Content Server Enterprise Edition

Version: 5.5

Architecture Guide

Document Revision Date: Oct. 30, 2003

FatWire SOFTWARE

Architecture Guide
Document Revision Date: Oct. 30, 2003
Product Version: 5.5

FatWire Technical Support
Web: http://www.fatwire.com/Support

FatWire Headquarters
FatWire, inc.
330 Old Country Road
Suite 207
Mineola, NY 11501
Web: www.fatwire.com

Table of

Contents

1 Architecture Overview 7

Content Server Enterprise Edition: Basic Concepts 7

Modular Page Design 8

Caching 10

Assets 11

Content Server Enterprise Edition Architecture 11

Three-Tier Architecture 11

The CSEE Product Stack 12

Implementation Models 13

Content Server Enterprise Edition as a Central Data Repository 14

CSEE Integrated with Other Applications 15

Content Server Distributed Over Multiple Sites 17

The Design Process 18

CSEE Environments 18

Design for Performance 20

Performance Test Early and Often 20

Project Scheduling and Staffing 21

2 Choosing Hardware and Software 23

Hardware for Different Environments 23

Development Environment 24

Management Environment 25

Delivery System 26

Testing System 26

Sizing Your Hardware 26
Hardware for a Small System 27
Hardware for a Medium-Sized System 27
Hardware for a Large System 28
Hardware to Make Your Site Scalable 28
Hardware for a Multinational Site 28
Time Difference and Hardware 29
Content Overlap and Hardware 29
Geographic Distance and Hardware 29
Hardware Guidelines 30
Choosing Software 30

3 Page Design, Caching, and Publishing 33
Page Caching Overview 33
Content Server Page Caching 33
Content Server Satellite Caching 35
Caching and CacheManager 35
Caching and Session 35
Publishing Overview 36
Site Design and Export to Disk Publishing 36
Page Design Best Practices 37
Choose a Coding Language 37
Determine Which Pagelets to Cache 38
Code Your Pages 39
Page Caching and Publishing Best Practices 40
Assessing Existing Page Designs 40
Determine What Can Be a Pagelet 40
Determine Whether the Content of a Pagelet Should Be an Asset 41
Determine Which Asset Type Is Best for Each Pagelet 42
. 45

4 Data Design and Resultset Caching 47
Data Design 47
Basic Assets 48
Flex Assets 48
Choosing Basic or Flex Assets 50
Designing Assets 51
Designing Basic Assets 51
Designing Flex Asset Families 51
Data Retrieval 53
Data Retrieval in Content Server and CS-Direct 53
Data Retrieval in CS-Direct Advantage 54
Resultset Caching 54
Table Updates and Resultset Caching 54
Setting Resultset Caching Timeouts 55

Setting Resultset Sizes 55
Database Maintenance 56

5 Security and Personalization 57
Security Overview. 57
Securing Content Management Systems 58
Securing the Web Site 58
CSEE Authentication 59
Content Server Authentication 59
LDAP Authentication. 59
Windows NT Authentication 59
Segmentation and Personalization 59
Segmentation and CSEE Products 60

Index 61

Chapter 1

Architecture Overview

Developing a system architecture is not a linear process; it is a circular one, where the hardware, software, and design decisions you have made earlier in the project must be reassessed based upon the design decisions you are currently making.

Architects must consider a client's business requirements, budget, legacy systems, and a number of other factors in order to design an architecture for a site that is:

- Fast
- Secure
- Reliable
- Scalable
- Easily maintained

This chapter provides an overview of the concepts you will need to know as you and your team create a system architecture. The chapter contains the following sections:

- Content Server Enterprise Edition: Basic Concepts
- Content Server Enterprise Edition Architecture
- Implementation Models
- The Design Process
- Project Scheduling and Staffing

Content Server Enterprise Edition: Basic Concepts

The Content Server Enterprise Edition product family (CSEE) consists of Content Server and all of its related applications:

- CS-Direct
- CS-Direct Advantage
- CS-Bridge Enterprise

- CS-Bridge XML
- CS-Engage
- CS-Satellite
- Analysis Connector

CSEE's ability to manage and serve large amounts of content efficiently is dependent upon three main concepts:

- Modular page design
- Caching
- Assets

The following sections describe each of these concepts in greater detail.

Modular Page Design

The CSEE product set allows you to separate the format of your web pages from their content. This is accomplished, in part, through modular page design, where web pages are composed of blocks of code called **elements**.

Consider, for instance, an online newspaper with the following format:

The stories on the site change from day to day, perhaps even hour to hour, but the basic format remains the same.

Elements

To achieve this flexibility, the newspaper web page is composed of 10 blocks of HTML, XML, JSP, or ASP code called **elements**. The elements are called from a **containing element** that determines the layout of the final web page.

You can create four main types of elements with CSEE:

- Content elements, which call content from the file system or database
- Structural elements, which determine how content is displayed and allow visitors to navigate the site
- Logic elements, which contain the application's business and display logic
- Maintenance elements, which maintain the system's databases and caches

Some elements fall into more than one category. For example, a containing element is a structural element because it determines page layout, yet it can also contain logic to display specific content to specific visitors—which makes it a logic element, too.

An element can call other elements. For example, an element that implements publishing can call an element that deletes outdated content from the CSEE database.

You can write your elements in XML, JSP, or ASP. Use XML for elements that determine page layout, and JSP for elements that contain logic. When a visitor to your web site requests a page, Content Server converts the XML and JSP elements into the output format that you choose; the default output is HTML.

All of the elements that make up a CSEE site are recorded in the ElementCatalog table. Most elements also have entries in the SiteCatalog table, allowing them to be cached on Content Server and on CS-Satellite.

Pagelets

The output generated by evaluating a CSEE element is called a **pagelet**. One pagelet can be used in many pages. Common design elements such as headers, footers, and navigation bars can be written once and called by many pages of the web site. In addition to being reusable, each pagelet can be cached to disk.

The following diagram shows how pagelets are combined to make up the finished web page:

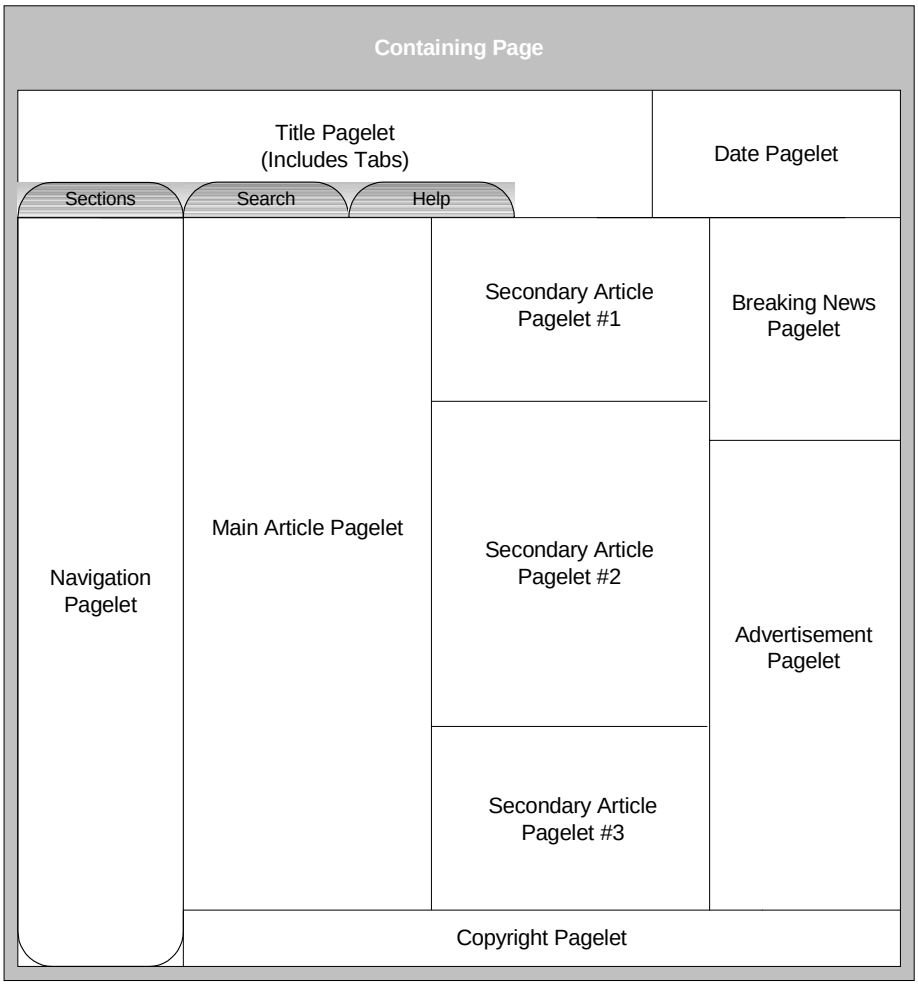


Figure 1: Modular Page Design

For information about implementing modular page design on your web site, see Chapter 3, "Page Design, Caching, and Publishing."

Caching

Content Server supports two main types of caching:

- Page caching, which caches pages and pagelets to disk (described further in Chapter 3, "Page Design, Caching, and Publishing")
- Resultset caching, which caches the results of SQL queries, search engine queries, and CS-Direct Advantage searchstates to memory (described further in Chapter 4, "Data Design and Resultset Caching")

You should design your site to use both types of caching; caching reduces load on Content Server and improves system performance.

Assets

The content management functions of CS-Direct, CS-Direct Advantage, and CS-Engage are based, in large part, on the concept of **assets**. An asset is a Java object that allows you to manage content and organize your site. There are two types of assets, which work with different products in the product set:

- Basic assets, provided by CS-Direct.
- Flex assets, provided by CS-Direct Advantage

All assets, whether basic or flex, fall into one of four general categories:

- Content assets, which represent content, such as image or article assets.
- Site design assets, which allow you to organize your site, such as page or template assets.
- Flex assets, which both represent and organize content, such as flex definition assets.
- Marketing assets, which allow you to group users by various criteria and target content to them, such as segment assets.

For more information about assets and asset design, see Chapter 4, “Data Design and Resultset Caching.”

Content Server Enterprise Edition Architecture

CSEE’s architecture is layered in a **stack**. The stack comprises Content Server, which sits on top of a database and an application server, and the rest of the CSEE content applications, which sit on top of Content Server. Each product utilizes and builds upon the capabilities of the ones below it in the stack. For example, CS-Direct adds a user interface for data entry to Content Server’s content management capabilities.

Three-Tier Architecture

CSEE products are Java 2 Enterprise Edition (J2EE) applications that take advantage of the benefits of the J2EE platform. J2EE is a three-tiered architecture:

- The bottom tier handles back-end tasks such as session management.
- The middle tier contains the business logic.
- The top tier handles front-end tasks such as displaying content.

CSEE products sit in the middle tier, on top of a J2EE application server. They retrieve content from the database at the back end, manage the content, and format that content for display to web site visitors at the front end. Because the CSEE products are J2EE applications, you can design custom Java code and servlets to separate business logic from content, and integrate other J2EE applications into your architecture.

The CSEE Product Stack

Figure 2 shows the architecture of the Content Server Enterprise Edition (CSEE) product stack.

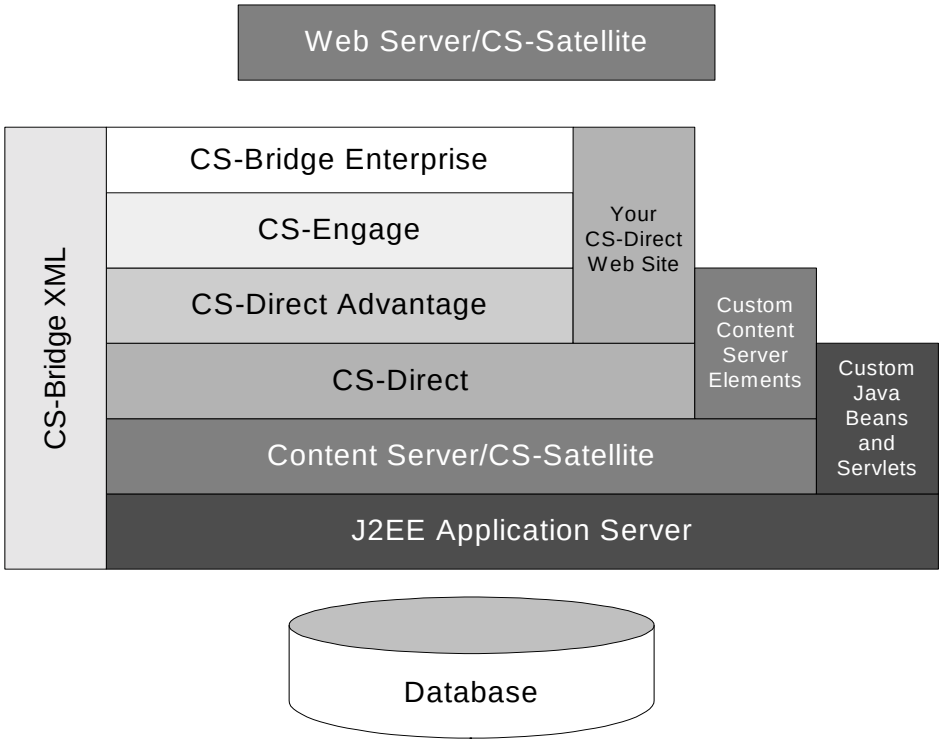


Figure 2: CSEE Product Architecture

Following is a description of each application in the Content Server Enterprise Edition product stack:

- Content Server is the foundation of FatWire's family of content management products. You use Content Server to create custom elements for publishing and database maintenance.
- Content Server Direct provides a user interface for data entry and various management tasks and introduces the concept of **assets**. Different types of assets represent content, organize content, and allow you to manage your site. Content Server Direct assets are **basic assets**.
- Content Server Direct Advantage makes it easy for users to drill down through multiple levels of information. It uses **flex assets** instead of basic assets. Additionally, Content Server Direct Advantage supports the importing of large amounts of data with its Bulk Loader utility.
- Content Server Engage allows business managers to collect information about their web site's visitors, and to use that information to display customized content or to provide special discounts and other promotions. Content Server Engage works in conjunction with Content Server Direct Advantage.
- Content Server Bridge XML enables the exchange of XML-encoded content with customers and enterprise partners. It is a configurable framework for receiving,

processing, and posting XML documents to and from other enterprise applications over the web. CS-Bridge XML comes with Content Server.

- Content Server Bridge Enterprise is a configurable processing architecture that enables you to call back-end application services from web pages delivered by Content Server. CS-Bridge Enterprise, which relies on webMethods Enterprise for system connectivity, provides the framework for developing and managing business logic within Content Server.
- Analysis Connector (not shown) collects web site events such as clickstream data, and loads that data into the CSEE database during low-traffic periods. The data is then ready to be analyzed by third-party analysis engines.
- Commerce Connector (not shown) integrates Content Server Direct Advantage and Content Server Engage with FatWire's payment processing and order management software, Transact.
- Content Server Satellite systems are low-cost caches for Content Server web pages. CS-Satellite comes with Content Server.

Implementation Models

You can apply Content Server and its related applications in any of three ways:

- As a central data repository
- As a component in a system where CSEE is integrated with other enterprise applications
- As a component in a distributed system where a central CSEE system supplies content to and receives content from remote CSEE systems

Content Server Enterprise Edition as a Central Data Repository

As shown in Figure 3, "CSEE as a Central Data Repository", files and data enter the CSEE system from other applications. This data can be input into CSEE through a variety of methods, including:

- The CS-Direct user interface
- Microsoft Word, by using CS-Desktop
- Dreamweaver, by using CS-Designer
- Content Server DocLink
- The XMLPost utility
- The BulkLoader utility
- Enterprise JavaBeans

CSEE, the application server, and the database store and manage that information, and output it for display in various formats.

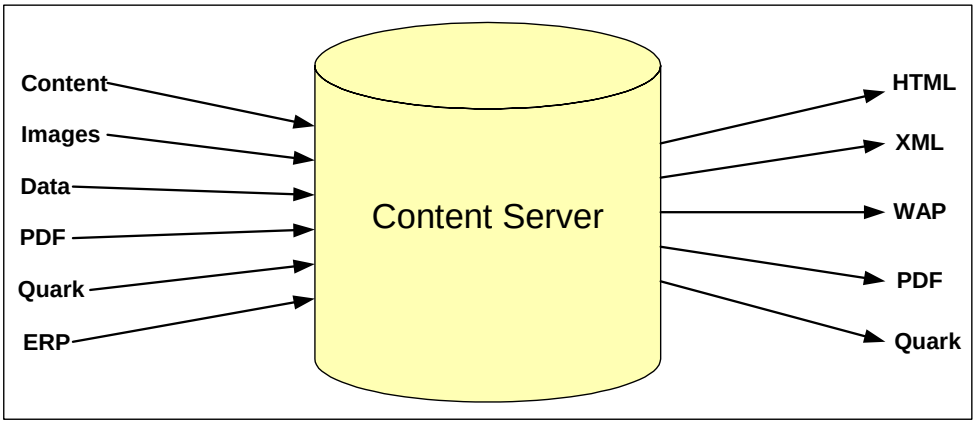


Figure 3: CSEE as a Central Data Repository

CSEE Integrated with Other Applications

CSEE supports two separate application integration solutions:

- Integration via enterprise middleware
- Integration via XML document exchange

The middleware solution requires enterprise middleware and, depending on your application, either the CS-Bridge Enterprise product, the Content Server Adapter for webMethods, or both. The CS-Bridge Enterprise framework enables tight application integration through a collection of APIs presented by the middleware.

Business-to-business integration using XML document exchange requires the CS-Bridge XML product, which is included with Content Server. CS-Bridge XML provides a looser integration based on processing and routing XML-encoded information between applications.

Like all products in the CSEE stack, these integration options can be applied separately or used in conjunction with each other. Content Server also includes a documented Java API that enables further custom integration.

Enterprise Application Integration via Standard Middleware

As shown in Figure 4, “Enterprise Application Integration Using Middleware,” middleware and CS-Bridge Enterprise jointly provide the means to deliver services from enterprise applications to Content Server, using XML as the common exchange format.

Middleware, such as webMethods Enterprise, provides the integration logic and system connectivity, and CS-Bridge Enterprise provides an environment for creating and managing business logic within Content Server. CS-Bridge Enterprise supplies the framework to make requests against any enterprise application supported by the middleware, and synchronously passes resulting data back to Content Server. Content Server then manages that data and outputs it for display.

Separately, the Content Server Adapter for webMethods serves as a configurable trigger to handle asynchronous requests from any system supported by the middleware.

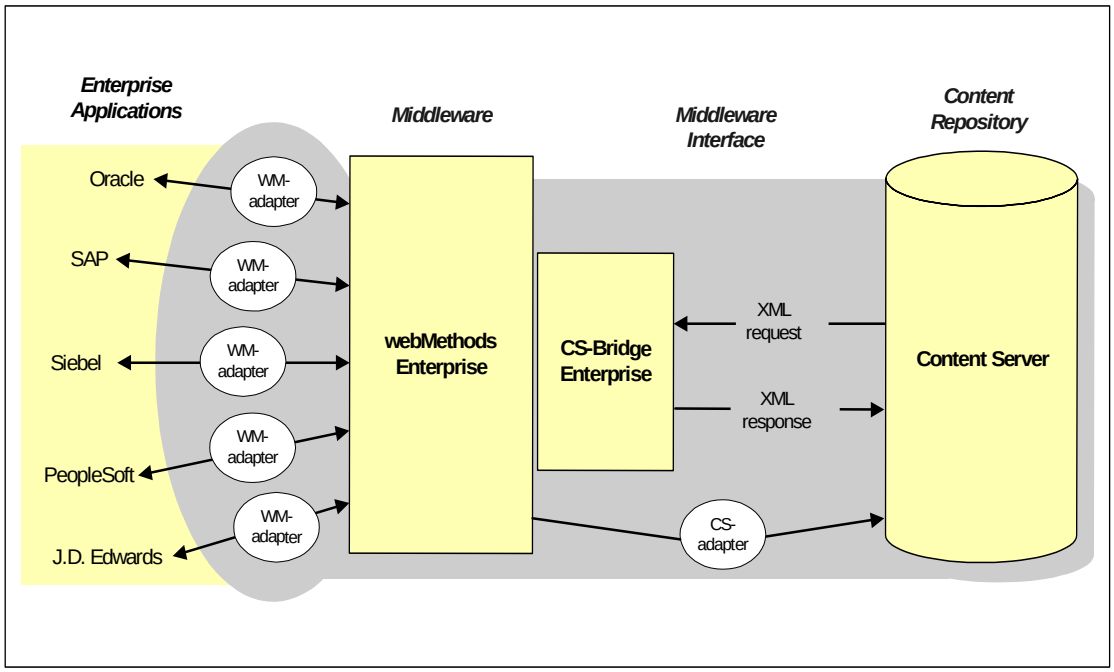


Figure 4: Enterprise Application Integration Using Middleware

Business-to-Business Integration via XML Document Exchange

Figure 5, "Business-to-Business Application Integration Using XML Document Exchange," shows a loose B2B-integration that makes use of the built-in XML document processing capabilities of Content Server. In this process, XML documents move over HTTP and are automatically parsed, authenticated by user, queued, and processed by whatever custom business logic you develop. Automated document transformations using the supplied XSLT processor complement the processing engine's ability to manipulate inbound and outbound documents and leverage Content Server's logic development environment. Once processed, documents can be optionally routed to a specified URL.

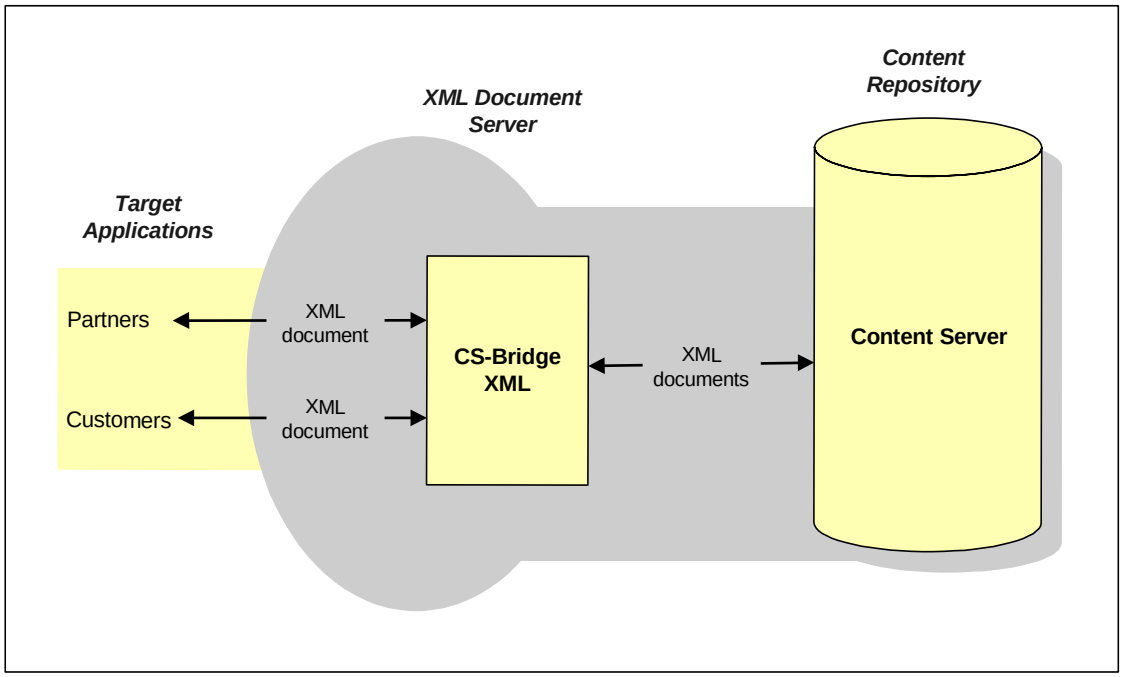


Figure 5: Business-to-Business Application Integration Using XML Document Exchange

Content Server Distributed Over Multiple Sites

In the third model, shown in Figure 6, "Distributed Content Server Sites," Content Server is modified to support multiple web sites implemented with multiple instances of Content Server. For example, Content Server can be configured for web sites in different geographic locations, which contain content specific to their regions.

As before, data passes to Content Server, but instead of creating and serving all output through the primary Content Server system, data is mirrored to three other Content Server systems, which support web sites in three different regions. The regional Content Server systems can receive content from or pass content to the main data repository, and they can also pass content to and from one another.

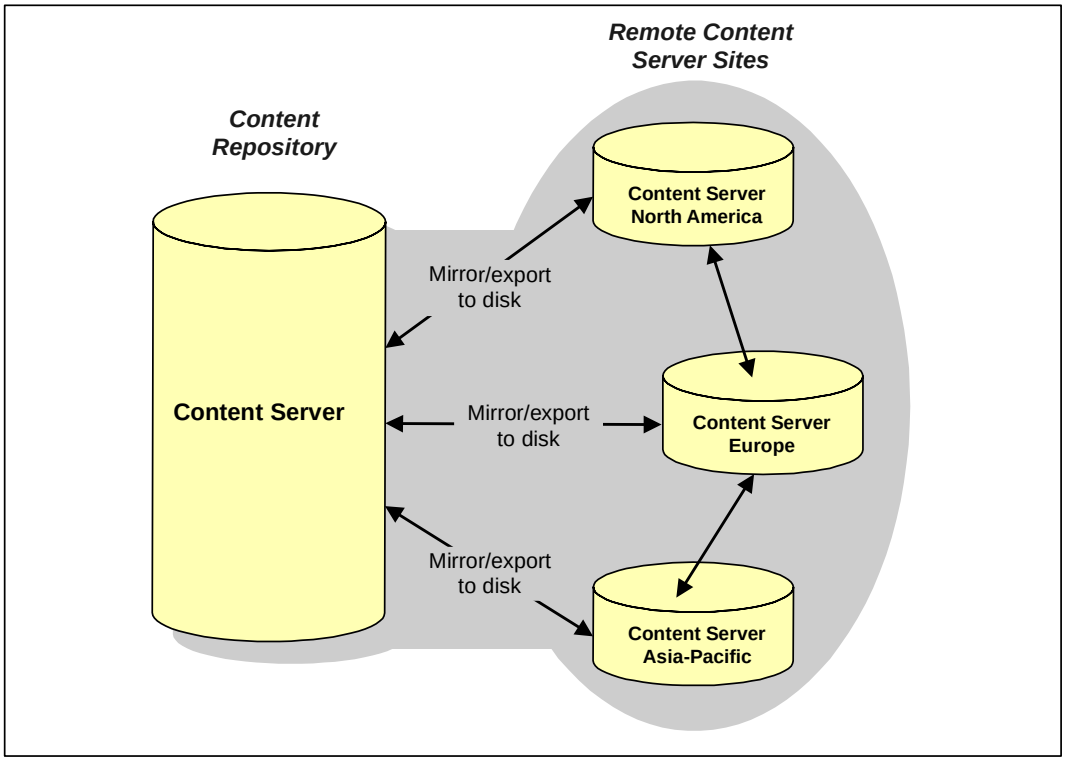


Figure 6: Distributed Content Server Sites

The Design Process

As you design a system architecture, consider the following key aspects of the project:

- Environments – A Content Server system has at least three separate environments, each with different users who have different needs.
- Performance – Your design should consider perfomance issues for each environment.
- Testing – Plan to performance test each environment regularly as you build the system.

CSEE Environments

Many people think of CSEE applications as creating a live web site that web surfers can view. In reality, CSEE applications create a minimum of three environments:

- A development environment, where template developers create the elements that provide the live web site's structure
- A management environment (staging), where content providers create, edit, and manage content
- A delivery environment (production), which delivers the live web site to people surfing the Web

In addition to these three environments, FatWire recommends that you implement a dedicated testing environment, for performance and functional testing.

A Three-Environment System

Figure 7, "A Three-Environment System," shows these three main environments:

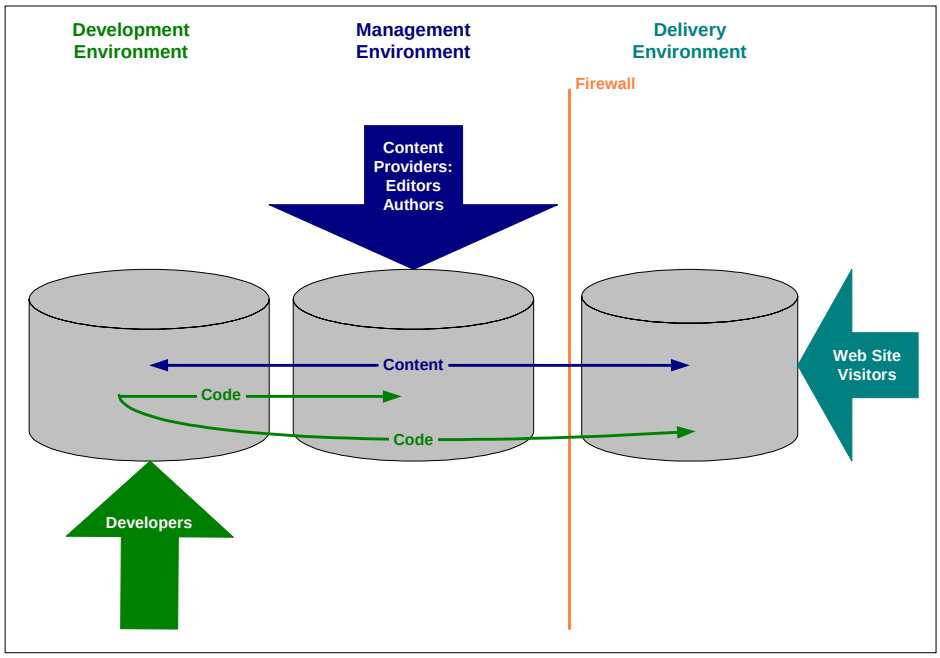


Figure 7: A Three-Environment System

Developers create the code that composes the delivery web site's structure and implements its business logic in the development environment. Web developers create the elements that provide the web site's structure. Java developers use Java classes and JSPs to implement much of the site's business logic. The developers then send their completed code to the other two environments:

- The management environment, so that the content providers can preview the content they are working with in the context of the web site
- The delivery environment, so that the live web site reflects the changes that the developers make

Content providers create, edit, and manage the content that will be used on the live web site. They then publish the completed content to the other two environments:

- The development environment, so that the developers have content with which to test their code
- The delivery environment, so that the completed content is displayed on the live web site

As you develop a system architecture, remember that the different environments that you create have different users with different needs. The management environment, for example, must be easy for content providers to use and must be able to handle a high

volume of writing to the database. The delivery environment, however, must be set up to handle serving web pages quickly.

A Four-Environment System

In addition to the development, management, and delivery environments, it is a good idea to add a fourth environment for testing. Figure 8, "A Four-Environment System," shows a four-environment system.

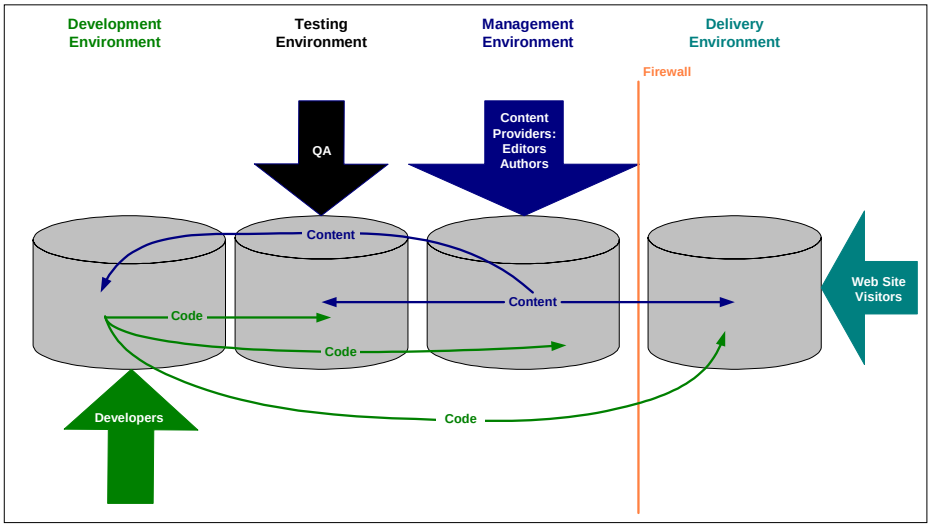


Figure 8: A Four-Environment System

A separate testing environment allows QA to run performance and functional tests without disrupting the developers' or content providers' work.

Design for Performance

Design with performance as a priority from the earliest stages of the project. This guide will provide some design tips that promote good performance. You should also check the *CSEE Developer's Guide* to find coding practices that ensure good performance.

Note that designing for good performance extends beyond the delivery environment. Developers and the management staff need development and management environments that perform well too.

Performance Test Early and Often

Designing for performance is not an exact science; you will improve your system's performance greatly by running frequent performance tests, modifying properties and settings for the application server, the database, and Content Server, then retesting, until you find the best settings for each environment.

You can use a load generator such as WebLoad to test your system's performance. Be sure to test the application server and database alone, as well as the development, management, and delivery environments with all of their hardware and software.

To ensure maximum performance for all of the system's environments, test with the testing environment tuned to replicate the development, management, and delivery environments.

Project Scheduling and Staffing

A typical project will take four to six months to complete from the time you begin to design the site until the site goes live. If the project is a simple one, it may take as little as two months. If the project is complex or must be integrated with legacy systems, it may take longer. Budget two to four weeks of this time for architectural design.

Some phases of the project can be undertaken simultaneously, as shown in the following diagram:

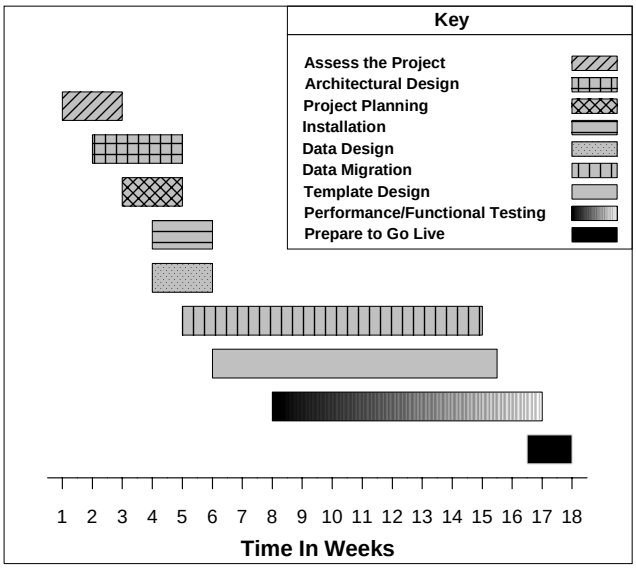


Figure 9: An Example of Project Scheduling

For example, you can install the software as you work on a data design, and migrate data from other sources while graphic designers and web developers design the web pages and code the templates.

You will need the help of various Subject Matter Experts (SMEs) as you design an architecture for a site. The following table lists the SMEs that you will need as you design the various components of the site:

Design Task	Required
Content Design/Site Features	Business Analysts, Web and Java Developers, DBAs
Data/Asset Design	DBAs, Developers (for custom assets)
Performance Testing and Tuning	An expert who can tune your application server, QA staff

Design Task	Required
Page Design	Graphic Designers, Web and Java Developers

The total staff for an average project is between 7-10 people:

- 1 Architect
- 1 Project Manager
- 3-4 Web/Java Developers to develop templates and business logic
- 1 or more DBAs to help with data design and migration
- 1 expert who can tune the application server

Note that not all of these staff members will be needed full time or for the duration of the project—the development staff, for example, will not be coding the site full time while the architect is assessing the project and developing an architecture.

Chapter 2

Choosing Hardware and Software

Choosing the correct hardware and software is integral to a project's success. Appropriate hardware provides the performance and scalability that clients want, and appropriate software makes accommodating a client's business requirements much simpler. This chapter provides guidelines for choosing hardware and software for your projects. It contains the following sections:

- Hardware for Different Environments
- Sizing Your Hardware
- Hardware to Make Your Site Scalable
- Hardware for a Multinational Site
- Choosing Software

Hardware for Different Environments

As you recommend hardware for a site, note that you will need hardware suited to at least three different environments:

- Development
- Management
- Delivery

FatWire strongly recommends hardware for a fourth environment: Testing.

Each environment supports different users with different needs, and the hardware for each environment must be tailored to those needs.

The following diagram shows a hardware design for a large web site; a smaller web site would require smaller, less powerful hardware:

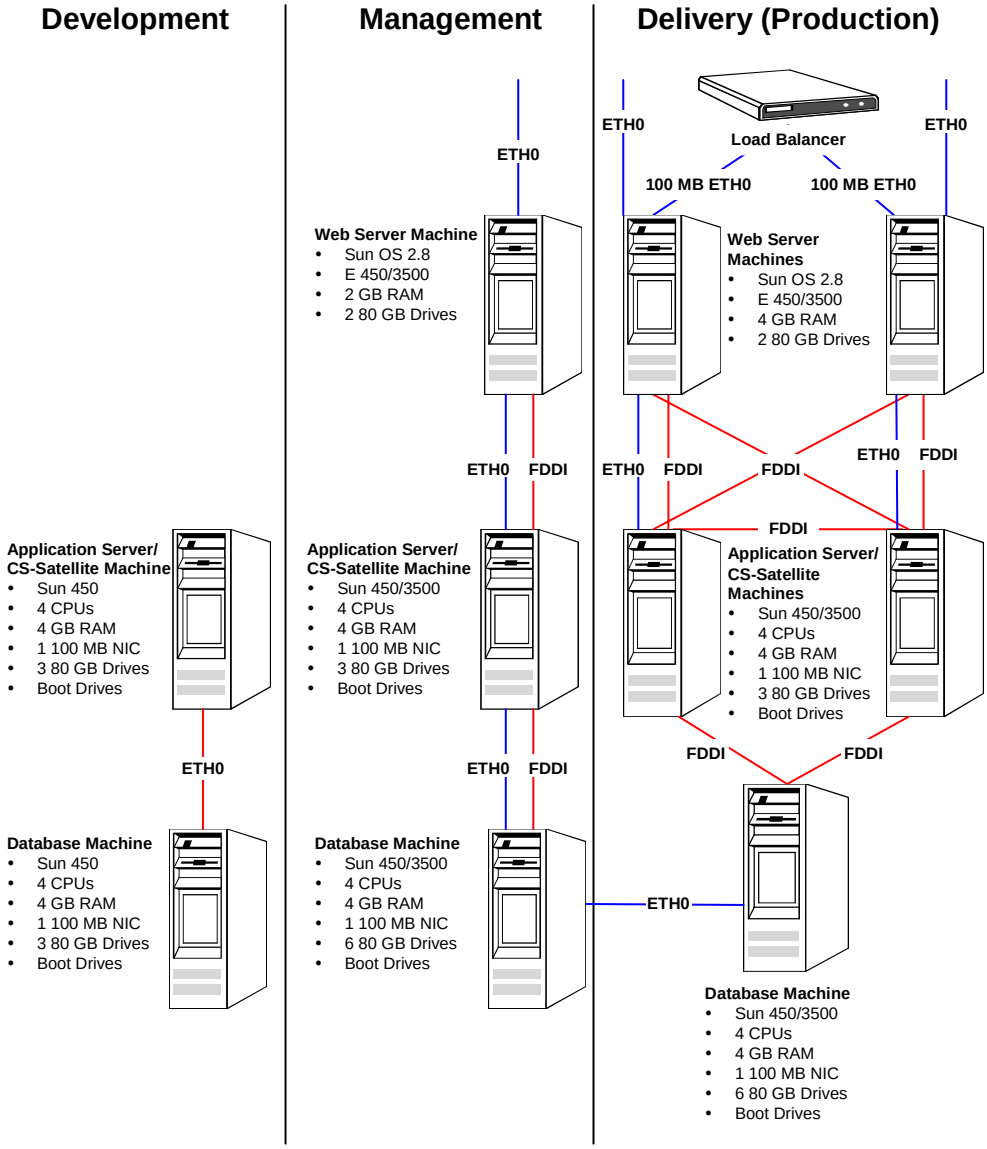


Figure 10: Three-Tiered Hardware Design

The following sections describe the hardware design for each environment in more detail.

Development Environment

Java and web developers use the development environment to code the elements and business logic for the finished web site. If your site is small, you may be able to develop on Windows 2000 machines, rather than on a Solaris development machine, as depicted in the following diagram.

If you only have a few developers, you can even create the development environment on a single laptop computer. If you have many developers working on your project, however,

the developers will log in to the Content Server user interface, or use Content Server Explorer to log on to the central development machine from their own laptops, as shown in the following diagram:

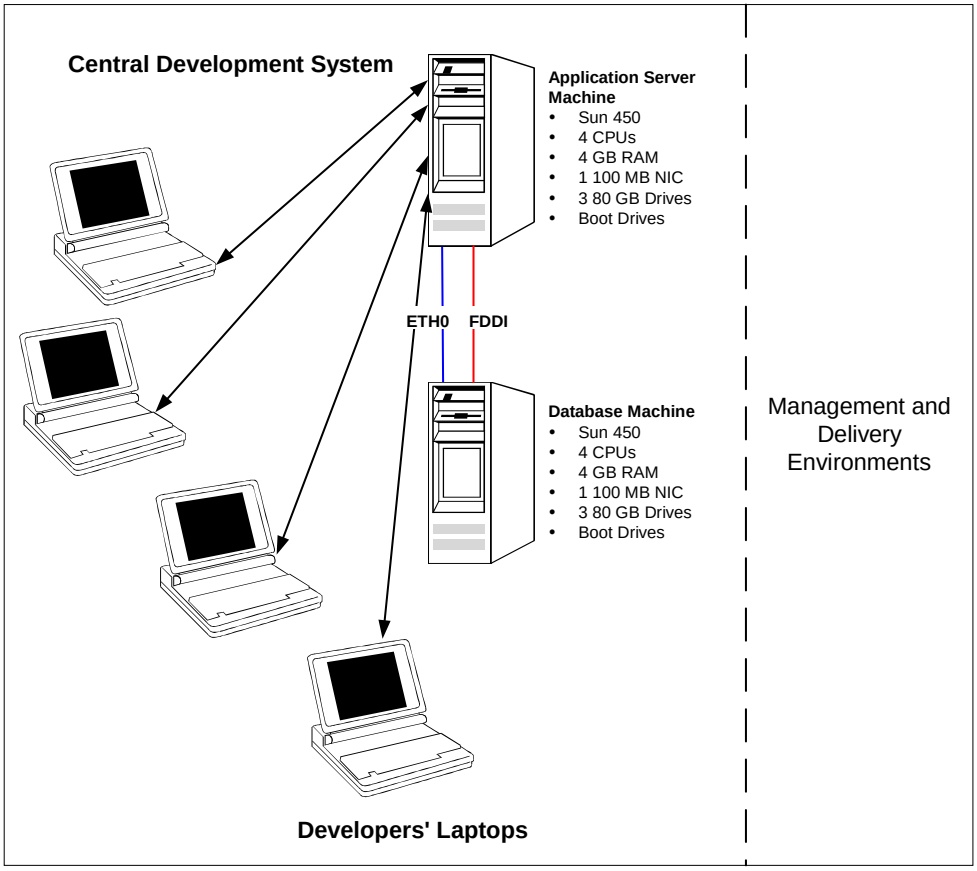


Figure 11: An Environment with Many Developers

If you have multiple developers sending code to a central system, you should have separate machines for your database and application server. Having a separate database improves performance on the development environment and makes maintaining it easier.

Management Environment

The management environment will require at least three Solaris machines: one for the database, one for the application server, and one for the web server. The database machine for the management environment should be the most powerful database machine you use in your architecture, as editors will be writing to it frequently.

If your management environment must run 24x7, you should include an additional application server machine so you can cluster your application server to provide failover. In addition, a cluster requires a shared file system, typically on the database machine.

Delivery System

The delivery system will probably run 24x7, and thus needs failover. To provide failover, you will need two application server machines and two web server machines. Figure 10, "Three-Tiered Hardware Design," shows this type of system, where the application servers are clustered to provide failover.

You will also need a machine for the database and file server. Generally, you can use one database/file server machine for a site; but for a large site where you need frequent access to files, a dedicated file server can improve performance.

If your site is "near real time" and publishes many times a day, you should consider adding another machine to handle the publishing load.

Additionally, you need a firewall between the delivery web server/CS-Satellite machines and the rest of your delivery system.

Testing System

In addition to the development, management, and delivery systems, you may want to add a dedicated testing system. A dedicated testing site is a necessity if both your management and delivery environments must run 24x7. The testing environment requires a database machine, an application server machine, and a web server machine. You should make your testing hardware as similar to your delivery environment as possible, though you will not need a failover system for a testing environment.

You should tune your testing environment to replicate the development, management, and delivery systems and test your site under each environment—it is as important for the development and management systems to perform well as a delivery system to perform well.

Because you need to test for performance as well as functionality, you will also need a load generator such as Web Load.

Sizing Your Hardware

Determining how to size your system's hardware can be a challenge. The following sections provide examples of hardware for small, medium, and large systems—to help you choose hardware for your own system. Each sample configuration includes hardware for a clustered management and delivery environment.

The systems described in this section use the following benchmarks:

- 1400 HTTP transactions per second on a Sun 4500 8-way system.
- 175 transactions per second per CPU.
- Each CPU supports 60 concurrent users .
- 8 transactions per page.

Note that these sample configurations provide guidelines for sizing systems; the actual size of the machines that you choose for your system depends on many factors, including:

- The number of attributes per asset type
- Page and template design
- The number of queries per page

- Network latency
- The configuration of the application server
- The number of uncached components in your site

Hardware for a Small System

The system described in the following table supports 60 concurrent users on the management environment. The delivery environment supports:

- 15 million transactions per day with 90% of components cached
- 1.9 million dynamic page views per day with 90% of components cached

	Management Environment	Delivery Environment
Web Servers	2 E250s, 1 CPU and 512 MB of memory each	2 E250s, 1 CPU and 512 MB of memory each
Application Servers with Content Server	2 E250s, 1 CPU and 1 GB of memory each	2 E250s, 1 CPU and 1 GB of memory each
Database Servers	1 E250, 1 CPU and 1 GB of memory each	1 E250, 1 CPU and 1 GB of memory each

Hardware for a Medium-Sized System

The system described in the following table supports 120 concurrent users on the management environment. The delivery environment supports:

- 30 million transactions per day with 90% of components cached
- 3.8 million dynamic page views per day with 90% of components cached

	Management Environment	Delivery Environment
Web Servers	2 E450s, 2 CPUs and 1 GB of memory each	2 E450s, 2 CPUs and 1 GB of memory each
Application Servers with Content Server	2 E250s, 1 CPU and 1 GB of memory each	2 E450s, 2 CPUs and 2 GB of memory each
Database Servers	2 E450s, 2 CPUs and 2 GB of memory each	2 E450s, 2 CPUs and 2 GB of memory each

Hardware for a Large System

The system described in the following table supports 240 concurrent users on the management environment. The delivery environment supports:

- 120 million transactions per day with 90% of components cached
- 15 million dynamic page views per day with 90% of components cached

	Management Environment	Delivery Environment
Web Servers	4 E450s, 2 CPUs and 1 GB of memory each	4 E450s, 2 CPUs and 1 GB of memory each
Application Servers with Content Server	2 E4500s, 4 CPUs and 2 GB of memory each	2 4500s, 8 CPUs and 8 GB of memory each
Database Servers	2 E4500s, 4 CPUs and 2 GB of memory each	2 4500s, 4 CPUs and 4 GB of memory each

Hardware to Make Your Site Scalable

Designing a scalable site is a matter of foresight—you choose hardware that is sufficient to run the web site today, and that is powerful and flexible enough to adapt to conditions in a year, when the company adds 30 new editors and visitor traffic doubles.

Recommend hardware that allows the client "room to grow," enabling them to add more processors, memory, and network connections as necessary.

The following scenarios provide examples of how to scale a site:

- To handle additional visitor traffic add CS-Satellite systems to the delivery environment.
- To handle additional editors or a business requirement that the management environment be available 24x7, cluster the application servers. Because clustering slows performance, consider adding more processors (to run multiple JVMs) and/or more RAM.
- To handle additional authors or editors who do not need to work on the management environment directly, allow them to edit their content outside of Content Server, then load the edited content into the Content Server database as a batch with XML Post.
- To handle a design that increases the number of uncachable pagelets (like some types of personalization),add more processors to the delivery cluster.

Hardware for a Multinational Site

Another factor to consider when choosing hardware for a site is where the web site will be viewed. There are three factors to consider when choosing hardware for a multinational site:

- Time difference
- Content overlap
- Geographic distance

Time Difference and Hardware

The time difference between the areas where your site will be viewed affect the hardware design. If the time difference is such that peak time for visitors on one version of the site coincides with a slow period, when large-scale updates to a site's content normally occur, consider using a separate Content Server system for each version of the site. Using two systems allows you to update each version of the site separately, when visitor traffic is low, thus improving system performance.

For example, if your site has a U.S. version and a Japanese version, you should consider having two systems so that you can publish new content for the US site when traffic is low in the U.S. (but high in Japan), and publish new content for the Japanese site when traffic is low in Japan (but high in the U.S.).

Content Overlap and Hardware

The amount of shared content between versions of a site also affects hardware design. On sites with a large amount of content overlap, use one Content Server system; on sites with a small amount of content overlap, use multiple systems.

The following diagram illustrates how the amount of shared content between sites helps determines the number of Content Server systems that you need:

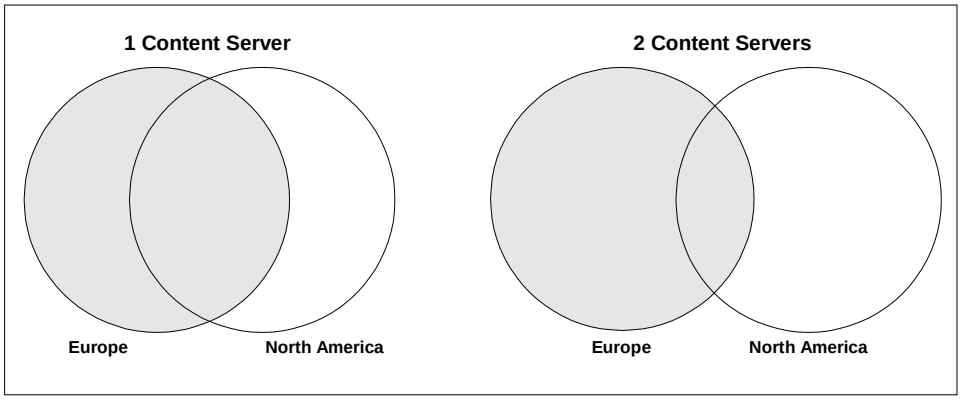


Figure 12: Web Site Content Overlap

Geographic Distance and Hardware

The geographic distance between the audiences for your web site affects your hardware design. Having an additional server closer to your intended audience improves performance. There are two main ways to move content closer to its audience:

- Add additional Content Server systems when you are creating multiple web sites with very different content.
- Add CS-Satellite systems when you are creating one web site, or when you are creating multiple web sites with very similar content.

Hardware Guidelines

The following guidelines will help you to determine how many systems you need:

Table 1: Hardware Guidelines

Site Characteristics	Number of Systems
Multiple versions for display in different time zones	Multiple Content Server systems
Large amount of shared content between versions	One Content Server system
Small amount of shared content between versions	Multiple Content Server systems
Significant distance between your server and your audience (which may affect performance)	Multiple Content Server systems or add CS-Satellite systems

Choosing Software

The various members of the Content Server Enterprise Edition (CSEE) product family are designed to support different types of web sites. Use the following guidelines to help you determine which software to recommend to a client:

Table 2: Software Guidelines

Product:	Site Requirements:
CS-Direct	• Your site is static, dynamic, or some combination of the two. • An asset can be one row in a database table.
CS-Direct Advantage	• Your site is dynamic only. • The design of individual instances of an asset type varies widely. • An asset needs to inherit traits from multiple parents. • Your assets have many fields. • You will import large amounts of data into the system on a regular basis.
CS-Engage	• You are using CS-Direct Advantage to create your web site. • You want to recommend content or items for sale to web site visitors based on criteria that you select. • You want to create promotions for certain types of web site visitors.

Product:	Site Requirements:
Analysis Connector	• You are collecting data from your web site, for use by a third-party analysis engine.
Commerce Connector	• You are using CS-Direct or CS-Engage. • You need to sell products or services. • You have Transact.
CS-Bridge Enterprise	• You are using Content Server. • You want to connect to an ERP, SAP, or other legacy system.

Chapter 3

Page Design, Caching, and Publishing

Page design, page caching, and publishing are interrelated topics that will profoundly affect how your finished web site performs. This chapter contains an overview of each topic and tips that will help you design an efficient web site. It contains the following sections:

- Page Caching Overview
- Publishing Overview
- Page Design Best Practices
- Page Caching and Publishing Best Practices
- Assessing Existing Page Designs

Page Caching Overview

An effective page caching strategy allows your site to perform well and influences your hardware design by relieving load on Content Server and the database. The Content Server Enterprise Edition (CSEE) product family contains two products that cache web pages:

- Content Server, which caches pages either on disk or in Java memory
- Content Server Satellite, which caches pages on remote servers

For optimum performance on the delivery server, use the caching capabilities of both products in tandem, as described in the following sections.

Content Server Page Caching

FatWire recommends that you cache your pages based on the following two principles:

- Cache most pages
- Use uncached pages only where necessary

The caches contain the pagelets, which contain HTTP headers and body content (usually in HTML) that are generated when elements are evaluated by Content Server.

Page-cached items are stored in memory and in the database's cache tracking tables. Uncached elements are evaluated by Content Server each time they are requested. This slows performance and puts additional load on Content Server, so you should try to cache as many elements as possible.

FatWire recommends that you design your pages so that 75 - 90% of the content can be cached. You should try to cache as many componants as you can. However, there are some valid reasons not to cache componants:

- If content changes frequently.
- If logic in the element needs to be evaluated whenever the page is called.
- If the application requires current data.

For instructions on how to implement Content Server caching, see the *CSEE Developer's Guide*.

BlobServer Caching

Content Server caches the output generated by page evaluation; it does not, however, cache images and other binary large objects (blobs). You have three options for managing blobs:

- Use BlobServer to serve and cache blobs.
- Use BlobServer to serve uncached blobs.
- Place blobs on your delivery web server, instead of using BlobServer.

The following guidelines will help you determine how to handle blobs on your site:

Use BlobServer with Caching	Use BlobServer without Caching	Put Blobs on the Web Server
• If security is unimportant—cached blobs are not bound by Content Server security. • If you have sufficient Java memory—blobs are cached in Java memory.	• If security is more important than the performance improvement that caching provides—when Blob Server security is enabled, blobs cannot be cached.	• If security is unimportant—blobs served by the delivery server are not bound by Content Server or BlobServer security. • If speed is important—blobs served by the delivery server increase performance.

For more information on BlobServer, see the *CSEE Developer's Guide*.

Content Server Satellite Caching

CS-Satellite is a FatWire software product that improves the performance of Content Server, especially for dynamic, personalized sites. It adds fast and affordable additional caches to the Content Server system. CS-Satellite also includes throttle code to help limit the load on Content Server.

Each copy of Content Server comes with a copy of CS-Satellite, which is automatically installed on your Content Server machine. You can also install additional copies of CS-Satellite on remote machines, moving your content closer to its intended audience, which speeds performance.

With CS-Satellite, you use Satellite XML or JSP tags to mark individual pagelets for caching in the Satellite cache.

Note that pages cached on CS-Satellite are not automatically protected by Content Server security; for information on how to code your pages so that they are protected by ACLs, and for more information about CS-Satellite caching in general, see the *CSEE Developer's* Guide.

For information on configuring CS-Satellite, see the *CS-Satellite Installation Guide*.

Caching and CacheManager

Content Server's CacheManager object maintains both the Content Server and CS-Satellite caches. CacheManager can do the following:

- Log pagelets in the cache tracking tables.
- Keep a record of the content (assets) that pages and pagelets contain by recording cache dependency items in cache-tracking tables. Cache dependency items are items that, when changed, invalidate the cached pages and pagelets that contain them. A cache dependancy item is logged as a dependancy for the current page and all of that page's parent pages.
- Remove pages and pagelets containing invalid items from the Content Server and CS-Satellite caches.
- Rebuild the Content Server and CS-Satellite caches with updated pages and pagelets after the invalid pages have been removed.

CacheManager completes these operations automatically—after you have mirror published new content to the delivery system, for example—ensuring that the pages that web site visitors see are always up to date.

For web sites that use Content Server alone, CacheManager's cache tracking and flushing are not automatic; however you can use CacheManager's Java API to implement similar functionality on your site. See the CSEE Developer's Guide and the CSEE Java API Reference for more information about the CacheManager object and Java API.

Caching and Session

By default, Content Server stores session information in cookies. This can cause a problem if users have cookies turned off.

If cookies are disabled, Content Server can rewrite the URL to include the session information. You can turn on URL rewriting by setting the

`cs.requiresessioncookies` property in `futuretense.ini`. Note, however, that pages with session information encoded in the URL cannot be cached; therefore performance can be affected if cookies are disabled.

Publishing Overview

Content Server products support two methods of **publishing**, or transferring assets from the management environment to the delivery environment:

- **Mirror to Server**, which copies content assets to the delivery environment. Mirror to Server is for dynamic web sites.
- **Export to Disk**, which evaluates Content Server pages and outputs the resulting HTML files to the file server. Administrators then use FTP or some other method to transfer the generated files to your delivery web server. Export to Disk is for static web sites.

Note that Mirror to Server transfers assets only to the other delivery machines; Java code and other items must be transferred to the delivery machine using some other method. See the *CSEE Administrator's Guide* for more information on publishing, and transferring these items to the delivery system.

Site Design and Export to Disk Publishing

Export to Disk publishing requires that your site administrator set one or more export starting points. The export starting point designates the asset to start with and the template to use for that asset when the publish process starts rendering. Without a starting point, the Export to Disk process cannot begin.

You must design your site so that one or more export starting points allow the publish process to work its way down the site hierarchy and publish each page in the web site. Consider, for example, two sites with the designs depicted in the following diagrams:

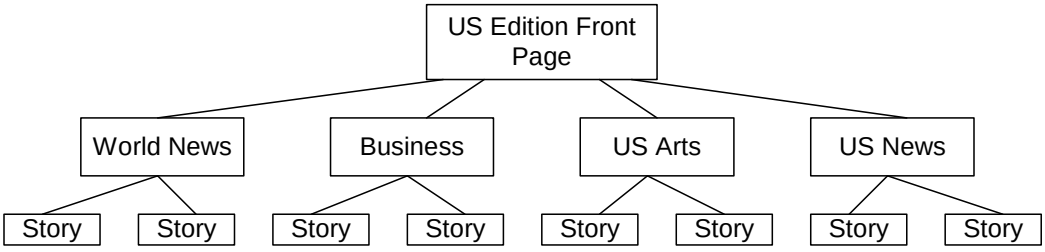


Figure 13: A Site with One Publish Starting Point

The site shown in Figure 13, "A Site with One Publish Starting Point," requires one export starting point, placed at the US Edition Front Page. From this starting point, the publish process can access all of the pages that make up the site.

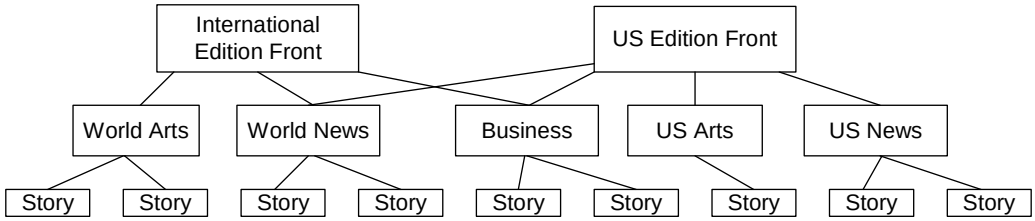


Figure 14: A Site with Multiple Publish Starting Points

The site shown in Figure 14, "A Site with Multiple Publish Starting Points,"requires two export starting points—one placed at the International Edition Front page and one placed at the US Edition Front page—in order for the publish process to access all of the pages in the site.

Note that dynamic sites, which use the Mirror to Server publish method, do not need publish starting points.

For more information about publishing and publish starting points, see the *CSEE Administrator's Guide* and the *CSEE User's Guide*.

Page Design Best Practices

FatWire's recommended page design and caching strategy has three parts:

- Choose the coding language.
- Determine which pagelets to cache.
- Code the pages with Satellite tags.

Choose a Coding Language

CSEE supports both XML and JSP as a language for creating elements. You can call elements written in JSP from an XML element and the reverse. You cannot, however, mix XML and JSP code in the same element. Note that the performance impact of invoking a JSP element is slightly greater than the impact of calling an XML element. Keep this fact in mind when deciding what language to code an element in.

You can improve your site's performance by using XML and JSP in specific types of elements:

Use XML	Use JSP
• For page layout • For displaying text or images • In elements with few loops or conditionals	• In elements with a large number of loops and conditionals

CSEE also supports Java. Use the CSEE Java API to code your business logic.

Determine Which Pagelets to Cache

In general, you should build modular pages where the container page is not cached but the pagelets that it contains are cached. This page caching strategy has two benefits:

- It allows you to put your lightweight business logic, security, and other things that require either page evaluation or a trip to Content Server to function properly, into the uncached containing page.
- The cached pagelets allow you to retain some of the performance benefits of caching.

The following diagram illustrates this strategy:

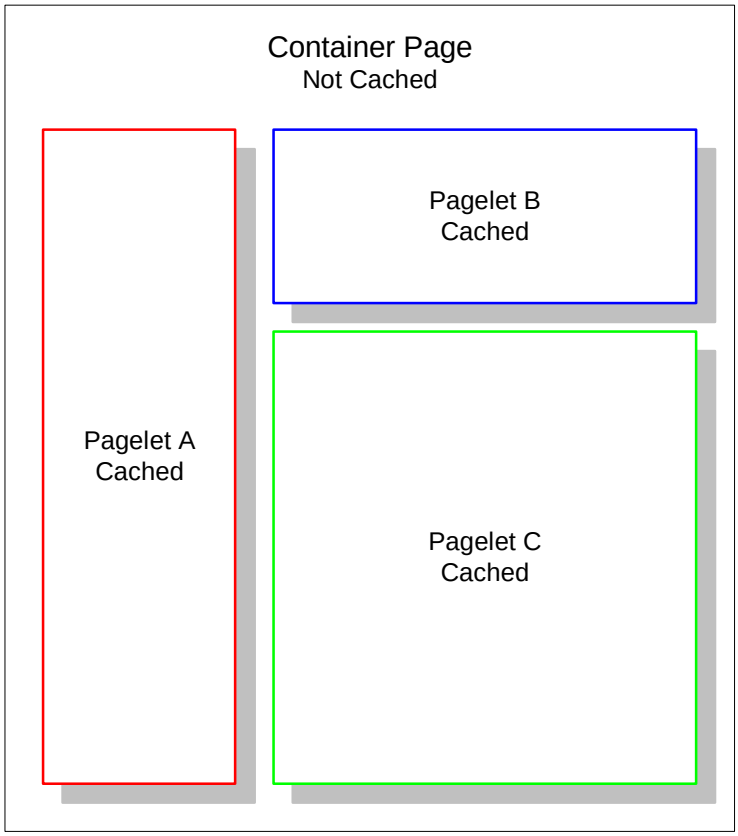


Figure 15: Page Caching Strategy

The container page, is uncached to allow developers to put ACL checking code there.

The other pagelets, which contain content which will change as new articles are added to the database, are cached.The page that is displayed, however, will never show outdated content, because CacheManager prevents the display of old content and removes it from the cache.

The following table summarizes the guidelines for caching pagelets:

Cache a Pagelet . . .	Do Not Cache a Pagelet . . .
• If the content seldom changes. • If the pagelet does not contain logic that requires evaluation to work.	• If the content must be “real time”. • If the pagelet contains logic that requires evaluation to work.

Code Your Pages

FatWire recommends that you code your pages with Satellite tags even if you are not running CS-Satellite. If you are not running CS-Satellite, the Satellite tags will behave as the corresponding Content Server tags.

For example, the code for a containing page element looks similar to the following:

Line one is the standard opening for FatWire XML pages.

```
1   <FTCS>
```

Line 2 calls another element that contains the display logic for this web page.

```
2   <callelement... "content_logic">
```

Line 3 opens the `satellite.tag` tag. The `satellite.tag` tag alerts Content Server that portions of this page will be cached on the CS-Satellite systems, as well as on Content Server, and that the Content Server hyperlinks on the page will be automatically converted to Satellite links.

```
3   <satellite.tag .../>
4   <table>
5   <tr>
```

Line 6 calls an element with the `satellite.page` tag. An element called with the `satellite.page` tag will be cached on both Content Server and your CS-Satellite systems.

```
6           <td><render.satellitepage ...."header"></td>
7   </tr>
8   <tr>
9           <td><render.satellitepage .... "content_asset"></td>
10  </tr>
11  </table>
12  </satellite.tag>
13  </FTCS>
```

For more information about CS-Satellite and coding with Satellite tags, see the *CSEE Developer’s Guide.*

Page Caching and Publishing Best Practices

If you are developing a system that includes the CS Content Applications, CS-Satellite and Content Server caches in work in tandem on your management and delivery systems. The goal of this strategy is to serve as many pagelets as possible from the CS-Satellite and Content Server caches.

The strategy has three major benefits:

- CS-Satellite includes throttle code to help control the load on Content Server.
- Cached items mean less evaluation, and therefore less load, for Content Server.
- Cached items are served more quickly than those that must be evaluated before they are served.

For more information about the CacheManager object, see the *CSEE Developer's Guide*.

Assessing Existing Page Designs

In many projects, the appearance of the visitor site's pages has been developed long before the architect arrives. In this case, the architect must evaluate the existing page design. The following steps outline the evaluation process:

1. Determine what can be a pagelet.
2. Determine what can be an asset.
3. Determine what asset type is best for each pagelet.
4. Determine what can be cached.

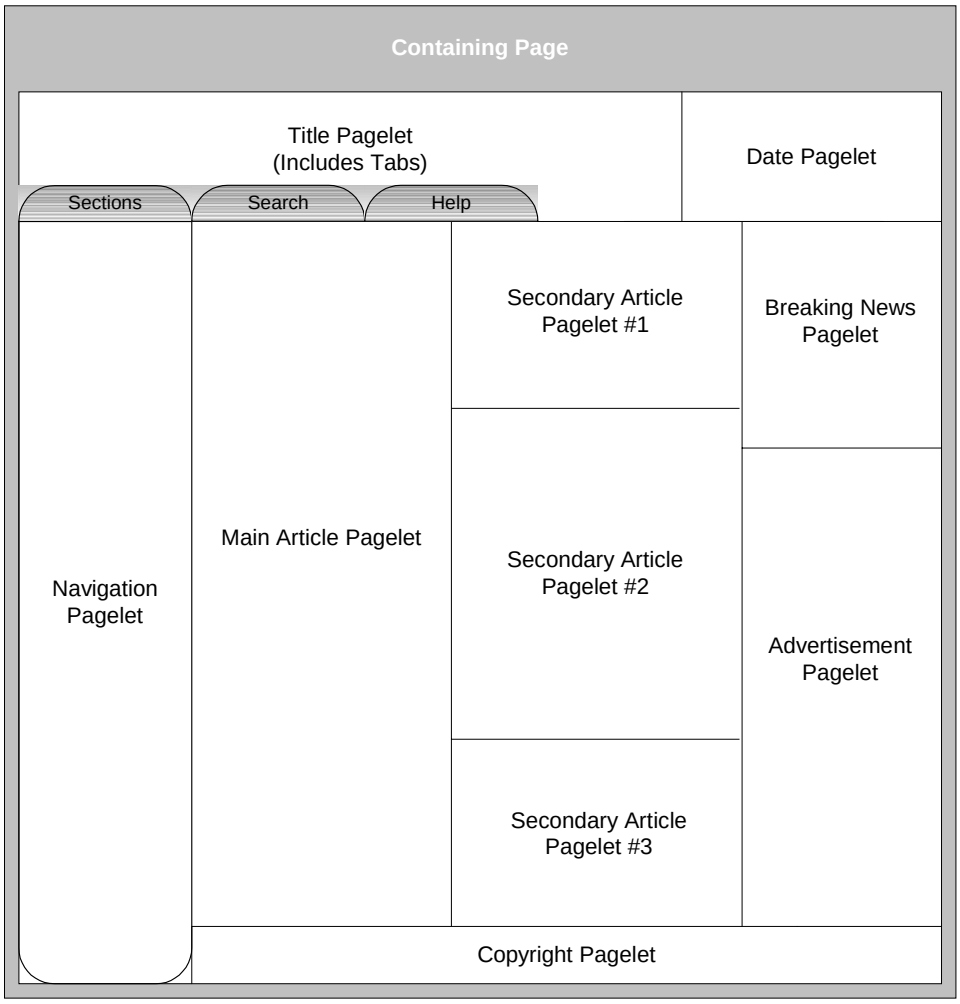
Determine What Can Be a Pagelet

The first step in assessing an existing page design is to break down that design into pagelets. Each item on a page will be part of a pagelet; the question is how to determine which items go into each pagelet. The following list provides guidelines for determining what is a pagelet:

- A section can be reused on multiple pages of the web site—a navigation bar, for example.
- A section is a cohesive unit of information—a newspaper article, or an article and its related image, for example.
- A section is a list—a list of links or abstracts of current articles, for example.

Note, however, that the dynamic assembly of pagelets by CS-Satellite can become a performance bottleneck if the number of pagelets per page is very high. If the CPUs on your CS-Satellite boxes are at their maximum, you may have too many pagelets in your page. To avoid this, try to design your pages so that they are composed of no more than than 10 pagelets.

For example, the Daily Standard page can be broken down into 10 pagelets and a containing page:



Any pagelet can be displayed on multiple pages, if necessary.

Determine Whether the Content of a Pagelet Should Be an Asset

The second step in assessing an existing page design is to determine which portions of the page should be assets. The following general guidelines will help you make this determination:

The content of a pagelet *should* be an asset:

- If the content must go through workflow.
- If the content needs to have restricted access.
- If the content changes frequently, so that the developers don't need to alter the templates each time things change.

The content of a pagelet might *not* be an asset:

- If the content of the pagelet will not change.
- If the item does not need to go through workflow.
- If the content of the pagelet is the result of business logic.

Example

For example, the Daily Standard's page breaks down as follows:

- The Title and Navigation pagelets do not need to be assets because they change very infrequently.
- The Date pagelet is a piece of business logic hard-coded into a template.
- The Main Article and Secondary Article pagelets should be assets because their content changes frequently.
- The Breaking News pagelet is an asset or a CS-Direct Advantage searchstate. Make the Breaking News pagelet an asset if you want non-developers, such as business managers or editors, to be able to change the query and put it through workflow.
- The Advertisement pagelet is a piece of business logic that displays random ads, rather than an asset. The advertisements that the Advertisement pagelet displays, however, are assets, as they must go through workflow.
- The Copyright pagelet is an asset, because although its content does not change frequently, any changes must be approved by the legal department. This means that the text must go through workflow.

Determine Which Asset Type Is Best for Each Pagelet

Once you have determined which portions of the page are assets, you must determine which asset type is appropriate for each asset.

FatWire products include a variety of sample assets that are installed if you install the sample sites. If none of the sample assets suit your needs, you can create custom asset types. For more information on creating assets, see Chapter 4, "Data Design and Resultset Caching."

The following tables describe a subset of the assets that come with CS-Direct, CS-Direct Advantage, and CS-Engage. Core assets are integral to the functioning of their respective products. Sample assets are included with a product's sample site and are only installed if you install your product's sample site.

The tables describe the asset types that you can display on your web pages, and do not include the site design assets, such as the CS-Direct Page asset, that will never be displayed on your finished web pages.

Table 3: CS-Direct Asset Types

Asset Type	When to Use
Collection Asset (Core Asset)	A collection asset is a group of assets of one asset type, chosen based on criteria that you select using a query asset. An editor manually orders the assets in a collection. You can create collections of any asset type except template. Use a Collection asset when an editor must manually order the assets in a collection.
Query Asset (Core Asset)	A query asset is made up of index or database queries that select a set of items. Use a query asset when: • You are creating a collection asset. The group of assets in a collection asset is created using a query asset. • You are creating a manual recommendation asset in CS-Engage. The group of assets in a manual recommendation asset is created using a query asset. • You want to display a list of items which do not need to be ordered manually.
Article Asset (Sample Asset)	The article asset type is included with the Burlington Financial sample site. It represents an article on the site.
ImageFile Asset (Sample Asset)	The image asset is included with the Burlington Financial site. It contains the information needed to retrieve an image from the file system.
SiteEntry Asset (Core Asset)	The SiteEntry asset type represents an entry in the SiteCatalog table. SiteEntry assets are associated with a CSElement asset.
CSElement Asset (Core Asset)	The CSElement asset type represents an entry in the ElementCatalog table. Use a CSElement asset when you want to do the following: • Code that is not for rendering an asset and that you want to reuse in more than one place and/or call from more than one type of template. For example, you have six templates that use the same top banner so you create a CSElement asset for the code in the banner and call that element from each template. This way, if you decide to change the way the banner works, you only have to change it in one place. • Recommendations for CS-Engage. If you create a dynamic list recommendation, you must create a CSElement asset to build the dynamic list. For more information, see the elements and templates chapter in the *CSEE Developer's Guide*.

Asset Type	When to Use
Page Asset (Core Asset)	The Page asset type stores references to other assets. Unlike the collection asset type, the assets that you refer to in the page asset do not need to be of the same type. Note that a page asset and a Content Server page are quite different. The page asset is an organizational construct that you use in the Site Plan tab as a site design aid and that you use to identify data in your elements. A Content Server page is a rendered page that is displayed in a browser or by some other mechanism.

Table 4: CS-Direct Advantage Flex Asset Types

Asset Type	Description
Product Asset (Sample Asset)	A product asset is an individual item with an associated set of flex attributes. Each product has a flex definition and one or more flex parents.
Article Asset (advanced) (Sample Asset)	An advanced article asset is a named asset with text content, similar to a CS-Direct article asset. Each advanced article asset has an associated flex definition that determines what attributes of the article appear on your web page. You define the flex attributes, and determine which attributes apply to an article by defining its associated flex definition. Like product assets, advanced article assets have one or more flex parents, from which they inherit attributes. An advanced article asset can have its own flex attributes, in addition to those that it inherits from its parents.
Image Asset (advanced) (Sample Asset)	An advanced image asset is a named asset with image content, similar to a CS-Direct image asset. Each advanced image asset has an associated flex definition, which determines the attributes of the image. You define the flex attributes, and determine which attributes apply to an image by defining its associated flex definition. Like product assets, advanced image assets have one or more flex parents, from which they inherit attributes. An advanced image asset can have its own flex attributes, in addition to those that it inherits from its parents.

Table 5: CS-Engage Asset Types

CS-Engage includes a recommendation asset type. The assets described in the following table are not different asset types; rather they are variations on the recommendation asset type.

Asset Type	Description
Static List Recommendation Asset	A static list recommendation asset holds a static, preselected list of assets. When a site page invokes the recommendation, it returns the items in this list.
Dynamic List Recommendation Asset	A Dynamic List Asset is associated with a CSElement asset that uses CS-Direct Advantage `searchstate` and `assetset` tags to query the database. When a site page invokes the recommendation, it runs the element and returns the items that match the query's conditions.

Asset Type	Description
Related Item Recommendation Asset	A related item recommendation holds the name of a relationship. When a site page invokes a relate item recommendation, items are retuned (recommended) only if they are manually configured to have a relationship with an asset on the page—usually a cross-sell or up-sell relationship. For example, you can create a relationship called `CrossSell` that displays a list of handbags on pages that display women's shoes, because marketing has determined that women who buy shoes also buy handbags.

Chapter 4

Data Design and Resultset Caching

The database is the backbone of your content management system; it is where much of your web site's content is stored. This chapter provides an explanation of data design and maintenance in the Content Server Enterprise Edition (CSEE) product family, and also describes how to implement resultset caching, which stores the results of your database queries in Java memory.

This chapter contains the following sections:

- Data Design
- Designing Assets
- Data Retrieval
- Resultset Caching
- Database Maintenance

Data Design

The CSEE database tables contain several types of information:

- Structural information, which provides the information used to create the structure and business logic of your web site. Examples of structural information include the templates used to display content on the site.
- System information, which allows CSEE to run. Examples of system information include login information and workflow information. Content Server system tables, such as the ElementCatalog table and SystemACL table, should not be modified.
- Content, such as articles that you want to display on the site.

You will need to add new database tables to the tables already provided with CSEE. The design of these tables has a close correlation with the design of the assets that will populate them. Because asset creation affects the database schema, you should have a DBA present as you design the assets.

Basic Assets

CS-Direct provides the **basic asset** model. Basic assets generally represent content, though there are also basic assets that allow you to organize and manage your web site.

Consider a basic asset as one row in a database table or spreadsheet. For example, all of the information for a newspaper article—the title, author, an abstract, a URL to where the text of the article is stored on the file system, the byline, and source of the article—and the IDs and other information needed to manipulate the asset within CS-Direct can be recorded in a single row of a database table, as a basic asset.

CS-Direct's sample site comes with several basic asset types already defined. You can modify these asset types to suit your needs, or create custom assets for your site. See the *CSEE Developer's Guide* for more information on creating and modifying asset types.

Flex Assets

CS-Direct Advantage provides the **flex asset** model. Flex assets simultaneously represent and organize content. Unlike basic assets, where the information for one instance of an asset is stored in one row of a database table, the information for one instance of a flex asset is stored in multiple database tables.

Flex Families

Flex assets are grouped into **flex families**. The members of a flex family form an asset inheritance tree, where child assets inherit various attributes from their parents. The following diagram represents an asset inheritance tree for an online newspaper:

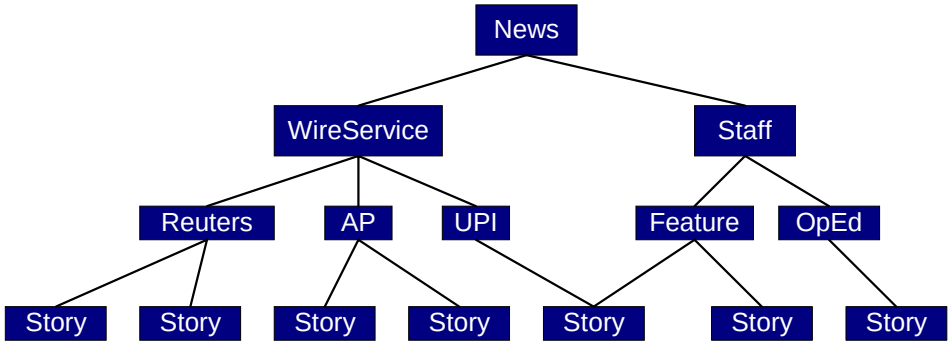


Figure 16: An Asset Inheritance Tree

This asset inheritance tree makes entering new stories into the system faster and easier, as some of the attributes for each story are inherited from its parents and grandparents, and so do not need to be entered by the editor. For example, The WireService node of the asset inheritance tree has a service attribute—a field that holds the name of the wire service that the story came from. This attribute and its value are inherited by all of the node's children and grandchildren, so all stories with Reuters as a parent inherit the information that they were supplied by Reuters.

Flex assets also support multiple inheritance, meaning that they can inherit attributes from more than one set of ancestors. Imagine, for example, that you are developing a system for a company which owns two newspapers. The papers share wire service stories, as shown in the following diagram:

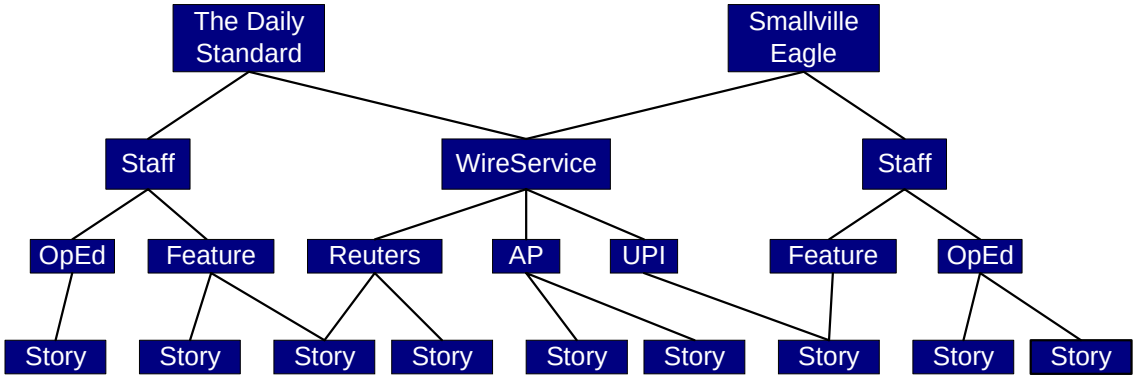


Figure 17: Multiple Inheritance Trees

Each story includes the logo of its respective paper. Flex assets support this by allowing wire service stories to inherit logos from both The Daily Standard and the Smallville Eagle. Developers then code the web site for each newspaper to display the applicable logo.

Note that the hierarchy of these asset inheritance trees has no effect on the hierarchy you build on the delivery web site to allow visitors to drill down to the information they need. Because the asset inheritance trees and your web site navigation are completely independent of each other, you can create asset inheritance trees suited to the needs of your editors, who enter stories into the system based upon the source that supplied them, without forcing your web site visitors to browse the finished web site based on a story's source, rather than the newspaper section that the story belongs in.

Flex Asset Types

Flex families are composed of five flex asset types:

- Flex Asset - The individual items at the bottom of an asset inheritance tree. Flex assets are composed of flex attributes.
- Flex Attribute - An individual unit of information that composes a flex asset. For example, "color" or "abstract" can be flex attributes.
- Flex Definition - A set of attributes that defines a kind of flex asset. You create a named flex definition that then serves as a template to create individual flex assets with similar characteristics.
- Flex Parent - An association of individual flex assets. You create item parents that help you organize or manage a set of flex assets.

  Each flex parent has its own set of flex attributes. All the children of the parent inherit these attributes. Each flex parent has a single flex parent definition, which defines the set of attributes that make up the flex parent.
- Flex Parent Definition - Similar to a flex definition, a flex parent definition serves as a template to define which attributes make up a flex parent.

Asset Variability

Flex assets support more fields than basic assets do, and allow individual instances of an asset type to vary widely. Imagine, for example, that you are using flex assets to design an online catalog for a company that sells housewares and linens. You create an flex parent asset type called `product` to represent the items for sale. The individual products for sale, however, vary greatly—sheets, for instance, are quite different from toasters.

For example, the information necessary to create an individual instance of a "sheet" product and a "toaster" product is as follows:

toaster	sheet
product_name: ToastOMatic 3000	product_name: Pima Flat Sheet
manufacturer: Appliance Co	manufacturer: US Linens
type: slot	size: Twin
SKU: 84756532	color: White
	fiber: Cotton
	thread_count: 310
	SKU: 9380547854

Flex assets have the flexibility to accommodate this difference; your developers create groups that only have fields for entering information appropriate to a specific product type. For this example, the developer creates a `toaster` flex definition and a `sheet` flex definition. The flex definitions only include the fields needed for each product type, so that the `toaster` definition does not include a `fiber` field.

Choosing Basic or Flex Assets

The following table contains guidelines for choosing standard or flex assets:

Use Basic Assets:	Use Flex Assets:
• If the asset can be a single row in one database table • If individual instances of an asset type have the same structure • If your web site visitors will be browsing the web site	• If you want the asset to inherit traits from its ancestors • If individual instances of an asset type vary widely • If you need attributes to have multiple values • If your web site visitors will be searching the web site

Designing Assets

The following sections contain information about developing basic and flex assets.

Designing Basic Assets

The following list contains design tips for basic assets:

- Asset design is constrained by the database that you use. For example, in Oracle, only one column (and hence one field in the asset) can have a `long` datatype.
- Try to minimize the number of fields you use in an asset by keeping the information they contain in useful units. For example, use one field for a telephone number, as each component of the number is of little use alone, but use two fields for a person's name, as you may want access to both the first and last name separately.
- Consider the asset's users—the content providers and editors who are entering data into the CS-Direct user interface. Do not display fields for information that is inappropriate for their jobs; create another asset to hold this information.

Designing Flex Asset Families

Most management systems will utilize multiple flex asset families. How you design these families and the assets that compose them affects both your database and the usability of the management system.

As you design flex asset families you must create a balance:

- Limit the number of flex parent definitions associated with a given flex definition, so that content providers and editors will not have to choose between hundreds of flex parent definitions to find the one appropriate to the information that they need to enter.
- Create enough flex parent definitions and flex definitions so that content providers and editors can find a definition with a minimum of fields that are not applicable to their task.

Designing the flex asset families for your project is a process that consists of the following steps:

1. Determine all of the attributes that you will need.
2. Determine which items have attributes with unique values.
3. Determine the number of flex definitions that you need
4. Determine the number of flex families that you need

Determine Which Attributes You Need

The first step in designing a flex asset family is to determine all of the attributes that you need for your site.

Note that this means more than determining the attributes that you need for your business requirements or the attributes that you want to display to web site visitors. You must also determine how you want content to be displayed on the finished web site and how you want web site visitors to "drill down" to items that they want.

For example, if you want to display a list of the ten most recent stories submitted to a newspaper, you must include an attribute that holds the date and time that the story was submitted, allowing your developers to create logic that retrieves the most recent

submissions. Similarly, if you would like web site visitors to search on articles based on the section they fall under—Sports, for example—you must include a "section" attribute.

Note

Use a multi-value field if an item is in more than one category—for example, if a news story can be classified as both "business" and "international," or if a movie can be classified as both "romance" and "musical."

Determine Which Attribute Values Are Unique

Children in the asset inheritance tree inherit attributes and their values from their parents and grandparents. Attributes where the values must be unique for each instance of an item—SKU, for example—are included in the flex definition, near the bottom of the asset inheritance tree. Conversely, attributes with common values are candidates for being item parents and item parent definitions, so that those values can be inherited by the individual items that need them.

Determine the Number of Flex Definitions That You Need

The number of flex definitions you create affects the number of data fields that appear in the asset forms on the management system. The number of definitions that you must create is determined by how different the individual flex assets at the bottom of the asset inheritance tree are.

Note that the number of attributes that make up a Flex Definition affects the amount of time it takes for that Flex Definition's form to load. It takes between 50 and 350 milliseconds to display one attribute field (depending on your attribute editor), so displaying many attribute fields can create slow forms.

In an online catalog, for example, you could create one flex definition called `item` which would act as the template for all items that the editors enter into the system. If, however, the catalog contains very different items, such as sheets and toasters, a universal product definition is not the best choice; sheets require many fields that toasters do not, forcing editors to leave many fields empty. A better solution is to create two flex definitions, one for toasters and one for sheets, where each flex definition contains only the fields necessary for that type of item.

Determine the Number of Flex Families That You Need

The database schema for Content Server Direct Advantage includes "mungo" tables. A mungo table contains attribute values for an associated Flex Family. Mungo tables can grow very large, often containing more than a million records.

Creating multiple Flex Families is a good way to separate your data into several mungo tables in your database, thus differentiating your data and allowing you to control security and archiving through individial Flex Families.

Note, however, that searching for content across multiple mungo tables is slower than searching for content in a single mungo table.

Data Retrieval

After you design your site's data structure, you can begin designing the queries that will help to build your site and populate it with content. The different CSEE products have different ways of retrieving data from the database, and these methods are outlined in the following sections.

Data Retrieval in Content Server and CS-Direct

The CSEE Java API includes a number of methods that generate SQL queries for use with CSEE. In general, you will use these methods to create your SQL queries, rather than creating queries manually. See the *CSEE Java API Reference* for more information about these methods.

You can either embed SQL queries in templates and other elements, or you can use the Query basic asset type to manage the query. The following guidelines will help you determine whether to use an embedded SQL query or the Query asset type:

Use an embedded SQL query:	Use the Query asset:
• If the results of the query can be used "as is" • If the query does not need to go through workflow • If the query will not change	• If you may need to manually order the results of the query • If the query needs to go through workflow • If the query may need to be changed to reflect changing business requirements • If you need to retrieve an element from the database

Designing SQL Queries

The guidelines for creating a SQL query are the same whether you are creating an embedded query or are using the Query asset type: create queries that retrieve only the information you need.

For example, if you are designing a query which will generate a list of abstracts of the most recent newspaper articles in your database, you only need the titles of the articles and the text of the abstract. Your SQL statement, therefore, should look like the following example:

```
SELECT title,abstract FROM articles WHERE
createddate=Variables.date
```

Using `SELECTTO` for your queries can help improve performance. When you create a query with `SELECTTO`, Content Server Enterprise Edition creates a perpared statement in addition to caching the resultset. This allows your database to cache the query itself, allowing you to retrieve content quickly.

Data Retrieval in CS-Direct Advantage

If you are using Content Server Direct Advantage to design your site, you create **searchstates** to retrieve the data that appears on the delivery web site, rather than creating SQL queries. A searchstate describes the type of data that you want to retrieve from the database, much like a SQL query does. When they are evaluated, searchstates produce **assetsets**. Assetsets are objects that represent groups of assets, and are roughly analogous to resultsets.

Resultset Caching

Content Server allows you to cache the resultsets generated by database queries in Java memory. Implementing resultset caching on a site improves performance and reduces the load on Content Server and the database. Outdated resultsets are removed from the cache in one of three ways:

- Resultsets are deleted from the cache automatically when a table changes.
- Resultsets are deleted from the cache using CatalogManager's `flushcatalog` command.
- Resultsets time out and are deleted based on values set in the property file, `futuretense.ini`.

Values set in `futuretense.ini` also control how many resultsets are held in cache for a given table. For more information about Content Server properties and how to set them, see the *CSEE Administrator's Guide*.

You can control resultset caching on a table-by-table basis. FatWire recommends that you assess each table in the schema, determine how it should be cached, and set the caching properties accordingly. As with page caching, which is described in Chapter 3, "Page Design, Caching, and Publishing," your resultset caching strategy will be different for the development, management, and delivery (production) environments.

Table Updates and Resultset Caching

FatWire recommends that you update your elements using one of three update methods:

- Content Server Explorer
- CatalogMover
- The CSEE user interface

If you are writing elements that modify the data in your database tables programmatically, use CatalogManager commands to change the data.

Resultsets generated by one of these update methods are cached automatically. Outdated resultsets are deleted from the cache automatically if you are using any of these methods. For more information on resultset caching, see the *CSEE Developer's Guide*.

Setting Resultset Caching Timeouts

The following guidelines will help you determine how to set your resultset caching timeouts:

- Use a low timeout value if the data in the database changes frequently.
- Use a high or infinite timeout value of the data in the database seldom changes.

For example, on the development system:

- The `ElementCatalog` table should have its resultset cache timeout set to a low value: 1 minute, for example. This allows a developer to view alterations to an element that has changed, while maintaining some of the load-reducing benefits of caching.
- A table that contains content, such as a table containing newspaper articles, should have an indefinite timeout value, because the content in the articles seldom changes on the development system.

On the management system:

- The timeout on tables that contain content should be low, so that unneeded information does not linger in the cache.
- The timeout on tables containing structural information, such as the `ElementCatalog` table, should be high, because this information seldom changes.

On the delivery system:

- Timeouts for tables that contain structural information, like the `ElementCatalog` table, should be long, because the elements seldom change.

  Experiment in order to find the appropriate value for your system; you want to cache as many structural resultsets as possible for as long as possible, but not to constantly purge the least recently used resultset.
- The timeout on tables that contain content should be low, so that unneeded information does not linger in the cache.

Setting Resultset Sizes

As you can control resultset timeouts, you can also control how many resultsets are held in Java memory. Because Java memory is limited, you should assess each table in your schema and determine the appropriate number of resultsets to save.

The `cc.cacheResults` property controls the default number of resultsets that are cached for all tables. You can override this default on a table-by-table basis by adding properties for those tables using the following convention: cc.`my_tablenameCSz`.

Setting resultset sizes for simple tables, such as the `ElementCatalog` table, is relatively easy to do. In general, the number of queries you want to store is equal to the number of records in the table, because queries to the `ElementCatalog` table seldom return more than one record.

Resultset caching sizes for content asset tables are more difficult to tune because a typical query returns more than one record. If you know that you will always have a small number of records returned, you can specify that a large number of resultsets be stored in the resultset cache.

Database Maintenance

The following tips will help you maintain your database:

- Only use CatalogManager commands, Content Server Explorer, or CatalogMover to modify database tables.
- Write a custom element that deletes assets that are flagged for deletion (have a `status=VO`) from the database.
- Consider moving "mungo" tables and other large tables to separated database machines. This makes these large tables easier to maintain and secure, and increases system performance.

Chapter 5

Security and Personalization

Although they may seem like unrelated features, security and personalization in CSEE are both implemented using the same mechanism—**access control lists** (ACLs). This chapter provides an overview of CSEE security and personalization, and contains the following sections:

- Security Overview
- CSEE Authentication
- Segmentation and Personalization
- Segmentation and CSEE Products

Security Overview

As you design a web site with CSEE, you must implement security for two components of the project:

- The content management systems (development, management, and delivery)
- The visitor web site that you are designing

CSEE provides security through access control lists (ACLs) for both the content management systems and the public web site. ACLs are groups of internal users and web site visitors who share the same access privileges. ACLs have two functions:

- They limit access to tables, page entries (SiteCatalog entries), and assets.
- They limit the functions that users can perform.

By default, CSEE stores its ACLs in the SystemACL table and its user information in the SystemUsers table. Note that you must modify the user interface to Content Server's administrative module to use CSEE ACLs with a large number of users.

CSEE provides the ability to integrate other authentication methods with CSEE, most notably LDAP.

The following guidelines will help you determine whether to use CSEE ACLs and authentication or LDAP authentication:

Use CSEE ACLs and Authentication . . .	Use LDAP Authentication . . .
• If the number of users is small.	• If the number of users is large. • If you need to use a protocol that is available across many platforms.

Securing Content Management Systems

For any content product, you should design your administrative ACLs so that you grant access to tables, pages, assets, and functions only for specific users who require that access.

With Content Server Direct, access to assets is handled by a **site**. A CS-Direct site is a tool to organize the pages and collections for your web site—it does not effect the appearance of your web site in any way. A CS-Direct site can correspond exactly to a live web site that you produce, but does not have to do so. For example, you can create a site that only editors can access, which contains only the content that they need to edit, and a site that only developers can access, which contains only the elements they are developing. The content from both CS-Direct sites is combined to create the live web site. For more information on sites, see the *CSEE Developer's Guide*.

You can also provide ACL protection at the asset instance level by adding an ACL column to any custom asset that you create, and writing code to check that ACL. For more information about internal security, see the *CSEE Administrator's Guide*.

Securing the Web Site

To implement security with ACLs for your web site, you must code login pages for the web site's visitors. When visitors log in to the site, they are associated with one or more ACLs that you select. If you use CatalogManager to log visitors in and out, their login information is kept in Content Server's SystemUser's table. ACLs are kept in the SystemACL table.

You can protect your site by adding code to an element that checks a user's ACL, or you can protect pages without adding ACL checking code by setting ACLs in Content Server's SiteCatalog table.

Note that if you use automatic ACL checking with the SiteCatalog instead of writing your own ACL checking code to secure your pages, pages served from the Content Server cache are protected by Content Server ACLs, but pages served from the CS-Satellite cache are not. For more information on CS-Satellite and security, see the *CSEE Developer's Guide*.

CSEE Authentication

CSEE supports three user authentication mechanisms:

- Content Server Authentication
- LDAP Authentication
- Windows NT Authentication

You choose which of these three mechanisms you want to use for your system when you install an authentication plug-in during the CSEE installation. For more information about the authentication plug-ins, see *Installing the CS Content Applications*. The following sections describe the three authentication mechanisms in greater detail.

Once you have chosen and installed the appropriate authentication method for your system, you interact with the authentication system by using Content Server's directory services tags. For more information about directory services and the directory services tags, see the *CSEE Developer's Tag Reference* and the *CSEE Developer's Guide*

Content Server Authentication

Content Server's default user management facilities support arbitrary user attributes such as email or phone number. However, arbitrary attributes cannot be assigned to ACLs, nor can ACLs be hierarchical. If you want to assign user attributes to ACLs or have hierarchical ACLs, FatWire recommends that you configure Content Server to use LDAP user management.

LDAP Authentication

You can use an LDAP database to store usernames, passwords, and other information about the users of either your content management systems or your site. Refer to *Installing the CS Content Applications* for information on how to configure Content Server to use LDAP.

Windows NT Authentication

Content Server may use Windows NT for authentication, using the default facilities for storing other user information. Refer to the *CSEE Administrator's Guide* for information on how to configure Content Server to use Windows NT.

The default facilities support arbitrary user attributes (such as email, address, etc.). However, arbitrary attributes cannot be assigned to ACLs, nor can ACLs be hierarchical.

Segmentation and Personalization

CSEE supports two methods of tailoring content to the visitors who will view it:

- Segmentation, where visitors are assigned to a group, and content is displayed based on that group
- Personalization, where different content is displayed for each visitor

Implementing segmentation on a site is similar to implementing security with ACLs—visitors log in to the site and are assigned an ACL based on criteria that you select. You then put code into your templates, which display content based on the visitor's ACL.

Because the pagelets you need for different groups can be cached, a well-designed segmentation and caching strategy allows you to customize your pages for different viewers without a large impact on performance or a need for more hardware.

True personalization, however, does have a performance and hardware impact. If the content of each page is unique to each visitor, the amount of material that you can cache is small, putting load on Content Server and slowing performance. The additional load on Content Server can be partially offset by adding more hardware, but the best practice is to use true personalization sparingly and design sites that rely on segmentation and an efficient caching strategy.

Segmentation and CSEE Products

Although you can implement segmentation with Content Server alone, FatWire provides a product that makes implementing segmentation on your site much simpler: CS-Engage.

CS-Engage has a **segment** asset type that defines a group of web site visitors who share a common trait—zip code, for example. Each segment is associated with a set of **rules** that determine what content is displayed for members of that segment. For example, you could create a segment for people who have an 01803 zip code and associate a rule with that segment that displays ads for businesses located in the same zip code.

To learn more about CS-Engage, see the *CSEE Developer's Guide*.

Index

A

access control list (ACL)
  overview 57
asset inheritance tree 48
assets
  and pagelets 41, 42
  asset types 42
  basic 48
  choosing basic or flex 50
  flex 48

B

basic assets
  See also assets
  choosing 50
  designing 51
  overview 48
best practices
  page caching 40
  publishing 40
BlobServer
  caching 34

C

CacheManager object 35
caching
  See also page caching and resultset caching
  and BlobServer 34
  and CacheManager 35
  and session 35
  overview 10
  pagelets 38
Content Server
  and CS-Bridge Enterprise 16
  BlobServer caching 34
  designing SQL queries
  page caching 33, 40
Content Server Direct
  basic assets 48
  designing basic assets 51
  designing SQL queries
Content Server Direct Advantage
  asset variability 50
  designing flex asset families 51
  flex asset types 49
  flex assets 48
  searchstates 54
Content Server Enterprise Edition (CSEE)
  and middleware 15
  architecture 11
  coding languages 37
  design overview 18
  environments 18
  hardware sizing 26
  implementation models 13
  integrating with other applications 15
  J2EE 11
  LDAP 57
  page design 37
  publishing 36
  security 57

Content Server Satellite
coding 39
page caching 35, 40
CS Content Applications
flex assets 11

D

data design
basic assets 48
flex assets 48
overview 47
database queries
searchstates 54
SQL 53
database tables
database maintanance 56
ElementCatalog 9
security 57
SiteCatalog 9
SystemAcl 57
delivery system
hardware 26
page caching 39
development system
hardware 24

E

elements
and tables 9
overview 9
environments
hardware 23
management 25
overview 18
Export to Disk publishing
See also Mirror to Server publishing
overview 36

F

flex assets
asset inheritance tree 48
asset variability 50
attributes 51, 52
choosing 50
designing flex asset families 51
flex asset types 49
flex definitions 52
flex families 48
overview 48

H

hardware
content overlap 29
delivery system 26
development system 24
different environments 23
geographic distance 29
management environment 25
management system 25
multinational sites 28
purchasing guidelines 30
scalability 28
sizing 26
sizing guidelines 26
testing system 26
time difference 29

I

integration
and CS-Bridge Enterprise 16
and middleware 15
overview 15

J

J2EE 11
Java 11, 37
JSP 9
choosing 37

L

LDAP
and security 57

M

management environment
hardware 25
management system
hardware 25
Mirror to Server publishing
See also Export to Disk publishing
overview 36
multinational sites
hardware 28

P

page caching
  See also caching
  best practices 40
  delivery system 39
  guidelines 39
  on Content Server 33
  with Content Server Satellite 35
page design
  best practices 37
  caching 10
  elements 9
  overview 8
  pagelets 9
pagelets
  and assets 41, 42
  and web pages 40
  caching 38
  overview 9
performance
  overview 20
  testing 20
project
  scheduling 21
  staffing 21
publishing
  best practices 40
  Export to Disk 36
  Mirror to Server 36
  overview 36

R

resultset caching
  See also caching
  overview 54
  resultset sizes 55
  table updates 54
  timeouts 55

S

scalability
  hardware 28
scheduling 21
searchstates
  designing 54
security
  access control lists 57
  LDAP 57
  overview 57
sessions
  and caching 35
site design
  and Export to Disk publishing 36
sizing
  hardware 26
software
  choosing 30
SQL
  See database queries
staffing 21
system design
  assessing page design 40
  choosing a coding language 37
  choosing basic or flex assets 50
  choosing software 30
  coding pages 39
  data design 47
  designing flex asset families 51
  designing searchstates 54
  hardware purchasing guidelines 30
  multinational sites 28
  overview 18
  page caching guidelines 39
  page design 37
  performance 20
  publishing 36
  scalability 28
  security 57
  with Content Server Satellite 35
systems
  delivery 26
  development 24
  management 25
  testing 26

T

tables
  See database tables
testing system
  hardware 26

W

web pages
  and pagelets 40
  assessing page design 40
  coding 39

X

XML 9
  choosing 37