially is to return a specific row from a child table, use the value 2x, where x is the count of the table for which you want to return the row; to return the entire table, use the value 2x-1.</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Parameter Name</th><th>Description</th><th>Allowed Values</th></tr>
<tr><td>sys_view</td><td>Specifies the set of fields to display, metadata, content data, or a single field.</td><td>
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>sys_All</td><td>Display all fields, including all child entries.</td></tr>
<tr><td>sys_ItemMeta</td><td>Displays only system fields. For local and shared fields, the server sets displayType=”sys_hidden”.</td></tr>
<tr><td>sys_Content</td><td>Displays only content fields (fields defined in the local and shared definition files). For system fields, the server sets displayType=”sys_hidden”.</td></tr>
<tr><td>sys_SingleField:<fieldname></td><td>Displays a single field, specified by <fieldname>. For all other fields in the Content Type, the server sets displayType=”sys_hidden”</td></tr>
</table>
</td></tr>
<tr><td>sys_childrowid</td><td>Specifies the row in a child table to display. Used when the value of sys_pageid is an even integer; otherwise, omit.</td><td>Integers</td></tr>
</table>

The server returns all fields of the specified Content Item. If the extension is XML, the response result must conform to the sys_ContentEditor.dtd (`rxroot/DTD/sys_ContentEditor.dtd`).

Preview Command

This command displays an assembled Content Item in preview mode. The value of the `isReadOnly` parameter for all fields is always true when the command is `preview`.

Parameter Name	Description	Allowed Values
sys_contentid	The ID of the Content Item to be edited. If this value is not specified, the server returns an empty results set.	Integers

<table>
<tr><th>Parameter Name</th><th>Description</th><th>Allowed Values</th></tr>
<tr><td>sys_revision</td><td>The ID of the Revision of the Content Item to return.</td><td>Integers</td></tr>
<tr><td>sys_pageid</td><td>Specifies whether to retrieve data for all fields on the Content Item or for a specific child table. This parameter is optional. If not included, the server returns all fields (as if the value of the parameter were 0).</td><td>
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>0</td><td>Return all Content Item fields, including all child table entries.</td></tr>
<tr><td>1</td><td>Return the summary of the first child table in the Content Item (in the order the child tables are specified in the Content Item definition); or all child entries for the first child table.</td></tr>
<tr><td>2</td><td>Return the editing information for a specific row in the first child table in the Content Item (in the order the child tables are specified in the Content Item definition). The row to return is specified by the sys_childrowid parameter.</td></tr>
<tr><td>3</td><td>Return the summary of the second child table in the Content Item (in the order the child tables are specified in the Content Item definition); or all child entries for the second child table.</td></tr>
<tr><td>4</td><td>Return the editing information for a specific row in the second child table in the Content Item (in the order the child tables are specified in the Content Item definition). The row to return is specified by the sys_childrowid parameter.</td></tr>
</table>
And so on. Odd values for this parameter specify to return the summary or all child entries of a table (1 is the first table, 3 is the second table, 5 is the third table, etc.). Even values specify to return a specific row from a child table (2 is the first table, 4 is the second table, 6 is the third table). The rule essentially is to return a specific row from a child table, use the value 2x, where x is the count of the table for which you want to return the row; to return the entire table, use the value 2x-1.
</td></tr>
</table>

<table>
<tr><th>Parameter Name</th><th>Description</th><th>Allowed Values</th></tr>
<tr><td>sys_view</td><td>Specifies the set of fields to display, metadata, content data, or a single field.</td><td>
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>sys_All</td><td>Display all fields, including all child entries.</td></tr>
<tr><td>sys_ItemMeta</td><td>Displays only system fields. For local and shared fields, the server sets displayType=”sys_hidden”.</td></tr>
<tr><td>sys_Content</td><td>Displays only content fields (fields defined in the local and shared definition files). For system fields, the server sets displayType=”sys_hidden”.</td></tr>
<tr><td>sys_SingleField:<fieldname></td><td>Displays a single field, specified by <fieldname>. For all other fields in the Content Type, the server sets displayType=”sys_hidden”</td></tr>
</table>
</td></tr>
<tr><td>sys_childrowid</td><td>Specifies the row in a child table to display. Used when the value of sys_pageid is an even integer; otherwise, omit.</td><td>Integers</td></tr>
</table>

The server returns all fields of the specified Content Item. If the extension is XML, the response result must conform to the sys_ContentEditor.dtd (`rxroot/DTD/sys_ContentEditor.dtd`).

Modify Command
This command modifies a Content Item or one of its fields.

<table>
<tr><th>Parameter Name</th><th>Description</th><th>Allowed Values</th></tr>
<tr><td>DBActionType</td><td>Specifies the database action to take.

NOTE: IF you specify a value other than those listed under “Allowed Values”, no database operation occurs.</td><td>
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>INSERT</td><td>Inserts a new Content Item with the specified values. Creates a new contentid and revision id.</td></tr>
<tr><td>UPDATE</td><td>Modifies an existing Content Item. When using this value, the parameters sys_contentid and sys_revision are required.</td></tr>
<tr><td>SEQUENCE_DECREMENT</td><td>Decrements the sortrank of the specified child row. This Value is ignored if the value of the Allow user to reorder entry field for the Content Editor is false.</td></tr>
<tr><td>SEQUENCE_INCREMENT</td><td>Increments the sortrank of the specified child row. This Value is ignored if the value of the Allow user to reorder entry field for the Content Editor is false.</td></tr>
</table>
</td></tr>
<tr><td>sys_contentid</td><td>The ID of the Content Item to be updated. Required if DBActionType=UPDATE.</td><td>Integers</td></tr>
<tr><td>sys_revision</td><td>The revision ID of the Content Item to be updated. Required if DBActionType=UPDATE.</td><td>Integers</td></tr>
<tr><td>sys_childid</td><td>The ID of the child table of the Content Item to be updated.</td><td>Integers</td></tr>
</table>

Parameter Name	Description	Allowed Values
sys_childrowid	The row in the child table to be updated.	Integers
fieldname	The name of the field to be updated.	Alphanumeric string of the updated value of the field.
*fieldname*_clear	Used for binary fields. Removes the existing value of the field. Simply leaving the upload field (which specifies the path to the binary object to upload), does not clear the binary object. You must use the "_clear" suffix with the field name to clear the binary object. This parameter does not take a value. For example, to clear a field called `imgbody`, you would include this parameter as `imgbody_clear`.	Does not take a value.

Workflow Command
The workflow command performs a Workflow Action on a Content Item. Workflow Actions fall into two categories

- Transitional requests perform the stated Transition, changing the Workflow State of the Content Item.
- Non-transitional requests check out or check in the Content Item, but do not change the Item's State.

<table>
<tr><th>Parameter Name</th><th>Description</th><th>Allowed Values</th></tr>
<tr><td>WFAction</td><td>Specifies the database action to take.</td><td>
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>checkout</td><td>Non-Transitional action; checks out the specified Content Item.</td></tr>
<tr><td>checkin</td><td>Non-Transitional action; checks in the specified Content Item.</td></tr>
<tr><td>forcecheckin</td><td>Non-Transitional action; checks in the specified Content Item, even if the Item is currently checked out by a user.</td></tr>
<tr><td>transition or trigger name</td><td>Transitional action. Requires the sys_transitionid parameter to specify the Transition to perform.</td></tr>
</table>
</td></tr>
<tr><td>sys_contentid</td><td>The ID of the Content Item on which to perform the Workflow Action.</td><td>Integers</td></tr>
<tr><td>sys_revision</td><td>The revision ID of the Content Item to be updated.</td><td>Integers</td></tr>
<tr><td>sys_transitionid</td><td>The ID of the Transition to perform. Not required for non-transitional actions.</td><td>Integers</td></tr>
<tr><td>commenttext</td><td>Comment for the Workflow request. May be empty.</td><td>Alphabetic string of the comment on the Transition.</td></tr>
</table>

Binary Command
Equivalent to a non-text request. See the discussion of non-text requests for details.

Parameter Name	Description	Allowed Values
sys_submitname	Name of the Binary field.	Alphanumeric string.
sys_contentid	ID of the Content Item.	Integers
sys_revision	ID of the Revision.	Integers

Parameter Name	Description	Allowed Values
sys_childrowid	ID of the child row; required if the binary object is stored in a child table.	Integers

Clone Command

The clone command creates a duplicate of the specified Content Item. The response document contains the same data as the cloned Item, which conforms to the sys_ContentEditor.dtd.

Parameter Name	Description	Allowed Values
sys_contentid	ID of the Content Item.	Integers
sys_revision	ID of the Revision.	Integers
sys_wfAction	Workflow Action to perform on the newly-created Content Item. The only valid value for this parameter is *checkin*. If this parameter has any other value, or if no value is specified for the parameter, the checkin action is not performed on the newly-created Content Item.	*checkin*

Relate Command

The relate command constructs or modifies a Relationship. The relate command can take subcommands:

Subcommand	Description
create	If the command specifies locators for both an Owner and a Dependent, this subcommand creates a new Relationship of the specified type between the specified Content Items if no Relationship of that type already exists between the two Items. If a Relationship of the specified type already exists between the specified Content Items, the Relationship is updated. If no Dependent locator is specified and the Relationship configuration allows cloning, the Owner is cloned and a Relationship of the specified type is created between the owner and the new clone.
insert	If the command specifies locators for both an Owner and a Dependent, this subcommand creates a new Relationship of the specified type between the specified Content Items if no Relationship of that type already exists between the two Items. If no Dependent locator is specified and the Relationship configuration allows cloning, the Owner is cloned and a Relationship of the specified type is created between the owner and the new clone.
query	Retrieves the specified Relationships.

Note that you can use the `relate` command without a subcommand. The Rhythmyx server treats such a request as if it included the `create` subcommand.

All subcommands use the same set of parameters.

Parameter Name	Description	Allowed Values
sys_relationshiptype	Specifies the type of relationship to create or modify.	Any Relationship type (for example, Active Assembly or Translation – Mandatory) defined in the system
sys_contentid	Specifies the ID of the Owner Content Item in the Relationship.	Integers
sys_revision	Specifies the Revision of the Owner Content Item in the Relationship.	Integers
sys_dependentid	Specifies the ID of the Dependent Content Item in the Relationship.	Integers
sys_dependentrevision	Specifies the Revision of the Dependent Content Item in the Relationship.	Integers

Non-Text Request Parameters

The parameters of a non-text resource are determined by the design of the resource, but must include at least the search key that specifies the row to return.

Spring Configurations

To add Spring beans to Rhythmyx, add Spring bean configuration files to the Spring configuration directory: `<Rhythmyxroot>/AppServer/server/rx/deploy/rxapp.ear/rxapp.war/WEB-INF/config/user/spring/`. During initialization, the server loads any Spring beans configuration files found in this directory.

Note that beans added to this configuration are added to the server's Spring context. The user dispatcher servlet also has a separate Spring context. Beans added to one context are not available in the other context.

The names of system Spring beans begin with the prefix "sys_". This prefix should not be used when naming custom Spring beans. However, best practice is to use a consistent naming convention for your project beans. For information about naming conventions, see "Design Object Naming Conventions" in the *Rhythmyx Implementation Guide*.

Alternate Hibernate Session Connections to the Rhythmyx Datasource

Rhythmyx connects to the Repository using a default Hibernate session that uses the Rhythmyx datasource. You can also configure custom connections to this datasource.

To connect to the Rhythmyx datasource programmatically, use the method `PSConnectionHelper.getHibernateInfo(PSConnectionInfo info)` where `info` is the name of the datasource connection for which you want to return properties. The object returned includes the following properties:

Property	Description
hibernate.connection.datasource	Name of JNDI datasource.
hibernate.default_catalog	Database name, if specified.
hibernate.default_schema	Schema or origin, if specified.
hibernate.dialect	Hibernate dialect used to connect to the database, based on the driver configured in the JNDI datasource.

Additional properties defined in the Rhythmyx datasource are also returned.

If you need to create a different Hibernate session for a Spring bean, use PSDatasourceSessionFactoryBean to create a new session factory for your bean. For details, see the Javadoc for this method. Note that Hibernate mappings cannot be specified with wildcards. The Javadoc for PSDatasourceSessionFactoryBean describes this issue, or see the jboss.org Web site for details.

Logging for Custom Implementations

Custom implementations should use the Apache commons logging interfaces to perform logging. Logging output from this interface will automatically use the log4j configuration defined in the Rhythmyx server. For additional details about logging, see ***Configuring Logging*** (see page 178).

Defining Non-Rhythmyx Datasources

If your implementation needs a datasource, you should configure it in the Rhythmyx Server Administrator (see "Maintaining Datasources"). If you need to configure options in your datasource that are not available in the Rhythmyx Server Administrator, you can create another JBoss datasource file and deploy it to the directory `<Rhythmyxroot/>/AppServer/server/rx/deploy` and add the new datasource to it. The new datasource will be picked up the next time the server is restarted. These datasources are not available for use in Rhythmyx applications.

Security

This section describes security issues in Rhythmyx.

Rhythmyx, JBoss, and JAAS

Rhythmyx implements the security model of its container, JBoss. Like many J2EE application servers, JBoss uses Java Authentication and Authorization Service (JAAS). By default, Rhythmyx uses its own login module, which uses the security providers shipped with Rhythmyx. JBoss includes a number of login modules that you can use instead, or you can implement your own login module. (For details about configuring these modules, see ***Rules for Custom Login Modules***.)

Implementing Custom Authentication

If you need to implement a custom login module, you will need to configure it, and you may need to implement customer Role Catalogers, Subject Catalogers, or both Role and Subject Catalogers. This section describes how to configure custom login modules correctly and how to implement custom Role and Subject catalogers.

Rules for Custom Login Modules

If you add a new login module, you must configure it in the file `<Rhythmyxroot>/AppServer/server/rx/conf/login-config.xml`. Changes to this file require a server restart to take effect.

If the Rhythmyx entry in this file is removed or commented out, any security providers configured in Rhythmyx will be disabled.

If you cannot use LDAP to return Role and Role Member data, and the data cannot be configured manually in the Role or Member registration in the Rhythmyx Server Administrator, you will need to implement a custom Role cataloger, a custom subject cataloger, or both. Role Catalogers return Role data, including lists of Role Members and Role properties. Subject catalogers return data about the individual Members of the Roles. For details about Role and Subject catalogers, see "***Role and Subject Catalogers***".

To return Roles correctly, the Rhythmyx login module should be configured last in the configuration file. The value of the `flag` attribute of the Rhythmyx login module must be *sufficient*, while the value of this attribute for all other modules must be *optional*.

If you want to return Role and subject data from all Role and Subject catalogers, do not configure any custom login module to return a list of Roles to Rhythmyx. Rhythmyx will then query all Role and subject catalogers once authentication processing is complete. (If any login modules is configured to return Role and Subject data, Rhythmyx will return Role and Subject data from that login modules and from the default rxmaster Backend Table Security Provider.)

If you use a <meta> tag to override the default content type, character encoding on the page must be UTF-8.

Role and Subject Catalogers

You do not need to implement a custom Role cataloger or subject cataloger when you implement a custom login module if you can use LDAP to retrieve Role and Member data, or if you can configure the data manually for the Roles and Members in the Server Administrator. If neither of these cases is true, you will need custom Role and subject catalogers.

Role catalogers are Rhythmyx extensions that return Role data, including the list of Role Members and Role properties. These extensions implement the interface IPSRoleCataloger. Subject catalogers are Rhythmyx extensions that return data for specific Members of Roles. These extensions implement the interface IPSSubjectCataloger.

The cataloger class requires a no argument constructor. The names of any properties must begin with lower-case letters. Properties require a setter method following Java Bean patterns. The name of the setter method must be camel-cased.

The cataloger must be registered on the Rhythmyx Server Administrator (see "Maintaining Catalogers" in the Rhythmyx Server Administrator Help) , with all properties specified. The server must be restarted to initialize the cataloger extension.

Implementing Custom Login Pages

Rhythmyx provides a default login page, including both login and logout functionality, but you can implement your own login page if you prefer. To implement a custom login page, add the following files to `<Rhythmyxroot>/AppServer/ server/rx/deploy/rxapp.ear/rxapp.war/`:

- login.jsp

  This file must include the following:

  - A field called `j_username`, in which the user enters the login name. The value should be defined with the tag`<%= request.getParameter("j_username") %>` to allow the value to be restored after a failed attempt.
  - A field called `j_password`, in which the user enters the password. The value should be defined with the tag `<%= request.getParameter("j_password") %>` to allow the value to be restored after a failed attempt.

  The form should not specify an action so it is submitted for login again after an error.

- error.jsp

  This file is optional. Include it if you want to display the error message from the server explaining why login failed. To display this message, include the tag JSP tag `<%= request.getParameter("j_error") %>`.

  This file should also include a way to return to the login page (such as a a link to `../login` or or a form that posts back to the login servlet by not specifying an action.

  Note that you can also display error messages on the login form by including the `j_error` field.

- logout.jsp

  This file must make a request to `/Rhythmyx/logout` (or a relative path based on the location from which `/Rhythmyx/logout` is called).

Security Extensions

Rhythmyx provides for three types of extensions for security:

- Role Catalogers
- Subject Catalogers
- Password Filters

For details about Role and Subject Catalogers, see ***Role and Subject Catalogers***. Note that no Role or Subject Cataloger extensions are shipped with Rhythmyx.

Password Filters

A Password Filter extension is used when passwords are stored in encrypted form. A Password Filter encrypts the password entered by the user so it can be compared to the encrypted password stored in the security provider.

A Password Filter extension must implement the interface IPSPasswordFilter.

IMPORTANT: A Password Filter extension must provide a meaningful no arguments constructor that will produce a working filter.

sys_DefaultPasswordFilter

Context:

Java/global/percussion/filter/

Description:

This exit takes a plain text string (a password) and encrypts it for a Rhythmyx security provider.

Class name:

com.percussion.filter.DefaultPasswordFilter

Interface:

com.percussion.security.IPSPasswordFilter

Parameters:

No user-supplied parameters. The server automatically supplies the password to the extension.

Security for Custom Web Applications

Custom web applications implemented as Rhythmyx JSPs or dispatched Spring MVC Controllers use the security configured for Rhythmyx. No additional security configuration is required for these applications. For details about configuring these applications, see Implementing Custom Java Server Pages and Servlets.

Security must be configured for non-Rhythmyx customizations. Configure security for each web application in the file `<Rhythmyxroot/AppServer/server/rx/deploy/rxapp.ear/rx-app.war/WEB-INF/config/user/security/user-security-conf.xml`. Entries in this file define the authentication requirement for each custom web application. Each web application is specified in a `path` node. The value of this node is the path to the web application for which you are defining security. The `authType` attribute of the `path` node specifies the type of authentication used for the web application. Options include `form`, `basic`, and `anonymous`.

The path nodes are nested in `securityConfiguration` nodes. The `securityConfiguration` node specifies whether the web applications contained in the node require secure login using SSL. If the value of the `forceSecureLogin` attribute is *yes*, authentication data will be transmitted using SSL, otherwise it is transmitted unencrypted via HTTP. A default `securityConfiguration` node is included with a `forceSecureLogin` attribute whose value is *no*.

Configuring Logging

Rhythmyx uses Log4j to provide logging functionality. The Log4j configuration file is stored in the directory `<Rhythmyxroot>/AppServer/server/rx/conf/log4j.xml`. This file includes extensive comments describing the default configuration.

For additional details about Log4j, see ***http://logging.apache.org/log4j***.

CHAPTER 8

Extensions

Rhythmyx extensions allow you to modify or enhance the functionality available in the base product by adding your own functionality. In many cases, Rhythmyx functionality is based on extensions, so you can customize the system to produce the behavior you need for your implementation.

This chapter outlines the general requirements for all extensions and explains how to register them. Two consolidated references to extensions are provided, one by type, the other alphabetical.

General Requirements of Extensions

All extensions must implement an interface, either IPSExtension or, in most cases, a more specialized interface. The interface required for each extension is documented with that extension type.

All extensions must be registered in the system so they can be initialized when the system starts. In most cases, the server must be restarted to initialize an extension. For details about registering an extension, see ***Registering an Extension*** (see page 181).

When designing and implementing an extension, evaluate its potential impact on performance. In most cases, the simplest way to gauge this impact is to consider how frequently it will be called. For example, a JEXL function extension is likely to be called whenever the associated Template is assembled, whether for preview or when publishing; when publishing, the function could be called hundreds, even thousands of times. A poorly designed and implemented extension can thus have a significant impact on performance.

IMPORTANT: Rhythmyx extensions must be thread safe. For information about thread safety, consult any standard Java reference.

Registering an Extension

A Rhythmyx extension can be implemented in either Java or JavaScript, but in Rhythmyx Version 6.0 and later, Java extensions are most common.

A Java extension usually has two sets of parameters:

- Initialization parameters are used to initialize the extension. You must specify both the name and the value of initialization parameters when registering the extension.
- Runtime parameters define data input to the parameter when it is called. You must specify the name and data type of these parameters. Ideally, when you register the extension you should also include a description of the parameter to explain what it does, what values are valid, and any default values.

The registration should also note any files (such as a properties file) or Rhythmyx applications needed to support the processing of the extension. Note that these supporting resources must exist before you can add them to the extension registration.

To register an extension:

1. In the Rhythmyx Workbench, go to the System view and select the Folder to which you want to add the extension registration.
2. In the Menu bar, choose *File > New*.

The Rhythmyx Workbench displays the New Extension wizard.

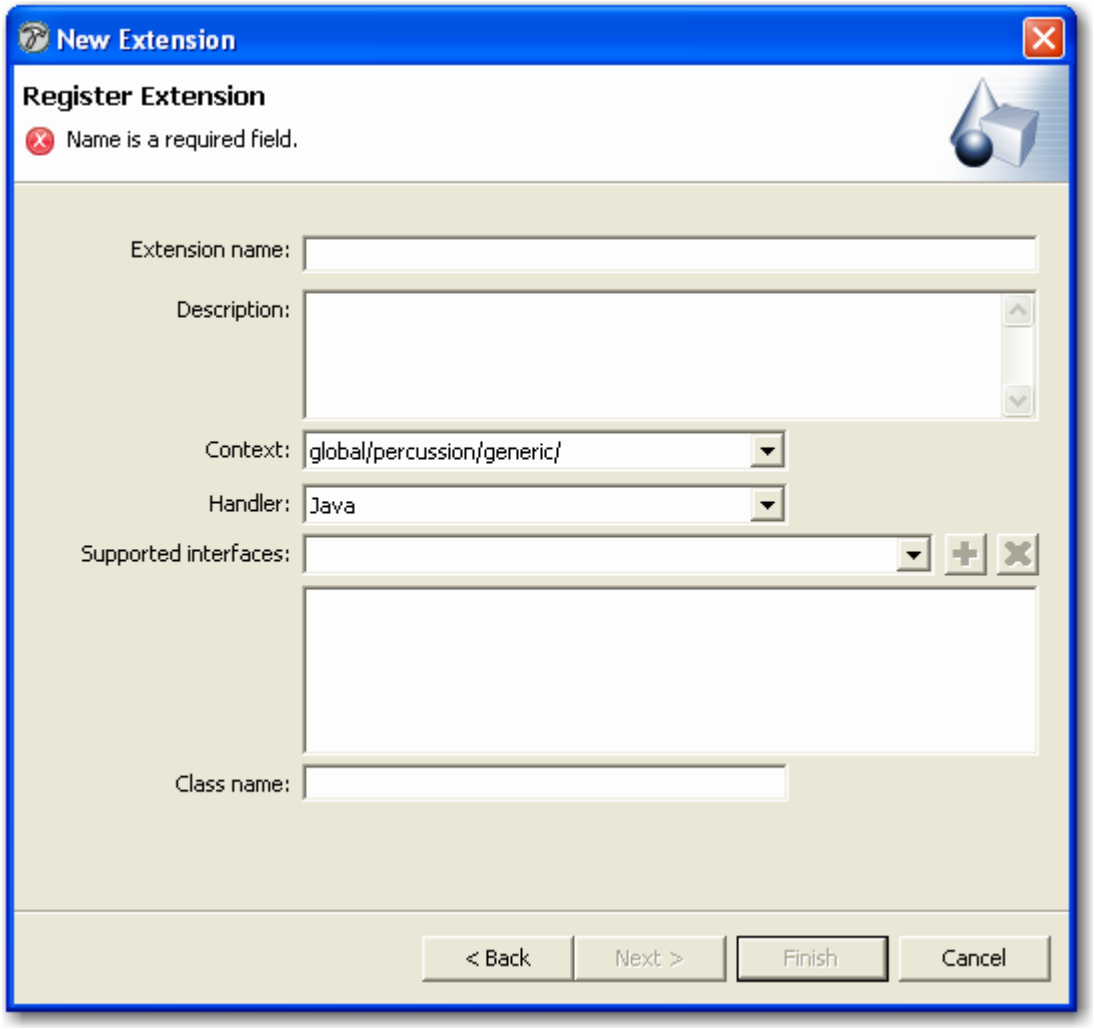


Figure 31: New Extension wizard

3 Enter the Extension name and, optionally, Description.

4 Leave *Java* selected in the Handler field.

5 To add a supported interface:

   a) In the Supported interfaces drop list, select the interface required for your extension.

   b) Click the [+] button next to the field to add the supported interface.

   c) Repeat for each supported interface you want to add to the extension.

6 Enter the Class name.

7 Click the [**Finish**] button.

The Rhythmyx Workbench displays the Extension editor.

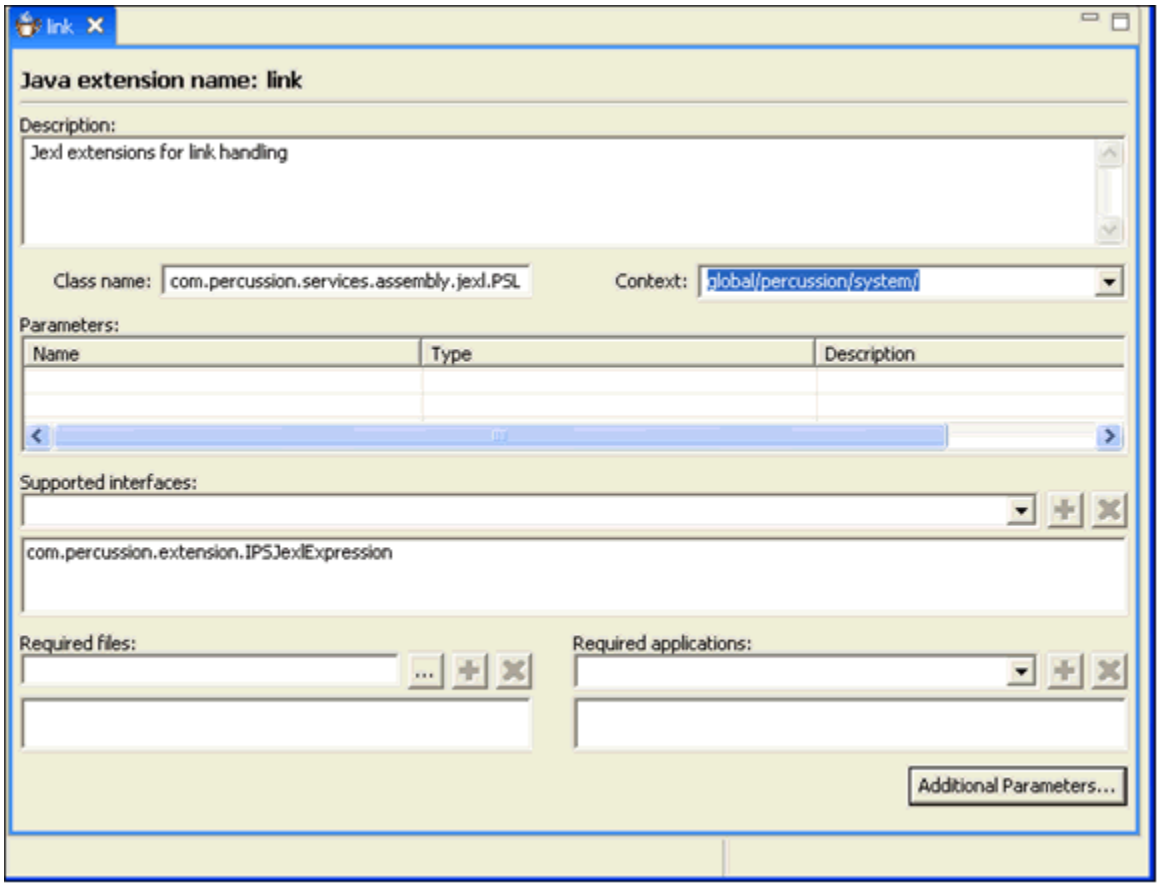


Figure 32: Extension Registration Editor

8 To add initialization parameters to the extension registration:

a) Click the [**Additional Parameters**] button.

The Rhythmyx Workbench displays the Additional Parameters dialog.

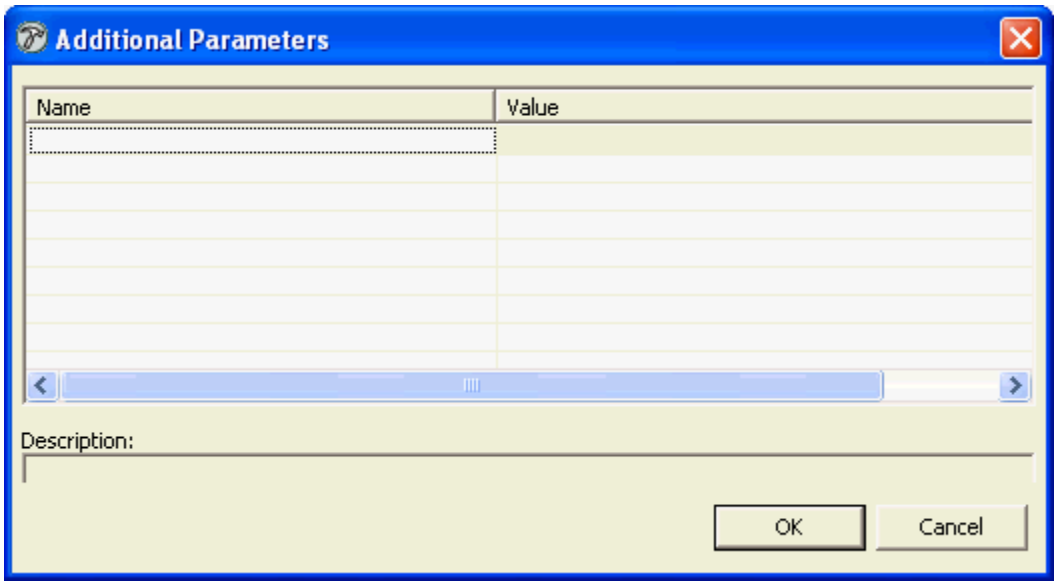


Figure 33: Additional Parameters dialog

b) For each initialization parameter in your extension, enter the Name of the parameters and its Value in the same row.

c) When you have entered all initialization parameters, click the [**OK**] button to save initialization parameters.

9 Add runtime parameters to the Parameters table on the Extensions editor. For each parameter, enter the Name and the data Type. Optionally, enter a Description.

10 In Required files, enter any number of files that the extension uses. Required files are generally .class, .jar, or .zip files. A required file might, for example, specify acceptable formats for dates or valid entries for a field.

a) Enter the path to the file in the Required files field, or use the browse button to find the file.

b) Click the [+] button next to the Required files field to add the path to the list of required files.

c) Repeat for each required file.

11 In Required applications, choose all of the applications that use the extension. If you fail to choose an application that uses the extension, you will not effect the running of the extension; however, the Rhythmyx Multi-Server Manager will not automatically package the extension with the application. For more information see the *Rhythmyx Multi-Server Management* document:

a) In the Required applications drop list, select the Rhythmyx application required for your extension.

b) Click the [+] button next to the field to add the application.

c) Repeat for each application you want to add to the extension.

12 In the Button bar of the Rhythmyx Workbench, click the save button.

Extensions Reference by Type

Assembly Plugins

binaryAssembler (see page 99)

databaseAssembler (see page 99)

debugAssembler (see page 100)

dispatchAssembler (see page 100)

legacyAssembler (see page 101)

velocityAssembler (see page 101)

Content List Generators

sys_PublishedSiteItems (see page 126)

sys_SearchGenerator (see page 128)

sys_SelectedItemsGenerator (see page 127)

Field Validations

sys_ValidateDateRange (see page 20)

sys_ValidateJexlFieldExpression (see page 20)

sys_ValidateNumberRange (see page 21)

sys_ValidateRequiredField (see page 22)

sys_ValidateStringLength (see page 22)

sys_ValidateStringPattern (see page 23)

Field Input Transformers

sys_MapInputValue (see page 24)

sys_NormalizeDate (see page 24)

sys_OverrideLiteral (see page 25)

sys_Replace (see page 25)

sys_ToHash (see page 26)

sys_ToLowerCase (see page 27)

sys_ToProperCase (see page 27)

sys_ToUpperCase (see page 28)

sys_TranslateJexlExpressionValue (see page 28)

Field Output Transformers

Item Filter Rules

Java Expression Language (JEXL) Functions

Location Scheme Generators

Slot Content Finders

sys_RelationshipContentFinder (see page 104)

sys_TranslationContentFinder (see page 105)

Template Expanders

sys_ListTemplateExpander (see page 127)

sys_SiteTemplateExpander (see page 129)

Workflow Actions

sys_createTranslations (see page 113)

sys_PublishContent (see page 114)

sys_TouchParentItems (see page 116)

Alphabetical Reference to Rhythmyx Extensions

Legacy Extension Reference

This section documents extensions that were shipped with earlier versions of Rhythmyx. These extensions are installed to provide backward compatibility in upgraded systems. In general, when implementing Rhythmyx Version 6.0, the newer extensions documented in ***Extensions Reference by Type*** (see page 185) and ***Alphabetical Reference to Extensions*** (see page 188) should be used.

Result Document Processing

rxs_SiteFolderContentListBuilder

Name

rxs_SiteFolderContentListBuilder

Context

global/percussion/fastforward/sfp/

Description

This exit builds a Content List for Site Folder Publishing from Content Items in a Content Explorer Site Folder tree. The exit's parameters let users customize which Content Items are selected for publishing.

Class name

com.percussion.fastforward.sfp.PSSiteFolderContentListExit

Interface

com.percussion.extension.IPSResultDocumentProcessor

Parameters

Name	Data Type	Description
filenameContext	string	Delivery location for Content Items. By default, takes the value of the locationCtx parameter in the Content List URL. If a delivery location is not specified, Rhythmyx delivers to the URL in the Assembly Context.
deliveryType	string	Type of delivery. Options are filesystem or ftp. Takes the value of the delivery parameter in the Content List URL if specified. Default value is ftp.

Name	Data Type	Description
isIncremental	string	Incremental publishing flag. Options are y or n. Takes the value of the inc parameter in the Content List URL if specified. Default value is n.
contentValidValues	string	Indicates in which States it is valid to publish this content. For more information, see "Extending Publishable States" and "Edit State Page" in the online CMS Help. Takes the value of the valid parameter in the Content List URL if specified.
MaxRowsPerPage	String	Specifies the maximum number of Content Items to appear on a single page of the Content List. Default is –1 (unlimited number of Content Items).

rxs_SiteFolderContentListBulkBuilder

Name

rxs_SiteFolderContentListBulkBuilder

Context

/global/percussion/fastforward/sfp/

Description

This exit builds a Content List for Site Folder Publishing from Content Items in a Content Explorer Site Folder tree. The exit's parameters let users customize which Content Items are selected for publishing. Also flushes all caches on a Publishing Hub server.

Class Name

com.percussion.fastforward.sfp.PSSiteFolderContentListBulkExit

Interface

com.percussion.extension.IPSResultDocumentProcessor

Parameters

Name	Data Type	Description
filenameContext	String	Delivery location for Content Items. By default, takes the value of the locationCtx parameter in the Content List URL. If a delivery location is not specified, Rhythmyx delivers to the URL in the Assembly Context.
deliveryType	String	Type of delivery. Options are filesystem or ftp. Takes the value of the delivery parameter in the Content List URL if specified. Default value is ftp.
isIncremental	String	Incremental publishing flag. Options are y or n. Takes the value of the inc parameter in the Content List URL if specified. Default value is n.
contentValidValues	String	Indicates in which States it is valid to publish this content. For more information, see "Extending Publishable States" and "Edit State Page" in the online CMS Help. Takes the value of the valid parameter in the Content List URL if specified.
MaxRowsPerPage	String	Specifies the maximum number of Content Items to appear on a single page of the Content List. Default is –1 (unlimited number of Content Items).
contentResourceName	String	Name of the resource that looks up Content Items and their Variants for publishing. The rxs_SiteFolderContentListBulkBuilder only processes the SQL results set from this resource, it does not process the XML result document. Therefore, if the resource includes any post-exits, the rxs_SiteFolderContentListBulkBuilder exit will ignore them.
ParamListToPass	String	Comma separated list of all non standard HTML parameters to pass from request to the content URL for each item in the content list.

rxs_SiteFolderAssembly

Name:

rxs_SiteFolderAssembly

Context:

global/percussion/fastforward/sfp/

Description:

This extension retrieves a Site Folder path to build a Site Folder publishing location. You can use this as a location scheme generator or as a UDF mapped to a path variable in an application resource that builds a publishing location. For more information see the FastForward documentation.

Class name:

com.percussion.fastforward.sfp.PSSiteFolderAssembly

Interfaces:

com.percussion.extension.IPSAssemblyLocation
com.percussion.extension.IPSUdfProcessor

rxs_AutoSiteItemFilter

Name:

rxs_AutoSiteItemFilter

Context:

global/percussion/fastforward/sfp/

Description:

This exit removes Content Items that are not associated with sys_siteid from an autoindex.

Class name:

com.percussion.fastforward.sfp.PSAutoSiteItemFilter

Interfaces:

com.percussion.extension.IPSResultDocumentProcessor

Parameters:

Name	Data Type	Description
ItemSiteFolderURL	string	The url of a resource that retrieves the site folder root xml file. Default is ../rx_supportSiteFolderContentList/lookupSiteFolderRoot.xml

rxs_NavAutoSlot

Name:

rxs_NavAutoSlot

Context:

global/percussion/fastforward/managednav/

Description:

This post-exit is attached to an assembler resource after the sys_casAddAssemblerInfo exit for any Content Type that has navigation bars. It adds the correct links to the Navigation Slots on an assembled Page. The exit finds Folders which contain the Content Item being assembled and a Navon. If one such Folder is found, a Variant of the Navon is inserted into the appropriate navigation Slot on the page. If more than one such Folder is found, the exit locates a Folder which is a descendent of the Site Folder root and inserts a Variant of the Navon into the appropriate navigation Slot on the page. If users supply a value for sys_folderid or rx_folder (as an HTML parameter to the assembler application), the exit finds this Folder and inserts a Variant of the Navon into the Navigation Slot.

Class name:

com.percussion.fastforward.managednav.PSNavAutoSlotExtension

Interface:

com.percussion.extension.IPSResultDocumentProcessor

Parameters

None

rxs_NavAddAttribute

Name:

rxs_NavAddAttribute

Context:

global/percussion/fastforward/managednav/

Description:

This exit adds an attribute from a Navon to the Navon node in the Managed Navigation output XML. Add this exit to Navon assembly applications when you want to add Navon fields that are not included in the XML by default. In general, this exit is best used to avoid a degradation in performance that could occur if you added the attributes using a document call.

Class name:

com.percussion.fastforward.managednav.PSNavAddAttribute

Interface:

com.percussion.extension.IPSResultDocumentProcessor

Parameters:

Name	Data Type	Description
attributeName	string	The name of the attribute to add to the output XML.
queryResource	string	The URL of the query resource that returns the attribute, usually in the form of appname/queryresourcename.
index	string	The index of the column in the query resource, 1 based indexing
relativeLevelAttribute	string	The name of the relative level attribute. Optional.

rxs_NavFolderSelector

Context:

global/percussion/fastforward/managednav/

Description:

Selects a given folder id by pathname and appends sys_folderid to the parent request. Useful in assemblers of Content Types which should be part of a navigation hierarchy, but are not in Folders. Most commonly used as a pre-exit of an assembler application.

Class name:

com.percussion.fastforward.managednav.PSNavFolderSelector

Interfaces:

com.percussion.extension.IPSResultDocumentProcessor
com.percussion.extension.IPSUdfProcessor
com.percussion.extension.IPSRequestPreProcessor

Parameters:

Name	Data Type	Description
pathname	string	Full path of the folder to select. Typically starts with //Sites

rxs_NavTreeSlotMarker

Context:

global/percussion/fastforward/managednav/

Description:

Use this exit with the rxs_NavTreeLink extension to generate a navigation tree for a specific navon. When this extension processes after rxs_NavTreeLink, it walks down the navtree and checks the info-url for each "ancestor" node. If it determines that the navon has content in a specified slot, it marks the navon element with an attribute set to "yes". You can use this attribute as a conditional in XSLT processing.

The purpose of this extension is to propagate links in custom slots on navon Variants down the ancestor tree and appear on each child navon. You can also use this extension for other logic in the XSL stylesheets that process the result document.

Process this exit after the rxs_NavTreeLink. You can be use it multiple times to create a marker for more than one slot.

Class name:

com.percussion.fastforward.managednav.PSNavTreeSlotMarker

Interfaces:

com.percussion.extension.IPSResultDocumentProcessor

Parameters:

Name	Data Type	Description
markerName	string	Name of the attribute that indicates that a navon has content in a specified slot.
slotName	string	Name of the Slot that the exit checks for content.

rxs_NavReset

This extension is used in internal Rhythmyx applications.

rxs_NavTreeLink

This extension is used in internal Rhythmyx applications.

rxs_NavTreeBuilder

This extension is used in internal Rhythmyx applications.

sys_casAddAssemblerInfo

Name:

sys_casAddAssemblerInfo

Context:

Java/global/percussion/contentassembler/

Description:

This post-exit adds information needed in content assembler stylesheets to the result document. It creates an XML document conforming to the DTD sys_AssemblerInfo.dtd and inserts it into the result document as its first child. Then it creates (or modifies) a separate set of stylesheets adding extra links for editing the related content in preview or WYSIWYG mode (active assembly mode). This exit should be added to most assembly application resources.

Class name:

com.percussion.cas.PSAddAssemblerInfo

Resource file:

classes

Interface:

com.percussion.extension.IPSResultDocumentProcessor

Example:

The following is an example of XML generated by this exit:

```
   <sys_AssemblerInfo previewurl="/Rhythmyx/casArticle/casArticle.xml"
sys_siteid="0" pssessionid="1a52d1b40cc8716577d33ce255d51e65d0e0cfdb"
sys_command="editrc" sys_contentid="310" sys_variantid="1"
sys_revision="1" sys_context="0" sys_authtype="0">
      <RelatedContent>
         <slotrceditlink>http://127.0.0.1:9992/Rhythmyx/sys_ComponentSup
port/componentabslink.xml?pssessionid=1a52d1b40cc8716577d33ce255d51e65d0
e0cfdb&amp;sys_componentname=rcsearch</slotrceditlink>
         <linkurl rxcontext="0" slotid="" relateditemid="" contentid=""
variantid="" slotname=""
moveuplink="http://winkelried:9992/Rhythmyx/sys_rcSupport/updaterelatedi
tems.html?sys_command=moveup&amp;sys_contentid=&amp;sys_variantid=1&amp;
sys_slotid=&amp;sys_context=0&amp;sys_revision=1&amp;sys_authtype=0&amp;
sysid="
movedownlink="http://winkelried:9992/Rhythmyx/sys_rcSupport/updaterelate
ditems.html?sys_command=movedown&amp;sys_contentid=&amp;sys_variantid=1&
amp;sys_slotid=&amp;sys_context=0&amp;sys_revision=1&amp;sys_authtype=0&
amp;sysid="
```

```
deletelink="http://winkelried:9992/Rhythmyx/sys_rcSupport/updaterelatedi
tems.html?sys_command=delete&amp;sys_contentid=&amp;sys_variantid=1&amp;
sys_slotid=&amp;sys_revision=1&amp;sys_context=0&amp;sys_authtype=0&amp;
sysid=" editlink=""
modifylink="http://winkelried:9992/Rhythmyx/sys_rcSupport/modifyslotitem
.html?sys_variantid=1&amp;sys_context=0&amp;sys_authtype=0&amp;sysid=">
            <Value current=""/>
         </linkurl>
      </RelatedContent>
      <AssemblerProperties>
         <Property name="rxcss">
            <Value current="../web_resources/xroads/resources/css"/>
         </Property>
         <Property name="rxjavascript">
            <Value current="../web_resources/xroads/resources/js"/>
         </Property>
         <Property name="rximage">
            <Value current="../web_resources/xroads/resources/images"/>
         </Property>
         <Property name="rxflash">
            <Value
current="../web_resources/xroads/resources/images/fla"/>
         </Property>
      </AssemblerProperties>
      <InlineLink
url="http://winkelried:9992/Rhythmyx/sys_casSupport/PublicationUrl.xml?s
ys_context=0&amp;pssessionid=1a52d1b40cc8716577d33ce255d51e65d0e0cfdb"/>
   </sys_AssemblerInfo>
```

Parameters:

None

sys_casAddChildInfo

Context:

Java/global/percussion/contentassembler/

Description:

Queries the specified URL and appends the content of the returned doc to the current doc. Used in Content Assemblers to add data from child tables to the assembled output. Attach this exit to Content Assemblers for Content Types that store data in child tables when you need to include the data from the child table in the formatted output. If the formatted output does not need data from the child table, the assembly resource does not need this exit.

To use this exit, you must create a resource that queries data from the child table.

Class name:

com.percussion.cas.PSAddChildInfo

Interface:

com.percussion.extension.IPSResultDocumentProcessor

Parameters

Name	Data Type	Description
resource	java.lang.String	(Required) URL of the resource (relative to the Rhythmyx root) that queries the child table. It should be of the form RhythmyxApplication/ResourceName

sys_casAutoRelatedContent

Name:

sys_casAutoRelatedContent

Context:

Java/global/percussion/exit/

Description:

This Exit is added to the Assembly resource of Automated Index Assembler applications. It adds the related content generated by the Automated Content Query to the Slots in the assembly template. This Exit must be added AFTER the sys_casAddAssemblerInfo exit.

Class name:

com.percussion.cas.PSAutoRelatedContent

Resource:

file:classes

Interface:

com.percussion.extension.IPSResultDocumentProcessor

Parameters

Name	Data Type	Description
LinkURL	java.lang.String	Name of the linkURL attribute. This must be an attribute of the root node
slotNameOverride	java.lang.String	Slot Name override (optional). Allows the caller to place the results in any slot.

Name	Data Type	Description
maxResults	java.lang.Integer	Optional. Defines the maximum number of related Content Items that can be added to the target Slot.

sys_ceDependencyTree

Context:

Java/global/percussion/contenteditor/

Description:

This exit reformats the result document as an XML tree by appending all child and parent items of the current content item to it. It makes repeated internal requests to expand the parents and children. The dependency viewer in the content editors uses this exit.

Class name:

com.percussion.ce.PSDependencyTree

Interface:

com.percussion.extension.IPSResultDocumentProcessor

Parameters:

None

sys_cmpAddAllParamsToUrl

Context:

Java/global/percussion/extensions/general/

Description:

This pre-exit adds all HTML parameters in the request to the specified URLs. The URLs are specified as the first children of the root element in the result document.

For example, if the request came with the HTML parameters param1 and param2, the result document:

```
<root>
  <url1>/Rhythmyx/sampleApp/samplePage1.htm</url1>
  <url2>/Rhythmyx/sampleApp/samplePage2.htm</url2>
</root>
```

becomes

```
<root>

<url1>/Rhythmyx/sampleApp/samplePage1.htm?param1=value1&amp;param2=value
2</url1>
```

```
<url2>/Rhythmyx/sampleApp/samplePage2.htm?param1=value1&amp;param2=value
2</url2>
</root>
```

The parameter pssessionid is always skipped.

Currently, this exit only modifies children and grandchildren of the root that have the element name specified. Also, it does not modify the URLs if they are the attributes of an element.

Class name:

com.percussion.extensions.general.PSAddAllParamsToUrl

Resource file:

classes

Interface:

com.percussion.extension.IPSResultDocumentProcessor

Parameters:

Name	Data Type	Description
UrlElementName	String	The name of the unique element name storing the URL value

sys_cmpMenuTree

Context:

Java/global/percussion/extensions/components/

Description:

This exit builds a cascaded menu item list XML document by making multiple internal requests to a Rhythmyx resource. The resulting tree depends on the data in the backend tables RXSYSCOMPONENT and RXSYSCOMPONENTRELATIONS.

The following is a sample of the XML document:

```
<menuitem name="ca_inbox" id="20" type="2">
      <displaytext>Inbox</displaytext>
      <description>Items assigned to me</description>
      <url>
         http://10.10.10.56:9992/Rhythmyx/sys_ca/camain.html?
         sys_sortparam=title&sys_componentname=ca_inbox
      </url>
      <userrolesurl>
         http://127.0.0.1:9992/Rhythmyx/sys_cmpUserStatus/
         userstatus.xml?pssessionid=
         8037ca1cbcc8bd31e3db8b392d4fff8c62c9dacc
      </userrolesurl>
```

```
      <contexturl>
         http://127.0.0.1:9992/Rhythmyx/sys_ComponentSupport/
         componentcontext.xml?pssessionid=
         8037ca1cbcc8bd31e3db8b392d4fff8c62c9dacc&sys_componentid=20
      </contexturl>
      <componentname>ca_inbox</componentname>
      <childitem id="1"/>
      <childitem id="2"/>
      <childitem id="6"/>
      <childitem id="7"/>
   </menuitem>
```

The exit makes multiple requests are made to expand each child item to menu item. Use it to generate the navigation bars in the Content Explorer.

Class name:

com.percussion.extensions.components.PSMenuTree

Resource file:

classes

Interface:

com.percussion.extension.IPSResultDocumentProcessor

Parameters:

None

sys_CollapseHTMLParameter

Context:

Java/global/percussion/generic/

Description:

This exit collapses a multi-value (array) HTML parameter by taking the first value. In other words, it replaces an entire array with the first value of the array.

The number of parameters is fixed at 8, but it can handle any number of parameters. This exit is not required if you use the Single HTML parameter option in the Workbench.

Class name:

com.percussion.extensions.general.PSCollapseHtmlParameter

Interface:

com.percussion.extension.IPSResultDocumentProcessor,
com.percussion.extension.IPSRequestPreProcessor

Parameters:

Name	Data Type	Description
p1	java.lang.String	First array value
p2	java.lang.String	Second array value
p3	java.lang.String	Third array value
p4	java.lang.String	Fourth array value
p5	java.lang.String	Fifth array value
p6	java.lang.String	Sixth array value
p7	java.lang.String	Seventh array value
p8	java.lang.String	Eighth array value

sys_DatabasePublisher

Context:

Java/global/percussion/contentassembler/

Description:

This exit is required on each database publisher parent table resource. This exit looks up the table definition specified in the parent table mapper and produces the XML file that conforms to the sys_DatabasePublisher.dtd.

Class name:

com.percussion.cas.PSDatabasePublisher

Interface:

com.percussion.extension.IPSResultDocumentProcessor

Parameters:

Name	Data Type	Description
action	java.lang.String	Action performed on the database: r - (default) Inserts the row. Deletes it first if it already exists. n - Inserts the row if it does not already exist. u - Updates the row if it already exists. d - Deletes the row if it already exists. Use d for unpublishing.
aliasmap	java.lang.String	Optional. Static XML file in the assembler application that contains table and column name mappings to be used if the table or column names contain characters that are not allowed in XML elements. The exit creates XML files that use the aliases, and then reinserts the real names in the output. Must conform to: ../dtd/aliasmap.dtd

sys_FormatFileTree

Context:

Java/global/percussion/generic

Description:

This exit reformats a list of file path names contained in an XML result tree into a true tree structure. Use this extension to display file lists in the Rhythmyx CMS.

Class name:

com.percussion.extensions.general.PSFormatFileTree

Interface:

com.percussion.extension.IPSResultDocumentProcessor

Parameters:

Name	Data Type	Description
inputListName	java.lang.String	Name of the input file list `<filelist>`
fileTreeName	java.lang.String	Optional. Name of the XML element that contains the output tree. Defaults to `<filetree>`.
fileElementName	java.lang.String	Optional. Name of the XML file elements found in the input list. Defaults to `<file>`.

Name	Data Type	Description
filePathName	java.lang.String	Optional. Name of the XML attribute that stores the full pathname. Defaults to `fullpath`.

sys_ftUploadAppendFileAttributes

Context:

Java/global/percussion/filetracker

Description:

This exit appends the size and modified datetime stamps to the update statistics document. This exit always goes with the ***sys_uploadFileAttributes*** (see "sys_DatabasePublisher" on page 203) preprocessor exit. The result statistics document has two elements added as first children, size and modified.

Class name:

com.percussion.uploadexits.PSUploadAppendFileAttrs

Resource:

file:classes

Interface:

com.percussion.extension.IPSResultDocumentProcessor

Parameters:

Name	Data Type	Description
FileSizeParam	String	The name of the HTML parameter that stores the file size. This must be the same as the name given for the preprocessor exit uploadFileAttributes. Always literal.
DateParam	String	The name of the HTML parameter that stores the modified date. This must be the same as the name given for the preprocessor exit uploadFileAttributes. Always literal.

sys_IncrementalContentFilter

Context:

Java/global/percussion/generic

Description:

This extension filters a content list, removing items which have already been published or unpublished. It performs an internal request to find the entry in the RXSITEITEMS table that corresponds to the current content item. If a valid entry is found, it removes the item from the content list. If no valid entry is found, it leaves the item in the content list.

This exit lets you use the same query resource for both full and incremental content lists. The second parameter, `switchparameter,` is optional.

If you specify the `switchparameter` name as `incremental` in the extension registration and your content list resource is `rx_pubContentLists/contentlist_generic.xml`:

- the resource returns a "filtered" content list if you include the parameter in the content list URL and set it to `yes`, for example: `/Rhythmyx/rx_pubContentLists/contentlist_generic.xml?variantid=101&incremental=yes`
- the resource returns a "full" content list if you do not include the parameter in the content list URL (or you include it but do not set it to `yes`), for example: `/Rhythmyx/rx_pubContentLists/contentlist_generic.xml?variantid=101`

If you do not specify the `switchparameter` name in the extension registration, the resource always returns a “filtered” content list.

You must create the internal request that finds the item in RXSITEITEMS. Add it to the content list application by performing the following steps:

1. Open the Content List application in the Rhythmyx Workbench.
2. Drag <Rhythmyx root>/DTD/contentlist.dtd onto the application window.
3. Rename the request itemstatus.
4. Open the Resource Editor and add the RXSITEITEMS table to the backend datatank.

5 Open the Selector and define the following conditions for the query:

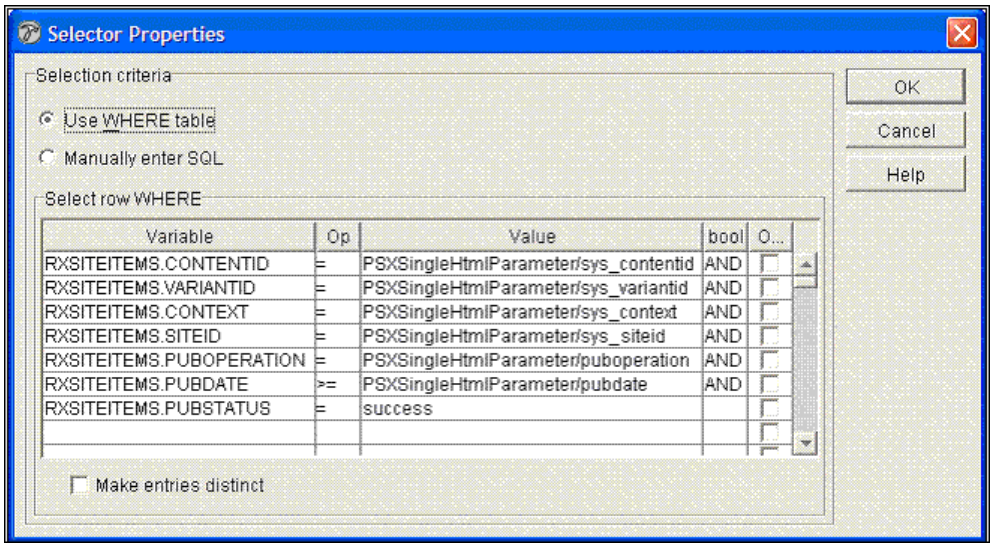


Figure 34: Selector to retrieve Items from RXSITEITEMS table.

This selection returns a single row (one content item) from the RXSITEITEMS table. REVISIONID is not required because only one revision of a Content Item should be present on a site.

6 Open the mapper and map RXSITEITEMS columns to their equivalents in contentlist.dtd. The mapping should resemble the following graphic, but exact mappings are not important, because the exit only tests the presence or absence of a result document.

It is important that you check Return empty XML at the bottom of the mapper. If there is no match on the query, the exit expects an empty document (which appears as `</contentitem>`) , not a document with empty subnodes (such as `<contentitem> <title/> <contenturl/> . . . <contentitem/>`). If it receives a document with empty subnodes, it will attempt to process it, which will result in an error.

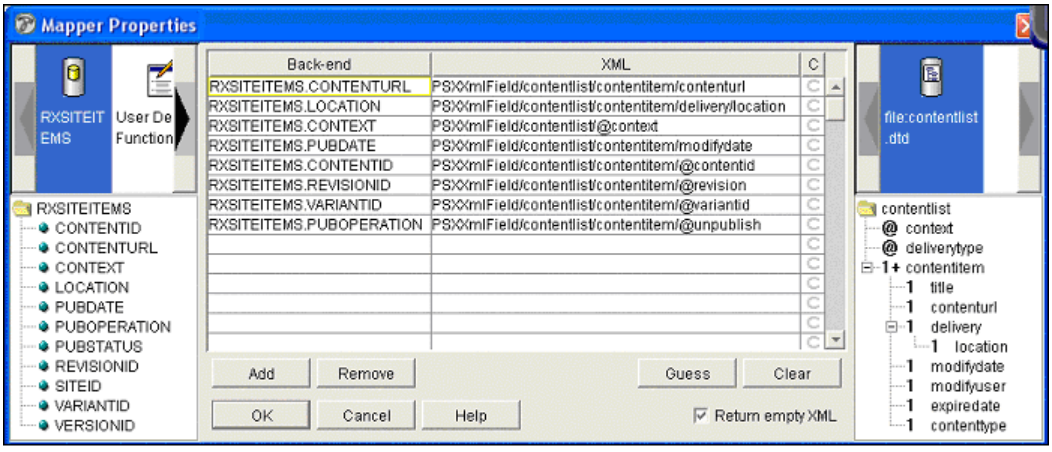


Figure 35: Mapping the support application for sys_IncrementalContentFilter

7 When you add the exit to a content list resource, set queryrequest equal to <application name>/itemstatus and set switchparameter as specified above.

Class name:

com.percussion.extensions.general.PSIncrementalContentFilter

Interface:

com.percussion.extension.IPSResultDocumentProcessor

Parameters:

Name	Data Type	Description
queryrequest	String	Name of internal request for site item lookup
switchparameter	String	Optional. Name of the HTML parameter for switching the filter on and off. Any name is valid. If the parameter is included and set to `yes,` the filter is turned on. If the parameter is included but omitted from the content list URL or set to any value other than `yes,` the filter is turned off. If the parameter is not included, the exit always returns a filtered content list.

sys_LoadChildData

Context:

Java/global/percussion/system/

Description:

This exit has a similar function as the sys_casAddChildInfo exit. Use it in Content Assemblers to add data from child tables to the assembled output.

The exit performs the query specified by the queryAttribute and replaces the childElement of the baseElement with the results of the query. Attach it to Content Assemblers for Content Types that store data in child tables when you need to include the data from the child table in the formatted output. If the formatted output does not need data from the child table, the assembly resource does not need this exit.

To use this exit, you must create a resource that queries data from the child table.

Class name:

com.percussion.cms.objectstore.server.PSLoadChildDataExit

Interface:

com.percussion.extension.IPSResultDocumentProcessor

Parameters

Name	Data Type	Description
baseElement	java.lang.String	Name of the element that will contain the child data.
childElement	java.lang.String	Name of the child element of the base element that will be replaced with the results of the query.
queryAttribute	java.lang.String	Name of the attribute of the child element that specifies the query to execute.

sys_ModifyXmlHierarchy

Context:

Java/global/percussion/generic

Description:

This extension is used to modify an XML documents hierarchy.

The concept of XML hierarchy modification is based on the need for setting up a discussion thread system. Each discussion topic submission can be considered separately, thus having no relationship with other submissions. However, most of the time, a submission is a "response" to a previously submitted topic. This creates a new discussion "thread," with response submissions becoming children to a parent submission topic. A relationship between the submissions is required to make this work.

To give a relationship to different submission topics, this exit uses a node-key pair comparison to provide a hierarchical relationship between submission topics.

Example:

(Pay special attention to the relationship between parentid and id attributes.)

```
node = Discussion/Topic

response key = Discussion/Topic/@parentid

parent node = Discussion/Topic

parent key = Discussion/Topic/@id
```

Original XML Document:

```
<Discussion>
   <Topic id="1" parentid="0">;
      <body>This is the first thread in the discussion<;/body>
      </Topic>
      <Topic id="2" parentid="0">;
      <body>This is the second thread in the discussion</body>
      </Topic>
      <Topic id="3" parentid="1">
```

```
        <body>;This is the first response to the first thread</body>
        </Topic>
   </Discussion>
```

After ModifyXmlHierarchyExtension exit:

```
   <Discussion>
     <Topic id="1" parentid="0">
       <body>This is the first thread in the discussion</body>
       <Topic id="3" parentid="1">
         <body>This is the first response to the first thread</body>;
       </Topic>
     </Topic>;
     <Topic id="2" parentid="0">
       <body>This is the second thread in the discussion</body>
     </Topic>
   </Discussion>
```

Class name:

com.percussion.extension.PSModifyXmlHierarchyExtension

Interface:

com.percussion.extension.IPSResultDocumentProcessor

Parameters:

Name	Data Type	Description
responseNode	java.lang.String	Required. The name of the XML node that contains the attribute, or "response key", for looking up the parent node of this submission topic.
responseKey	java.lang.String	Required. The attribute owned by the "response node". This key defines which node is this response node's parent by looking at the "parent key" defined in the "parent node". The value of the response key must be unique among all the response nodes.
parentNode	java.lang.String	Required. The name of the XML node that contains the attribute, or "parent key", for holding the key that response nodes use to look up to find the parent.
parentKey	java.lang.String	Required. The attribute owned by the "parent node". The value that the "response node" attempts to match with its "response key" to determine if this "parent node" is the parent.

sys_pubCreatePublisherConfig

Context:

Java/global/percussion/cms/publisher

Description:

This exit creates default configuration settings for a new remote publisher during registration. The default settings correspond to the records in the RXPUBLISHERCONFIG table for PUBLISHERID=0.

During registration of a new remote publisher, updating the RXPUBLISHER and RXPUBLISHERCONFIG tables by inheriting configuration settings corresponding to publisherid=0 for the new one requires complicated manual SQL. This exit replaces the manual SQL and simplifies upgrading the Rhythmyx application for registering a new remote publisher.

Class name:

com.percussion.publisher.server.PSExitCreatePublisherConfig

Interface:

com.percussion.extension.IPSResultDocumentProcessor

Parameters:

Name	Data Type	Description
htmlParamNewPublisherId	String	PublisherId of the new publisher being registered.

sys_PublishContent

Name:

sys_PublishContent

Context:

global/percussion/cms/publisher/

Description:

This extension publishes an Edition when a Content Item makes a Transition that is registered with sys_PublishContent as the Workflow Action. One common example of this is when a user wants all content to publish to a staging server immediately upon approval into a staging state.

The Workflow, Transition and Edition are specified in the file publish.xml. Before running the action, create this file in <Rhythmyx root>/rxconfig/Workflow/ in the format:

```
<?xml version="1.0"?>
```

```
<root>
     <PSXPublish>
          <PSXWorkflowId>5004</PSXWorkflowId>
          <PSXTransitionId>1</PSXTransitionId>
          <PSXEdition>5005</PSXEdition>
     </PSXPublish>
</root>
```

The Transition Id is unique only within the Workflow, not across Workflows.

If there are many requests to publish the same edition, this extension publishes one edition in addition to the one already running and ignores all other requests.

Class name:

com.percussion.extensions.publishing.PSPublishContent

Interface:

com.percussion.extension.IPSWorkflowAction

Parameters:

None

sys_PublishEditionForPreview

Name:

sys_PublishEditionForPreview

Context:

Java/global/percussion/cms/publisher/

Description:

This exit is used to in database publishing contexts when content is served using ASP/JSP, or similar applications. This exit creates a temporary "Edition" in the database so the user can preview Content Items in a live context. When the temporary "Edition" is published, the preview is displayed to the user, and the "Edition" data is removed from the database.

For details about the use of this exit, contact Percussion Software Technical Support.

Class name:

com.percussion.extensions.publishing.PSPublishEditionForPreview

Interface:

com.percussion.extension.IPSResultDocumentProcessor

Parameters

Name	Data Type	Description
editionId	java.lang.String	ID of the Edition to Publish. This Edition must be a Manual Edition.
previewVariant	java.lang.String	ID of the Variant the user will specify to preview.
assemblyVariant	java.lang.String	ID of the Variant that actually generates the data.
support application	java.lang.String	(Optional) Name of the application that supports the exit for this instance. Defaults to rx_pubPreviewEdition.
query resource	java.lang.String	(Optional) Name of the query resource in the support application. . Defaults to queryEdition.
update resource	java.lang.String	(Optional) Name of the update resource in the support application. Defaults to updateEdition.

sys_ReplaceResultDocument

Context:

Java/global/percussion/extensions/general

Description:

This exit enables an application to perform internal requests to different resources depending on the value of a condition. When the condition is met, a corresponding resource document replaces the original document. The Rhythmyx server does not support requests to resource names, but requires that you use the pipe name (internal request name).

To use this exit, do the following:

1. Create one or more resources to be executed conditionally by copying the original resource and modifying it and assigning pipe names in the Rhythmyx Workbench.
2. Place sys_ReplaceResultDocument as a post-exit on the original resource.
3. Assign the name of the conditional request document that you want to serve as a default to the parameter DefaultRequestName.
4. Assign the condition for choosing a conditional resource (rather than using the original document) to the parameter ConditionValue.
5. Assign conditions for choosing each conditional resource to FirstOptionValue, SecondOptionalValue, and so on, depending on the number of conditional requests.

6 Assign internal request names (pipe names of each resource) to FirstInternalRequestName, SecondInternalRequestName, and so on, depending on the number of conditional requests. FirstInternalRequestName should be the resource requested when FirstOptionValue is true, SecondInternalRequestName should be the resource requested when SecondOptionValue is true, and so on.

NOTE: If ConditionValue is not equal to any of the OptionValues, then DefaultRequestName is executed. If the resource that corresponds to a condition is null, when the condition is met, the exit does not replace the original document.

Class name:

com.percussion.extensions.general.PSReplaceResultDocument

Interface:

com.percussion.extension.IPSResultDocumentProcessor

Parameters:

Name	Data Type	Description
DefaultRequestName	String	Default resource name - choose this resource if the conditional statement is true, but none of the other conditions is met.
ConditionValue	String	Condition for choosing one of the conditional resources.
FirstOptionValue	String	First condition. (Condition for requesting FirstInternalRequestName)
FirstInternalRequestName	String	First conditional request name.
SecondOptionValue	String	Second condition. (Condition for requesting SecondInternalRequestName)
SecondInternalRequestName	String	Second conditional request name.
. . .		
NthOptionValue	String	Nth condition. (Condition for requesting NthInternalRequestName)
NthInternalRequestName	String	Nth conditional request name.

sys_ServerUserRoleSearch

Context:

Java/global/percussion/usersearch/

Description:

This exit can modify a result document by adding search results in the following cases:

1. Given the HTML parameter sys_command=GetRoles, it produces a list of server roles like:

```
<root>
  <role>role1</role>
  <role>role2</role>
</root>.
```

2. Given the HTML parameters sys_command=GetUsers and sys_role=roleName, it produces a list of users that are members of the role roleName like:

```
<root>
  <role>roleName
    <user>user1</user>
    <user>role2</user>
  </role>
</root>.
```

The element <root> is any Document element of the result document.

Class name:

com.percussion.extensions.usersearch.PSServerUserSearch

Resource file:

classes

Interface:

com.percussion.extension.IPSResultDocumentProcessor

Parameters:

None

sys_SetCookie

Context:

Java/global/percussion/generic/

Description:

This extension associates a cookie with the results to be returned to the requester in the HTML response document.

Class name:

com.percussion.extension.PSSetCookieExtension

Interface:

com.percussion.extension.IPSResultDocumentProcessor

Example:

```
name=MySessId2
value=1001
expires=12/31/1999 11:59 p
domain=www.percussion.com
path=/
isSecure=1
```

This associates the cookie named MySessId2 with all requests on the www.percussion.com Web server. The cookie is only sent over secure (SSL) connections. It has a value of 1001 and will expire on December 31, 1999 at 11:59:00 pm.

Parameters:

Name	Data Type	Description
name	java.lang.String	Required. The name of the cookie.
value	java.lang.String	Required. The value of the cookie.
expires	java.lang.String	Optional. The date the cookie expires. Use 'EEE, dd-MMM-yyyy hh:mm:ss z' as the date format (date or time may be omitted). Use 'literal' as the value type.
domain	java.lang.String	Optional. The domain name of the host from which the URI is accessed. For instance, to set a cookie for any Web server in the percussion.com domain, set the domain name to <code>percussion.com</code> To set a cookie for www.percussion.com, set the domain name to the full server name: www.percussion.com.
path	java.lang.String	Optional. Causes the exit to only send the cookie when accessing a URI under the specified path on the host. This includes the path and all descendents. For instance, using \"/\" matches all URI specifications on the host.

Name	Data Type	Description
isSecure	java.lang.String	Optional. A boolean value that determines the connection type. When set to \"1\", the cookie is only sent when a secure (SSL) connection has been established. When set to \"0\" or \"\", any connection type is acceptable.

sys_SetEmptyXmlStyleSheet

Context:

Java/global/percussion/generic/

Description:

This exit associates a style sheet with an empty XML document when there is no root node in the XML document. It is used primarily to return a static page when no data is found for a request.

Class name:

com.percussion.extension.PSSetEmptyXmlStyleSheetExtension

Interface:

com.percussion.extension.IPSResultDocumentProcessor

Parameters:

Name	Data Type	Description
styleSheet	java.net.URL	The URL of the stylesheet.

sys_wfAddPossibleTransitions

Context:

Java/global/percussion/workflow

Description:

This exit adds a node to the result document that contains actions appropriate for this document, including checkin/out, edit, preview and transitions.

Class name:

com.percussion.workflow.PSExitAddPossibleTransitionsEx

Interface:

com.percussion.extension.IPSResultDocumentProcessor

Parameters:

Name	Data Type	Description
UserName	java.lang.String	Name of the current user.
StatusDocumentElementName	java.lang.String	Node name (XML field name like root/document) of the status document.
ContentIDNodeName	java.lang.String	Node name (e.g. contentid or @contentid) of the content ID.

sys_wfAppendWorkflowActions

Context:

Java/global/percussion/workflow

Description:

This exit appends a list of all workflow actions registered by the server to the result XML document. It makes the <workflowactionlist> element a child of the root element in the document, and each action in the list a <workflowaction> element.

Class name:

com.percussion.workflow.PSGetWorkflowActionList

Interface:
com.percussion.extension.IPSResultDocumentProcessor

Parameters:
None

sys_wfExecuteActions

Context:
Java/global/percussion/workflow

Description:
This exit executes the assigned workflow actions for the transition.

Class name:
com.percussion.workflow.PSExecuteWorkflowActions

Interface:
com.percussion.extension.IPSResultDocumentProcessor

Parameters:
None

sys_wfPreviewWorkflow

Context:
Java/global/percussion/workflow

Description:
This exit transforms the result document into another DTD that the style sheet uses to generate the graphical view of the workflow. Use this exit in the workflow editor application.

Class name:
com.percussion.workflow.PreviewWorkflow

Interface:
com.percussion.extension.IPSResultDocumentProcessor

Parameters:
None

sys_wfSendNotifications

Context:
Java/global/percussion/workflow

Description:
This exit sends notifications to roles/ad-hoc users about the transition.

Class name:
com.percussion.workflow.PSExitNotifyAssignees

Interface:
com.percussion.extension.IPSResultDocumentProcessor

Parameters:

Name	Data Type	Description
ContentID	java.lang.Integer	Content ID.
UserName	java.lang.String	Name of the current user.

sys_wfUpdateHistory

Context:
Java/global/percussion/workflow

Description:
This exit updates content state history.

Class name:
com.percussion.workflow.PSExitUpdateHistory

Interface:

com.percussion.extension.IPSResultDocumentProcessor

Parameters:

Name	Data Type	Description
ContentID	java.lang.Integer	Content ID.
UserName	java.lang.String	Name of the current user.

sys_xdCopyDom

Context:

Java/global/percussion/xmldom

Description:

This post-exit copies a DOM tree fragment into the result document. This is similar to sys_xdDomToText, except that it copies a "fragment" of the source document under the root of the XML result document. Use this in document assemblers if the output is to be processed with a stylesheet that is aware of the XML document structure.

Class name:

com.percussion.xmldom.PSXdCopyDom

Interface:

com.percussion.extension.IPSResultDocumentProcessor

Parameters:

Name	Data Type	Description
sourceObjectName	java.lang.String	Name of source object. Defaults to XMLDOM.
sourceNodeName	java.lang.String	Name of source node. Leave blank or set to "." to copy the entire document
destNodeName	java.lang.String	Name of destination XML node. If the name is "." the new tree is copied under the root node.

sys_xdDomToText

Context:

Java/global/percussion/xmldom

Description:

Pre-exit or post-exit that transfers an XML document into a string for insertion as a single field either on insert or update or as the result of a query.

Class name:

com.percussion.xmldom.PSXdDomToText

Interface:

com.percussion.extension.IPSResultDocumentProcessor

Interface:

com.percussion.extension.IPSRequestPreProcessor

Parameters:

Name	Data Type	Description
SourceObjectName	java.lang.String	Name of source document object
SourceNode	java.lang.String	Name of node within source document. Use "InputDocument" if the source is an uploaded XML document. To copy the entire document, leave blank or set to "." Default is XMLDOM.
DestinationName	java.lang.String	Field or node where exit stores results. When this is used as a pre-exit, an HTML parameter name; when this is used as a post-exit, the name of an XML node added beneath the "Document Element" of the result document.

sys_xdRemoveElements

Context:

Java/global/percussion/xmldom

Description:

Post-exit that removes one or more XML elements from the result document. Use this to clean up the result document by removing unnecessary nodes after transformation.

Class name:

com.percussion.xmldom.PSXdRemoveElements

Interface:

com.percussion.extension.IPSResultDocumentProcessor

Parameters:

Name	Data Type	Description
element1	java.lang.String	Optional. Name of element to remove.
element2	java.lang.String	Optional. Name of element to remove.
element3	java.lang.String	Optional. Name of element to remove.
element4	java.lang.String	Optional. Name of element to remove.
element5	java.lang.String	Optional. Name of element to remove.
element6	java.lang.String	Optional. Name of element to remove.
element7	java.lang.String	Optional. Name of element to remove.
element8	java.lang.String	Optional. Name of element to remove.
element9	java.lang.String	Optional. Name of element to remove.
element10	java.lang.String	Optional. Name of element to remove.

sys_xdTextToDom

Context:

Java/global/percussion/xmldom

Description:

Pre- or post-exit that parses an input text source and produces a DOM document.

Class name:

com.percussion.xmldom.PSXdTextToDom

Interface:

com.percussion.extension.IPSResultDocumentProcessor

Interface:

com.percussion.extension.IPSRequestPreProcessor

Parameters:

Name	Data Type	Description
sourceName	java.lang.String	For a pre-exit, the name of the HTML parameter or attached file containing the source. For a post-exit, the name of the node containing the source.
DOMName	java.lang.String	Name of Temporary DOM Object. Default is "XMLDOM."
tidyProperties	java.lang.String	Optional. Name of Tidy Properties file.
serverPageTags	java.lang.String	Optional. Name of ServerPageTags file
encodingDefault	java.lang.String	Optional. Java name for character encoding of the source text. The value only affects uploaded files and overrides any value supplied by the browser.

sys_xdTextToTree

Context:

Java/global/percussion/xmldom/

Description:

Post-exit that parses source text and replaces it with a tree of XML nodes. The document element of the newly parsed document is attached underneath the original node.

Class name:

com.percussion.xmldom.PSXdTextToTree

Interface:

com.percussion.extension.IPSResultDocumentProcessor

Parameters:

Name	Data Type	Description
SourceNode	java.lang.String	Name of source node in the XML result document.
TidyProperties	java.lang.String	Name of tidy properties file.
ServerPageTags	java.lang.String	Name of ServerPageTags file.

sys_xdMultiTextToTree

Context:

Java/global/percussion/xmldom/

Description:

Post-exit to use in place of sys_xdTextToTree when a field occurs multiple times. If a content editor has a child editor with controls that require sys_xdTextToTree (e.g. <Rich_Text_Control_Name), use sys_xdMultiTextToTree. sys_xdTextToTree will only pick up the first occurrence of a field, but sys_xdMultiTextToTree will pick up all occurrences of the field.

Class name:

com.percussion.xmldom.PSXdMultiTextToTree

Interface:

com.percussion.extension.IPSResultDocumentProcessor

Parameters:

Name	Data Type	Description
SourceNode	java.lang.String	Name of source node in the XML result document.
TidyProperties	java.lang.String	Name of tidy properties file.
ServerPageTags	java.lang.String	Name of ServerPageTags file.

sys_xdTransformDom

Context:

Java/global/percussion/xmldom

Description:

Pre-exit or post-exit that runs the source DOM through an XSL stylesheet. It parses the result with the XML parser and stores it in the destination object. To ensure that the output is well-formed, use <xsl:output method="xml">.

The XSL stylesheet must reside in the current application directory. To do this, attach it to a query in the current application.

Class Name:

com.percussion.xmldom.PSXdTransformDom

Interface:

com.percussion.extension.IPSResultDocumentProcessor,
com.percussion.extension.IPSRequestPreProcessor

Parameters:

Name	Data Type	Description
sourceObjectName	java.lang.String	Optional. Source object name. Default is "XMLDOM." When used as a pre-exit, the special XML document name *InputDocument* may be used to refer to the input XML document (usually, this document is provided by the PSXmlUploader). When used as a post-exit, the special XML document name *ResultDocument* may be used. This name refers to the document passed as an argument to the exit (the document created by the Rhythmyx mapper).
StyleSheet	java.lang.String	Stylesheet name within current application.
destObjectName	java.lang.String	Optional. Destination object name. Can be the same as the source DOM name.

sys_xdTransformDomToText

Context:

Java/global/percussion/xmldom

Description:

Pre-exit or post-exit that transforms an XML document and stores the result as text. The output is not parsed, and therefore does not have to be well-formed. The stylesheet may create XML, HTML or plain text.

Class name:

com.percussion.xmldom.PSXdTransformDomToText

Interface:

com.percussion.extension.IPSResultDocumentProcessor,
com.percussion.extension.IPSRequestPreProcessor

Parameters:

Name	Data Type	Description
sourceObjectName	java.lang.String	Name of source DOM object. May be the special XML document name *InputDocument* when used as a pre-exit. This name refers to the input XML document. Usually, this document is provided by the PSXmlUploader. May be the special XML document name *ResultDocument* when used as a post-exit. This name refers to the document passed as an argument to the exit (the document created by the Rhythmyx mapper).
StyleSheet	java.lang.String	Stylesheet within the current application. This file must be stored in the current application's directory.
destObjectName	java.lang.String	Name of destination parameter or node. In a pre-exit, the value is always an HTML parameter. In a post-exit, the name of an XML node added beneath the "Document Element" of the result document. If you use a multiple-level name, only the last node is replaced. For example, if you use the name, *category/firstnode, category* must exist in the document. The exit creates *firstnode*, or replaces its first occurrence.

Request Preprocessing

sys_AddCurrentDateTime

Context:

Java/global/percussion/generic/

Description:

This exit adds the current date and time (relative to the Rhythmyx server) as an HTML parameter to the provided request. It formats the date and time according to the provided format pattern or to the default (yyyy-MM-dd HH:mm:ss) if no format pattern provided. Any Java ***SimpleDateFormat patterns*** (http://java.sun.com/j2se/1.3/docs/api/java/text/SimpleDateFormat.html) are acceptable.

Use this exit to add the date/time when building time dependent selection conditions (for example, during incremental publishing of content.)

Class name:

com.percussion.extensions.general.PSAddCurrentDateTime

Interface:

com.percussion.extension.IPSRequestPreProcessor

Parameters:

Name	Data Type	Description
htmlParamName	java.lang.String	The name of the HTML parameter to be added to the provided request. May be null or empty. If not provided, the default name `sys_NOW` is used.
formatPattern	java.lang.String	The pattern to use to format the current date and time. May be null or empty. If not provided, the exit uses the default of yyyy-MM-dd HH:mm:ss
dateOffset	java.lang.String	A negative or positive integer that indicates the number of days to offset the current date by. For example, if the current date is 5/30/2002, and the offset is –2, the current date is returned as 5/28/2002.
truncate	java.lang.String	Indicates whether or not to truncate the current time setting to the hour. For example, if the current time is 12:36:01 and truncate = yes, the current time is returned as 12:00:00 Values: `yes` = truncate `no` or `null` = do not truncate.

sys_caDeleteContent

Name:

sys_caDeleteContent

Context:

Java/global/percussion/ca/

Description:

This exit builds a content list for deletion by an update resource after deleting data from the content type specific tables by making internal requests to the content editor's purge resource. If the attempt to delete the data fails, the exit adds the content item to the skipped item list for deletion. Place this exit on an update resource that deletes rows from all system tables.

The DTD for the document is:

```
<!ELEMENT deleterows (row*, skipped) >
<!ELEMENT row (#PCDATA) >
<!ATTLIST row pkey CDATA #IMPLIED >
<!ELEMENT skipped (row*) >
```

Class name:

com.percussion.extensions.ca.PSDeleteContent

Resource file:

classes

Interface:

com.percussion.extension.IPSRequestPreProcessor

Parameters:

Name	Data Type	Description
keyParameterName	String	Name of the html parameter that holds the primary key from the backend table. Map keyParameterName to the backend table's primary key.

sys_casConcatAssemblyLocation

Name:

sys_casConcatAssemblyLocation

Context:

Java/global/percussion/contentassembler/

Description:

This exit concatenates all parameters in the RXLOCATIONSCHEMEPARAMS table to generate the assembly location. If no parameters are specified, it returns an empty string. It makes no checks, and transforms backslashes in parameters to forward slashes.

Use this as a generator exit.

Example:

```
params[0] + params[1] + . . . + params[n].
```

Class name:

com.percussion.cas.PSConcatAssemblyLocation

Interface:

com.percussion.extension.IPSAssemblyLocation

Parameters:

None

sys_casConcatWithIdAssemblyLocation

Name:

sys_casConcatWithIdAssemblyLocation

Context:

Java/global/percussion/contentassembler/

Description:

This exit concatenates values to generate an assembly location. It uses the first parameter as an index that specifies where to append the second parameter (contentid) in a concatenated list made up of parameters from the RXLOCATIONSCHEMEPARAMS table.

It requires at least 2 parameters from the table and handles as many parameters as provided. It checks that the minimum number of parameters are provided and that the index is in the range of the provided parameters. It transforms all backslashes in parameters to forward slashes. Use this as a generator exit.

Example:

If the index parameter = 1, a location string like this will be created:

```
params[1] + contentid + params[2] + . . . + params[n].
```

Class name:

com.percussion.cas.PSConcatWithIdAssemblyLocation

Interface:

com.percussion.extension.IPSAssemblyLocation

Parameters:

Name	Data Type	Description
index	java.lang.Object	Required. Number that specifies where, sequentially, to append the contentid in the list of other parameters. This is a string or an object which can be converted to a string using the toString method. When toString parses the string as an integer it must return a valid integer.
contentid	java.lang.Object	Optional. The contentid for which this extension creates the location URL. If not provided, takes the contentid of the current request.

sys_casDefaultAssemblyLocation

Name:

sys_casDefaultAssemblyLocation

Context:

Java/global/percussion/contentassembler/

Description:

This exit concatenates the specified parameters to generate the assembly location and adds the contentid before the suffix. (root+path+contentid+suffix). Use this as a generator exit.

Class name:

com.percussion.cas.PSDefaultAssemblyLocation

Interface:

com.percussion.extension.IPSAssemblyLocation

Parameters:

Name	Data Type	Description
root	java.lang.Object	Required. The root part of the location URL to be created. Forward and backward slashes are permitted. If this parameter does not end with a path delimiter, the exit adds one.
path	java.lang.Object	Required. The resource path part of the location URL to be created. Forward and backward slashes are permitted. Acceptable with or without start and end path delimiters.
suffix	java.lang.Object	Required. The suffix part of the URL location to be created. Acceptable with or without delimiter.

sys_casGenericAssemblyLocation

Name:

sys_casGenericAssemblyLocation

Context:

Java/global/percussion/contentassembler/

Description:

Builds a delivery location by concatenating all the text nodes of the XML document returned by the specified resource.

Class name:

com.percussion.cas.PSGenericAssembly

Interface:

com.percussion.extension.IPSAssemblyLocation

Parameters:

Name	Data Type	Description
resource	java.lang.String	Required parameter. URL of the resource (relative to the Rhythmyx root) that supplies the location. It should be of the form RhythmyxApplication/ResourceName

Name	Data Type	Description
contentid	java.lang.String	Optional parameter. It is the content id of the item. If specified, it will be added as an html parameter (sys_contentid) when querying the specified resource
revision	java.lang.String	Optional parameter. It is the revision id of the item. If specified, it will be added as an html parameter (sys_revision) when querying the specified resource

sys_casModifyRelatedContent

Name:

sys_casModifyRelatedContent

Context:

Java/global/percussion/contentassembler/

Description:

This exit handles all modification requests for related content items, including inserting items into a slot, moving an item up in a slot, deleting an item from a slot, moving an item down in a slot, and moving the item to another slot or changing the item variant within a slot. Creates an XML document that is input to the update resource.

The exit sets the DBActionType parameter based on whether the modification is inserting new rows, updating rows, or deleting existing row(s). It uses an internal query request to get the required information about the slot items.

Class name:

com.percussion.cas.PSModifyRelatedContent

Interface:

com.percussion.extension.IPSRequestPreProcessor

Parameters:

None

sys_CollapseHTMLParameter

Context:

Java/global/percussion/generic/

Description:

This exit collapses a multi-value (array) HTML parameter by taking the first value. In other words, it replaces an entire array with the first value of the array.

The number of parameters is fixed at 8, but it can handle any number of parameters. This exit is not required if you use the Single HTML parameter option in the Workbench.

Class name:

com.percussion.extensions.general.PSCollapseHtmlParameter

Interface:

com.percussion.extension.IPSResultDocumentProcessor,
com.percussion.extension.IPSRequestPreProcessor

Parameters:

Name	Data Type	Description
p1	java.lang.String	First array value
p2	java.lang.String	Second array value
p3	java.lang.String	Third array value
p4	java.lang.String	Fourth array value
p5	java.lang.String	Fifth array value
p6	java.lang.String	Sixth array value
p7	java.lang.String	Seventh array value
p8	java.lang.String	Eighth array value

sys_commAuthenticateUser

Name:

sys_commAuthenticateUser

Context:

Java/global/percussion/communities

Description:

This exit authenticates a user's community. Add sys_commAuthenticateUser to resources in applications that produce HTML pages that compose the CMS interface, including all resources in the sys_ca application and all resources that are attached to non-default stylesheets in the Publishing, Workflow and System applications.

If the Communities feature is disabled (communities_enabled=no in the server.properties file), the exit passes the authentication, sets the user's communityid to 0, and stores the Community as the user's session object (sys_community).

If the Communities feature is enabled (communities_enabled=yes in the server.properties file) the exit performs the following:

1. Tries to obtain the user Community from the session; if it cannot, tries to recover it from Cookies. Recovering communityid from Cookies is required when a session times out while a user is on a Rhythmyx page, because the server creates a new session, but the communityid is not available in it.
2. If it cannot obtain a user Community, it assigns the system Community (communityid=1). In this case, authentication automatically succeeds, and the exit goes to step 3.
3. After the exit obtains the user Community, if the user Community is not the system Community, the exit makes an internal request that produces a list of all of the user's Communities. If the list contains the user Community, authentication succeeds; otherwise, it fails.
4. If the user Community is the system Community, authentication automatically succeeds.
5. If authentication succeeds, the exit stores the user's Community as the session object.

Class name:

com.percussion.community.PSAuthenticateUser

Interface:

com.percussion.extension.IPSRequestPreProcessor

Parameters:

None

sys_ConvertCustomSearchOperator

Name:

sys_ConvertCustomSearchOperator

Context:

Java/global/percussion/cx/

Description:

Converts the custom search operator sent by the Content Explorer applet to the proper backend SQL operator. Can also convert the operator and value(s) sent to the appropriate SQL where clause syntax.

Class:

name:com.percussion.extensions.cx.PSConvertCustomSearchOperator

Interface:

com.percussion.extension.IPSRequestPreProcessor

Parameters

Name	Data Type	Description
operatorParamName	java.lang.String	Required. Name of the HTML parameter containing the operator to convert. If the valueParamName parameter does not have a value, the operator is converted and assigned to the HTML parameter specified in this parameter. In that case, only text operators are valid. If the valueParamName does contain a value, the HTML parameter specified in this parameter is not modified. For additional behavior, see the description of the valueParamName parameter.
valueParamName	java.lang.String	Optional Name of the HTML parameter containing the value or values to convert. If this parameter has a value, the operator specified in the operatorNameParameter and the value of this parameter are used to construct a SQL fragment that can be used in a WHERE clause. This SQL fragment is stored in the HTML parameter specified sqlFragmentParamName parameter. If the specified HTML parameter does not contain a value, no processing occurs.
sqlFragmentParamName	java.lang.String	Required if a value is defined in the valueParamName parameter. Name of the HTML parameter used to store a generated SQL fragment. For details about the SQL fragment, see the description of the valueParamName
backendColumnName	java.lang.String	Required if a value is defined in the valueParamName parameter. Specifies the backend database table column to use in the SQL fragment.

Name	Data Type	Description
backendColumnDataType	java.lang.String	Required. Datatype of the backend table column specified in the backendColumnName parameter. Valid options include TEXT(default), NUMBER, and DATE.
connectorOperator	java.lang.String	Optional. Specifies the operator or prepend to the SQL fragment. Valid values are AND and OR. If no value is specified in this parameter, no operator value is prepended to the SQL fragment. The value of this parameter is ignored if no value is specified for the valueParamName parameter, or if the HTML parameter specified in that parameter is NULL or invalid.
dateFormatString	java.lang.String	Required. Format to use if the backend column type is DATE. Must match the date format of the of your RDBMS. Formats must conform to one of the formats specified in the java.text.SimpleDateFormat. Defaults to *yyyy-MM-dd*.
useHtmlParamDoc	java.lang.String	Required. Flag specifying where to derive the HTML parameter values. If the value of this flag is *y*, HTML parameter values are derived from the submitted XML document. This document must conform to the format expected by PSHtmlParamDocument.fromXml(Element). If the value of this flag is *n*, the parameter values are derived from the HTML parameters in the submitted request.

sys_CopyParameter

Description:

This exit copies the request parameter named by the exit parameter "source" into the request parameter named by the exit parameter "destination"

Class Name:

com.percussion.extensions.general.PSCopyParameter

Interface:

com.percussion.extension.IPSRequestPreProcessor

Parameters

Name	Data Type	Description
source	java.lang.String	Request parameter name to be copied.
destination	java.lang.String	Request parameter name to receive copy.

sys_FileInfo

Context:

Java/global/percussion/generic/

Description:

This exit scans the server's HTML parameter map. For each file object it finds, it creates 0 or more metadata parameters and adds them to the map. The information it attempts to add is: filepath, filename, extension, MIME type, character encoding, and size. If it can find the information,it adds the parameter; otherwise it adds nothing for that property. The naming convention for the additional parameters is originalname_property. The corresponding property parameter suffixes are: _fullFilepath, _filename, _ext, _type, _encoding, _size, respectively.

The following table lists the sys_FileInfo suffixes, sample field names formed using the suffixes, and the fields' contents.

Suffix	Sample Field	Content
_fullFilepath	imagebody_fullFilepath	Original file path and name of the uploaded file.
_filename	imagebody_filename	The original filename of the uploaded file.
_ext	imagebody_ext	The file extension.
_type	imagebody_type	The MIME type and subtype.
_encoding	imagebody_encoding	The character encoding.
_size	imagebody_size	The length of the file, in bytes.

When you add a sys_File control or a sys_WebImageFX control to a field in a Content Type, Rhythmyx automatically adds sys_FileInfo as a dependency of the Content Type.

The _ext and _type fields provide information that helps browsers display your uploaded files correctly when you use the sys_file control.

The _filename and _type fields provide information that helps browsers display your uploaded files correctly when you use the sys_WebImageFX control.

See ***sys_File control*** (see "sys_File" on page 60) or ***sys_WebImageFX control*** (on page 65) for information.

Class name:

com.percussion.extensions.general.PSFileInfo

Interface:

com.percussion.extension.IPSRequestPreProcessor

Parameters:

None

sys_FlushCache

Context

Java/global/percussion/system/

Description

On a server specified as a Publishing hub, flushes all caches (Assembler cache, Resource cache, and Folder cache). On other servers, this exit is not activated.

This exit should be added to any Content List resource to ensure that all caches are flushed prior to publishing content.

Class Name

com.percussion.server.cache.PSExitFlushCache

Interface

com.percussion.extension.IPSRequestPreProcessor

Parameters

None

sys_FlushAssemblerCache

Context:

global/percussion/system/

Description:
Pre-exit that flushes all items from the server cache or flushes only the items specified by the parameters. If caching is not enabled for the server, calling this exit has no effect, but does not produce an error. Add this exit to any content list that includes an auto index to force the server to flush the variants of auto index content items before publishing them.

Class name:

com.percussion.server.cache.PSExitFlushCache

Interface:

com.percussion.extension.IPSRequestPreProcessor

Parameters

Name	Data Type	Description
htmlParamName	java.lang.String	The name of the application. The exit flushes items that match this application name. To omit the parameter, set this value to an empty string or null.
contentid	java.lang.String	A numeric content id. The exit flushes items that match this content id. To omit the parameter, set this value to an empty string or null.
revisionid	java.lang.String	A numeric revision id. The exit flushes items that match this revision id. To omit the parameter, set this value to an empty string or null. If contentvalue is null or empty, then this parameter's value must be null or empty also.
variantid	java.lang.String	A numeric variant id. The exit flushes items that match this variant id. To omit the parameter, set this value to an empty string or null.

Examples:

The following parameters cause the exit to flush all pages that the application casArticle assembles:

Parameter	Value
appname	"casArticle"
contentid	""
revisionid	""
variantid	""

The following parameters cause the exit to flush pages that include any Variant of the content item with content ID 125, revision ID 1:

Parameter	Value
appname	""
contentid	"125"
revisionid	"1"
variantid	""

The following parameters cause the exit to flush pages that include any variant with the specified variant ID:

Parameter	Value
appname	""
contentid	""
revisionid	""
variantid	"14"

sys_GetSessionVariable

Context:

Java/global/percussion/generic/

Description:

This pre-exit gets a variable from session object and populates an html parameter with it in response to a query. Use this with sys_SetSessionVariable, which stores a variable from an html query parameter.

Class name:

com.percussion.extensions.general.PSGetSessionVariable

Interface:

com.percussion.extension.IPSRequestPreProcessor

Parameters:

None

sys_imageInfoExtractor

Name:

sys_imageInfoExtractor

Context:

Java/global/percussion/exit/

Description:

Automatically extracts Image height and width when uploaded using a sys_file control, in addition to the filename, ext, type, and size parameters extracted by sys_FileInfo Exit.

Class name:

com.percussion.extensions.general.PSImageInfoExtractor

Interface:

com.percussion.extension.IPSRequestPreProcessor

Parameters

None

sys_MakeDeleteTableRowsXMLDoc

Context:

Java/global/percussion/general/

Description:

This exit builds an XML document that consists of a list of content items for deletion by the Rhythmyx update resource. The DTD for the document is:

```
<!ELEMENT deleterows (row*) >
<!ELEMENT row (#PCDATA)>
<!ATTLIST row pkey CDATA #IMPLIED>
```

Place this exit on a Rhythmyx update resource that deletes the rows from one or more backend tables. Map the XML element pkey to the primary key in the backend table(s).

Class name:

com.percussion.extensions.general.PSMakeDeleteRowsXmlDoc

Interface:

com.percussion.extension.IPSRequestPreProcessor

Parameters

Name	Data Type	Description
keyParameterName	String	Name of the html parameter that has the key parameter value(s)

sys_Nullif

Context:

Java/global/percussion/extensions/general

Description:

Sets the specified fields to null if their value matches the compareTo value. The comparison is case-sensitive.

For example, if you import the table values “Title,” “Mr.,” “Mrs,” and “Ms,” into the field Customer_Title in a drop list, you could use this exit to reset the field with value “Title,” which should not be included in the drop list, to null. In this example, you would set compareTo to Title and p1 to Customer_Title.

Class name:

com.percussion.extensions.general.PSNullIf

Interface:

com.percussion.extension.IPSRequestPreProcessor

Parameters

Name	Data Type	Description
compareTo	java.lang.String	Value that will be converted to null.
p1	java.lang.String	Name of first parameter to check.
p2	java.lang.String	Name of second parameter to check.
p3	java.lang.String	Name of third parameter to check.
. . .		
pN	java.lang.String	Name of nth parameter to check.

sys_ParameterTokenizer

Context:

Java/global/percussion/generic

Description:

This pre-exit splits input parameters with delimiters into a series of lists for input. This exit supports 3 delimiters: semicolon, period, and comma.

Example:

The related content search screen contains a series of checkboxes. The value of each checkbox contains the contentid and variantid of the inserted child document separated by a delimiter. All of the checkboxes have the same name. This results in a list of values in an ArrayList.

The function of this exit is to parse the delimited array into two or more other arrays. The number of arrays parsed depends on the number of parameters passed.

There are N parameters:

```
CheckBoxArrayName
FirstOutputArrayName
SecondOutputArrayName
```

etc, etc.

Class name:

com.percussion.extensions.general.PSParameterTokenizer

Interface:

com.percussion.extension.IPSRequestPreProcessor

Parameters:

Name	Data Type	Description
InputListName	java.lang.String	Name of the input HTML parameter
FirstOutputName	java.lang.String	Name of the first output HTML parameter
SecondOutputName	java.lang.String	Name of the second output parameter
ThirdOutputName	java.lang.String	Name of the third output parameter
FourthOutputName	java.lang.String	Name of the fourth output parameter
FifthOutputName	java.lang.String	Name of the fifth output parameter
SixthOutputName	java.lang.String	Name of the sixth output parameter
SeventhOutputName	java.lang.String	Name of the seventh output parameter

sys_PrepareInClause

Context:

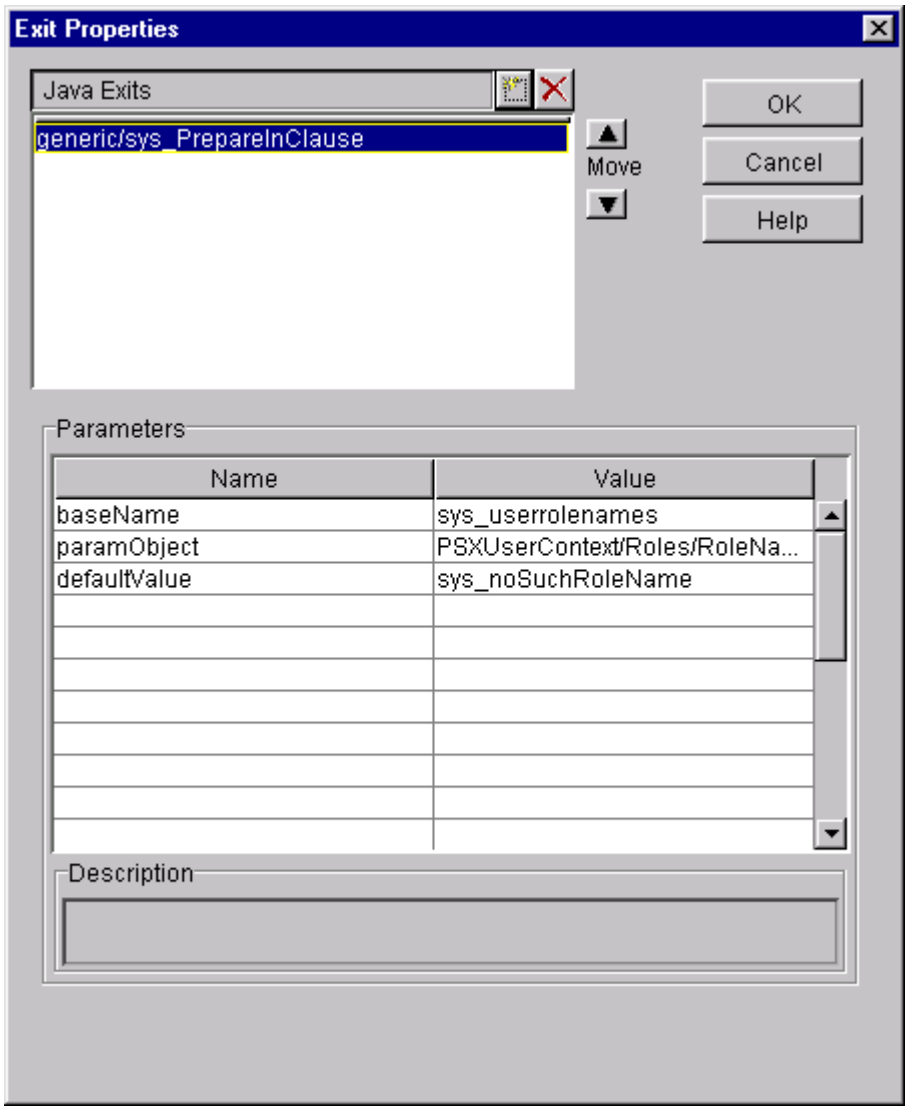
java/global/percussion/generic

Description:

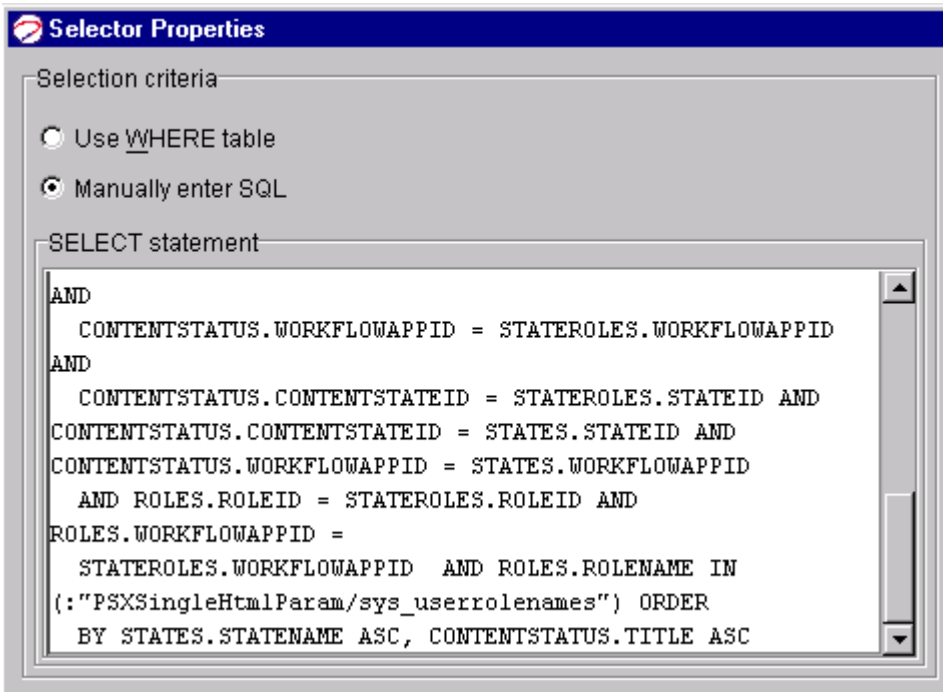
This exit formats a string from all objects in a Collection that is a valid "IN" clause for a SQL query. It stores the result in an HTML parameter, and it performs a toString() on each object in the Collection. The result does not include the parentheses.

Example:

The sys_ca application uses this exit to convert the "RoleNames" object into an HTML parameter for use in a select statement. As the following Exit Properties dialog shows, paramObject stores the object, and baseName stores the HTML parameter created from paramObject. In this example, the list object has one value, "RoleNames" and the select parameter created from the value is "sys_userrolenames." If paramObject had no value, the HTML parameter would take the default value of "sys_noSuchRoleName."


Exit Properties
Java Exits
generic/sys_PrepareInClause
Move
OK
Cancel
Help
Parameters
Name
Value
baseName
sys_userrolenames
paramObject
PSXUserContext/Roles/RoleNa...
defaultValue
sys_noSuchRoleName
Description

The select clause that uses the HTML parameter is the following:



Class name:

com.percussion.extensions.general.PSPrepareInClause

Interface:

com.percussion.extension.IPSRequestPreProcessor

Parameters:

Name	Data Type	Description
baseName	java.lang.String	The base name to use for the parameter created. May not be null or empty.
paramObject	java.lang.Object	An object implementing the java.util.Collection interface containing the values necessary for creating the values of the "IN" clause. May not be null, but may be empty.
defaultValue	java.lang.String	A value to use in the event that the Collection is null or empty. If this value is not null, it is used if the Collection does not have at least one value that resolves to a non-empty string when toString() is called.

sys_removeControlChars

Name:

sys_removeControlChars

Context:

Java/global/percussion/contenteditor/

Description:

This exit will remove control characters from all fields in a content editor. These characters are illegal in XML and will cause an exception if they are left in.

This exit should be added to Content Editor resources that use third-party applications (such as Microsoft Word) to edit text, and then only if the text edited in those applications contain control characters. If you encounter “Invalid XML character” errors when editing Content Items, try adding this exit to the Content Editor resource.

Class name:

com.percussion.ce.PSRemoveControlChars

Interface:

com.percussion.extension.IPSRequestPreProcessor

Parameters:

None

sys_SetArrayHtmlParameter

Name:

sys_SetArrayHtmlParameter

Context:

global/percussion/extensions/general/

Description:

Sets an HTML Parameter to the request with an array list of values. Makes an internal request to the request specified in the resourceName parameter and returns a list of values from the element specified in the elementName parameter. The number of result values returned can be limited by specifying a value for the maxNumber parameter in the exit, or by including the maxNumber parameter in the HTML request. The maxNumber parameter in an HTML request has a higher precedence than the maxNumber parameter in the exit..

Class:

com.percussion.extensions.general.PSSetArrayHtmlParameter

Interface:

com.percussion.extension.IPSRequestPreProcessor

Parameters:

Name	Data Type	Description
paramName	java.lang.String	Required. Name of the HTML parameter whose value you want to set.
resourceName	java.lang.String	Required. Specifies the name of the resource from which to request the data for the HTML parameter.
elementName	java.lang.String	Required. The name of the XML element from which the values will be extracted.
maxNumber	java.lang.String	Optional. If specified, only the specified number of values will be added to the HTML parameter defined in the paramName parameter of the exit. If this value is not specified, and if the HTML request does not include a maxNumber parameter, the list of values returned will be unrestricted. To control whether the first or last values in the list are returned, define a sort order in the Rhythmyx request. This exit always takes the first n values in the returned set.

sys_SetProviderTypeInstance

Context:

Java/global/percussion/system/

Description:

This pre-exit splits the security provider information into provider type and instance, thus creating 2 HTML parameters for queries.

This exit is necessary because the request gives the security provider type and instance as a single parameter in the format 'providerType/instance,' but the backend treats them as separate fields.

Class name:

com.percussion.security.PSSetProviderTypeInstance

Resource file:

classes

Interface:

com.percussion.extension.IPSRequestPreProcessor

Parameters:

Name	Data Type	Description
securityParameterName	java.lang.String	Optional. Security Provider Information. If null or empty, defaults to sys_securityProvider.
separator	java.lang.String	Optional. Separator that splits the security provider information. If null or empty, defaults to /.
providerTypeParamName	java.lang.String	Optional. HTML Parameter name for security provider type query. If null or empty, defaults to sys_spType.
securityInstanceParamName	java.lang.String	Optional. HTML Parameter name for security provider instance query. If null or empty, defaults to sys_spInstance.

sys_SetSessionVariable

Context:

Java/global/percussion/generic/

Description:

This pre-exit stores the value of a private session object based upon the value in an html request parameter. Use this with sys_GetSessionVariable, which accesses the information requested by a query.

Class name:

com.percussion.extensions.general.PSSetSessionVariable

Interface:

com.percussion.extension.IPSRequestPreProcessor

Parameters:

Name	Data Type	Description
param_name	java.lang.String	Name of html parameter

sys_TextExtraction

Name:

sys_TextExtractor

Context:

Java/global/percussion/contenteditor/

Description:

This pre-exit extracts the text and metadata in a binary or HTML (XML files cannot be processed by Text Extraction) 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 plain text.

An exit that uploads external files to Rhythmyx, either ***sys_fileInfo*** (on page 238) or ***sys_imageInfoExtractor*** (on page 242) always precedes sys_TextExtractor.

For information about performing text extraction with this exit, see "Implementing Text Extraction" in the document *Rhythmyx Implementation Guide*.

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.

Name	Data Type	Description
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.
OutputEncoding	java.lang.String	Encoding to use for output character set. Default is WINDOWS-1252. If you are using a multi-byte character type, you must specify the correct output encoding. Valid values are: 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 Note: Multi-byte characters are commonly used to represent ideograms in Asian languages such as Chinese.

If the implementer overrides any text extractors used for the full-text search, the new text extractors are used in this exit. For more information about overriding the default text extractors, see the *Search Configuration* section in the Rhythmyx Server Administrator online help.

sys_UploadFileAttributes

Context:

Java/global/percussion/generic/

Description:

This exit calculates the file size of an uploaded file in bytes and optionally gets the current date/time to be used as a modified date. It appends the modified date using an HTML parameter. Use this exit on an update resource that uploads a file to the database.

Class name:

com.percussion.extensions.general.PSUploadFileAttrs

Interface:

com.percussion.extension.IPSRequestPreProcessor

Parameters:

Name	Data Type	Description
FileNameParam	String	HTML Parameter name from the form that posts the file to server. For example, if FileNameParam is *contentbody*, the file upload/download manager always uploads the file with *contentbody* as a form field.
FileSizeParam	String	Name of the HTML parameter that stores the file size. Always literal. This is used in the mapper to put the size value into the database.
DateParam	String	Optional. Name of the HTML parameter that gets the current datetime stamp. Always literal. This is used in the mapper to put the file modified date in the database.
DateFormatString	String	Optional. Datetime format string. Always literal. For example, yyyy/MM/dd hh:mm:ss. Note that this may depend on the backend in which the file is saved. Default is MM/dd/yyyy hh:mm:ss a
FileSizeMax	String	Optional. Maximum size limit for the file. If the file exceeds this value, the exit throws an exception.

sys_wfAuthenticateUser

Context:

Java/global/percussion/workflow

Description:

This exit authenticates the current user for the user's role(s).

Class name:

com.percussion.workflow.PSExitAuthenticateUser

Interface:
com.percussion.extension.IPSRequestPreProcessor

Parameters:

Name	Data Type	Description
ContentID	java.lang.Integer	Content ID.
UserName	java.lang.String	Name of the current user.
RoleNameList	java.lang.String	Comma separated list of roles for this user.
CheckInOutCondition	java.lang.String	Whether or not to process. Continue process if this condition is met. Valid values are "ignore", "checkin" and "checkout."
RequiredAccessLevel	java.lang.Integer	Minimum access level required to authenticate the user. 1 - None, 2 - Reader and above, 3 - Assignee and above

sys_wfDisallowUpdatePublished

Context:
Java/global/percussion/workflow

Description:
This exit prevents updating of a document that is in the publish state.

Class name:
com.percussion.workflow.PSExitDisallowUpdatePublished

Interface:
com.percussion.extension.IPSRequestPreProcessor

Parameters:

Name	Data Type	Description
ContentID	java.lang.Integer	Content ID.

sys_wfNextNumber

Context:
Java/global/percussion/workflow

Description:
This exit gets the next number required for new unique IDs in the table inserts.

Class name:
com.percussion.workflow.PSExitNextNumber

Interface:
com.percussion.extension.IPSRequestPreProcessor

Parameters:

Name	Data Type	Description
htmlParamName	java.lang.String	Name of the html parameter to return the next number value.
htmlParamKey	java.lang.String	Name of the key for which the number is attributed. For example, the table name.

sys_wfNextNumberSecondary

Context:
Java/global/percussion/workflow

Description:
This exit the gets next number required for new unique IDs in the table inserts.

Class name:
com.percussion.workflow.PSExitNextNumber

Interface:
com.percussion.extension.IPSRequestPreProcessor

Parameters:

Name	Data Type	Description
htmlParamName	java.lang.String	Name of the html parameter to return next number value.

Name	Data Type	Description
htmlParamKey	java.lang.String	Name of the key for which the number is attributed. For example, the table name.

sys_wfPerformTransition

Context:

Java/global/percussion/workflow

Description:

This exit performs a valid transition and changes the content state accordingly.

Class name:

com.percussion.workflow.PSExitPerformTransition

Interface:

com.percussion.extension.IPSRequestPreProcessor

Parameters:

Name	Data Type	Description
ContentID	java.lang.Integer	Content ID.
UserName	java.lang.String	Name of the current user.
ActionTriggerName	java.lang.String	Unique action trigger (checkin, checkout or any transition trigger).

sys_wfPrepareQueryFilter

Context:

Java/global/percussion/workflow

Description:

Prepares filter(s) for the query depending on the user's roles and stores the filter(s) in an HTML parameter.

Class name:

com.percussion.workflow.PSExitPrepareQueryFilters

Interface:

com.percussion.extension.IPSRequestPreProcessor

Parameters

Name	Data Type	Description
UserName	java.lang.String	Name of the current user.
RoleNameList	java.lang.String	Comma separated list of user's roles.

sys_xdDomToFile

Context:

Java/global/percussion/xmldom

Description:

This pre-exit copies a temporary XML document as a text object into a temporary file. The user can map this file to a backend column using the destination name. This exit differs from sys_xdDomToText because it inserts the result in a temporary file. Use this exit if you have an XML file that you have converted to a DOM object using sys_xdTextToDom, and want to store the DOM object in the database as a file.

Class name:

com.percussion.xmldom.PSXdDomToFile

Interface:

com.percussion.extension.IPSRequestPreProcessor

Parameters

Name	Data Type	Description
SourceObjectName	java.lang.String	Name of source document object (the temporary XML document object). Use "InputDocument" to insert an uploaded XML document. Default is XMLDOM.
SourceNode	java.lang.String	Name of node within the source document to copy. To copy the entire document, leave blank or set to "." .
DestinationName	java.lang.String	The name of the HTML parameter name that stores the reference to the temporary file. The exit stores the object as a temporary binary file.

Name	Data Type	Description
Encoding	java.lang.String	Java name of encoding the exit uses when writing the file. If this is not specified, the exit uses the default platform.

sys_xdDomToText

Context:

Java/global/percussion/xmldom

Description:

Pre-exit or post-exit that transfers an XML document into a string for insertion as a single field either on insert or update or as the result of a query.

Class name:

com.percussion.xmldom.PSXdDomToText

Interface:

com.percussion.extension.IPSResultDocumentProcessor

Interface:

com.percussion.extension.IPSRequestPreProcessor

Parameters:

Name	Data Type	Description
SourceObjectName	java.lang.String	Name of source document object
SourceNode	java.lang.String	Name of node within source document. Use "InputDocument" if the source is an uploaded XML document. To copy the entire document, leave blank or set to "." Default is XMLDOM.
DestinationName	java.lang.String	Field or node where exit stores results. When this is used as a pre-exit, an HTML parameter name; when this is used as a post-exit, the name of an XML node added beneath the "Document Element" of the result document.

sys_xdDomToParams

Context:

Java/global/percussion/xmldom

Description:

This pre-exit copies the children of <PSXParam> elements to HTML parameters. Use this exit to simplify processing of multiple HTML parameters (instead of performing multiple calls to sys_xdDomToText). The exit assumes the input document has the format:

```
<PSXParam>
  <param1>value1</param1>
  <param2>value2</param2>
  <param3>value3</param3>
<PSXParam>
```

It creates an HTML parameter from each element in the source XML document using the element name as the parameter name and the element value as the parameter value. The new parameters are then set into the HTML parameter map.

If you include appendParameter and set it to "yes," the exit converts <PSXParam> elements with repeating nodes by storing each repeating node value in an array; otherwise, the exit replaces the value of the HTML parameter with each new value that it finds for it, so only the last repeating value is saved. For example, if an application simulates checkboxes, and produces an input document formatted like:

```
<PSXParam>
  <checkbox>value1</checkbox>
  <checkbox>value2</checkbox>
  <checkbox>value3</checkbox>
<PSXParam>
```

if you include appendParameter="yes" the checkbox parameter = [value1,value2,value3] (an array list). Otherwise, the checkbox parameter = value3 (a string).

Class name:

com.percussion.xmldom.PSXdDomToParams

Interface:

com.percussion.extension.IPSRequestPreProcessor

Parameters:

Name	Data Type	Description
sourceName	java.lang.String	Name of source XML document object. Use "InputDocument" if the source is an uploaded XML document.
appendParameter	java.lang.String	Optional. If appendParameter is set to "yes," all values of elements in the input doc are stored in an array. If appendParameter is not included or set to "no," each value of an element in the input doc replaces the previous value found for the element.

sys_xdProcessRelatedLinks

Name:

sys_xdProcessRelatedLinks

Context:

Java/global/percussion/xmldom/

Description:

This pre-exit scans a DOM tree for inline related links and images. It processes related links that are in the format:

```
<a href="http://RXServer:RxPort/Rhythmyx/AppName/Request.html?
      sys_contentid=123&sys_variantid=1">
```

and adds extra parameters for `sys_contentid` and `sys_variantid.` It performs this processing for all links and images, or any other `<html>` element that contains `src=` or `href=` attributes.

Class name:

com.percussion.xmldom.PSXdProcessRelatedLinks

Interface:

com.percussion.extension.IPSRequestPreProcessor

Parameters:

Name	Data Type	Description
SourceObject	java.lang.String	Name of XMLDOM private object. Use "InputDocument" if the source is an uploaded XML document. Default is XMLDOM.

sys_xdTextToDom

Context:
Java/global/percussion/xmldom

Description:
Pre- or post-exit that parses an input text source and produces a DOM document.

Class name:
com.percussion.xmldom.PSXdTextToDom

Interface:
com.percussion.extension.IPSResultDocumentProcessor

Interface:
com.percussion.extension.IPSRequestPreProcessor

Parameters:

Name	Data Type	Description
sourceName	java.lang.String	For a pre-exit, the name of the HTML parameter or attached file containing the source. For a post-exit, the name of the node containing the source.
DOMName	java.lang.String	Name of Temporary DOM Object. Default is "XMLDOM."
tidyProperties	java.lang.String	Optional. Name of Tidy Properties file.
serverPageTags	java.lang.String	Optional. Name of ServerPageTags file
encodingDefault	java.lang.String	Optional. Java name for character encoding of the source text. The value only affects uploaded files and overrides any value supplied by the browser.

sys_xdTextCleanup

Name:
sys_xdTextCleanup

Context:
Java/global/percussion/xmldom/

Description:

This pre-exit parses an input text source and produces a DOM document instead of a private object, then turns the <body> field back into a text object, replacing the original text field. The input text source can be an HTML parameter (for example, the DHTML editor) or an uploaded file. The exit scans the tree for inline related links unless the InLineDisable parameter is set to "Y".

Class name:

com.percussion.xmldom.PSXdTextCleanup

Interface:

com.percussion.extension.IPSRequestPreProcessor

Parameters:

Name	Data Type	Description
SourceName	java.lang.String	Name of source parameter. For a pre-exit, an HTML parameter or an attached file. For a post-exit, a node.
TidyProperties	java.lang.String	Optional. Name of Tidy Properties file.
ServerPageTags	java.lang.String	Optional. Name of ServerPageTags file.
encodingDefault	java.lang.String	Java encoding name to use for files. The value only affects uploaded files and overrides any value supplied by the browser.
DisableInlineLink	java.lang.String	Flag for disabling scanning of inline links. Set to "Y" to disable scanning of Inline Related Links.
AvoidTidyPrettyPrint	java.lang.String	Optional. Flag for using Document Builder's toString function instead of tidy's pretty print. Set to "yes" to use Document Builder's toString function; set to anything else to use tidy's pretty print. When you include sys_xdTextCleanup on a content editor application that uses the editor, Rhythmyx automatically sets this parameter to "yes" to avoid loss of blank lines in the HTML editor.

sys_xdTransformDom

Context:

Java/global/percussion/xmldom

Description:

Pre-exit or post-exit that runs the source DOM through an XSL stylesheet. It parses the result with the XML parser and stores it in the destination object. To ensure that the output is well-formed, use <xsl:output method="xml">.

The XSL stylesheet must reside in the current application directory. To do this, attach it to a query in the current application.

Class Name:

com.percussion.xmldom.PSXdTransformDom

Interface:

com.percussion.extension.IPSResultDocumentProcessor,
com.percussion.extension.IPSRequestPreProcessor

Parameters:

Name	Data Type	Description
sourceObjectName	java.lang.String	Optional. Source object name. Default is "XMLDOM." When used as a pre-exit, the special XML document name *InputDocument* may be used to refer to the input XML document (usually, this document is provided by the PSXmlUploader). When used as a post-exit, the special XML document name *ResultDocument* may be used. This name refers to the document passed as an argument to the exit (the document created by the Rhythmyx mapper).
StyleSheet	java.lang.String	Stylesheet name within current application.
destObjectName	java.lang.String	Optional. Destination object name. Can be the same as the source DOM name.

sys_xdTransformDomToText

Context:

Java/global/percussion/xmldom

Description:

Pre-exit or post-exit that transforms an XML document and stores the result as text. The output is not parsed, and therefore does not have to be well-formed. The stylesheet may create XML, HTML or plain text.

Class name:

com.percussion.xmldom.PSXdTransformDomToText

Interface:

com.percussion.extension.IPSResultDocumentProcessor,
com.percussion.extension.IPSRequestPreProcessor

Parameters:

Name	Data Type	Description
sourceObjectName	java.lang.String	Name of source DOM object. May be the special XML document name *InputDocument* when used as a pre-exit. This name refers to the input XML document. Usually, this document is provided by the PSXmlUploader. May be the special XML document name *ResultDocument* when used as a post-exit. This name refers to the document passed as an argument to the exit (the document created by the Rhythmyx mapper).
StyleSheet	java.lang.String	Stylesheet within the current application. This file must be stored in the current application's directory.
destObjectName	java.lang.String	Name of destination parameter or node. In a pre-exit, the value is always an HTML parameter. In a post-exit, the name of an XML node added beneath the "Document Element" of the result document. If you use a multiple-level name, only the last node is replaced. For example, if you use the name, *category/firstnode*, *category* must exist in the document. The exit creates *firstnode*, or replaces its first occurrence.

User Defined Function Processing

sys_Add

Context:

Java/global/percussion/generic/

Description:

This exit adds 2 UDF-supplied operands and returns the result.

Class name:

com.percussion.extensions.general.PSSimpleJavaUdf_add

Interface:

com.percussion.extension.IPSUdfProcessor

Parameters:

Name	Data Type	Description
leftOp	java.lang.Number	Left hand operator
rightOp	java.lang.Number	Right hand operator

sys_casGenerateAssemblerLink

Name:

sys_casGenerateAssemblerLink

Context:

Java/global/percussion/assemblers/

Description:

Generates an internal URL to the assembler for the specified variant that includes the parameters sys_contentid, sys_revision, sys_context, sys_variantid, sys_authtype, and pssessionid.

Class name:

com.percussion.cas.PSGenerateAssemblerLink

Interface:

com.percussion.extension.IPSUdfProcessor

Parameters

Name	Data Type	Description
variantid	java.lang.String	Variant id of the desired assembler (required)
contentid	java.lang.String	Optional override of sys_contentid
revision	java.lang.String	Optional override of sys_revision
authtype	java.lang.String	Optional override of sys_authtype

sys_casGeneratePubLocation

Name:

sys_casGeneratePubLocation

Context:

Java/global/percussion/contentassembler/

Description:

This exit generates the public location for context sensitive data in a Rhythmyx resource. The preview generator is hardcoded. The exit obtains the generator for all other contexts from the table RXLOCATIONSCHEME.

Class name:

com.percussion.cas.PSGeneratePubLocation

Interface:

com.percussion.extension.IPSUdfProcessor

Parameters:

Name	Data Type	Description
variantid	java.lang.Object	Required. The variantid for which this exit creates location URLs.
contentid	java.lang.Object	Optional. The contentid for which this exit creates location URLs. If not provided the exit uses the contentid of the current request.
revision	java.lang.Object	Optional. The revision for which this exit creates location URLs. If not provided the exit uses the revision of the current request.
context	java.lang.Object	Optional. If supplied, the exit uses this context instead of the context specified by the sys_context parameter.
siteid	java.lang.Object	Optional If supplied, the specified target siteid is used instead of the default value specified by the HTML parameter sys_siteid.
siteFolderid	java.lang.Object	Optional. If supplied, the specified folderid is used instead of the default value specified by the HTML parameter sys_folderid.
authtype	java.lang.Object	Optional. If supplied, the specified authtype overrides the value specified by the HTML parameter sys_authtype.

sys_DefaultPasswordFilter

Context:
Java/global/percussion/filter/

Description:
This exit takes a plain text string (a password) and encrypts it for a Rhythmyx security provider.

Class name:
com.percussion.filter.DefaultPasswordFilter

Interface:
com.percussion.security.IPSPasswordFilter

Parameters:
No user-supplied parameters. The server automatically supplies the password to the extension.

sys_Base64Decoder

Context:
Java/global/percussion/generic/

Description:
This exit decodes a base64 string to a string, and optionally character encodes the return string.

Class name:
com.percussion.extensions.general.Base64Decoder

Interface:
com.percussion.extension.IPSUdfProcessor

Parameters:

Name	Data Type	Description
encodedText	java.lang.String	The text to decode.
charEncoding	java.lang.String	The character encoding to use when creating the return string.

sys_Base64Encoder

Context:

Java/global/percussion/generic/

Description:

This exit encodes a normal text string to base64 string, and optionally character encodes the return string. Use this to encrypt passwords.

Class name:

com.percussion.extensions.general.Base64Encoder

Interface:

com.percussion.extension.IPSUdfProcessor

Parameters:

Name	Data Type	Description
encodedText	java.lang.String	The text to encode.
charEncoding	java.lang.String	The character encoding to use when creating the return string.

sys_Concat

Context:

Java/global/percussion/generic/

Description:

This exit concatenates up to 5 text strings.

Class name:

com.percussion.extensions.general.PSSuperConcat

Interface:

com.percussion.extension.IPSUdfProcessor

Parameters

Name	Data Type	Description
p1	java.lang.String	First text string

Name	Data Type	Description
p2	java.lang.String	Second text string
p3	java.lang.String	Third text string
p4	java.lang.String	Fourth text string
p5	java.lang.String	Fifth text string

sys_DateAdjust

Context:

Java/global/percussion/generic/

Description:

This exit updates the date according to the command of a corresponding user defined function (UDF) call. There are up to six calendar fields which can be adjusted: year, month, day, hour, minute, and second. These fields are integers; non-integers will be truncated. (Users are responsible for making these fields integers.)

Prior to the exit running, the user must define seven objects through the GUI. The first object is a string representing a date. The other six objects are numbers representing the quantity by which to adjust the date. The date string should be in a format recognizable by the Rhythmyx server's PSDataConverter, otherwise the exit throws an exception.

Class name:

com.percussion.extensions.general.PSSimpleJavaUdf_dateAdjust

Interface:

com.percussion.extension.IPSUdfProcessor

Parameters:

Name	Data Type	Description
date	java.util.Date	The date to modify
years	java.lang.Number	The number of years to adjust date
months	java.lang.Number	The number of months to adjust date
days	java.lang.Number	The number of days to adjust date
hours	java.lang.Number	The number of hours to adjust date
minutes	java.lang.Number	The number of minutes to adjust date
seconds	java.lang.Number	The number of seconds to adjust date

sys_Divide

Context:

Java/global/percussion/generic/

Description:

This exit divides operand 1 by operand 2 and returns the result as a float. operand 1 and operand 2 are supplied by a UDF.

Class name:

com.percussion.extensions.general.PSSimpleJavaUdf_divide

Interface:

com.percussion.extension.IPSUdfProcessor

Parameters

Name	Data Type	Description
leftOp	java.lang.Number	Operand 1 (dividend)
rightOp	java.lang.Number	Operand 2 (divisor)

sys_GetBase64EncodedBody

Context:

Java/global/percussion/generic

Description:

This exit retrieves the HTML document specified by the URL parameter, extracts the information between the <BODY> tags, base64 encodes it, and returns it as a String.

This enables users to publish partial pages, such as snippets or SSIs to a database during database publishing.

Class name:

com.percussion.extensions.general.PSGetBase64EncodedBody

Interface:

com.percussion.extension.IPSUdfProcessor

Parameters

Name	Data Type	Description
resource	Java.lang.String	The resource URI. Full, partial and relative URI's are supported and can have parameters. The supplied parameters are appended to the end. The pssessionid is always appended. Only HTTP requests are made, even if the fully qualified URI uses HTTPS. Relative URI's must be relative from the application root directory.
paramName1	java.lang.String	Optional. Name of the first HTML parameter.
paramValue1	java.lang.String	Optional. Value of the first HTML parameter.
ParamName2	java.lang.String	Optional. Name of the second HTML parameter.
ParamValue2	java.lang.String	Optional. Value of the second HTML parameter.
ParamName3	java.lang.String	Optional. Name of the third HTML parameter.
ParamValue3	java.lang.String	Optional. Value of the third HTML parameter.
ParamName4	java.lang.String	Optional. Name of the fourth HTML parameter.
ParamValue4	java.lang.String	Optional. Value of the fourth HTML parameter.
ParamName5	java.lang.String	Optional. Name of the fifth HTML parameter.
ParamValue5	java.lang.String	Optional. Value of the fifth HTML parameter.
ParamName6	java.lang.String	Optional. Name of the sixth HTML parameter.
ParamValue6	java.lang.String	Optional. Value of the sixth HTML parameter.
ParamName7	java.lang.String	Optional. Name of the seventh HTML parameter.
ParamValue7	java.lang.String	Optional. Value of the seventh HTML parameter.
ParamName8	java.lang.String	Optional. Name of the eighth HTML parameter.
ParamValue8	java.lang.String	Optional. Value of the eighth HTML parameter.
ParamName9	java.lang.String	Optional. Name of the ninth HTML parameter.
ParamValue9	java.lang.String	Optional. Value of the ninth HTML parameter.

Name	Data Type	Description
ParamName10	java.lang.String	Optional. Name of the tenth HTML parameter.
ParamValue10	java.lang.String	Optional. Value of the tenth HTML parameter.

sys_GetBase64Encoded

Context:

Java/global/percussion/generic/

Description:

This exit takes the same parameters as the exit sys_MakeIntLink, but instead of returning the URL string, it gets the contents with the built URL, and returns it as a base64 encoded string.

Class name:

com.percussion.extensions.general.PSGetBase64Encoded

Interface:

com.percussion.extension.IPSUdfProcessor

Parameters:

Name	Data Type	Description
resource	Java.lang.String	The resource which will be looked up through an internal request and its base64 encoded response will be returned.
paramName1	java.lang.String	Optional. Name of the first HTML parameter.
paramValue1	java.lang.String	Optional. Value of the first HTML parameter.
ParamName2	java.lang.String	Optional. Name of the second HTML parameter.
ParamValue2	java.lang.String	Optional. Value of the second HTML parameter.
ParamName3	java.lang.String	Optional. Name of the third HTML parameter.
ParamValue3	java.lang.String	Optional. Value of the third HTML parameter.
ParamName4	java.lang.String	Optional. Name of the fourth HTML parameter.
ParamValue4	java.lang.String	Optional. Value of the fourth HTML parameter.
ParamName5	java.lang.String	Optional. Name of the fifth HTML parameter.
ParamValue5	java.lang.String	Optional. Value of the fifth HTML parameter.
ParamName6	java.lang.String	Optional. Name of the sixth HTML parameter.
ParamValue6	java.lang.String	Optional. Value of the sixth HTML parameter.
ParamName7	java.lang.String	Optional. Name of the seventh HTML parameter.
ParamValue7	java.lang.String	Optional. Value of the seventh HTML parameter.

Name	Data Type	Description
ParamName8	java.lang.String	Optional. Name of the eighth HTML parameter.
ParamValue8	java.lang.String	Optional. Value of the eighth HTML parameter.
ParamName9	java.lang.String	Optional. Name of the ninth HTML parameter.
ParamValue9	java.lang.String	Optional. Value of the ninth HTML parameter.
ParamName10	java.lang.String	Optional. Name of the tenth HTML parameter.
ParamValue10	java.lang.String	Optional. Value of the tenth HTML parameter.

sys_GetFileSize

Context:

Java/global/percussion/extensions/general

Description:

This exit is required for the BEA accelerator. If you want to publish to the standard BEA setup, you need the size of the document (it is a non-nullable column in their DOCUMENT table).

Class Name:

com.percussion.extensions.general.PSGetFileSize

Interface:

com.percussion.extension.IPSUdfProcessor

Parameters:

Name	Data Type	Description
resource	Java.lang.String	The resource which will be looked up through an internal request and its size will be returned.
paramName1	java.lang.String	Optional. Name of the first HTML parameter.
paramValue1	java.lang.String	Optional. Value of the first HTML parameter.
ParamName2	java.lang.String	Optional. Name of the second HTML parameter.
ParamValue2	java.lang.String	Optional. Value of the second HTML parameter.
ParamName3	java.lang.String	Optional. Name of the third HTML parameter.
ParamValue3	java.lang.String	Optional. Value of the third HTML parameter.
ParamName4	java.lang.String	Optional. Name of the fourth HTML parameter.
ParamValue4	java.lang.String	Optional. Value of the fourth HTML parameter.
ParamName5	java.lang.String	Optional. Name of the fifth HTML parameter.

Name	Data Type	Description
ParamValue5	java.lang.String	Optional. Value of the fifth HTML parameter.
ParamName6	java.lang.String	Optional. Name of the sixth HTML parameter.
ParamValue6	java.lang.String	Optional. Value of the sixth HTML parameter.
ParamName7	java.lang.String	Optional. Name of the seventh HTML parameter.
ParamValue7	java.lang.String	Optional. Value of the seventh HTML parameter.
ParamName8	java.lang.String	Optional. Name of the eighth HTML parameter.
ParamValue8	java.lang.String	Optional. Value of the eighth HTML parameter.
ParamName9	java.lang.String	Optional. Name of the ninth HTML parameter.
ParamValue9	java.lang.String	Optional. Value of the ninth HTML parameter.
ParamName10	java.lang.String	Optional. Name of the tenth HTML parameter.
ParamValue10	java.lang.String	Optional. Value of the tenth HTML parameter.

sys_Literal

Context:

Java/global/percussion/generic/

Description:

This exit converts the UDF-supplied parameter to a string and returns it.

Class name:

com.percussion.extensions.general.PSSimpleJavaUdf_literal

Interface:

com.percussion.extension.IPSUdfProcessor

Parameters:

Name	Data Type	Description
p1	java.lang.Object	The source object

sys_MakeAbsLink

Context:

Java/global/percussion/generic

Description:

This exit creates an absolute URL with up to 6 name/value pairs specified in the parameters.

A URL has the following pieces for purposes of this description

```
<scheme>://<host><path-segments><resource>?<query>#<fragment>
```

Five basic forms are allowed for the supplied URL:

Fully qualified (e.g. `http://server:9992/Rhythmyx/approot/res.html`

Partially qualified (e.g. `/Rhythmyx/approot/res.html`)

Relative (e.g. `../myApp/res.html`)

Resource name only (e.g. `res.html`)

An empty string

Any of these forms may contain a query and fragment part. The exit assumes that any relative url is relative from the originating request's app root. If the supplied URL is fully qualified and the protocol is not 'http', the exit returns the supplied URL, unmodified. Otherwise, it substitutes any pieces supplied. If the supplied URL is not fully qualified,the exit adds the missing parts using the values from the originating request. For a partially qualified name, it adds the http protocol, server and port to the supplied name. For an unqualified name, it adds these items, plus the Rhythmyx request root and the originating application request root. For a relative name, it adds the http protocol, server, port, and Rhythmyx root, assuming the name is relative from the originating request's app root. For an empty string, it uses all parts of the originating request, substituting the supplied parameters. If the port is 80, it does not add a port number to the generated url.

Multiple name/value pairs may be specified for the parameters. For example, if the following were supplied as parameters:

```
resource = query1.html
param1 = city
value1 = Boston
param2 = state
value2 = MA
```

then the exit would generate the following URL (assuming the request was targeted directly at the Rhythmyx server):

```
http://rxserver:9992/Rhythmyx/MyApp/query1.html?city=Boston&state=MA</p
>
```

Note: The resource may contain parameters defined on it, in which case the exit appends the supplied parameters after the last parameter defined.

Class name:

com.percussion.extensions.general.PSMakeAbsLink

Interface:

com.percussion.extension.IPSUdfProcessor

Parameters:

Name	Data Type	Description
resource	java.lang.String	Relative resource without the parameters.
paramName1	java.lang.String	Optional. Name of the first HTML parameter.
paramValue1	java.lang.String	Optional. Value of the first HTML parameter.
paramName2	java.lang.String	Optional. Name of the second HTML parameter.
paramValue2	java.lang.String	Optional. Value of the second HTML parameter.
paramName3	java.lang.String	Optional. Name of the third HTML parameter.
paramValue3	java.lang.String	Optional. Value of the third HTML parameter.
paramName4	java.lang.String	Optional. Name of the fourth HTML parameter.
paramValue4	java.lang.String	Optional. Value of the fourth HTML parameter.
paramName5	java.lang.String	Optional. Name of the fifth HTML parameter.
paramValue5	java.lang.String	Optional. Value of the fifth HTML parameter.
paramName6	java.lang.String	Optional. Name of the sixth HTML parameter.
paramValue6	java.lang.String	Optional. Value of the sixth HTML parameter.

sys_MakeAbsLinkSecure

Context:

Java/global/percussion/generic

Description:

This exit creates an absolute URL with up to 6 name/value pairs specified in the parameters. It is identical to sys_MakeAbsLink, except, if the supplied URL is fully qualified and specifies the ***https*** (javascript:BSSCPopup('https.htm')) protocol, the link is generated using https instead of ***http*** (javascript:BSSCPopup('http.htm')) . If the supplied URL is not fully qualified or does not specify https protocol, the link is generated using http.

A URL has the following pieces for purposes of this description:

```
<scheme>://<host><path-segments><resource>?<query>#<fragment>
```

Five basic forms are allowed for the supplied URL:

Fully qualified (e.g. https://server:9443/Rhythmyx/approot/res.html

Partially qualified (e.g. /Rhythmyx/approot/res.html)

Relative (e.g. ../myApp/res.html)

Resource name only (e.g. res.html)

An empty string

Any of these forms may contain a query and fragment part. The exit assumes that any relative url is relative from the originating request's app root. If the supplied URL is fully qualified and the protocol is not 'http' or 'https', the exit returns the supplied URL, unmodified. Otherwise, it substitutes any pieces supplied. If the supplied URL is not fully qualified, the exit adds the missing parts using the values from the originating request. For a partially qualified name, it adds the http or https protocol, server and port to the supplied name. For an unqualified name, it adds these items, plus the Rhythmyx request root and the originating application request root. For a relative name, it adds the http or https protocol, server, port, and Rhythmyx root, assuming the name is relative from the originating request's app root. For an empty string, it uses all parts of the originating request, substituting the supplied parameters. If protocol of the URL is http and the port is 80, it does not add a port number to the generated URL.

The first parameter, useSecure, specifies whether to use https or http. If the value of this parameter is `yes`, and the original request used a secure channel or the supplied URL specifies https, the UDF uses https; if the value of the parameter is anything else, it uses http regardless of the protocol used by the original request.

Multiple name/value pairs may be specified for the parameters. For example, if the following were supplied as parameters:

```
useSecure=yes
resource = query1.html
param1 = city
value1 = Boston
param2 = state
value2 = MA
```

then the exit would generate the following URL (assuming the request was targeted directly at the Rhythmyx server and was made on a secure server):

https://rxserver:9443/Rhythmyx/MyApp/query1.html?city=Boston&state=MA</p>

Note: The resource may contain parameters defined on it, in which case the exit appends the supplied parameters after the last parameter defined.

Input parameters and resulting protocol and port used

useSecure	Original Request Protocol	Supplied URL Protocol	Resulting Protocol	Resulting port
no	HTTP	none	HTTP	originating request's port
no	HTTPS	none	HTTP	Rhythmyx server's default port
yes	HTTP	none	HTTP	originating request's port
yes	HTTPS	none	HTTPS	originating request's port
no	HTTP	HTTP	HTTP	port from supplied URL
no	HTTPS	HTTP	HTTP	Rhythmyx server's default port
no	HTTP	HTTPS	HTTP	originating request's port

Input parameters and resulting protocol and port used				
useSecure	**Original Request Protocol**	**Supplied URL Protocol**	**Resulting Protocol**	**Resulting port**
no	HTTPS	HTTPS	HTTP	Rhythmyx server's default port
yes	HTTP	HTTP	HTTP	port from supplied URL
yes	HTTPS	HTTP	HTTP	port from supplied URL
yes	HTTP	HTTPS	HTTPS	port from supplied URL
yes	HTTPS	HTTPS	HTTPS	port from supplied URL

Class name:

com.percussion.extensions.general.PSMakeAbsLinkSecure

Interface:

com.percussion.extension.IPSUdfProcessor

Parameters:

Name	Data Type	Description
useSecure	java.lang.String	Required. Flag specifying whether or not to use a secure connection. Enter `yes` to specify use of https for a secure connection; enter `no` (or any value other than `yes`) to specify use of http for a non-secure connection.
resource	java.lang.String	Optional. Relative resource without the parameters.
paramName1	java.lang.String	Optional. Name of the first HTML parameter.
paramValue1	java.lang.String	Optional. Value of the first HTML parameter.
paramName2	java.lang.String	Optional. Name of the second HTML parameter.
paramValue2	java.lang.String	Optional. Value of the second HTML parameter.
paramName3	java.lang.String	Optional. Name of the third HTML parameter.
paramValue3	java.lang.String	Optional. Value of the third HTML parameter.
paramName4	java.lang.String	Optional. Name of the fourth HTML parameter.
paramValue4	java.lang.String	Optional. Value of the fourth HTML parameter.
paramName5	java.lang.String	Optional. Name of the fifth HTML parameter.
paramValue5	java.lang.String	Optional. Value of the fifth HTML parameter.
paramName6	java.lang.String	Optional. Name of the sixth HTML parameter.
paramValue6	java.lang.String	Optional. Value of the sixth HTML parameter.

sys_MakeAbsLinkSecureEx

Context:

Java/global/percussion/generic

Description:

This exit creates an absolute URL with up to 10 name/value pairs. It is identical to sys_MakeAbsLinkSecure except that it allows you to specify a host name and port. Typically, the host specified is the name of the secure DNS server.

This UDF is preferred to the sys_MakeAbsLinkSecure UDF when using SSL for publishing.

Class name:

com.percussion.extensions.general.PSMakeAbsLinkSecureEx

Interface:

com.percussion.extension.IPSUdfProcessor

Parameters

Name	Data Type	Description
useSecure	java.lang.String	Optional. Flag specifying whether or not to use a secure connection. Enter `yes` to specify use of https for a secure connection; enter `no` (or any value other than `yes`) to specify use of http for a non-secure connection. Defaults to `yes`.
host	java.lang.String	(Optional) The host name to be used to produce the output url. If not specified, the host name of the originating request will be used.
port	java.lang.String	(Optional) The port to be used to produce the output url. If not specified, the port of the originating request will be used, subject to the value of the \"useSecure\".
resource	java.lang.String	Relative resource without the parameters.
paramName1	java.lang.String	(Optional) Name of the first HTML parameter.
paramValue1	java.lang.String	(Optional) Value of the first HTML parameter.
paramName2	java.lang.String	(Optional) Name of the second HTML parameter.
paramValue2	java.lang.String	(Optional) Value of the second HTML parameter.
paramName3	java.lang.String	(Optional) Name of the third HTML parameter.
paramValue3	java.lang.String	(Optional) Value of the third HTML parameter.
paramName4	java.lang.String	(Optional) Name of the fourth HTML parameter.

Name	Data Type	Description
paramValue4	java.lang.String	(Optional) Value of the fourth HTML parameter.
paramName5	java.lang.String	(Optional) Name of the fifth HTML parameter.
paramValue5	java.lang.String	(Optional) Value of the fifth HTML parameter.
paramName6	java.lang.String	(Optional) Name of the sixth HTML parameter.
paramValue6	java.lang.String	(Optional) Value of the sixth HTML parameter.
paramName7	java.lang.String	(Optional) Name of the seventh HTML parameter.
paramValue7	java.lang.String	(Optional) Value of the seventh HTML parameter.
paramName8	java.lang.String	(Optional) Name of the eighth HTML parameter.
paramValue8	java.lang.String	(Optional) Value of the eighth HTML parameter.
paramName9	java.lang.String	(Optional) Name of the ninth HTML parameter.
paramValue9	java.lang.String	(Optional) Value of the ninth HTML parameter.
paramName10	java.lang.String	(Optional) Name of the tenth HTML parameter.
paramValue10	java.lang.String	(Optional) Value of the tenth HTML parameter.

sys_MakeIntLink

Context:

Java/global/percussion/generic

Description:

This exit creates an absolute URL with up to 10 name/value pairs and adds user session information. The URL locates an internal resource.

The exit constructs a URL that the Rhythmyx server uses to make an internal request. Therefore, it differs from sys_MakeAbsLink because it always constructs the URL using 127.0.01 (the local server address) and the Rhythmyx server port (usually 9992) regardless of what is supplied or what the originating request used.

A URL has the following pieces for purposes of this description:

```
<scheme>://<host><path-segments><resource>?<query>#<fragment>
```

Five basic forms are allowed for the supplied URL:

Fully qualified (e.g. http://server:9992/Rhythmyx/approot/res.html

Partially qualified (e.g. /Rhythmyx/approot/res.html)

Relative (e.g. ../myApp/res.html)

Resource name only (e.g. res.html)

An empty string

Any of these forms may contain a query and fragment part. The exit assumes that any relative url is relative from the originating request's app root. If the supplied URL is fully qualified and the protocol is not 'http', it returns the supplied URL, unmodified. Otherwise, it substitutes any pieces supplied. If the supplied URL is not fully qualified, it adds the missing parts using the values from the originating request (except for the server address, which is always 127.0.0.1 and the port which is always the one on which the Rhythmyx server is listening). For a partially qualified name, it adds the http protocol, server and port to the supplied name. For an unqualified name, it adds these items, plus the Rhythmyx request root and the originating application request root. For a relative name, it adds the http protocol, server, port, and Rhythmyx root, assuming it is relative from the originating requests app root. For an empty string, it uses all parts of the originating request, substituting the supplied parameters. If the port is 80, it does not add the port number to the generated url.

Multiple name/value pairs may be specified for the parameters. For example, if the following were supplied as parameters:

```
resource = query1.html
param1 = city
value1 = Boston
param2 = state
value2 = MA
```

and the session identifier were sessionid, then the exit generates the following URL (the params do not necessarily appear in the order presented):

http://rxserver:9992/Rhythmyx/MyApp/query1.html?pssessionid=sessionid&city=Boston&state=MA</p>

NOTE: The resource may contain parameters defined on it, in which case the exit appends the sessionid after the last parameter defined.

Class name:

com.percussion.extensions.general.PSMakeIntLink

Interface:

com.percussion.extension.IPSUdfProcessor

Parameters:

Name	Data Type	Description
resource	java.lang.String	Relative resource without the parameters. Must be relative from the application root directory.
paramName1	java.lang.String	Optional. Name of the first HTML parameter.
paramValue1	java.lang.String	Optional. Value of the first HTML parameter.
paramName2	java.lang.String	Optional. Name of the second HTML parameter.
paramValue2	java.lang.String	Optional. Value of the second HTML parameter.
paramName3	java.lang.String	Optional. Name of the third HTML parameter.
paramValue3	java.lang.String	Optional. Value of the third HTML parameter.
paramName4	java.lang.String	Optional. Name of the fourth HTML parameter.

Name	Data Type	Description
paramValue4	java.lang.String	Optional. Value of the fourth HTML parameter.
paramName5	java.lang.String	Optional. Name of the fifth HTML parameter.
paramValue5	java.lang.String	Optional. Value of the fifth HTML parameter.
paramName6	java.lang.String	Optional. Name of the sixth HTML parameter.
paramValue6	java.lang.String	Optional. Value of the sixth HTML parameter.

sys_MakeIntRequest

Context:

global/percussion/extensions/general/

Description:

This UDF lets users map an XML element assembled in an exit or a Rhythmyx application in the Rhythmyx Workbench mapper. You can map any element in the XML to this UDF. The root of the XML document returned by this request must match the name of the XML element to which the UDF is mapped. Otherwise, the Rhythmyx server returns a runtime error.

For example, suppose you had the following mapping for this UDF:

In this example we map the element <RelatedContent> through sys_MakeIntRequest. This request must return an element <RelatedContent>, similar tot he following example code:

```
<RelatedContent>
 <infourls actionlisturl="Text" activeiteminfourl="Text"
contentstatusurl="Text" userstatusurl="Text"/>
 <linkurl sys_context="Text" sys_slotid="Text" relateditemid="Text"
sys_contentid="Text" sys_variantid="Text" slotname="Text">
   <Value current="Text"/>
 </linkurl>
</RelatedContent>
```

The following code shows this XML included with the final output of the application:

```
<sys_AssemblerInfo previewurl="Text" sys_siteid="Text"
pssessionid="Text" sys_command="Text" sys_contentid="Text"
sys_variantid="Text" sys_revision="Text" sys_context="Text"
sys_authtype="Text" type="Text" outputformat="Text" xml:lang="Text">
   <RelatedContent>
      <infourls actionlisturl="Text" activeiteminfourl="Text"
contentstatusurl="Text" userstatusurl="Text"/>
      <linkurl sys_context="Text" sys_slotid="Text" relateditemid="Text"
sys_contentid="Text" sys_variantid="Text" slotname="Text">
         <Value current="Text"/>
      </linkurl>
   </RelatedContent>
   <AssemblerProperties>
```

```
        <Property name="Text">
           <Value current="Text"/>
        </Property>
     </AssemblerProperties>
     <InlineLink url="Text"/>
  </sys_AssemblerInfo>
```

Class name:

com.percussion.extensions.general.PSMakeInternalRequest

Interface:

com.percussion.extension.IPSUdfProcessor

Parameters:

Name	Data Type	Description
resource	java.lang.String	Required. The Rhythmyx resource to which to make an internal request. Specifies the application and page of the dataset to which the internal request is to be made. May be as brief as `appName/pageName` or as extensive as `http://127.0.0.1:9992/Rhythmyx/AppTest/nov.xml?alpha=bravo&test=5`.
stylesheet	java.lang.String	Optional. The name of the stylesheet to be applied to the request result document. The stylesheet must be stored in a Rhythmyx application. If stored in the current application, just the file name is needed (e.g. transform.xsl). For other applications use the relative path (e.g. ../sys_resources/stylesheets/transform.xsl).
inheritParams	java.lang.String	Optional. A flag to specify whether or not to inherit the original request parameters for the internal request. Default is yes. Value is case-insensitive.
paramName0	java.lang.String	Optional. Name of the first HTML parameter. Parameter parsing stops at the first NULL or empty parameter name.
paramValue0	java.lang.String	Optional. Value of the first HTML parameter.
paramName1	java.lang.String	Optional. Name of the second HTML parameter.
paramValue1	java.lang.String	Optional. Value of the second HTML parameter.
paramName2	java.lang.String	Optional. Name of the third HTML parameter.
paramValue2	java.lang.String	Optional. Value of the third HTML parameter.
paramName3	java.lang.String	Optional. Name of the fourth HTML parameter.
paramValue3	java.lang.String	Optional. Value of the fourth HTML parameter.
paramName4	java.lang.String	Optional. Name of the fifth HTML parameter.
paramValue4	java.lang.String	Optional. Value of the fifth HTML parameter.
paramName5	java.lang.String	Optional. Name of the sixth HTML parameter.

Name	Data Type	Description
paramValue5	java.lang.String	Optional. Value of the sixth HTML parameter.
paramName6	java.lang.String	Optional. Name of the seventh HTML parameter.
paramValue6	java.lang.String	Optional. Value of the seventh HTML parameter.
paramName7	java.lang.String	Optional. Name of the eighth HTML parameter.
paramValue7	java.lang.String	Optional. Value of the eighth HTML parameter.
paramName8	java.lang.String	Optional. Name of the ninth HTML parameter.
paramValue8	java.lang.String	Optional. Value of the ninth HTML parameter.
paramName9	java.lang.String	Optional. Name of the tenth HTML parameter.
paramValue9	java.lang.String	Optional. Value of the tenth HTML parameter.

sys_MakeLink

Context:

Java/global/percussion/generic

Description:

This exit creates a URL (as a string) with up to 6 name/value pairs. It creates the URL from the supplied parameters and returns it. Up to 6 name/value pairs may be specified for the arguments. For example, if the following were supplied as arguments:

```
base = query1.html
param1 = city
value1 = Boston
param2 = state
value2 = MA
```

then it generates the following URL:

```
query1f.html?city=Boston&state=MA
```

Note: The base may contain parameters defined on it, in which case the exit appends the supplied parameters after the last parameter defined.

Class name:

com.percussion.extensions.general.PSMakeLink

Interface:

com.percussion.extension.IPSUdfProcessor

Parameters:

Name	Data Type	Description
baseUrl	java.lang.String	URL without the parameters.

Name	Data Type	Description
paramName1	java.lang.String	Optional. Name of the first HTML parameter.
paramValue1	java.lang.String	Optional. Value of the first HTML parameter.
paramName2	java.lang.String	Optional. Name of the second HTML parameter.
paramValue2	java.lang.String	Optional. Value of the second HTML parameter.
paramName3	java.lang.String	Optional. Name of the third HTML parameter.
paramValue3	java.lang.String	Optional. Value of the third HTML parameter.
paramName4	java.lang.String	Optional. Name of the fourth HTML parameter.
paramValue4	java.lang.String	Optional. Value of the fourth HTML parameter.
paramName5	java.lang.String	Optional. Name of the fifth HTML parameter.
paramValue5	java.lang.String	Optional. Value of the fifth HTML parameter.
paramName6	java.lang.String	Optional. Name of the sixth HTML parameter.
paramValue6	java.lang.String	Optional. Value of the sixth HTML parameter.

sys_Multilpy

Context:

Java/global/percussion/generic/

Description:

This exit multiplies operand 1 by operand 2 and returns the result. Operand 1 and operand 2 are supplied by a UDF.

Class name:

com.percussion.extensions.general.PSSimpleJavaUdf_multiply

Interface:

com.percussion.extension.IPSUdfProcessor

Parameters:

Name	Data Type	Description
leftOp	java.lang.Number	Operand 1
rightOp	java.lang.Number	Operand 2

sys_Subtract

Context:

Java/global/percussion/generic/

Description:

This exit subtracts operand 1 from operand 2 and returns the result.

Class name:

com.percussion.extensions.general.PSSimpleJavaUdf_subtract

Interface:

com.percussion.extension.IPSUdfProcessor

Parameters:

Name	Data Type	Description
leftOp	java.lang.Number	Operand 1
rightOp	java.lang.Number	Operand 2

sys_wfGetCheckOutUserStatus

Context:

Java/global/percussion/workflow

Description:

This exit returns a String that represents the status of the current document. Three values are possible:

0 - Not checked out

1 - Checked out by current user

2 - Checked out by another user

Class name:

com.percussion.workflow.PSGetCheckOutStatusUdf

Interface:

com.percussion.extension.IPSUdfProcessor

Parameters:

Name	Data Type	Description
userName	java.lang.String	The name of the user that currently has the document checked out. Usually obtained from a backend column in the CONTENTSTATUS table.

Workflow Action Processing

sys_TouchParentItems

Name:

sys_TouchParentItems

Context:

Java/global/percussion/extensions/general/

Description:

This action touches all "parent" (Owner) items of the current item in Relationships whose Category is Active Assembly. It finds all Ancestors of the Content Item in Active Assembly Relationships and updates them by putting the current date/time and current user name in the CONTENTLASTMODIFIEDDATE and CONTENTLASTMODIFIER columns of the CONTENTSTATUS table.

This exit uses the following resources in the sys_ceDependency application:

- parents.xml query - this resource must have a "pipe name" of parents.
- touchitem.xml - an update resource (with a pipe name of touchitem. this resource updates the CONTENTSTATUS table. The only parameter of touchitem.xml is sys_contentid. This parameter specifies a list of content IDs as a {link java.util.ArrayList ArrayList} object.

Class name:

com.percussion.extensions.general.PSTouchParentItems

Resource file:
classes

Interface:
com.percussion.extension.IPSWorkflowAction

Parameters:
None

sys_createTranslations

Name:
sys_createTranslations

Context:
global/percussion/workflow/

Description:
This action creates a Translation Content Item of the original Content Item in each Locale in which the original Content Item does not already have a corresponding Translation Content Item. The action uses a configuration file, sys_createTranslations.properties, which is located in the directory <Rhythmyxroot>/rxconfig/i18n. This file defines the type of Translation Relationship to create between the original Content Item and the Translation Content Item for each Locale. It also defines a list of Locales for which Translation Content Items will not be generated.

Class Name;
com.percussion.workflow.PSCreateTranslations

Resource File:
rxconfig/I18n/sys_createTranslations.properties

Interface:
com.percussion.extension.IPSWorkflowAction

Parameters:
None

Index

M

N

O

P

Q

R

S

T

U

V

W

X