Rhythmyx

Modeling a Content Management System

5 .7

Printed on 26 September, 2005

Copyright and Licensing Statement

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

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

Licenses and Source Code

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

Netscape Public License

Apache Software License

IBM Public License

Lesser GNU Public License

Other Copyrights

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

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

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

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

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

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

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

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

This product includes software developed by CDS Networks, Inc.

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

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

Modeling a Content Management System

Modeling a Rhythmyx Content Management System is the process of analyzing your business processes, Web site, and document output to define your Content Model and determine the system infrastructure you need to create, maintain, and publish your content.

The Content Model defines the Content Types and Variants you need. The Content Types define templates for content authoring and review. Variants define templates for assembling formatted output as snippets, pages, and documents. You can define a Content Model for an entire site or a section of a site, or for a complete set of documents or only selected documents.

The infrastructure model defines the hardware configuration, Communities, and Workflows required to implement the Content Model.

It is strongly recommended that you develop your CMS model before beginning implementation so you know what you need to implement. It is tempting to skip modeling and jump straight in to developing Content Editors and Content Assemblers. However, time invested modeling results in a faster, more efficient implementation because you have a road map to follow in developing your system. Modeling your CMS also helps you develop a clearer picture of how the various elements of your CMS should work together. With that stronger understanding, you can develop all of the elements of the CMS to work together smoothly rather than spending time troubleshooting elements that you developed independently.

Once you develop your model, you can begin implementing it. To implement your Content Model, you will build and implement a Content Editor for each Content Type. The Content Editor facilitates the creation, review, and life-cycle management of content items. You will also build and implement at least one content assembly application for each Content Type to define the outputs, or Variants, for content of that Content Type.

To implement your infrastructure model, you will set up your hardware and install the appropriate Rhythmyx licenses for each machine. You will also define the Communities and Workflows used in your system.

CHAPTER 2

Modeling Content

Content Types form the intersection between the needs of the business users and the needs of the web site. Business users need to create and disseminate content. The website needs to display that content in a manner that is usable and that communicates effectively with the customer. Content types are the mechanism by which business users enter content that is assembled by Rhythmyx into the final web pages.

You can approach modeling in two ways. In many cases, you can start with documents that already exist, such as press releases, job postings, or product listings. The business units that produce these documents probably already have templates for them. These templates, or sample copies of typical documents, can be useful tools in the process of developing your content type model.

If such documents or templates are not already available, you will need to examine your website to determine your content model. As you examine your website, try to look past the site structure to see the content behind the structure. The system handles structure and reformatting of content for different purposes automatically by assembling different Variants of the content types. Because Rhythmyx permits you to develop a broad variety of variants, or formatted outputs, you do not need a large number of content types, or input mechanisms. Thus, when you look at a page, try to envision the content item that delivered each item.

Suppose you have a page similar to the following graphic:

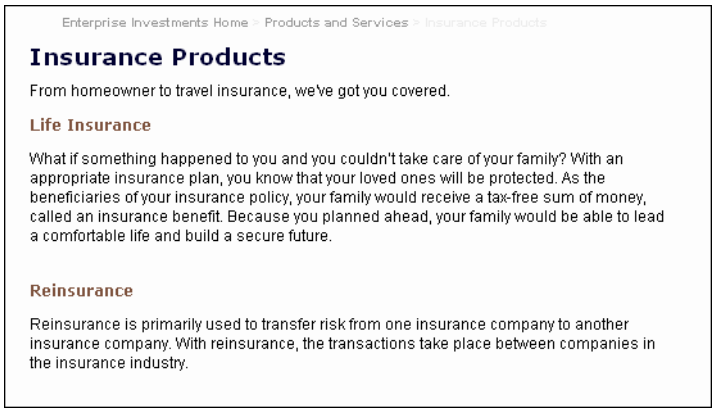


Figure 1: Generic Page Example

Note that in addition to the body content titled "Insurance Products" the page includes other content titled "Life Insurance" and "Reinsurance". You could include fields for these items on the content item, but Rhythmyx makes this unnecessary since it can pull the information in from other content items. You only need to be sure that content items are defined to provide this content.

As you review your site, compose a check-list to ensure that you create content types to deliver all content on the site. But when devising your content model, forget about the site. Focus instead on devising content types that meet the needs of the business users. Who creates the content? Who modifies it? What information do they need to communicate. How do they perceive this information? The content type definition should provide a mechanism to enter all the information needed that fits the perceptions of the business users.

Consider, too, that producing the information you want may require more than one content type. For example, producing a job posting may require a legal disclaimer, but different disclaimers might be required for different locations or different position. Human resources knows the correct disclaimer for given circumstances, but the posting department might not. You might therefore want two content types to produce the job posting. One content type might be the job description from the posting department, while human resources uses the other to produce the legal disclaimers. Items of these two content types would be merged to create the final job posting.

Most of your content types should provide mechanisms for business users to add content to the site, but you will need some content types that support the site as a whole. These content types generally aggregate other content types by defining relationships among them. Most commonly, these are index and automated index content types, which are typically used by editors who define the associations among content for specified portions of the site. You may also define site-specific content types that simplify the maintenance chores of the web master, such as maintaining banners. When defining these content types, you need to ask the same questions that you ask when defining content types for business users: Who creates the content? Who modifies it? What are they trying to communicate? How do they perceive the information they are trying to communicate.

Fleshing Out Content Types

To flesh out the content types in your content model you will need to:

- define the fields in the content type.
- define the metadata in the content type.
- define the behavior of each field in the content type.

Defining the Fields in a Content Type

A field is a distinct piece of information. When you look at the template or sample document, it may be easy to determine the fields that comprise the Content Type. In many cases, however, the fields will not be so obvious, or you may need to decide how you want your Content Type to work.

For example, a news release might consist of a title, dateline, abstract, body, and contact information. The dateline of a news release typically appears in-line with the body text, but you may decide you want to define it as a separate field, or even as two separate fields, since a dateline consists of a location and a date.

When defining fields, consider whether the content type includes information that you might want to assemble independently of other content. In the new release example, the blurb could be included as part of the body, but frequently, abstract are used in lists of related content. Therefore, you might want to define the abstract as a separate field so you can specify it for assembly more easily.

The way the information is created may also help define different fields. For example, information defined by choice lists or check boxes should generally be unique fields. You might want to limit the location choices in the dateline of the news release to a specific set of cities, for example, so you define the location as a separate field with a choice list that includes only those cities.

A third consideration is the type of data. The most common types of data are text, numbers, dates, and binary data, such as image files. Different data types require different fields. Thus, while the location in the dateline of the news release is text, like the abstract and body, the date is a different format, and therefore must be a separate field.

Finally, if some of the data will be processed differently, you will want to define a different field for it. For example, if some of the data has different validation rules than other data, it should have a different field.

The disadvantage to defining a content type with more fields is that it requires more structure for the content type, and more effort on the part of the user to understand it.

Rhythmyx comes with several system-wide fields already defined. These fields are part of all content types by default, unless they are specifically excluded. Further, you may find that several content types share fields defined locally. Rhythmyx includes a mechanism for defining fields shared by more than one content type, and including them in the content editor for the appropriate content types.

Defining the Metadata for a Content Type

Metadata is data about data; it provides additional definition for the data in the content type. Metadata generally serves one of the following purposes:

- Defining usage or processing of the data for the content type; where and when can the data be used.
- Facilitating searches, particularly by defining keywords.
- Targeting or categorization of content items.

Typically, the allowable values for metadata are strongly limited.

Rhythmyx includes several pieces of metadata as system fields, including start date, expiration date, and internal title.

Defining the Behavior of a Content Type

Once you have defined the fields and metadata for the content type, you must determine how users enter them and how the data will be processed. These factors will determine the controls you use to enter data and the processing rules applied to the data.

A content editor interface is a set of controls for entering data. Controls available include:

- Edit box: a text field that holds up to 50 alphanumeric characters.
- Text area: a free-form field that holds an unlimited number of alphanumeric characters.
- Drop-down: a field that allows users to select values from a drop list.
- Checkbox: a set of fields that the user can select or deselect.
- File: a text field of up to 50 characters that defines the path to a binary file for upload.
- Hidden input: a field that stores data that is hidden from the user. Data for hidden input controls is entered automatically by Rhythmyx.
- Simple calendar: a date field with a pop-up calendar to simplify date entry.
- HTML editor: a "what you see is what you get" HTML editor.
- Table: a set of fields in the form of a table. Generally, data edited in a table is stored in a child table in the database and is edited through a detail editor.

Controls define the structure of the data entered and can prevent the entry of "junk" data. The more structure a field has, the more important it is to select the correct control. For example, a Location field might be a simple edit box, but if you want to limit the options to a specific set of locations, you should use a drop-down control that limits the options for the field to prevent users from entering incorrect data.

Note that if you use a different environment to edit your content, you must use methods other than controls to define the Content Type's behavior. For example, Rhythmyx's Word Connector lets you create Content Editors that convert Word documents to text content. Content Editors that use the Word Connector convert specific Word styles to Content Editor fields. You must edit the Word template and the Word Connector files to define the behavior of Word styles and Content Editor fields.

Processing rules perform operations on data. It is easier to assign these rules to data if your fields are well structured. Rhythmyx supports the following processing rules:

- Input and output translation: converts data from one form to another when entered or displayed.
- Validation: confirms that the data in a field is valid for that field.
- Visibility: defines the circumstances in which a field is or is not displayed.

Content Models and Related Content

Related content marries your content model to the assembly process, but the real effort is developing and registering the variants for the content types and the slots available on those variants. One critical issue to keep in mind at this stage, however, is that some page elements that may appear to be fields may actually be slots. Images are a prime example. Because images are binary files, they generally have their own content type and are associated with content by assigning them to a slot on the content item rather than as a field on the content item.

CHAPTER 3

Modeling Communities

Communities let you add efficiency to your Rhythmyx Content Management System by isolating the content users work with. Each content item is a member of one, and only one Community. A content item is a member of the Community in which it is created, and exists only in that Community. Only users logged in to a Community can work with content in that Community. Communities also control the interface components and Active Assembly Variants available to the user.

Communities are not, however, a substitute for security. Communities simplify the CMS for the user by filtering out information that is not relevant to them. Communities do not provide the security available through the server and ACL settings.

Determining the Communities You Need

Communities separate users into similar groups. Communities do not cross Site boundaries, so at a minimum you will need one Community for each Site that you manage in the CMS.

You may want to isolate some users or their content from the rest of the Site Community. For instance, you might want to isolate the content and management of the Human Resources section of an intranet from the rest of the intranet. The CMS for a power company or medical device company might need to isolate regulatory affairs from the rest of Site. Any group of users or content you want to isolate requires a unique Community.

Note that Communities are not hierarchical. You cannot have a Site Community and have, for example, a Human Resources Community subordinate to that Site Community. The Human Resources Community would be an entirely separate and unique Community.

You need to strike a balance when defining your Communities. The Communities need to be big enough that users do not have to change Communities constantly to work with different content. At the same time, you do not want your Communities so large that the user is overwhelmed by the amount of content they have to manage.

Assigning Elements to Communities

The following Rhythmyx elements must be associated with a Community to make them available to the Business User:

- Sites
- Roles
- Workflows
- Content Types
- Variants
- Interface Components

Sites can only be associated with a single Community. To implement ***Parallel Communities*** (see "Modeling Parallel Workflows" on page 1), you should have a unique Role for each Community as well. Users will be members of this Role when they log in to the Community. The remaining elements can be assigned to multiple Communities.

You might want to keep a running list of your Communities and the CMS elements associated with each. As you develop new CMS elements, add them to the list under the Communities with which you want to associate them.

Community Modeling as an Iterative Process

While you can quickly outline the Communities that you need, most of Community modeling is iterative. As you model and develop additional CMS elements, determine which Community each element belongs to. This process refines your Community model to include the new elements. If you extend your CMS to manage additional parts of your site, determine whether you need a new Community or can use an existing Community. As you model and develop new CMS elements for the new section, be sure you add them to the appropriate Community.

CHAPTER 4

Modeling Workflows

A Workflow is a business process that follows a sequence of events. In Rhythmyx, every Content Item must exist in a Workflow. The Workflow determines:

- the steps in the processing of the Content Item;
- which users can access the item at each phase in the process; and
- what the users can do with the item at that point.

In general, you need one Workflow per Content Type, although occasionally two or more Content Types can use the same Workflow. Different classes of Content Types, however, definitely need different Workflows. For example, text-rich Content Types require different Workflows than graphic or file management Content Types. Different users, generally with different skills, are responsible for the processes of creation and review of each class of content.

Developing a Basic Serial Workflow

When developing a Workflow, start simply. Look at the process of creating a content item, reviewing and approving it, and making it available for publishing. Look at the process only in one direction, from start to finish. What are the steps in the process? Each step will become a State in your Workflow.

Next, ask who acts on the content item at each step. The answer to this question determines the Roles in your Workflow. A user must be in a Workflow Role assigned to a State to have access to the content in that State.

Refining the Workflow Model

Once you have developed the basic Workflow, you can refine it.

Go back to each State.

- What happens if the content is rejected from that State?
  - Which State does it go to?
  - If the content moves forward, is it possible it might move to a different State than the one you initially specified?
- What happens to Archived content?
  - Can it be revived?
  - If so, what happens to it when it is reused? Must it go back to the beginning of the Workflow and repeat the entire process? Or does it return to some intermediate State? Or does it proceed through a new set of States?

These questions help you determine whether you need additional Transitions for the Workflow, and what those Transitions should be. Each Transition moves a content item from one State to another. In the process of examining your Workflow, you may also determine that you need additional States that you did not include in your original model.

- Once a content item is in a Public State, who can take it out of a Public State? In what circumstances?
- Do you only want to remove the item to archive it, or do you want to make it available for minor edits?

The answers to these questions may tell you that you need additional Roles, and perhaps more States as well.

Modeling Public States

Each Workflow must include at least one State that is Public. Content in a State specified as Public is eligible to be published when Rhythmyx produces output. Typically, a Workflow includes a set of States associated with the Public State.

The first of these States is a Pending State. Content in a Pending State is ready to be published, but related content items may not be ready yet. A Pending State is particularly important for content items that have a strong dependency relationship with another content item. A strong dependency is a content relationship in which both items in the association must enter a Public State together. For example, if you have a text content item with an associated graphic, you might want to ensure that both items are published at the same time. When you add graphic to the text item in Active Assembly, you would define a strong dependency relationship to ensure that the text item could only go Public if the graphic was ready to go Public as well. Typically, text and graphic items are in different Workflows and are managed by different individuals. In this situation, it would be easy for content to become trapped in the Workflow, neither item able to advance until the other item advanced.

A Pending State resolves this issue. Adding a Pending State prior to the Public State gives users a chance to marshal content before making it Public. In this example, if both Workflows included a Pending State that all content entered before entering the Public State, the text item could wait in Pending until the graphic was ready. When the graphic entered the pending State in its own Workflow, both items could be Transitioned to Public.

An Archive State should always accompany a Public State. An Archive State is a storage holding State for content that has expired. Content in this State may be available to Transition back to an active State.

Modeling Quick Edit States

Another State typically associated with the Public State is a Quick Edit State. Content in a Public State cannot be edited. You might need to make minor edits, however, such as to correct misspellings. To make these edits manually, you would need to Transition the Content Item to an editable State, then check it out. This process can be cumbersome, however, if the user only needs to fix a minor problem like a simple misspelling. To make the process easier, Rhythmyx provides a menu option *Edit* > *Quick Edit* for each Content Item in a Public State. This menu option provides a single action that Transitions the Content Item to an editable State and checks it out to the user.

To make Quick Edit available to Content Items in a Workflow, the Workflow must include:

- a Quick Edit State;
- a Quick Edit Transition from the Public State to the Quick Edit State; and
- a Transition from Quick Edit back to Public.

Be sure to include these elements in your Workflow.

Modeling Parallel Workflows

You can use the same Workflow across multiple Communities, which is called a parallel Workflow. Thus, if you have several Communities in your system that use the same Content Types, or whose Content Types follow the same Workflow, you can use the same Workflow for all Communities.

To gain access to a content item, a user must meet the following criteria:

- The user must be logged in to the Community.
- The user must be logged in to the Role associated with that Community.
- The user must be in a Workflow Role assigned to the current State of the item in the Workflow.

If a user does not meet all criteria, Rhythmyx does not grant access to the content item.

To make parallel Workflow work:

- you must associate a Community-specific Role with each Community;
- each State in the Workflow must have one or more Roles assigned;
- in each Transition in the Workflow, the Approval Type field must be set to Each Role;
- the Roles that can make the Transition must be specified.

Modeling Notification

Notification is a mechanism that can automatically send e-mail messages to users in States associated with a Transition. You can send Notification messages to users in the destination State (the State to which the content is Transitioning), the origin State (the State from which the content is Transitioning), both, or neither. You can also add lists of additional users you want to receive the messages.

When modeling Notification, consider the volume of e-mail you will be generating. Notifications for each item at each Transition can generate a large volume of e-mail. You can often assume that users in some Roles know they need to act on Content in States associated with their Roles. For example, if your Workflow has a QA State, users in the QA Role should know they need to test content in that State. Best practice is to reserve Notification for specific cases when Notification would be important, such as notifying infrequent users of content they need to act on. For example, you might have your lawyers review specific pieces of content. The lawyers may not use Rhythmyx regularly, so they would need to be notified when content entered a Legal Review State. Another case where you might want to use Notification is to handle exceptional situations, such as content Transitioning to a State used only when errors occurred. You might want to enable Notification to ensure that a user reviews the content.

As you review your Workflow, consider whether the users in each State would ordinarily know to act on content in that State. If so, you probably can get by without Notification for the Transitions associated with that State. If not, you should consider adding Notification.

Modeling Aging

Aging is a Rhythmyx Workflow feature that defines the amount of time content can reside in a State before Rhythmyx will act on it automatically. You implement Aging by defining special Transitions that act on content automatically after a specified period or time or a specified date has passed.

The most common use of Aging is to remove expired content from a Web site automatically by Transitioning it when its expiration date has passed.

Other cases when you might want to use Aging are if you want to remind users to act on content in a specific State, or when you want to push content forward automatically regardless of whether the user has acted on it. For example, if you have time-sensitive content, you might want to push it forward regardless of whether a user has acted on it. In this case, you might define an Aging Transition to remind the user that they have content they need to act on and another that would make the content Public on its Start Date.

As you review your Workflow, consider whether you want to remind users that they have content they need to act on, or whether time considerations might compel you to push content forward regardless of whether the users have acted on it.

Modeling Relationships

Relationships are used throughout Rhythmyx for purposes such as Active Assembly. The most common Relationships that implementers develop are Workflow associations that link content items. In many cases, you want to link Content Items so they become public at the same time. For example, an article may have one or more associated graphics. You don't want the article to appear on your Web site without the graphics. In this case, Rhythmyx already provides a Relationship (Related Content, Mandatory) for these related content associations.

You may, however, want to create your own versions of these Relationships. To model these Relationships, sketch out what you want your system to do. Do you want the Relationship to create copies of the original Content Item? Or is it an association between two existing Content Items? What special processing should the Relationship perform? Will existing Exits or Effects accomplish your goal? Or will you need to write custom Extensions? Will processing occur under specific circumstances? What are those circumstances? The answers to these questions define the properties of the Relationship, the Exits and Effects that implement the Relationship processing, and the conditions that trigger those Extensions.

CHAPTER 5

Modeling Assembly

Most sites use a hierarchical organization, dividing the site into sections, each section into sub-sections, and sub-sections into yet smaller units, as shown in the following graphic:

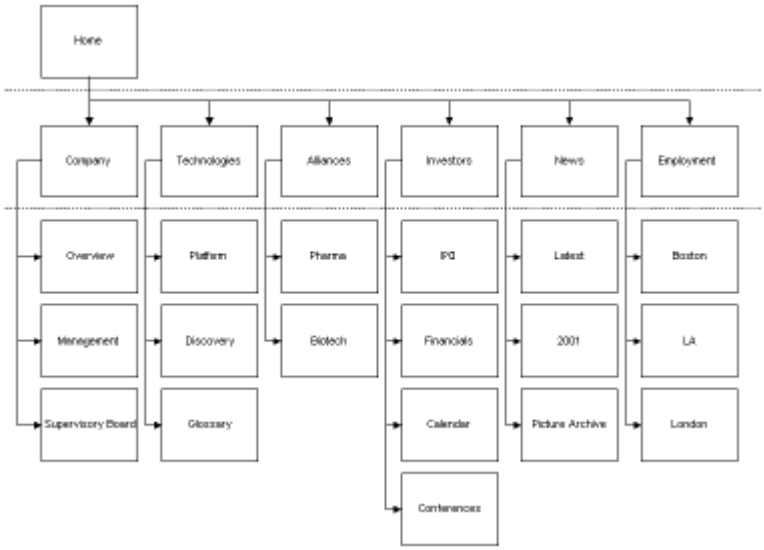


Figure 2: Typical Site Structure

Rhythmyx delivers a traditional structure, but Rhythmyx assembler applications are not typical templates. A Rhythmyx assembler application is a template that merges content extracted from the back-end database with formatting to produce a page or a portion of a page called a snippet. Snippets are the low-level building blocks of the page. In fact, most assembly variants are for snippets rather than for pages because this structure provides the most flexibility when producing a page. Contributors have more flexibility to move content to different pages. Site design becomes more fluid because redesign effort is minimized. Often, all that is required to redesign a site is to rearrange the snippet assemblers assigned to the page assemblers.

When developing your assemblers, always keep two questions in mind:

- What are the pages you want to produce?
- What are the snippets you want to produce?

As you develop more templates, you can narrow down the total number of templates on your site by recognizing duplicate templates and eliminating them.

Developing Templates for Page Variants

Developing the template for a page variant involves two major processes:

- Determine the parent content type of the page;
- Determine which formatting is local to the page and which is part of a slot on the page.

Determining the Parent Content Type

The first step in designing a Variant is to determine the parent content type for the page. A page generally consists of content items from several different content types, but one of these items is the container or "parent" for all the other content types on the page. The parent item stores all of the pointers to the subcomponents, or "child" items on the page. Any type of content item that allows for related content may act as a parent. Very often, the parent content type is an index content type, such as for home pages, but article or other text-rich content types may also be parents. The key issue is that the parent item owns the content relationship, not the child.

To determine the parent content type of the page:

1. Identify all the content types used within that page.
2. Determine which of those content types are edited by the user that ultimately decides what appears on the page.
3. Of those content types, find the content type that would have to be changed the most whenever its related content relationships were changed. This Content Type is the parent. This page is therefore a Variant of that content type.
4. If none of these content types needs to be changed when the relationships change, you may need another index content type (indexes are content types that have no local fields, only related content).

The child content types that appear on this Variant generally do not need to be edited just because this parent content has made a relationship to include them. The editor of the parent content item decides to use or relate to the child content items "as they are".

For example, suppose a page contains a featured article content type, a related link content type and an image content type. If the creator or editor of the featured article content items will define the related image and the related link, then the featured article content type should be the parent content type of the page. Changes to that article will have the greatest impact on the related images and links. The page generated in this case is a Variant of the feature article.

However, if the business process for the page dictates that a higher level editor defines the page by choosing any article, image, and related links, then the parent content type for that page is best modeled as an index content type. An index content type allows separate users to create images, articles, and links as unique content items, independent of how any given index author may later decide to combine them. In this case, the page generated is a Variant of that index, not a Variant of the featured article.

It is important to note that any content type could be used as a parent for one Variant, and a child for another Variant. In fact, a content type could be used as both the parent and a child on the same Variant. This is a property of the recursion that is fundamental to the assembly process.

For example, in the first case above, the related link could actually be another feature article. In that case, the parent content type is still a feature article. The page is still a variant of a feature article. However, a different variant of a feature article that produces the related link also appears on that page, as a child item.

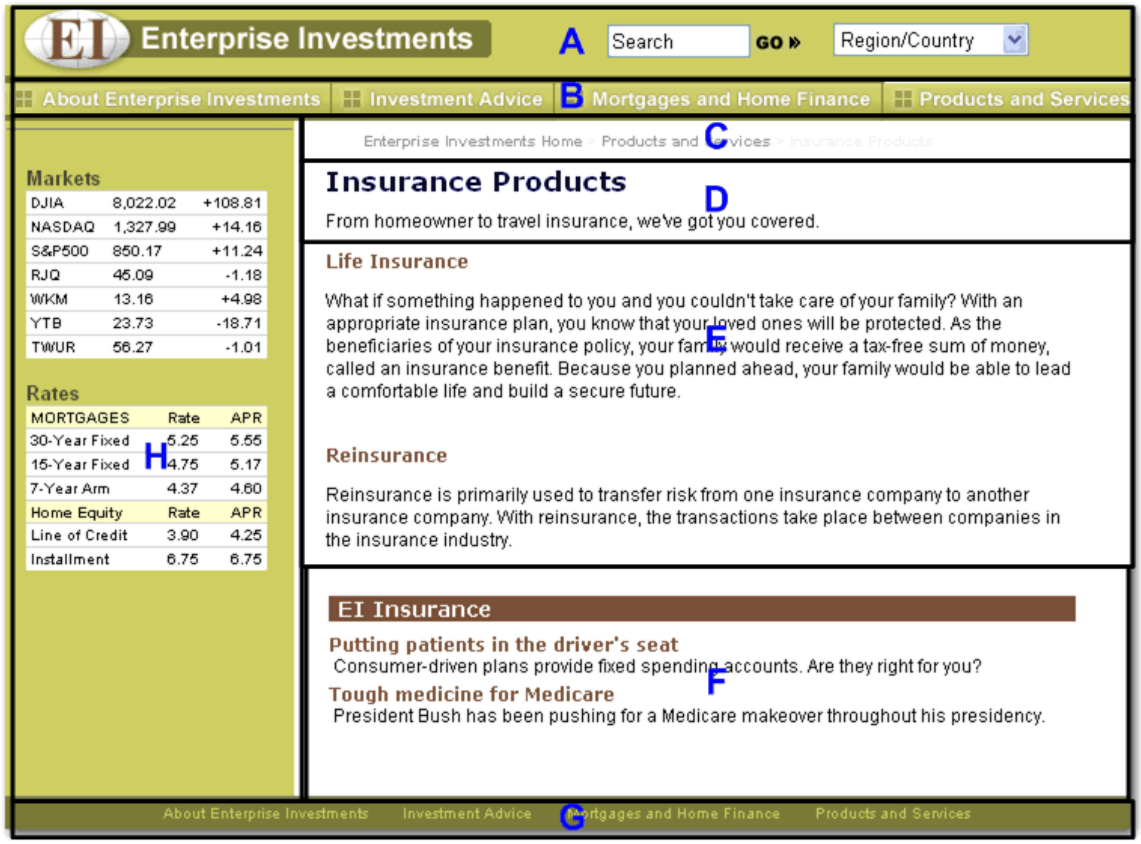
Determining Local Formatting Versus Slots and Global Template Information

Once you have determined the parent content type for the page, you need to determine the formatting and content that is local to the page and the content that comes from other content types. Content that comes from items other than the parent either fits into a space on the page termed a "Slot" or is generated by a Global Template. You may find a printed copy of your page useful to help you define the slots and global data, and you will probably want to make an electronic copy of your web page as well.

On your printed page, draw a box around each unique portion of the page. Elements you may want to isolate include:

- the banner, the footer, and navigation bar;
- any lists of used for navigation;
- any images;
- sidebars;
- any large blocks of text.

For convenience, you may want to identify each box with a letter or number. For example:



Note that the banner is identified (A), as well as the footer (G). The footer (G) is also a navigation snippet, as are (B) and (C). The sidebar on the page is labelled (H), and the other sections of non-local content are labelled (E) and (F). Only section (D) represents content from the local Content Item.

You must decide which of the non-local labelled portions of the page should represent Slots on the local Content Item's Variant that are filled with related Content Items, and which portions will be common to every page on your Web Site and should become part of a Global Template. In our example, portions (E) and (F) represent content that is related to the local Content Item, so their locations will become Slots on the Variant. In addition, the navigation section (C) displays breadcrumbs for the local content, so this will also become a Slot on the Variant.

The other sections of the page are not specific to the local Content Item, but create a generic background for any page on the Web Site. Therefore it is most efficient for these portions of the page (the banner (A), the Site navigation buttons (B) and (G), and the sidebar (H)) to appear on a Global Template.

Once you have sketched out the sections of your page on paper, you can turn to your HTML file and begin to identify which code should become part of the Global Template or part of a Slot. Mark each Slot and Global Template section with a comment identifying where it starts and where it ends.

Once you have identified all of the Slots and portions of the Global Template, you can cut these sections of code out of the parent page, and save them in separate html files. These files will become the templates for the Slot and Global Template Variants.

Developing Templates for Snippet Variants

Use the same steps to develop the templates for snippet variants that you used to develop the page variant templates: determine the parent content type of the snippet, then determine the local content and formatting as opposed to the slots contained in the snippet.

Note that snippets can, and frequently do, contain other snippets. For example, a snippet that includes an image usually requires a slot for that image, since a snippet assembler, like a page assembler, can only have one parent content type.

Snippets can also include hard-coded slots and custom XSL, though these are less common in snippets than in pages.

CHAPTER 6

Planning the CMS Environment

Multi-Tiered Environment

In a multi-tiered Rhythmyx environment, multiple servers and other components serve different functions in the development of your Rhythmyx system. Since development of the system progresses in stages from development to production, the server/component group used in each stage represents a tier in the development environment.In the recommended environment, implementers create new Rhythmyx elements and features on a development tier, perform user testing on an integration tier, and install new functionality to a production tier after testing is complete:

development tier-> integration tier-> production tier

The need for separate development and production tiers is obvious; developers cannot create new functionality in the environment where end users perform their work. However, the integration tier is also necessary because it provides a location for developers to install new features and test them as part of an integrated environment. The integration tier also minimizes the risk of application failure and production server downtime caused by introducing new features directly to a production tier. You should further reduce risks of downtime and application failure through implementation of standard source control, backup, and recovery mechanisms across all tiers.

The following graphic shows a configuration of the recommended three-tier environment that includes a single server on each tier. The development tier frequently includes multiple servers, and the other tiers may include multiple servers as well.

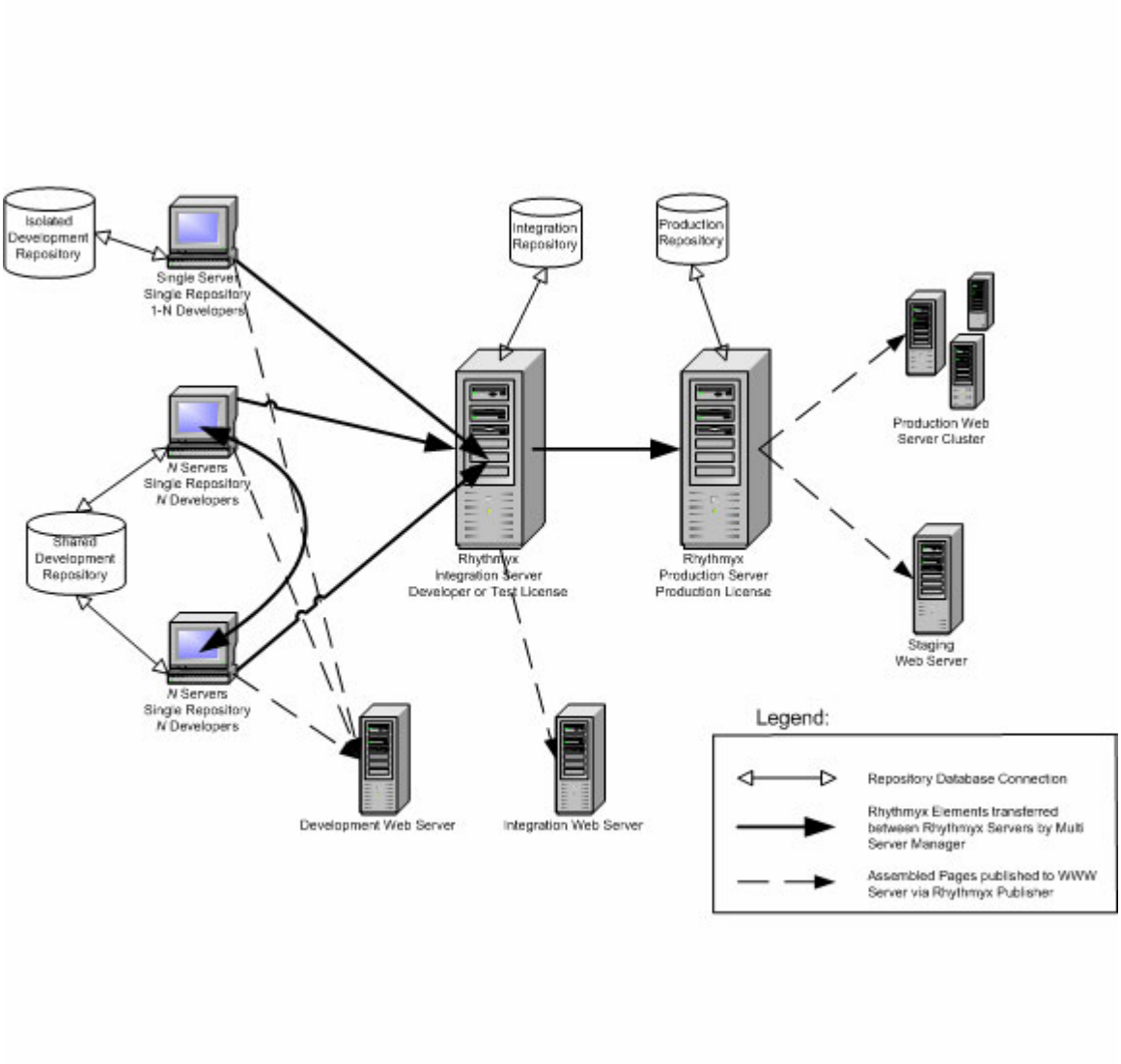


Figure 3: Multi-tiered Rhythmyx Environment

NOTE: The Rhythmyx Multi Server Manager helps you move Rhythmyx elements from one Rhythmyx server to another within the same tier or between tiers. See the Rhythmyx Multi Server Manager documentation for detailed information about the Multi Server Manager.

Development Tier

All Rhythmyx development tasks should take place on the development tier of the development environment. Tasks that developers can perform on the development tier include:

- Developing, testing, and fixing bugs in Rhythmyx elements including but not limited to:
  - Content Editors
  - Content Assemblers
  - Content Lists
  - Components
  - Workflows
  - Extensions
- Registering Rhythmyx elements (for example, Variants, Slots, Key Words, Content Types, Components, and Communities).
- Publishing Administration
- Registering Publishers, Sites, Contexts, Variables, Editions, and Content Lists.

NOTE: - Tiers should have different publishers to prevent unintended integration of their content.

- Purging Publication Logs
- Security and User Administration tasks performed through the Rhythmyx Server Administrator. In most cases users and roles will be different on each tier.
- Publishing to development servers and local file systems.
- Creating Multi-Server Manager archives.
- Installing Multi-Server Manager archives from other Development tier servers.
- Installing sample content items from production or integration tiers with the Multi-Server Manager for testing or bug fixing.

Use any combination of the following configurations for your development tier:

- Single server, single repository, single developer

  Use this configuration if your environment consists of one developer performing all development work.

- Single server, single repository, multiple developers

  Use this configuration if your environment consists of multiple developers who are working on the same project and must be aware of the applications and design elements created by one another.

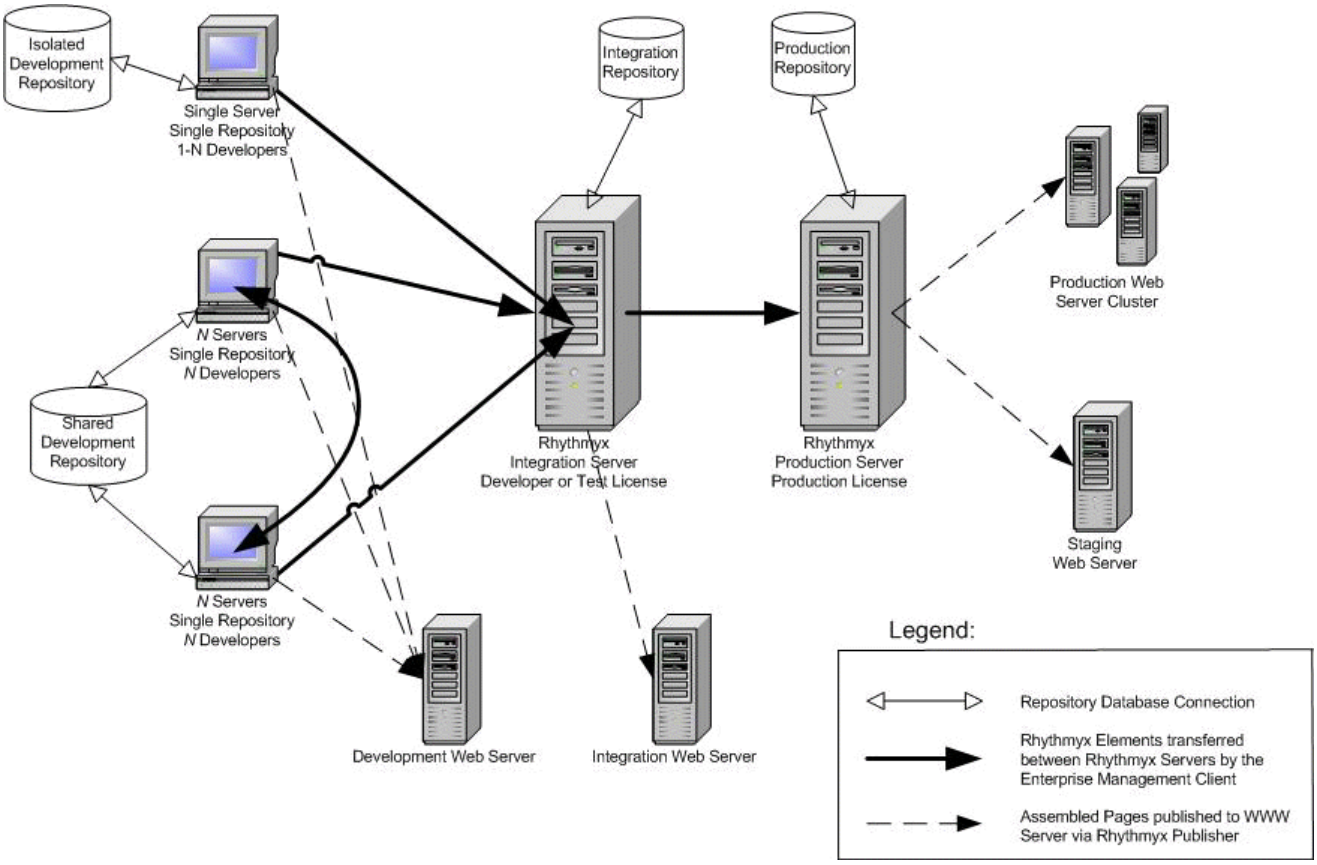
- Multiple servers, single repository, multiple developers

  Use this configuration if your environment includes multiple developers who are working on the same project and must be aware of the design elements created by one another. Each developer works on a different server, but the servers share a repository. The developers can move applications between servers with the Multi-Server Manager.

- Multiple servers, multiple repositories, multiple developers

  Use this configuration if your environment includes multiple developers who are not working on common or interdependent projects or who are geographically separated. Each developer works on a different server with a different repository. The developers can move applications between servers with the Multi-Server Manager.

The following graphic shows a development tier with multiple servers and multiple developers. One developer has an individual repository while two developers share a repository.



Integration Tier

Developers should perform testing of new features and bug fixes on the integration tier, and resolve any bugs found during testing on the development tier.

Tasks that developers can perform on the integration tier include:

- Publishing Administration
  - Registering Publishers, Sites, Site specific Context variables, and Editions. NOTE: - Tiers should have different publishers to prevent unintended integration of their content.
  - Purging Publication Logs.

- Security and User Administration tasks performed through the Rhythmyx Server Administrator. . In most cases users and roles will be different on each tier.
- Workflow Administration – Workflow notification cc lists may differ between tiers because, in general, different users receive notifications in development, integration, and production. All other Workflow components remain the same between tiers.
- User Acceptance Testing (UAT) on systems with a Test license. UAT may also be performed on an additional tier between integration and production.
- Publishing to an integration testing Web server.
- Creating Multi-Server Manager archives.

Production Tier

When integration testing is complete, developers may move Rhythmyx design elements to the production tier.

To provide failover redundancy to the production tier, you may include a hot standby server on the production tier. (An additional license is required.) The Rhythmyx environment on the hot standby server should be identical to environment on the production server and should be kept in sync with the production server. Use Multi-Server Manager to install archives created on the integration tier to both the production and the hot standby servers

Tasks that developers, administrators, and content contributors can perform on the production tier include:

- Publishing Administration
    - Registering Publishers, Sites, and Editions. IMPORTANT: - The production tier should always have a different publisher than the other tiers to prevent any integration of test and development content with production content.
    - Purging Publication Logs.
- Security and User Administration tasks performed through the Rhythmyx Server Administrator. In most cases users and roles will be different on each tier.
- Workflow Administration – Workflow notification cc lists may differ between tiers because in general, different users receive notifications in development, integration, and production. All other Workflow components remain the same between tiers.
- Content creation and approval
- Publishing to a staging Web server (to test your production configuration).
- Publishing to a production Web server.
- Creating Multi-Server Manager archives.

As a general rule, developers should not create content items on the development or integration tiers and use them on the production tier. Moving content between tiers is only recommended in the following situations:

- Content items are design elements – If Content Types build components of the CMS, such as navigation menus and editor controls, it is appropriate to create them on the development tier, test them on the integration tier, and when testing is complete, install them on the production tier.
- Users create functional content items in user acceptance testing on the integration tier. – Developers may install these content items on a production server to transfer users seamlessly from a test to a production system.
- Bugs found at the production or integration sphere involve particular content items - Most frequently, these errors occur during Active Assembly. To debug the errors, install the production or integration content items on the development server.

NOTE: Content that contains inline links or images will not be transformed between servers. This will cause the inline links/images to break; the content item must be manually edited again to correct the problem.

Scaling the Rhythmyx Environment

Depending on the size of your Web presence, the number of content contributors working on your Rhythmyx system, and the amount of content they produce, you may initially choose to purchase additional content hubs, publishing hubs, a test server, or a hot standby server, or you may purchase them later, as your system develops and grows.

Use the following guidelines for help determining when to expand your Rhythmyx system:

Optional Component	Scaling Guidelines
Additional Content Hub	If you have more than 400 or 500 content contributors working on Rhythmyx at the same time, you should consider adding an additional content hub to avoid performance problems.
Additional Publishing Hub	If the publishing runs in your incremental publishing cycle overlap each other (one publishing run begins before the previous one is finished) your system may begin to republish content items and waste resources. You may acquire an additional publishing hub to handle some of the publishing volume.
Database Publisher	If you store content in a database and deliver it dynamically, you should consider purchasing a Database Publisher when you purchase your Rhythmyx system. Otherwise, purchase a Database Publisher when you implement a database storage and delivery system for your Web site.
Test Server	Access to development servers is restricted to users with Workstation licenses; therefore a test server is necessary if you want business users to test your changes before you move the changes to a production server. A test server is also useful if: ▪ Your implementers need the ability to merge their development work on a common server. ▪ You want to make new functionality available to some users for testing, but not to others.
Hot Standby Server	If customer need or industry regulations require that you have a highly available and updated system, you may acquire hot standby servers to back up your System Master server and other servers that you have added to your Rhythmyx system.

Recommended Rhythmyx Server Location

Where you install your Rhythmyx server depends on the location of your content contributors and the level of security you require for Publishing.

- Install Rhythmyx on an external network that is accessible from the Web if:
  - Some of your content contributors do not have access to your LAN.
  - You are running Rhythmyx through a servlet interface.
  - You need the ability to publish securely using SSL.
- Install Rhythmyx on your local area network if:
  - All of your content contributors have access to your LAN.
  - You are not running through a servlet interface.
  - You do not plan to publish with SSL.

If you are using secure publishing, you can install your Rhythmyx server and your publishing hub on different networks:

- Install Rhythmyx on your local area network and your publishing hub on an external network that is accessible from the Web. You might choose this configuration if all of your content contributors have access to your LAN, but your LAN is behind a firewall, preventing direct network access to the Web server. The publishing hub on the external network can act as a bridge to the Web.

OR

- Install Rhythmyx on an external network that is accessible from the Web and install your publishing hub on your Web server. You might choose this configuration if some of your contributors do not have access to your LAN and you want to make requests to the Rhythmyx server via SSL.

Rhythmyx Environment with Commonly Used Component Locations and Licensing

The sample Rhythmyx environment displayed in the graphic uses the recommended configuration:

development server -> integration server -> production server

The environment uses the components provided by basic licenses for a Rhythmyx System Master, a Development Workstation (package of 5), and Rhythmyx Accelerator for Microsoft Word. It also uses components provided by optional licenses for a Test Server, a Database Publisher, an additional Publishing Hub, an additional Content Hub, and an additional File Publisher.

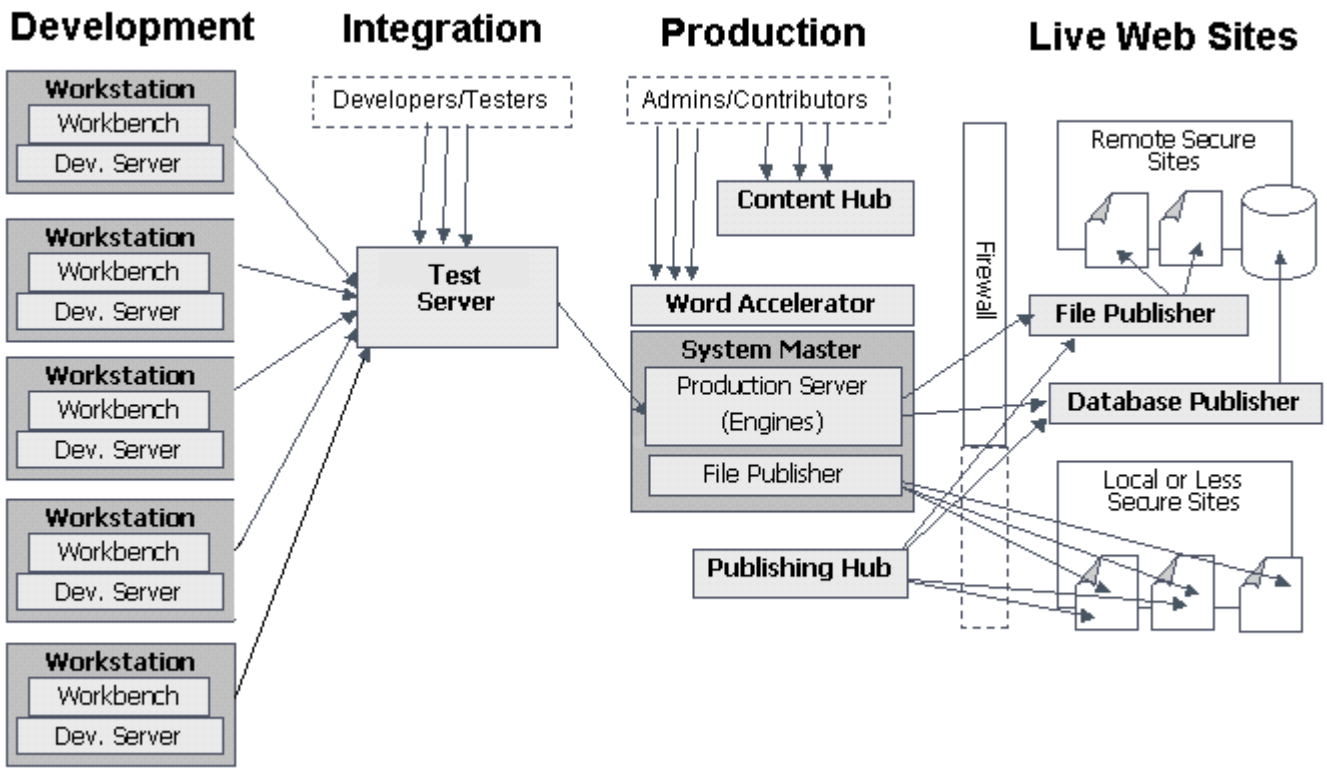


Figure 4: Multi-tiered Rhythmyx Environment showing the components on each tier

The Rhythmyx System Master and the File Publisher for local and less secure sites resides behind a firewall. The Database Publisher and File Publisher for remote, secure sites reside on the Web server or on networks that are accessible from the Web.

Index