



Sitecore CMS 6.1

Integrating External Data with Sitecore

A Developer's Introduction

Table of Contents

Chapter 1	Introduction	4
Chapter 2	Content Integration	5
2.1	Repositories	7
2.2	Communication Frameworks	8
2.3	Mechanisms for Integration	9
2.3.1	ASP.NET User Controls and Server Controls.....	9
2.3.2	Data Provider	10
2.3.3	Pipelines	11
2.3.4	Events.....	12
2.3.5	Commands.....	13
2.3.6	Custom Editors.....	14
2.3.7	Workflow Actions.....	16
2.3.8	Custom Validators	17
2.3.9	Sitecore Modules.....	18
SharePoint Connector.....		18
2.4	Integration Scenarios	20
2.4.1	Delivery Environment	20
Display, Insert or Update Content-Independent External Data		20
Display Content-Dependent External Data		20
Display Pages Not Processed By Sitecore		21
2.4.2	Authoring Environment	22
Use External Content as Metadata or Related Items.....		22
Update External Systems Based on Sitecore Events.....		22
Change Sitecore Behavior Based on External Data.....		22
Chapter 3	Media Integration.....	24
3.1	Repositories	28
3.2	Mechanisms for Integration	29
3.2.1	Media Browser	29
3.2.2	Text Fields	30
3.2.3	UI Upload Pipeline.....	31
3.2.4	Workflow Actions.....	31
3.2.5	Data Provider	32
3.3	Integration Scenarios	33
3.3.1	Delivery Environment	33
Display External Media.....		33
3.3.2	Authoring Environment	33
Upload Media to Content Delivery Network		33
Search and Select Media from Digital Asset Management System or File System		33
Chapter 4	Security Integration.....	35
4.1	Repositories	37
4.2	Mechanisms.....	38
4.2.1	.NET Membership Provider.....	38
4.2.2	Custom Membership Provider.....	38
4.2.3	Active Directory	39
4.2.4	Named Users	39
4.2.5	Member Import.....	39
4.2.6	CRM Security Providers	40
4.3	Integration Scenarios	41
4.3.1	Delivery Environment	41
Third-Party .NET Membership Provider		41

Simple Integration with Federated Authentication System.....	41
User Management for Intranets	41
4.3.2 Authoring Environment	41
Single Sign-On for Content Authors.....	41
Federated Authentication Using .NET Membership	41
Chapter 5 Conclusion	42

Chapter 1

Introduction

Most Sitecore projects involve integrating Sitecore with external data sources. This document provides a glimpse into the kinds of data integration options that Sitecore customers and partners have implemented in Sitecore solutions across the globe. In addition to our out-of-the-box integrations and connectors, almost every aspect of the Sitecore system can be extended to hook into external repositories and applications.

This document discusses several common integration requirements, including the integration of external content, media and security accounts. For each of these topics, we will look at common repositories for external assets, typical communication frameworks, mechanisms in Sitecore for implementing the integration and common integration scenarios.

Note that this document does not focus on specific case studies. Rather, it takes a look at a variety of integration options from a developer perspective and presents them generically. For specific case studies, visit the Product Resources section of the Sitecore Web site: <http://www.sitecore.net/en/Products/Resources.aspx>

Chapter 2

Content Integration

In many Sitecore solutions, customers choose to store some content in Sitecore and other content in external databases or applications. This is often preferable to importing all content into Sitecore for a variety of reasons. For example, there may be existing content that is used by multiple applications across the enterprise. This content may already have sophisticated administrative tools, workflows and business rules that work well in their current state. Users may already be well-trained in the existing application and may be satisfied with its effectiveness. Importing this content into Sitecore may not be necessary, given the possibility of data integration.

In other cases, the nature of the external application itself dictates that the content should not be stored in Sitecore. For example, ERP systems, e-commerce applications and CRM tools store data that is often transactional in nature and managed by complex business rules. Again, these are better candidates for content integration rather than content import.

What do we mean by content integration in Sitecore? It can mean a variety of things, but the assumptions are typically as follows:

- **Sitecore is not the repository of record for the integrated content. In other words, the authoritative source for the content is the external system.**
- **Sitecore does not manage the workflow for the integrated content. Content moves through workflow in the external system.**
- **Sitecore may or may not serve the integrated content on the public Web site.**

Given these assumptions, what are the actual points of integration? Integration scenarios can be divided into three major categories:

- **Integrating external content or applications into the published Web site.**
- **Integrating external content or applications into the authoring interfaces.**
- **Integrating external content or applications into background processes.**

Let's consider each of these in turn. Integrating external content into the published Web site is the most common and simplest scenario. There are several possibilities here. For example, there may be a need to display content from an external system on the same page as (or interspersed with) Sitecore content. Imagine an e-commerce site that

displays product inventory information on the same page as promotional copy for a new product offering. The inventory information may be stored in an e-commerce or ERP system while the promotional information may be native Sitecore content. Another example could be a portal application, where the site aggregates content from a variety of systems and information sources. Finally, consider a scenario where Sitecore content is displayed within applications managed by external systems. For example, a web form might have form field labels stored in Sitecore (to take advantage of multilingual and personalization features), but the submitted form data may get stored in an external system.

Integrating external content or applications into the authoring environment shifts our focus from the published Web site to the actual authoring interfaces. Here, there are a great deal of integration options and innumerable integration scenarios. For example, imagine creating a new content item in Sitecore and associating that content item with a product SKU in your e-commerce system. This could make it easy to let the product marketers edit content in Sitecore while associating the content with information stored in backend systems. Or, consider the scenario where content needs to be associated with metadata information stored in an external repository. For example, your enterprise taxonomy may be defined in an external system. With Sitecore, content authors can use simple pick-lists to create the association between a Sitecore content item and the external taxonomy values.

When integrating external content or applications into background processes in Sitecore, you may be hooking into system events or running your own tasks that affect Sitecore content. For example, you might hook into Sitecore's publishing events to update your search engine index when new content is available. You might schedule a task that creates static RSS feeds with both Sitecore content and content stored in other applications. You could create a workflow integration that updates the workflow status of Sitecore items based on events from external systems.

This chapter discusses some of the mechanisms that Sitecore provides to realize these integration scenarios. You will see that almost every component of the Sitecore application is designed to work in environments with multiple applications and heterogeneous platforms.

2.1 Repositories

For typical applications, data is stored in relational databases or in XML files. This data can be accessed through an application layer, direct database query or XPath. In general, Sitecore recommends accessing external data through the application layer of the external system rather than through direct database queries using ADO.NET. The application layer typically has essential business logic that makes sense of the data in the underlying repository. For example, the application layer may apply security permissions, versioning, workflow information or other application intelligence that is unique to the functioning of the system. As a result, content integration in Sitecore typically involves communicating with the application layer of another system. The type of underlying repository is less important than the availability of an API or web service layer that can provide data access capability to the Sitecore components.

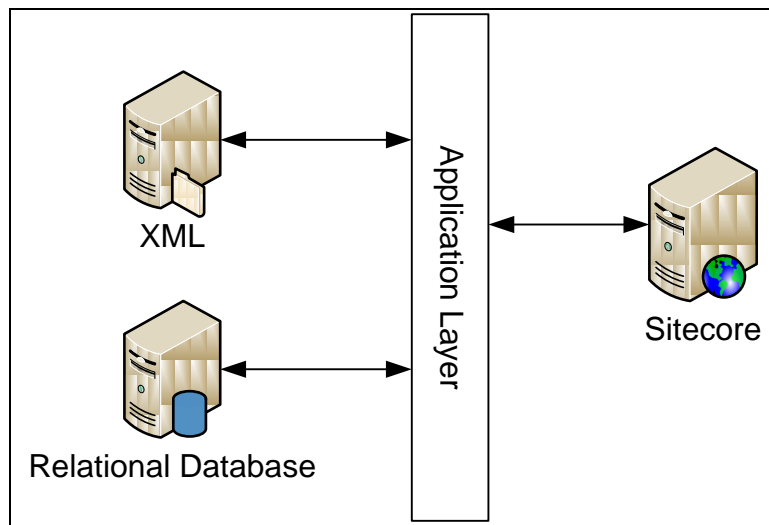


Figure 1. Sitecore connects to remote repositories through an application layer.

2.2 Communication Frameworks

Regardless of the repository in which you store data, your Sitecore components need to access that repository through either .NET APIs, web services or database access.

.NET APIs are the most convenient of any of these options, as there is little additional configuration required beyond referencing the proper assemblies and, possibly, adding entries to your web.config.

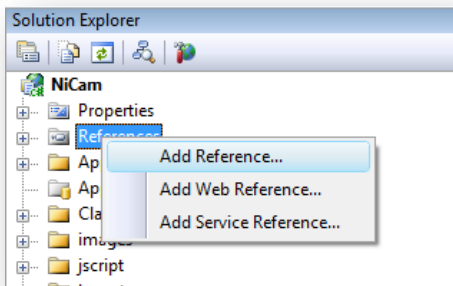


Figure 2. Adding a reference to a .NET assembly.

Database access is also straightforward, though the considerations in 2.1 should be kept in mind. The .NET Framework supports ADO.NET connections to Microsoft SQL Server, Oracle as well as ODBC-accessible repositories. Sitecore components such as Sublayouts (ASP.NET User Controls) or C# classes can be quickly developed to leverage the built-in capabilities of the .NET Framework.

Web services are a common solution when content repositories exist on separate physical hardware or are managed by Java-based applications (or other non-.NET systems). Microsoft has made it extremely easy to create web services using Visual Studio and developers can easily create components that access web services from external systems.

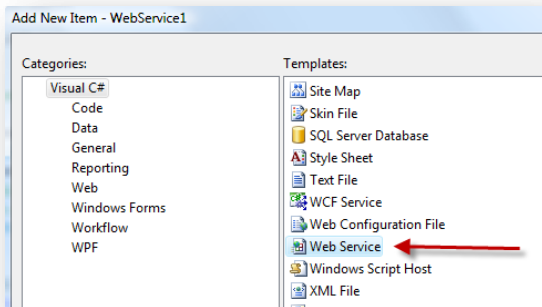


Figure 3. Creating a web service in Visual Studio.

2.3 Mechanisms for Integration

There are so many mechanisms for integration in Sitecore, it is difficult to describe them both comprehensively and at great depth. The following discussion will cover many of the common integration mechanisms in Sitecore and will point to detailed references on the Sitecore Developer Network ([SDN](#)) where applicable.

2.3.1 ASP.NET User Controls and Server Controls

Sitecore uses standard ASP.NET components such as User Controls and Server Controls in presenting page components on the public Web site. While Sitecore employs different terminology in describing these controls (User Controls are “Sublayouts”; Server Controls are “Renderings”), the underlying technology is identical to standard ASP.NET practices. User controls can be created in Visual Studio and can include a combination of declarative controls and code-behind. Server Controls (sometimes referred to as “Web Controls”) are created as classes in Visual Studio (such as a .cs file) and always inherit from `Sitecore.Web.WebControls.WebControl`.

A rendered page on the Web site includes a combination of controls. Each of these controls can render content from Sitecore or other systems using .NET calls in the code behind. In the following example, the rendered page includes a number of user controls and XSLTs:

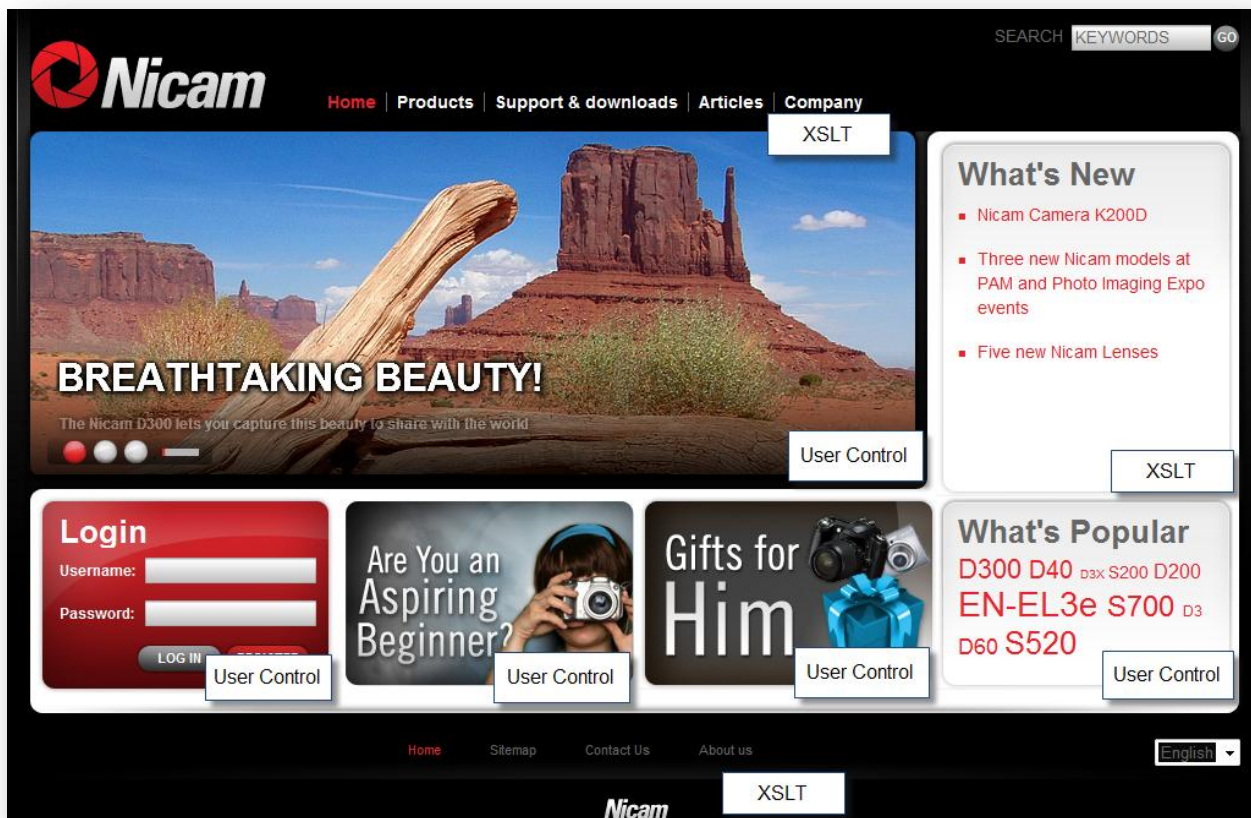


Figure 4. A page is composed of controls.

Each user control or XSLT can retrieve data from Sitecore or from an external repository. For example, the “What’s Popular” control can retrieve information from an ecommerce system and present popular items based on today’s sales.

Selected SDN References:
 Presentation Component Reference – <http://sdn.sitecore.net/Reference/Sitecore%206/Presentation%20Component%20Reference.aspx>
 Presentation Component Cookbook – <http://sdn.sitecore.net/Reference/Sitecore%206/Presentation%20Component%20Cookbook.aspx>

2.3.2 Data Provider

Sitecore works with a data abstraction model when accessing data. As a result, Sitecore can work with data stored in almost any repository. A standard Sitecore installation ships with data providers for SQL Server and Oracle. Sitecore also offers a Sharepoint data provider that allows Sharepoint lists and libraries to be accessed in Sitecore as though they were native Sitecore content.

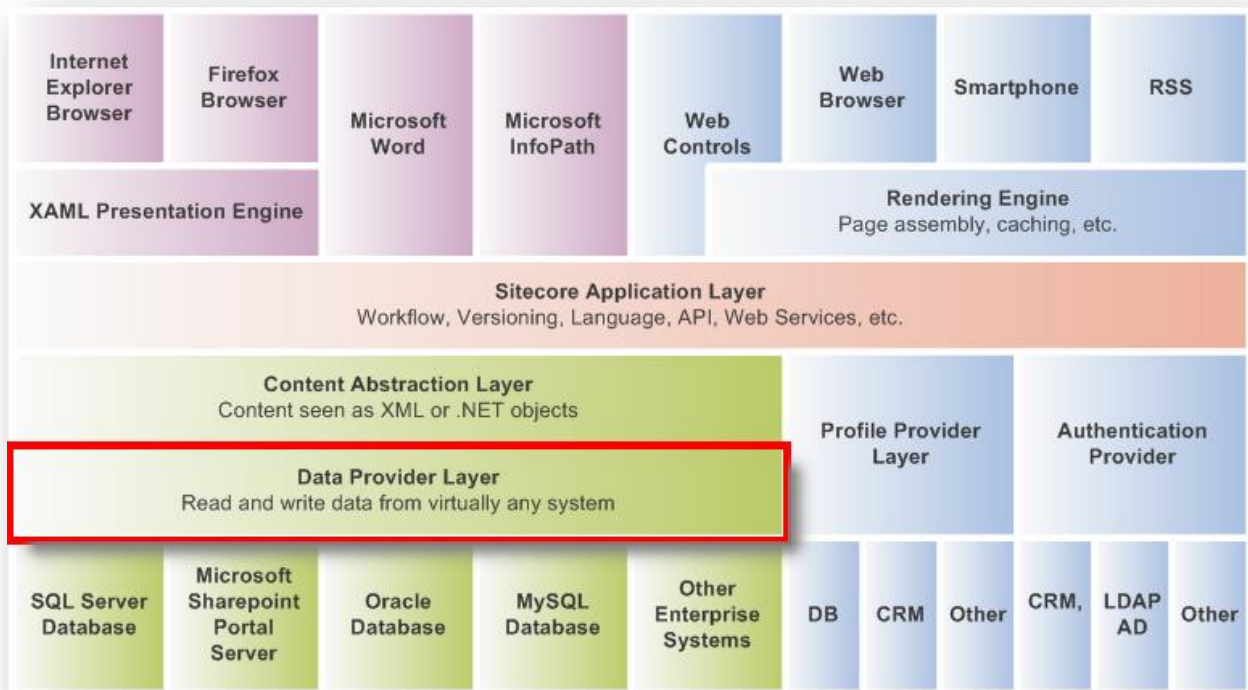


Figure 5. Sitecore's Data Provider Layer abstracts content storage.

In the diagram above, various clients – ranging from a standard Web browser to Microsoft Word – access content via the Sitecore Application Layer. The Application Layer, in turn, accesses the Content Abstraction Layer which addresses the underlying repositories via the Data Provider Layer.

Think of the Data Provider Layer as an interface that allows you to connect Sitecore to virtually any content repository. Developers create data providers by implementing several methods that define content types and content structure. Sitecore gives you the flexibility to implement a basic read-only data provider, a full-blown data provider that supports all of Sitecore's authoring features or anything in between.

In addition to the out-of-the-box SQL Server and Oracle implementations, Sitecore provides several data providers in its Shared Source Library. Sample providers are available for the SQL Server Northwind Database, XML sources and SOAP-accessible sources. Full source code is provided for you to review and customize.

Selected SDN References:

Integrating External Data Sources –

<http://sdn.sitecore.net/Developer/Integrating%20External%20Data%20Sources.aspx>

2.3.3 Pipelines

Pipelines are another avenue for integrating external data. A pipeline in Sitecore is a series of processing steps that the application executes when performing certain actions. In Sitecore 6, pipelines are executed in a number of circumstances, including the following:

- HTTP request is received
- Content is published
- Content is rendered
- Health monitoring statistics are generated
- Individual fields are rendered
- Search is executed
- UI actions are performed, such as item manipulation, image upload, etc.

In each of these pipelines, Sitecore executes processing steps that define the activities performed in the pipeline. For example, in the HTTP request pipeline, processors are executed that perform a number of important functions. These include:

Processor	Function
Item Resolver	Determines which item the user requests.
Language Resolver	Determines which language the user requests.
Device Resolver	Determines which device is making the request. For example, a mobile device, web browser or Flash movie could be initiating the request.
User Resolver	Determines which user sent the request.
Site Resolver	Determines which site the user requests.

Each of these processors is defined in a .NET class with a method called `Process()`. These processors are defined in the web.config. For example, the following defines the Item Resolver:

```
<processor type="Sitecore.Pipelines.HttpRequest.ItemResolver, Sitecore.Kernel" />
```

How does this relate to data integration? As a developer, you can define your own processors that participate in any of the Sitecore pipelines. To define a processor, create a class with the following method signature:

```
public void Process(Sitecore.Pipelines.HttpRequest.HttpRequestArgs args)
{
    // your code here
}
```

Depending on which pipeline you are customizing, the signature will look slightly different. The example above is appropriate when adding a processor to the HTTP request pipeline.

In a pipeline processor, you can execute whatever .NET code you would like. This includes code that accesses external applications. For example, when a content author uploads a new image to the Sitecore Media Library, you may want to push the image into an external Digital Asset Management system or Content Delivery Network. To accomplish this, you can add a processor to the uiUpload pipeline:

```
<uiUpload>
  <processor mode="on" type="Sitecore.Pipelines.Upload.CheckSize, Sitecore.Kernel" />
  <processor mode="on" type="Sitecore.Pipelines.Upload.ResolveFolder, Sitecore.Kernel" />
  <processor mode="on" type="Sitecore.Pipelines.Upload.Save, Sitecore.Kernel" />
  <processor mode="on" type="MyNamespace.MyClass, MyDll" />
  <processor mode="on" type="Sitecore.Pipelines.Upload.Done, Sitecore.Kernel" />
</uiUpload>
```

In this case, MyNamespace.MyClass contains the logic to push the uploaded file into the external system.

2.3.4 Events

Sitecore fires off events in a number of circumstances, typically those involving data manipulation. Developers can handle these events, cancel the events and even create their own events. Sitecore ships with approximately thirty events pre-defined. Here are some examples:

- item:adding
- item:added
- item:deleting
- item:deleted
- item:saving
- item:saved
- publish:begin
- publish:end
- security:loggingIn
- security:loggedIn
- security:loggingOut
- security:loggedOut

Events with the “-ing” suffix occur before data has been committed and can be cancelled. Events with the “-ed” suffix occur after the data has been committed.

As with pipeline processors, event handlers are defined in the web.config. Each event can have any number of handlers. You create your own handler with a class name, dll and method name.

```
<event name="item:saved">
  <handler type="Sitecore.Links.ItemEventHandler, Sitecore.Kernel" method="OnItemSaved" />
  <handler type="Sitecore.Tasks.ItemEventHandler, Sitecore.Kernel" method="OnItemSaved" />
  <handler type="Sitecore.Globalization.ItemEventHandler, Sitecore.Kernel" method="OnItemSaved" />
  <handler type="Sitecore.Rules.ItemEventHandler, Sitecore.Kernel" method="OnItemSaved" />
  <handler type="MyNamespace.MyClass, MyDll" method="MyMethod" />
</event>
```

You can use this event-driven model to integrate Sitecore with external systems. For example, imagine that you have unique auditing requirements. By default, Sitecore maintains an audit log of system access and data manipulation activities. But perhaps your organization has extended logging requirements or the log information needs to be written to a specific system for reporting purposes. By hooking in to the desired Sitecore events, you can write custom audit information to any log file, database system or application you choose.

Selected SDN References

Using Events – <http://sdn.sitecore.net/Articles/API/Using%20Events.aspx>

2.3.5 Commands

A command in Sitecore allows you to associate user interface actions with .NET code. With commands, users can easily click a button and fire off code that connects to an external system. For example, content authors may want to create a piece of content and associate it with a product SKU in your e-commerce system. Users could click a button in the content editor user interface that creates the association based on field values in the content item.

To understand this better, let's first take a look at the Ribbon in the Sitecore Content Editor. The Ribbon consists of multiple tabs, each of which contains commands and command groupings (a.k.a. "chunks").

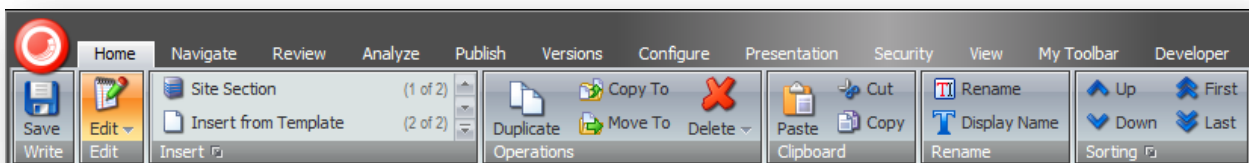


Figure 6. The Content Editor Ribbon with Commands.

The functionality for each of these buttons is defined in the core database and associated with a Sitecore Command. The commands themselves are defined in the Commands.config file, for example:

```
<command name="item:addversion"
  type="Sitecore.Shell.Framework.Commands.AddVersion, Sitecore.Kernel" />
```

To implement a command, your code should minimally include an Execute() method with your desired functionality. As in the other examples we have seen so far, you can use your code to access external applications, databases or web services.

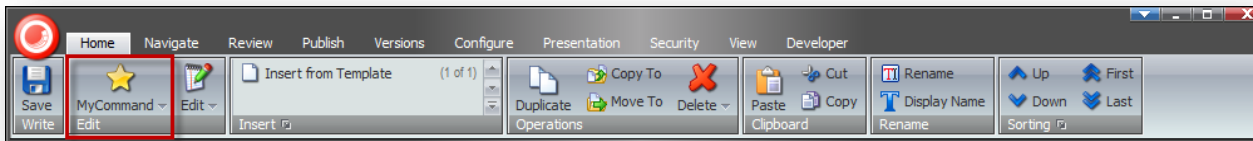


Figure 7. Ribbon with custom Command.

2.3.6 Custom Editors

A custom editor in Sitecore is a way to create an alternate editing interface in the Sitecore Content Editor. These alternate editing interfaces can be used to search external repositories, import external data or even access external administrative tools.

To understand a custom editor, let's first take a look at the standard Sitecore editor. When a user clicks on a content item in the content tree, Sitecore presents the standard editor. The standard editor represents content items as a set of structured fields. For example, in the screenshot below, the COOLPIX S700 stores data in several fields, including Image, Title, Price and Short Description. These fields are not standard *per se*; rather, each content type can have its own set of fields that best describe the data for that content type. For example, it makes sense for a camera to have a price and for a news story to have a release date. A camera and a news story are based on different underlying content types and, as such, may have completely different fields.

In Sitecore, authors typically edit content by updating fields rather than editing the underlying HTML in a page. This structured approach allows for content reuse and repurposing and allows for the enforcement of branding and usability rules on a site. This focuses authors on content rather than on graphic design or HTML editing.



Figure 8. The standard Sitecore editor.

So, what is a custom editor? A custom editor is an interface that does not present the standard structured fields of a content item. There are several examples of these in the out-of-the-box user interfaces. For example, the Media Library uses a custom editor that provides a visualization of media items along with the ability to upload new items.

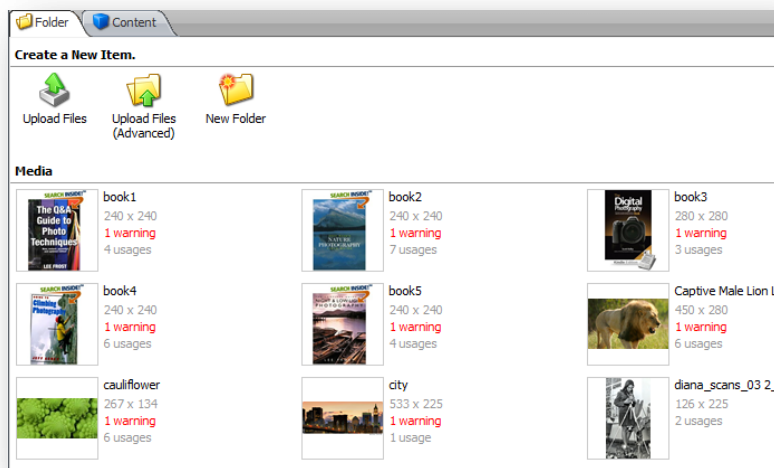


Figure 9. The Media Library editor.

Sitecore allows you to create custom editors and associate them with content items or content types. A Custom Editor is simply an ASPX file or XAML control that includes whatever functionality you would like. The Editor is created in Visual Studio, then referenced in the Core database. This makes the editor available in the Content Editor user interface.

An example of how this can be used is the case where you want to simplify the incremental import of content from another system. A developer can create a custom editor that allows the user to search the external system and select an item for import. A custom editor could also be used to expose administrative tools from external systems within the Content Editor. Any functionality you can define in .NET code can be created in a custom editor.

Selected SDN References:

Client Configuration Cookbook –

<http://sdn.sitecore.net/Reference/Sitecore%206/Client%20Configuration%20Cookbook.aspx>

2.3.7 Workflow Actions

In Sitecore, content items move through a workflow process prior to being made available on the published site. Workflows are organized conceptually into States, Commands and Actions.

Each content item is associated with a workflow state. A workflow state describes the work status of an item, such as Editing, Approval, Marketing Review, Translation, etc. Items move through workflow states on their way to being approved for publication. Most workflow states are non-final states, meaning that authors cannot publish content in these workflow states. When a content item reaches a final state, it is publishable.

Items are moved from one state to another using workflow commands. A workflow command appears in one of several locations in the Sitecore user interfaces. For example, commands appear in the Sitecore Workbox as well as the ribbon in the Content Editor.

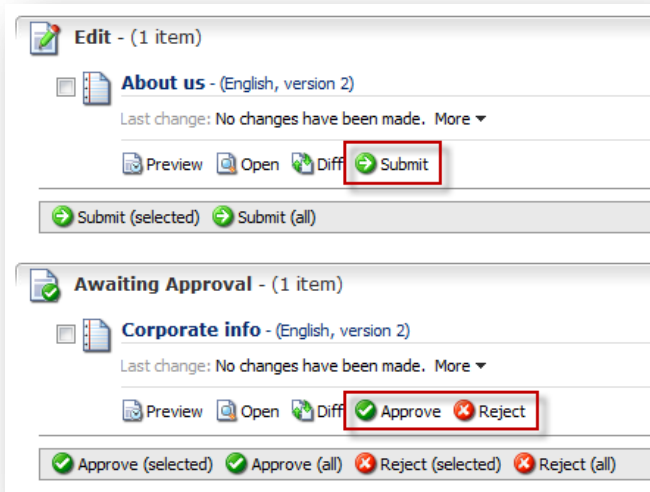


Figure 10. Workflow commands in the Sitecore Workbox.

You can integrate external data into Sitecore workflows using workflow actions. An action is .NET code that can be fired off upon a workflow state transition or the execution of a command. For example, when a user clicks the “Approve” command in the above screenshot, a workflow action can check the translation status of the item in an external system and cancel the command if the translation has not been completed.

Workflow commands are coded in Visual Studio and configured in the Sitecore user interfaces. As an example, Sitecore's out-of-the-box publishing action allows content to be published automatically when a content item enters a final workflow state. In the screenshot below, the AutoPublish action is defined using a class name and .dll. Note that Sitecore can pass parameters to your code when the workflow action is executed:



Figure 11. Defining a workflow action.

Selected SDN References:

Workflow Reference – <http://sdn.sitecore.net/Reference/Sitecore%206/Workflow%20Reference.aspx>

Workflow Cookbook – <http://sdn.sitecore.net/Reference/Sitecore%206/Workflow%20Cookbook.aspx>

2.3.8 Custom Validators

Sitecore ships with a fully-featured and fully pluggable validation engine. The validation engine can be activated at the item level, field level or page level. Sitecore ships with many out-of-the-box validators to test for broken links, W3C XHTML validation, ALT text for images, etc. Developers can create their own validators to apply validation rules based on data in other systems.

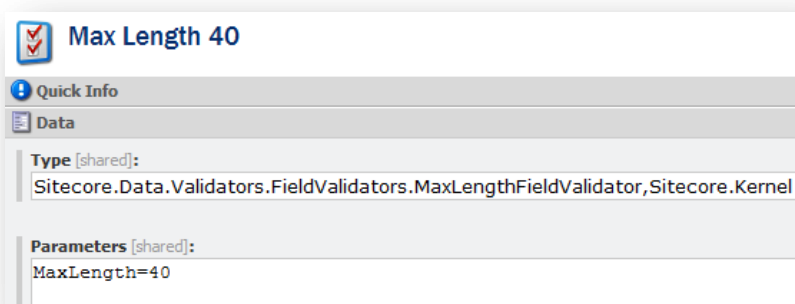


Figure 12. The Max Length validation rule.

Validation rules are defined in the Content Editor by providing a class name, dll and optional parameters. Your class should have an `Evaluate()` method that applies the required business rules, error message

and error level. Sitecore will provide visual feedback in the user interface to alert the user that a validation rule has been violated.

When will custom validators need to access an external system? The W3C validation is a good example. Sitecore uses a validator to send the data from a rendered page or an individual rich text field to an application that validates the XHTML content against the appropriate W3C schema. Similarly, you can use the validation engine to validate content against your own compliance standards that may be managed by non-Sitecore tools.

Selected SDN References:

Client Configuration Cookbook –

<http://sdn.sitecore.net/Reference/Sitecore%206/Client%20Configuration%20Cookbook.aspx>

2.3.9 Sitecore Modules

Sitecore provides several pre-built connectors to external systems. This section describes one of them. You can find more information on our pre-built connectors in the Products section of the SDN site (<http://sdn.sitecore.net/Products.aspx>) and in our Shared Source library (<http://trac.sitecore.net/Index>).

SharePoint Connector

The SharePoint Connector creates a connection between Sitecore and your SharePoint repository. Sharepoint lists and libraries can be represented as native Sitecore content. In essence, the SharePoint Connector is a data provider (see section 2.3.2) that provides bi-directional connectivity to SharePoint. The SharePoint Connector ships with an administrative interface that allows administrators to create connections to SharePoint and map fields in SharePoint list items to fields in Sitecore content types.

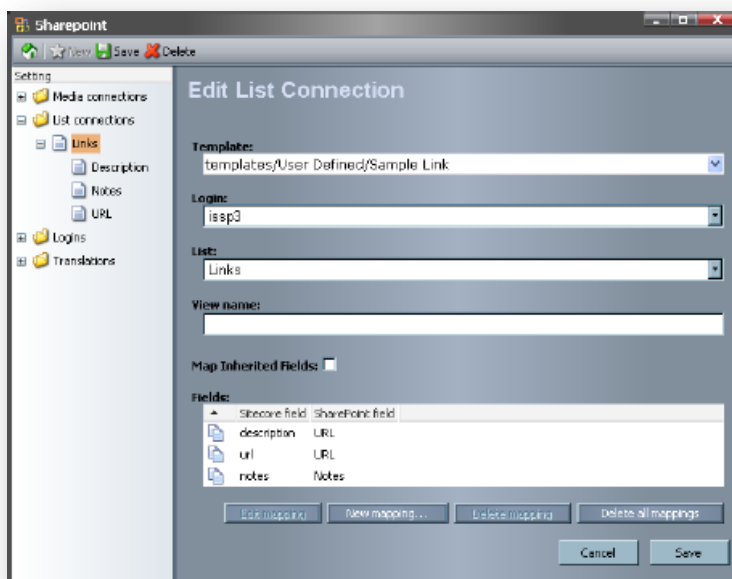


Figure 13. Administration screen for SharePoint Connector.

A typical usage scenario for this connector is when content authors make use of SharePoint's collaborative tools to author product manuals, spec sheets or marketing collateral. The content author may want the content to be accessible on the public Web site, but exposing the SharePoint instance to the public internet is not an option from a security and performance perspective. Sitecore allows those SharePoint assets to be represented as native Sitecore assets in the authoring environment and then published to the delivery environment without exposing SharePoint in the DMZ.

2.4 Integration Scenarios

To further illustrate the content integration possibilities with Sitecore, let's look at some common scenarios. This section divides these scenarios into those that involve the delivery environment (i.e. the published site) and those that involve the authoring environment.

2.4.1 Delivery Environment

Display, Insert or Update Content-Independent External Data

The term “content-independent” describes scenarios where content should be displayed on the page regardless of the Sitecore content item that has been requested. For example, a newsletter signup control might appear on the same page as a news article. The contents of the control are essentially independent of the news article, meaning that the text, images and functionality in the control don't vary based on which news article is selected.

For these scenarios, the best approach is to create a .NET user control or server control that accesses the external repository. For example, the newsletter signup control may call a web service that processes the submitted information. Though the newsletter subscription information is managed in an external system, you may actually be leveraging Sitecore content on your sign-up form. For example, if you want to make the form fields multilingual, you can save the form values in the Sitecore Dictionary and use those values in `<asp:Label>` controls. When the form is submitted, rather than saving the information to Sitecore, you can send the information to your email service provider.

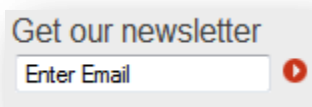


Figure 14. Example of a Newsletter Signup form.

This makes transitioning to Sitecore extremely easy for ASP.NET developers. If you have existing ASP.NET applications that are coded modularly, it is very likely that your existing user controls will be plug-and-play with Sitecore.

Display Content-Dependent External Data

Content-dependent data is stored in an external system and explicitly related to Sitecore content. A common example of this is integration with an e-commerce system. Typically, e-commerce implementations involve a combination of Sitecore content and e-commerce content. Product descriptions and images, for example, may reside in Sitecore while product SKUs and inventory are stored in the e-commerce system.

A simple way to address this requirement is to create a mapping between the Sitecore content item and the product ID in the e-commerce system. You can store the product ID as a Sitecore field value in a related content item.



Figure 15. Product ID from commerce system stored in Sitecore.

The Sitecore field is typically a read-only field that is hidden from business users (using Field Security settings). The field can be populated manually, through a Custom Editor or by using a Command or Command Template that fires off when the content item is created. Another alternative would be to leverage the Sitecore Wizard Framework that allows the user to search for products or create a new product in the e-commerce system.

For example, the user creates a new Product and the system presents a search screen. The user searches for a product in the e-commerce system and selects it. The system then creates a new content item and auto-populates the Product ID field with the product ID from the e-commerce solution.

An alternative approach would be to store the mappings between Sitecore items and e-commerce items in a mapping table. The mapping table can store the Sitecore item ID along with its related product ID.

To render the composite content on the site, your code will retrieve the product ID from the Sitecore repository and use that ID to retrieve pricing and inventory information from your e-commerce system.

Display Pages Not Processed By Sitecore

In some circumstances, you may want to display pages on your site that are not processed by Sitecore. There are two approaches to this: creating subdirectories with standard ASP.NET web forms or using a reverse proxy to route requests to appropriate application servers.

Let's consider the subdirectory scenario first. When a request comes in to IIS, Sitecore will look for the requested item in the content tree. If the system does not find the item in the content tree, it will look on the file system for the corresponding directory structure. If it finds a matching file, it will process the file and send an HTTP response. You can also explicitly point out directories for Sitecore to *ignore*. In this case, Sitecore will simply pass off requests to ASP.NET to handle through its standard page processing model. These directories (or even individual files) are specified in the `IgnoreUrlPrefixes` setting in the `web.config`.

```
<!-- IGNORE URLS
    Set IgnoreUrlPrefixes to a '|' separated list of url prefixes that should not be
    regarded and processed as friendly urls.
-->
<setting name="IgnoreUrlPrefixes"
    value="/sitecore/default.aspx/trace.axd/webresource.axd/sitecore/admin/upgrade/" />
```

Note that these values can be virtual directories or actual directories under the web root.

Sometimes, however the requirements are more complicated. If there is a section of your site that is managed by an entirely different server, you will need to use a reverse proxy approach. This can be useful when separate application servers (for example, Java-based applications) manage some sections of the site while Sitecore manages other sections of the site. A reverse proxy allows you to map URL paths to particular servers. So, for example, requests to `http://mysite.com/checkmybalance` get routed to a Java-based system while all other requests get routed to the Sitecore server. Reverse proxies can be set up in most load balancers and can also be configured as a separate server.

2.4.2 Authoring Environment

Use External Content as Metadata or Related Items

In this scenario, the user associates Sitecore content with content stored in an external system. For example, imagine you have a content item that you want to associate with related content stored in an external repository. There are many ways to accomplish this. One approach is to create a read-only data provider that connects to the external repository. The data provider items can be represented in a multilist or treelist control:

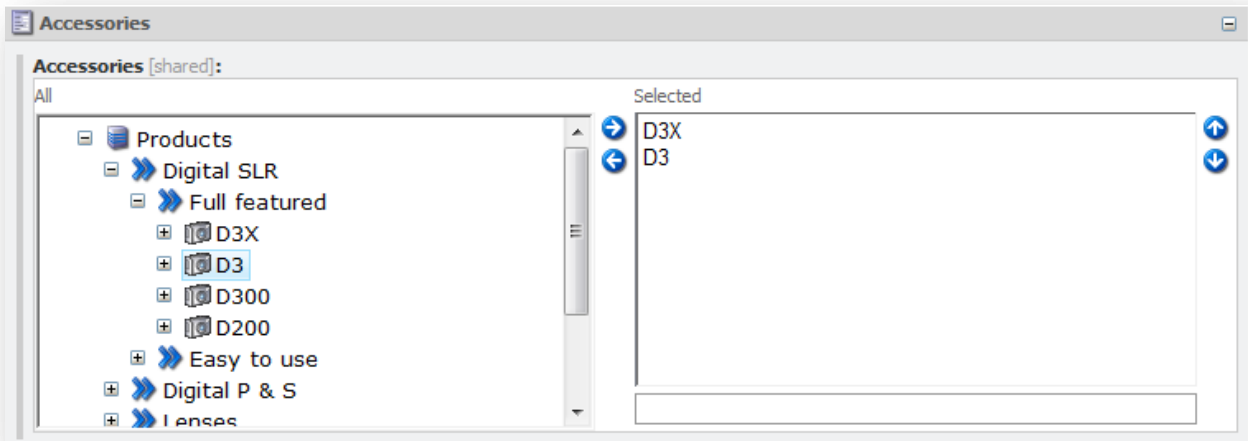


Figure 16. Selecting related items using a Data Provider.

This allows the user to select items from the external repository and associate them with the current item in the Content Editor (or split-screen mode).

Update External Systems Based on Sitecore Events

As mentioned earlier, Sitecore fires off events under a variety of circumstances, including all circumstances where data is manipulated. These events are fired off both from user interface actions as well as scripted actions. (Note that Sitecore provides pipelines to handle events specific to user interface actions.)

An example of this could be in the handling of the `publish:end` event. This event gets fired off when a publish operation has completed. When `publish:end` occurs, you may want to have your search engine re-spider your site. This will update your search index and allow public visitors to search for the new content.

Alternatively, you may want to use `publish:itemProcessed` to have your search engine incrementally update its index with the content that has been published. This event fires off every time an individual item moves through the publish pipeline.

In either case, you can hook in to the appropriate event and use, for example, web services to alert your search tool to index the desired content.

Change Sitecore Behavior Based on External Data

An example of changing Sitecore behavior based on external data could be in device detection. By default, Sitecore provides several mechanisms for detecting devices that access your site. These include domain, subdomain, query string and browser agent string. In some circumstances, however, you might want more granular control over how device detection works. This can be accomplished by overriding the

DeviceResolver in the HTTP request pipeline. The default device resolver is defined in the web.config:

```
<processor type="Sitecore.Pipelines.HttpRequest.DeviceResolver, Sitecore.Kernel" />
```

By removing the processor and replacing it with your own, you can apply custom device detection logic. This is the approach taken by Creuna, a Sitecore partner that has compiled a comprehensive database of hundreds of browser agent strings. Using Creuna Mobile Detect, Sitecore can apply unique presentation settings to any known mobile device.

Selected Sitecore.net Resources:

Mobile Detect -- <http://www.sitecore.com/en/News/NewsAndEvents/2008/9/Creuna-Mobile-Detect.aspx>

Chapter 3

Media Integration

Sitecore ships with a full-featured Media Library that supports all standard media file formats. In Sitecore, “media” is an umbrella term to describe any file that can be uploaded into the system. This includes images, Microsoft Word documents, Flash SWFs, Quicktime videos and more. Media can be multi-lingual, versioned and associated with any number of metadata attributes. Security permissions can be applied to image assets for both content authors and Web site visitors, ensuring that access rights are maintained for multiple audiences.

Sitecore also supports time-saving features like dynamic image resizing. This allows content authors to upload a high-resolution image and have it resized on-the-fly based on the required form factor for any given page. For example, content authors can upload a 1000x1000 image that may be presented as a 100x100 thumbnail on the homepage and a 300x300 image on a detail page. The full-sized, high-resolution version can be made available on a download page (for example a Press Room page).

The Media Library has many more features, but it is not the perfect match for all scenarios. For example, while users can upload video files into the Media Library, Sitecore is not optimized for streaming media. Full-fledged streaming media servers (such as Windows Media Server or Real Helix Server) provide functionality for bandwidth detection and dynamic stream adjustment, multicast capability and support for massive scalability scenarios for real-time broadcasts. These features fall outside of the scope of what content management systems offer.

Another way in which the Media Library may not always be appropriate is when customers want to leverage their enterprise Digital Asset Management (DAM) repository. A DAM can house digital assets across the enterprise for everything from marketing campaigns to print publications to Web site presentation. Strategically, organizations may prefer to use their existing DAM as the primary repository for media assets to simplify both the business user experience (i.e. a single user interface with a standardized metadata taxonomy) and system maintenance (i.e. storage and backup). Storing some assets in the DAM and other assets in the Sitecore Media Library may create confusion among business users when performing searches or uploading new media items.

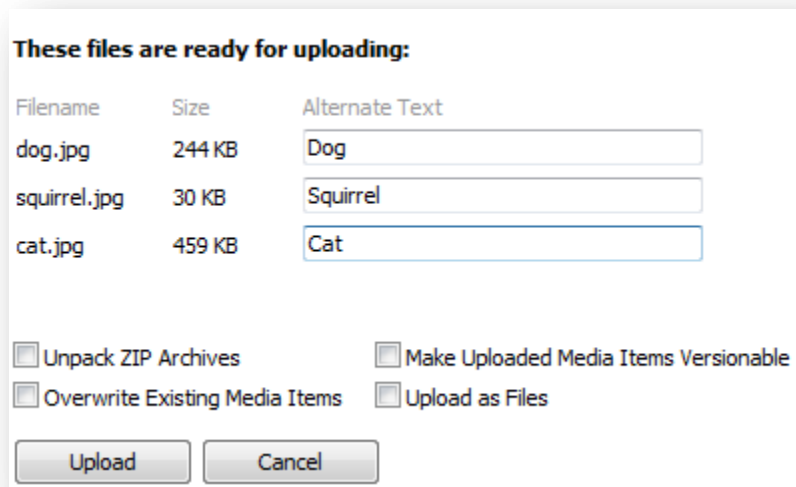
A final situation where the Media Library may not be sufficient on its own is when media must be served from locations that are geographically close to site visitors for performance and scalability purposes. Content Delivery Networks (CDNs) provide access to edge servers across the globe. These servers improve the end user experience by providing a more direct route for network traffic to requesting clients. The use of a CDN

does not preclude the use of the Sitecore Media Library. However, organizations will need to architect their Sitecore solution to support the use of the CDN.

In each of these circumstances, it may be preferable to integrate an external media repository with Sitecore's Media Library rather than making Sitecore the authoritative or exclusive system. Integration of an external media store can mean one of several things in a typical implementation:

1. All media is stored in an external system.
2. The external system is the authoritative source for media, but selected assets are imported into Sitecore.
3. Sitecore is the authoritative source for media, but selected assets are exported to an external system.

In approach 1, when all media is stored in an external system, the primary integration considerations involve the management of media and the creation of media links. The management of media refers to the upload, versioning and maintenance of media. Customers must consider which, if any, of these activities should occur using the Sitecore user interfaces. For example, should the standard Sitecore upload user interface be used to upload media into the external system? Or will all uploads occur via the user interfaces of the external system?



Filename	Size	Alternate Text
dog.jpg	244 KB	Dog
squirrel.jpg	30 KB	Squirrel
cat.jpg	459 KB	Cat

Unpack ZIP Archives Make Uploaded Media Items Versionable
 Overwrite Existing Media Items Upload as Files

Upload Cancel

Figure 17. Standard Sitecore upload interface.

Regarding the creation of media links, consider the scenario where a content author is populating an image field. When media is stored in the Media Library, users access the Media Browser to select a media item.

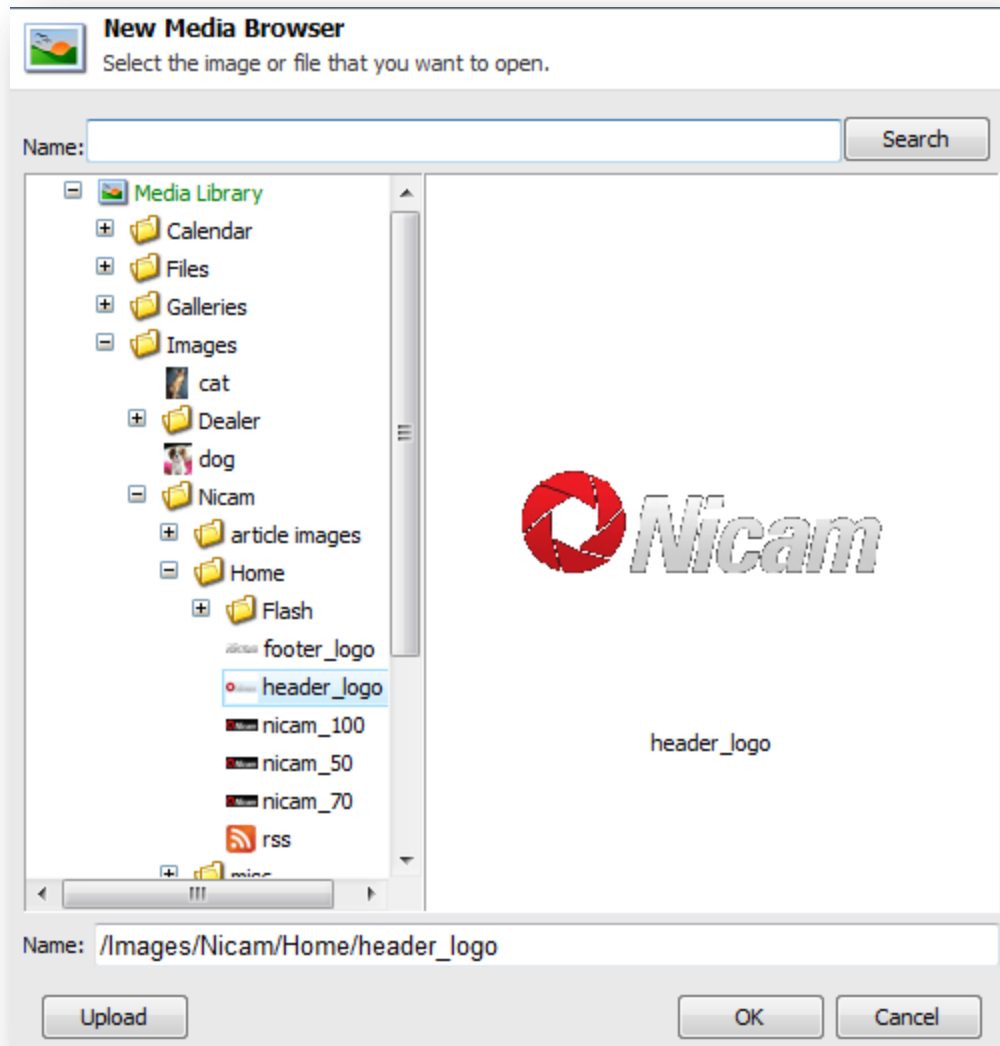


Figure 18. Media Browser user interface.

When integrating with an external media repository, content authors may wish to use this standard user interface. Alternatively, they may simply copy and paste a link to the appropriate file location.

In addition, consideration will need to be given to the “Insert Sitecore Media” option in the rich text editor. Should content authors use the standard Sitecore user interface, a custom interface or will they just copy and paste the appropriate path as the source for the image?

In approach 2, listed above, media is stored in an external system and selectively imported into Sitecore. The selective import approach gives content authors the flexibility to pull web-specific assets into the CMS and manage them separately. A DAM, for example, may have PSDs, TIFFs and other image types that are not typically found on a

Web site. Content authors can import just the JPGs, GIFs, PDFs and other media types that they plan to use in the Web context. In this case, content authors will require a Sitecore interface to search assets in the external repository and select items to import into Sitecore's Media Library.

Something of a hybrid between integration approaches 1 and 2 is found with Sitecore's SharePoint Connector. The SharePoint Connector allows either a live or staged view of SharePoint Document Libraries. In the live view, Sitecore directly queries the Sharepoint repository when accessing media assets. This is necessary in circumstances where up-to-the-minute access to SharePoint resources is required. This is similar to approach 1. In the staged view, SharePoint assets are imported into Sitecore's Media Library and a live connection with the SharePoint repository is not maintained. This reflects approach 2.

An additional circumstance comes into play with the SharePoint Connector. Some customers may prefer the live view in the authoring environment and a staged view in the delivery environment. The SharePoint server is often inaccessible in the DMZ and, even if it were accessible, should not be exposed to the traffic levels found on a busy website.

To address this situation, Sitecore can publish actual Media Library items based on the information in the virtual SharePoint items. As part of the standard publish process, Sitecore converts the live view of the SharePoint item to an actual Media Library item that can be served from Sitecore like any other Media Library asset. This approach also allows the SharePoint instance to remain shielded from public access.

With integration approach 3, Sitecore is the authoritative system for media and a content export utility is required. An example of this is when using a CDN that requires the upload of content assets to an FTP server. When a content author uploads a media item into Sitecore, the same media item should be uploaded to the CDN. Another option could be to hook into the publishing or workflow processes and upload items when they have been approved for publication.

The remainder of this chapter will discuss media integration in more depth, looking at the requirement from the perspective of typical repositories, mechanisms for integration and integration scenarios. The discussion will focus on several possible integration mechanisms; however, the options presented are not intended to be comprehensive. Please review the [Content Integration](#) section of this document for additional alternatives.

3.1 Repositories

Digital assets are stored in databases, file systems or “in the cloud.”

DAMs store media assets either in a database or on a file system. In addition to the asset storage, DAMs provide a solution for maintaining metadata and indexing assets. The DAM provides a user interface that allows contributors to upload assets, describe assets and search for assets. The DAM also provides maintenance utilities for disaster recovery. Some DAMs also provide utilities for repurposing content for the web. This can be as simple as using a standard protocol (such as FTP) to move assets from one server to another or can involve additional features such as image compression for web presentation.

Media servers (such as Windows Media Services or Real Networks Helix Server) store files on a file system. The system intelligence (such as bandwidth detection and multicasting capabilities) is built in to the server itself and is not a function of the storage medium. Additional capabilities can be built in to your media files (such as special encoding), but standard file system storage is the norm.

Edge networks, such as Akamai, serve media content from servers located geographically proximate to website visitors. This provides high performance and scalability across the globe. Depending on your arrangement with the CDN, you may upload media assets to the provider via FTP or you may rely on the provider to monitor your site to identify content that should be cached.

3.2 Mechanisms for Integration

3.2.1 Media Browser

An obvious UI touch point for integration is the Media Browser. The Media Browser is a user interface that Sitecore presents to content authors when they populate a media field (such as an Image field). In the following example, Banner Image is an image field.

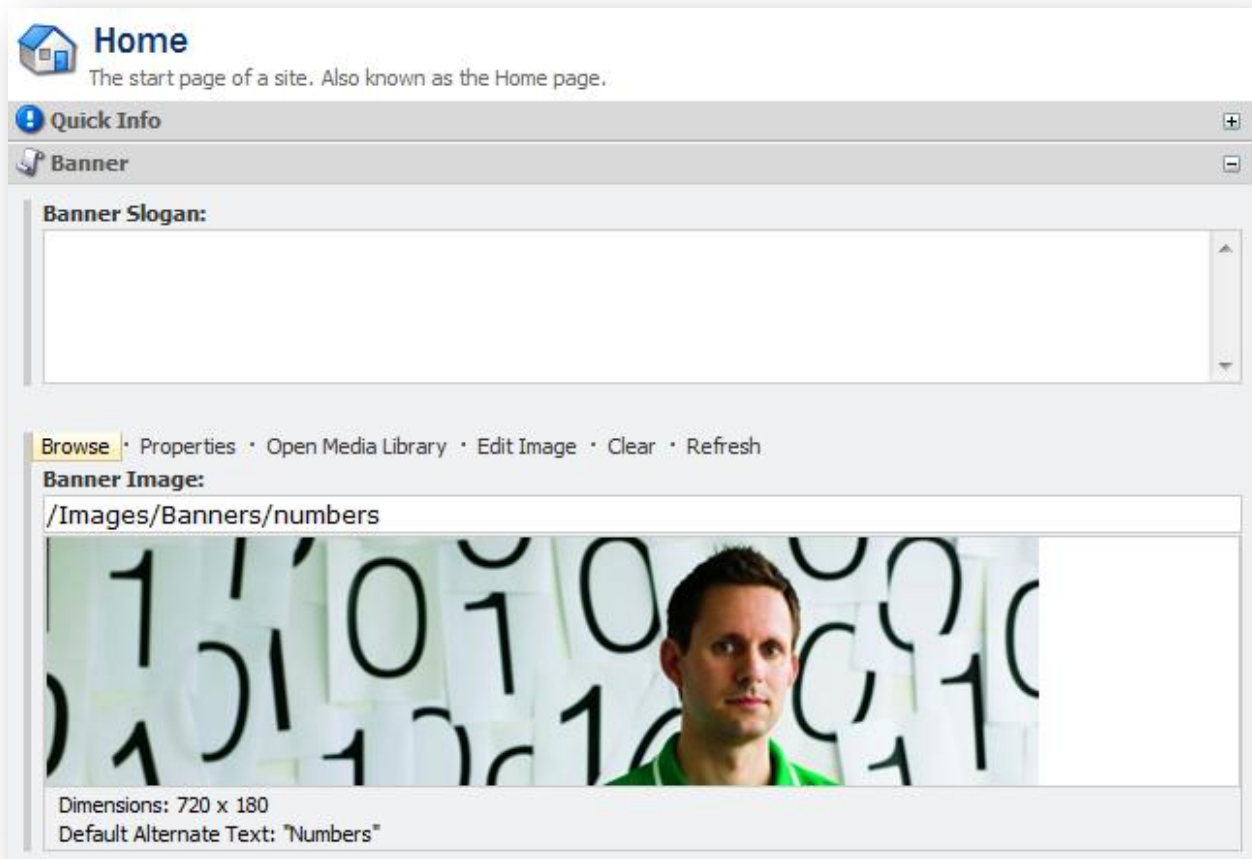


Figure 19. Banner Image field with image.

To select a new image, a content author clicks on the “Browse” button and the Media Browser pops up. The Media Browser allows the content author to select a new image from the Media Library.

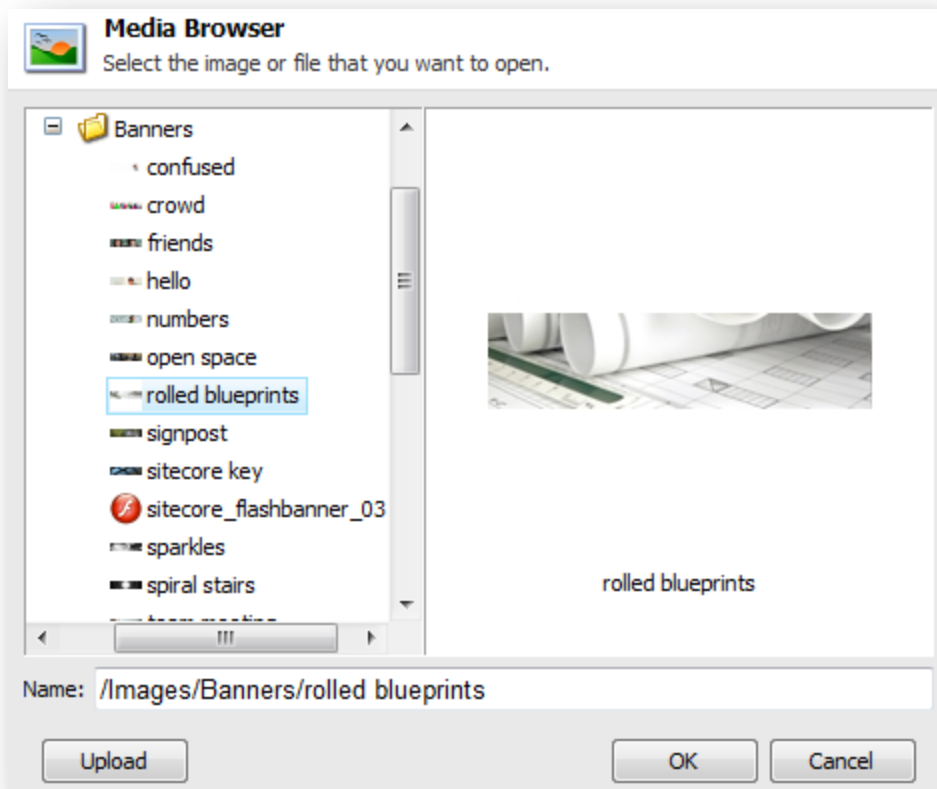


Figure 20. Selecting an image using the Media Browser.

The Media Browser is a simple XAML application that allows users to browse the Sitecore Media Library. You can modify the Media Browser to search your own repository and populate Sitecore media fields with the data of your choosing. For example, when the content author opens the Media Browser, the system can present a search box that allows them to search based on DAM metadata.

As a developer, you accomplish this by modifying the declarative code of the Media Browser and the code beside. The declarative code can be found in the Media Browser.xml file located in the standard Sitecore installation. Modifying this file allows you to change the appearance of the Media Browser. The functionality of the Media Browser is in

`Sitecore.Shell.Applications.Media.MediaBrowser.MediaBrowserForm`. By overriding methods in this class, you can change the functionality of the Media Browser to suit your needs.

Note that the Media Browser itself is just one example of a search form. You could also create the same functionality using a Custom Editor, a Wizard or your own XAML control.

3.2.2 Text Fields

Sometimes, a simple solution is the best solution. That may be the case if you use a text field to store the path to non-Sitecore media Items. In this case, you can have a field called “Media File” that stores a valid URL for an external media item. Your presentation logic can take the field value and render it in the context of an `` tag.

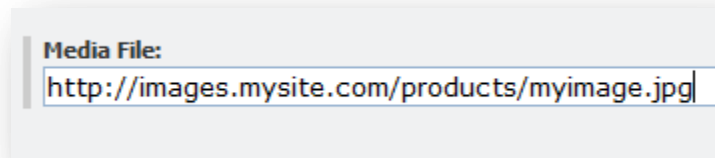


Figure 21. A simple text field used to store an image URL.

The advantage of this approach is that it involves no coding and can be implemented in about ten minutes. The disadvantage is that it provides no convenient UI to search or browse media and it doesn't offer media preview functionality. In some scenarios, however, those additional features will not matter and this very basic approach will address the requirement.

3.2.3 UI Upload Pipeline

The `uiUpload` pipeline is executed when a user uploads a file using the Sitecore user interface. This pipeline is useful when you want to intercept upload activity and push the uploaded files to a separate server. The pipeline is defined in the `web.config` and consists of several processing steps.

```
<uiUpload>
  <processor mode="on" type="Sitecore.Pipelines.Upload.CheckSize, Sitecore.Kernel" />
  <processor mode="on" type="Sitecore.Pipelines.Upload.ResolveFolder, Sitecore.Kernel" />
  <processor mode="on" type="Sitecore.Pipelines.Upload.Save, Sitecore.Kernel" />
  <processor mode="on" type="Sitecore.Pipelines.Upload.Done, Sitecore.Kernel" />
</uiUpload>
```

As discussed earlier in this document, adding a pipeline processor in Sitecore is straightforward. Simply add a processor to the list above, specifying your namespace, class and assembly. Your class should have a method called `Process()` that initiates your logic. The `Process()` method should have the following signature:

```
public void Process(UploadArgs args)
{
}
```

`UploadArgs` provides access to all the information you need about the upload. Most significantly, an `HttpFileCollection` object provides access to the file(s) that are being uploaded.

3.2.4 Workflow Actions

Workflow actions can be used to apply programmatic functionality to the workflow process. See the above discussion of [workflow actions](#) for a description of this standard Sitecore feature.

Workflow actions have a unique role in media integration scenarios, particularly with CDN integration. For example, consider a user creating a news article using the Sitecore user interfaces. The news article will likely have a title, descriptive text and metadata fields. The article may also have associated images that are uploaded when the article is created. If you make use of the `uiUpload` pipeline described above, the image will be uploaded to the CDN prior to the related content being approved through the workflow process. This may or may not be acceptable. If it is not, workflow actions provide a useful alternative. Instead of making use of the `uiUpload` pipeline, use a workflow action to identify all of the images

referenced in the news article. These related images only be uploaded to the CDN when the news story itself is approved for publication.

3.2.5 Data Provider

Data providers are described above in the content integration section. An additional mention is relevant here when integrating external media sources. Developers can implement data providers that represent external media within the Sitecore Media Library. From Sitecore's perspective, implementing a media data provider is no different from implementing a data provider for standard content. Media in Sitecore is simply another instance of a content item with structured fields. The media asset itself is represented in a blob field type that is implemented by the data provider.

In the example below, the YouTube data provider (available in Sitecore's Shared Source Library) treats YouTube videos as though they were native Media Library items.

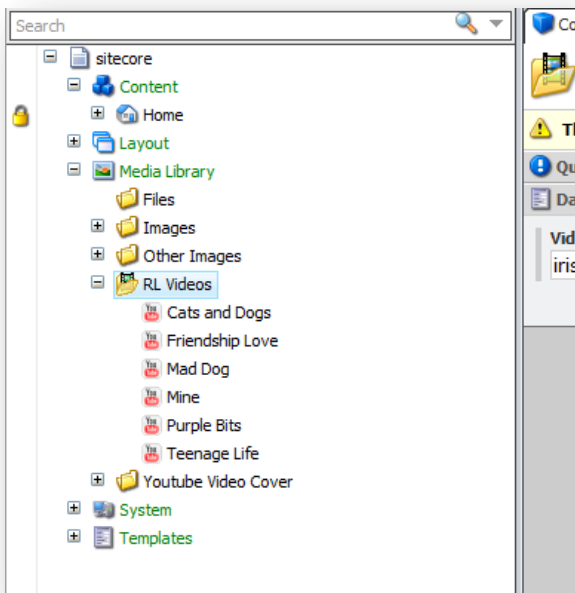


Figure 22. Media Library with YouTube Data Provider.

To create this data provider, the developer used YouTube's web service API and Sitecore's data provider interface to allow content authors to insert YouTube videos into their web pages.

Selected SDN References:

YouTube Data Provider –
<http://trac.sitecore.net/YouTubeIntegration>

3.3 Integration Scenarios

As with the content integration discussion, we will consider in turn media integration scenarios that involve the delivery environment (i.e. the published site) and those that involve the authoring environment.

3.3.1 Delivery Environment

Display External Media

Simply displaying external media on your published site is quite simple. There are three elements to the problem: media storage, media referencing and media rendering. As we have discussed previously, your media can be stored in any number of repositories from dedicated media servers to a DAM to a CDN. The stored media needs to be referenced somewhere, either in Rich Text fields, text fields or even in your .NET user controls. Finally, media needs to be rendered in a visitor's Web browser by sending an tag in the HTTP response.

In general, displaying external media is a simple scenario. Web browsers natively support the tag and its "src" attribute. If the path to an image is stored in a text field, you will need to use .NET code to retrieve the path and dynamically populate the image source attribute. If the path to an image is in a Rich Text field, the tag is already present in the content and no additional processing is needed.

Other types of media, of course, will require other HTML markup. Embedding a Quicktime video will require different code than embedding a Flash movie or a Windows Media clip. Regardless of the media type in question, however, we are still in the realm of standard HTML markup that will be used to tell the browser how to present the media content.

3.3.2 Authoring Environment

Upload Media to Content Delivery Network

To upload media to a CDN, there are a number of possibilities. First, different CDNs and even different packages with a single CDN work differently in regards to how media is provided to the CDN. With some systems, an explicit upload is required via FTP. With other packages, the CDN may scan your servers for content and ingest new media automatically.

If an explicit upload process is required, you can either hook in to the uiUpload pipeline or the workflow process to push the media item from the Sitecore server to the CDN server (see 3.2.3 and 3.2.4).

If no upload process is required, simply publishing the media and linking to it on your site is sufficient for the CDN to pull the asset into its network.

Search and Select Media from Digital Asset Management System or File System

If you want users to select external media using the Sitecore user interfaces, you may either customize the Media Browser or implement a data provider. This is useful in scenarios where you would like content authors to populate image fields in Sitecore using the standard Sitecore user interfaces.

A Media Browser customization is useful when you want to take advantage of search features built in to your DAM or file-system search engine. The interface can allow users to search across rich metadata to find potential images or other media assets that may be appropriate for the content at hand. The path to the desired media asset can be stored in an image field or text field.

A data provider can be useful when you want to represent external media as though it is native Sitecore media. In this case, no Media Browser customization is required and you can use standard Sitecore image fields, for example, when selecting media.

Chapter 4

Security Integration

Sitecore has a built-in system for managing users and roles across any number of security domains.

Sitecore manages authentication and authorization for system users in both the authoring and delivery environments. A user in the authoring environment accesses the system with the intention of authoring, editing or approving content. A user in the delivery environment accesses the system with the intention of accessing Web site content or applications that are only provided to registered users.

A Sitecore user can belong to any number of roles; similarly, a Sitecore role can belong to any number of roles. This “roles in roles” functionality allows for abstraction and simplification of user maintenance.

Sitecore supports two kinds of roles: content roles and user interface roles (a.k.a. Sitecore Client roles). Content roles describe what users are allowed to do with regards to content management. For example, users may have permissions to Read, Write and Create content in certain sections of the site but not in others. User interface roles refer to security descriptors placed on the user interface components themselves. For example, administrators can limit user access to specific user interfaces or even specific elements of those interfaces (such as a button in the UI).

A Sitecore domain is a grouping of users and roles. Your solution can include any number of Sitecore domains. By default, users for Sitecore’s authoring and delivery environments are managed under separate security domains. For multi-site solutions, each site may have its own domain. Sitecore provides locally-managed domains, which limit security administrators to managing users and roles for site-specific domains.

In the move from Sitecore 5.3 to 6.0, Sitecore transitioned from a proprietary user management system to the ASP.NET Membership framework. By default, Sitecore users and roles are stored in the aspnet membership tables (see below).

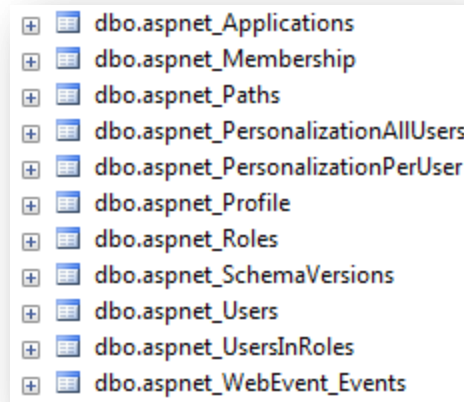


Figure 23. Default membership tables.

Sitecore has extended Microsoft's membership framework to support features such as "roles in roles" and caching. The adoption of ASP.NET Membership makes it very straightforward to integrate Sitecore with external systems that are already making use of this feature of ASP.NET.

Sitecore also leverages the Membership framework for Active Directory (AD) integration. This common integration is typically used to manage content authors and their roles. Sitecore's AD integration supports single sign-on, meaning that users can access the Sitecore UIs without having to enter additional credentials. For intranets, AD may also be used for website visitors.

While the ASP.NET Membership schema and Active Directory provide useful options for most implementations, there are some scenarios where these are not appropriate membership repositories. In these cases, your project may require integrating Sitecore with your external security provider. The most common cases that require security integration are the following:

- A large membership repository already exists that is not based on ASP.NET membership.
- A federated membership system controls authentication across multiple applications.

In the first case, you may have been building up your user base over a number of years in a home-grown system. Your system may be used for a variety of purposes, such as Web site authentication, email marketing and management reports. Porting your database over to ASP.NET Membership may require a re-coding of existing applications that may or may not be native .NET.

In the second case, a federated membership system controls authentication. Porting your users over to the ASP.NET schema could undermine single sign-on capabilities and would not be consistent with your enterprise authentication strategy.

The rest of this chapter will discuss security integration options with Sitecore in both the authoring and delivery environments. We will follow the same format as previous chapters, looking at membership repositories, mechanisms for integration and then reviewing common integration scenarios.

4.1 Repositories

As mentioned in the introduction to this chapter, Sitecore provides two default repositories for membership data: the ASP.NET Membership schema and Active Directory. The ASP.NET Membership tables store user authentication, user profile and role membership information. While ASP.NET Membership does not natively support the “roles-in-roles” feature, Sitecore has extended the membership provider to support this feature. Users can also be stored in Active Directory. Active Directory uses its own database to store account information. This database is placed on domain controllers along with other components that manage AD database access.

One of the advantages of Sitecore’s adoption of ASP.NET Membership is that Sitecore works natively with any system that implements the ASP.NET Membership Provider. In this sense, Sitecore is repository agnostic with regard to physical storage of membership data. Customers, for example, have integrated Sitecore with Microsoft Commerce Server’s membership provider and several other .NET e-commerce platforms.

In security integration scenarios, the physical repository is typically less significant than the mechanisms for communication between Sitecore and the system that manages the repository. With Sitecore’s AD integration, for example, Sitecore makes use of .NET APIs that communicate with the AD server using an LDAP connection string. Another common solution is to make use of web services to communicate between Sitecore and your authentication system.

4.2 Mechanisms

4.2.1 .NET Membership Provider

The .NET membership provider is an obvious point of integration between Sitecore and an external security system. If your authentication system supports ASP.NET Membership, it will likely be “plug and play” with Sitecore. If your vendor has only partially implemented ASP.NET Membership, it is possible that you will need to implement some required methods; but, in most cases, no additional work is required to get Sitecore working with your membership provider.

The configuration for a membership provider should look familiar to those who have worked with ASP.NET Membership. Sitecore allows you to specify a membership provider, role provider and an optional profile provider. For example, the following web.config element is for the default Sitecore membership provider:

```
<membership defaultProvider="sitecore">
  <providers>
    <add name="sitecore" type="Sitecore.Security.SitecoreMembershipProvider,
      Sitecore.Kernel" realProviderName="sql" providerWildcard="%" raiseEvents="true" />
  </providers>
</membership>
```

Role providers and profile providers are configured similarly to the membership provider, also in the web.config.

4.2.2 Custom Membership Provider

If your membership repository does not support ASP.NET membership out of the box, Sitecore makes it easy to implement your own provider. The topic is discussed in detail on the Sitecore Developer Network.

To understand the effort required to implement a membership provider, it is useful to distinguish a membership provider from a role or profile provider. A membership provider mainly provides authentication services. These services can include features such as authentication, password management and CRUD operations on user accounts. If you only require authentication services from the membership provider, implementing a read-only provider is sufficient. A read-only provider requires the implementation of two methods of `System.Web.Security.MembershipProvider: GetUser()` and `ValidateUser()`.

Sitecore also provides the option of implementing a read/write membership provider. This implementation is useful if you wish to use the Sitecore user interfaces to manage user accounts. As with the read-only provider, this requires that you overwrite methods of `System.Web.Security.MembershipProvider`.

Implementing a role provider is an analogous process. Based on your requirements, i.e. read-only or read/write, you can implement the appropriate methods of `System.Web.Security.RoleProvider`.

Microsoft provides ample documentation on implementing role and membership providers and Sitecore encourages developers to review relevant online documentation for more information.

Selected SDN References:

Sitecore CMS 6 Membership Providers –
<http://sdn5.sitecore.net/Reference/Sitecore%206/Membership%20Providers.aspx>

Low-Level Sitecore CMS Security and Custom Providers –
http://sdn.sitecore.net/Articles/Security/Low_level_Sitecore_Security_and_Custom_Providers.aspx

4.2.3 Active Directory

As mentioned in the introduction to this section, Sitecore ships with Active Directory integration. The Active Directory module can point to any number of AD domains using Microsoft's Active Directory Membership Provider. Sitecore has extended this provider to facilitate caching and the roles-in-roles capability.

Selected SDN References:

Active Directory Module Administrator's Guide –
<http://sdn.sitecore.net/Products/AD/Documentation.aspx>

4.2.4 Named Users

The simplest approach to membership integration is to implement named users. This approach is not a match for customers with sophisticated security requirements on the public Web site and it will provide less rich analytics data for customers using Sitecore's Online Marketing Suite (OMS). Also, it is only an appropriate solution for membership on the public Web site and doesn't make sense for content author authentication. Given this, it is probably the quickest way to integrate users from an external membership repository into a Sitecore solution.

The approach works as follows. Imagine that your public Web site has two kinds of users: anonymous users and registered users. Registered users can see sections of the site that are not available to anonymous users and can do things like manage their profiles, post comments, etc. You have a pre-existing membership repository of a million users that have registered on your site over the past 3 years.

With the named user approach, you only need to create one user in Sitecore. For example, you can create a user called "RegisteredUser." Since all registered users on the site are treated the same from a security perspective, all users can browse the site with the RegisteredUser identity (from Sitecore's perspective).

The flow of events is as follows:

1. The user enters their user name and password in a login form on the Web site.
2. The system authenticates the user against the existing membership repository.
3. Once authenticated, the system programmatically logs the user in to Sitecore as the RegisteredUser account.
4. As a result, the system has authenticated the user via the existing membership system and Sitecore applies the RegisteredUser security permissions as the user browses the site.

The advantage of this approach is that it requires very little code, is not complex and satisfies the requirements for many Web sites.

4.2.5 Member Import

If you are interested in migrating your legacy membership repository to ASP.NET Membership, you can accomplish this through a member import approach. To accomplish this, use the classes in the `System.Web.Security` namespace to create user accounts and roles based on the information in your existing membership repository.

4.2.6 CRM Security Providers

Sitecore offers two security providers that are useful for integrating CRM data: the Microsoft Dynamics provider and the Salesforce.com provider. These providers support user authentication on your public Web site from your CRM system.

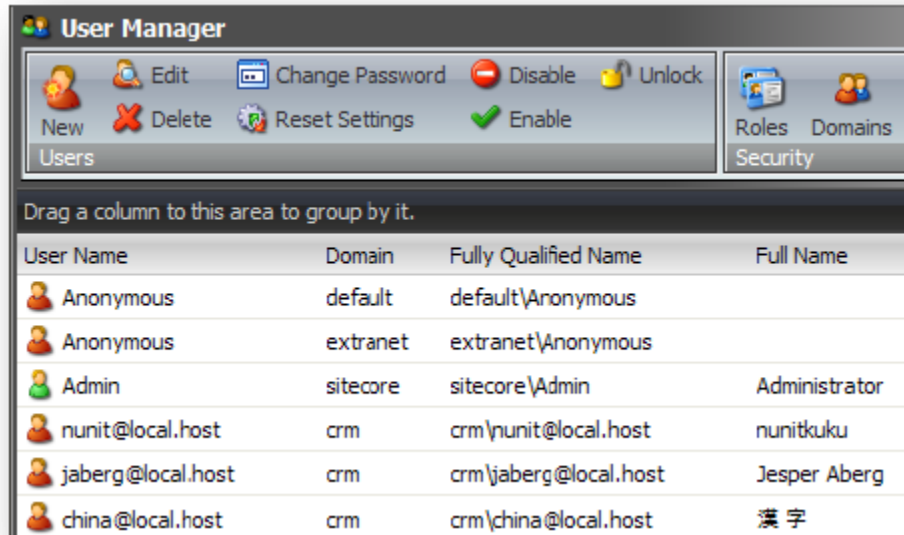


Figure 24. Microsoft Dynamics users in Sitecore User Manager.

There are several major benefits to Sitecore CRM connectors: First, as pre-built components, no coding is required for implementation. Second, by connecting your public Web site to your CRM, there is no need to house user accounts in multiple repositories. Third, your Web site can leverage CRM information to personalize site content and analytics data. This last point is of particular relevance to customers using Sitecore's OMS.

Sitecore uses Microsoft Dynamics integration on its public Web sites, such as the Sitecore Developer Network and the Sitecore Partner Network. User accounts and roles are configured in Microsoft Dynamics, which provides a common membership repository across Sitecore's various Web properties.

4.3 Integration Scenarios

Different approaches are available for security integration in the delivery and authoring environments.

4.3.1 Delivery Environment

Third-Party .NET Membership Provider

Because Sitecore's user management is based on Microsoft's provider model, it is simply a configuration exercise to integrate Sitecore with a third-party .NET membership provider. The provider is declared in the web.config as described in section 4.2.1. Issues may arise if the third-party vendor only partially implements their provider. In these cases, additional methods may need to be implemented to support complete integration with Sitecore.

Simple Integration with Federated Authentication System

If authentication is handled by a non-.NET federated authentication provider and a .NET membership provider is not a desirable solution (due to timelines or perceived complexity), the Named Users approach described in 4.2.3 may be the simplest approach. Sitecore authenticates site visitors against the federated system and then logs the user in as a generic Sitecore user (with appropriate roles pre-assigned). This approach is useful, for example, in scenarios where the site visitor interacts with multiple applications across disparate platforms. Authentication is handled outside of Sitecore, but Sitecore applies all of the necessary roles so that the user has a security-enforced browsing experience.

User Management for Intranets

In the intranet scenario, it may be desirable to manage user accounts with Active Directory. Site permissions can be derived from group assignments in AD or through content-specific roles configured in Sitecore. As a result, Sitecore presents a security-aware user experience for AD site visitors. For example, appropriate benefits information can be displayed to users based on the international AD domain or organizational unit of which they are part.

4.3.2 Authoring Environment

Single Sign-On for Content Authors

The simplest way to achieve single sign-on for content authors is with Sitecore's Active Directory module. Rather than accessing the standard Sitecore login screen, authors visit a single sign-on URL that uses Integrated Windows Authentication to authenticate users. Content authors are automatically redirected to the appropriate authoring interface without having to explicitly log in to the system. Note that content authors can still have the option of accessing the standard log in screen if they wish to access an alternate user interface.

Federated Authentication Using .NET Membership

In the enterprise environment, users may need to access many disparate systems and it is often easiest to manage authentication through a federated system. For the authoring environment, Sitecore recommends accomplishing this using a custom .NET membership provider. The membership provider should use web services or other mechanisms to authenticate users against the federated repository. Authentication success or failure is passed to Sitecore, which applies appropriate security permissions based on authentication status.

Chapter 5

Conclusion

More than any other Web CMS on the market, Sitecore was built from the ground up to support integration with external systems. Sitecore exposes almost all of its functionality through configuration settings or its API. As a result, almost every aspect of Sitecore's functionality can be customized and connected to any system addressable through .NET APIs or Web services.

Sitecore takes advantage of its framework flexibility as it develops new pre-built integrations. Sitecore currently provides integrations with social media and forum solutions, productivity applications, search applications, e-commerce platforms, translation applications and more. All of this is in addition to the numerous out-of-the-box components mentioned elsewhere in this document.

Whether the pre-built components address your requirements or a custom solution is more appropriate, Sitecore is an open platform that makes integration with external content, media, security and applications feasible, flexible and straightforward.