Sitecore CMS 6.0 or later

Content Reference

A Conceptual Overview of Content Management in Sitecore

Table of Contents

Chapter 1

Introduction

This manual is designed to give you a conceptual overview of the way that Sitecore manages content. The topics covered range from the content tree to versioning and publishing. There is also a section about editing the `web.config` file to enable running in live mode. There are also sections about managing proxies and aliases.

This manual contains the following chapters:

- **Chapter 1 — Introduction**
  This is a brief description of this manual.
- **Chapter 2 — Managing Content**
  This chapter describes how Sitecore manages content.
- **Chapter 3 — Publishing**
  This chapter describes how Sitecore copies content from the “work in progress” database to the “live” Web site database.

Chapter 2

Managing Content

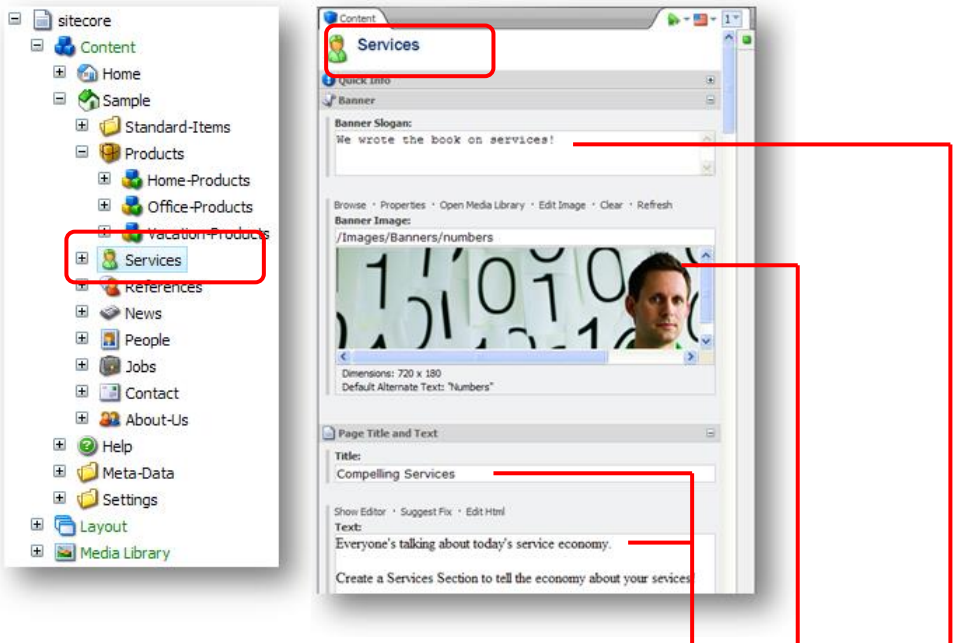
This chapter describes how Sitecore manages content.

This chapter contains the following sections:

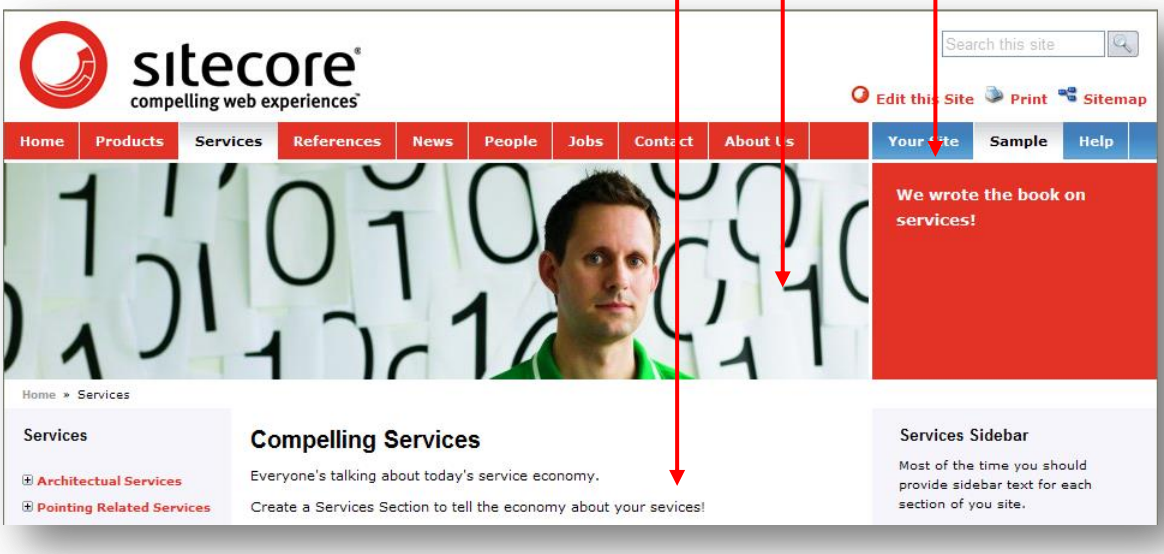
- Content Tree
- Databases
- Item Versioning
- Proxies
- Aliases
- Link Management

2.1 Content Tree

Sitecore stores content in items organized into a tree. The following example shows a part of the content tree with the same item opened in the editor:

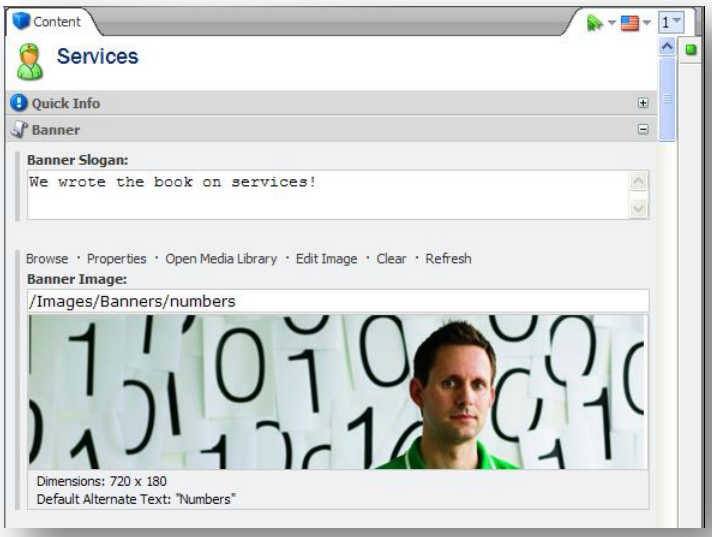


This is what this item looks like on the Web page:

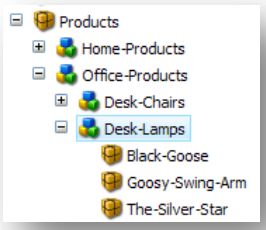


An item in Sitecore is a structured element of content. Items are similar to records in relational databases. Items contain one or more data fields of various types which are grouped in field sections.

For instance, in the following image, you can see the **Banner** field section which contains two fields — the **Banner Slogan** text field and the **Banner Image** field.



Any type of item can serve as a container for other items, providing structure for a site. For example, every item in the following image has fields which are rendered on the front-end and at the same time each of these items can be a container (like a folder in Windows Explorer) for other items. Content authors can also create the *folder* type items.



Some items are involved in servicing page requests but are not rendered themselves. For example, the various site setting items, such as cache setting items.

Different items have different types and functions and are therefore stored in different locations. For instance, the content items are stored under */sitecore/content/*, the layouts are stored under */sitecore/Layout/*, the workflow items are stored under */sitecore/system/Workflows/* and so on.

On the front-end, the URL to an item is constructed according to the following formula: “hostname” + “Sitecore path”. The root item is defined in the `web.config` configuration file. Sitecore also provides functionality for creating URL aliases.

Example of a URL:

http://localhost/sitecore/content/Sample/Products.aspx

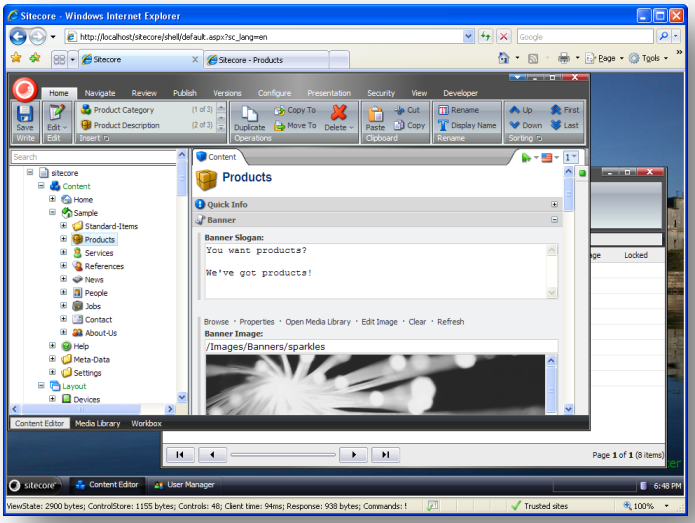
Sitecore stores item data in databases. Some types of items, such as layouts, sublayouts, XSL renderings, and file media, store a definition item in a Sitecore database which stores the path to corresponding file on the file system. When you duplicate an item like this, Sitecore duplicates the

reference to the file on the disk and does not duplicate the file itself. Moving, renaming or deleting such an item does not move, rename or delete the corresponding file on the disk — only the reference.

You can determine how items appear in the Content Editor by editing the properties that affect their appearance, such as, Icon and Display Name.

Note
Item names are used for URLs, not display names.

In Sitecore, users can define data structures through a browser-based user interface. No additional downloads are necessary to edit a Web site built with Sitecore.

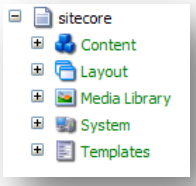


The back-end of a Web site built with Sitecore is typically accessed with the URL *hostname/sitecore*.

The root item in the content tree is called “sitecore”. The paths to Sitecore items are defined according to the content hierarchy, this document will mostly use absolute paths when referring to items, for example: */sitecore/content/Sample/Products*.

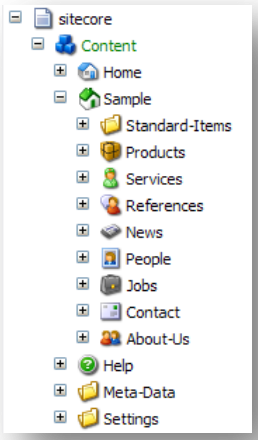
2.1.1 Standard Content Tree Areas

There are five standard content tree areas under the “sitecore” item: Content, Layout, Media Library, System, and Templates.



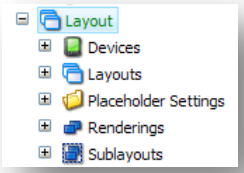
Content

The Content section contains the content of the site (or multiple sites). This content is used to render the Web pages.



Layout

The Layout section contains all the presentation components such as the layouts, sublayouts, renderings and placeholder settings.



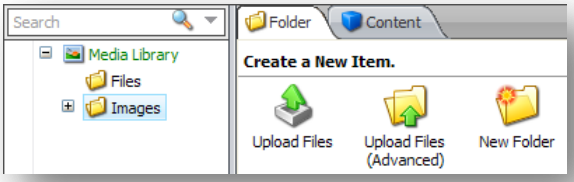
Media Library

The Media Library contains all the media content that can be used on the Web site — images, animations, movies, PDF documents, and so on.

The actual content of each media file can be either stored on the file system (in which case one of the fields in the media template references the location of the file on the disk) or encoded and stored in a field in the database.

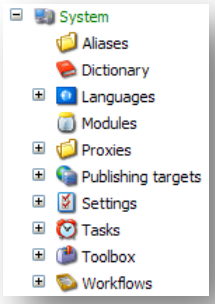
Sitecore provides a number of media templates that are used to manage metadata about the media, such as the alternate text for images. You can see the list of the media templates under the */sitecore/templates/System/Media* template folder item.

You can upload individual or multiple files to the media library. You can upload large numbers of files in a zip file and instruct Sitecore to automatically unpack the archive after uploading it.



System

The System section contains items which are used by the controls of the site but are not rendered themselves. The System section contains the following items:



Templates

The Templates section contains all the different types of template that are used to design the Web site.

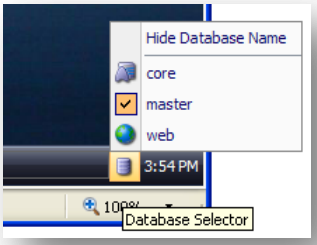
2.2 Databases

Each Sitecore installation uses a number of relational databases to store content objects, user credentials, and other data. Sitecore APIs and XML representation abstract these databases as hierarchical repositories of data items.

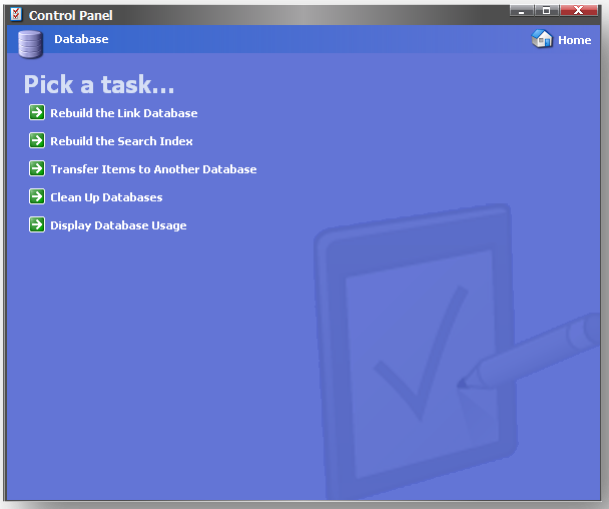
Sitecore 6 includes three databases by default:

- **Master** — contains every version of all the data items
- **Web** — contains the most recently published version of each publishable data item. By default the published Web site is rendered using the data from the Web database.
- **Core** — controls the Sitecore user interfaces

You can switch between databases by clicking the Database Selector button  in the lower right corner of the desktop:



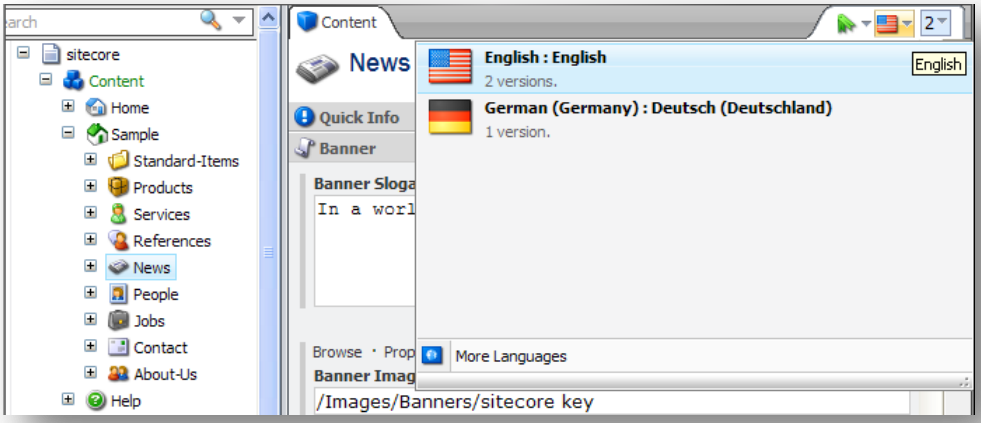
Sitecore contains various tools for working with databases. To access these database tools click Control Panel, Database.



You can configure Sitecore to use other databases and use data providers to attach databases from different vendors.

2.3 Item Versioning

Sitecore supports sites running in multiple languages and supports item versioning. A Web site can support any number of languages. You can also create many different numbered versions of each item in each language. For example, in the following image the *News* item has two versions in English and one version in German.



There are 2 English versions and one may be published already while the other version is still being edited in a workflow.

Language definition items are stored under */sitecore/system/Languages*.

The field values in items can be versioned, unversioned (all numbered versions within a given language contain the same value, but different languages may contain different values), or shared (all versions in all languages contain the same value for the field).

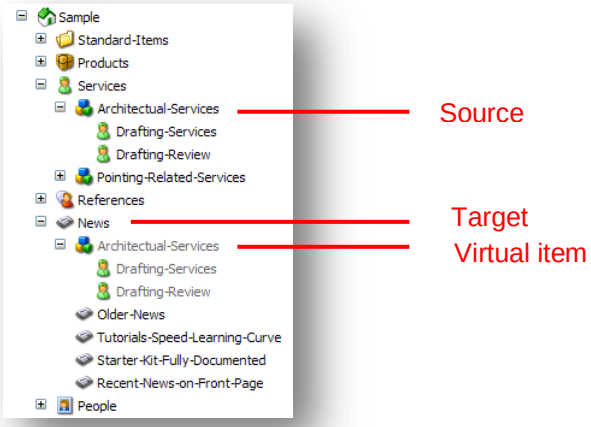
Versions can be created manually or automatically with workflows.

You can restrict publishing on a version level. For more information about publishing, see section 3.3, Publication Restrictions.

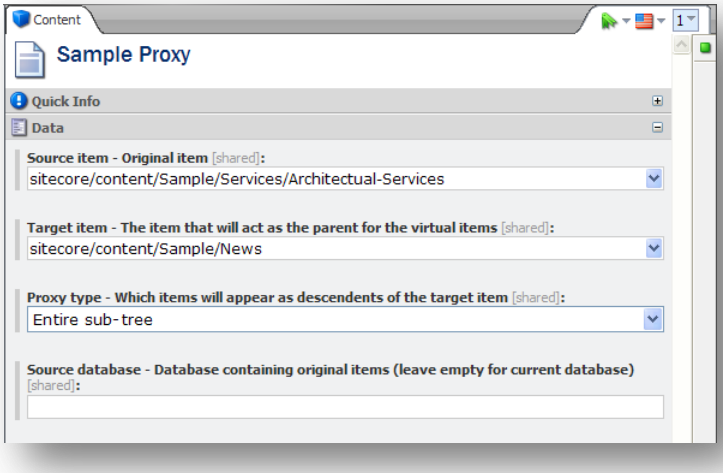
2.4 Proxies

Proxy items make it possible to show a source item (and potentially its children) in multiple places within the Sitecore content tree. Changes to virtual items are reflected immediately in the source item. Even deleting a virtual item or sub-item deletes the same item at the source.

When proxy items are enabled, a proxy item instructs Sitecore to generate a virtual item, which acts as though it is the source item, but is displayed in a different location. Virtual items are displayed in the Content Editor content tree with grey text, as shown.



Proxy items are defined in the */sitecore/system/Proxies* folder and specify a source item, the target item or parent to the virtual items generated, whether sub-items of the source should also appear as virtual items, and the database from which to retrieve the source item.



Sitecore places the following restrictions on proxies and virtual items:

- You can only create proxies for content items.
- You cannot add virtual items to packages.
- You cannot transfer virtual items to another database with the Transfer to Database feature.

2.4.1 Proxy Template

Proxy items are based on the */sitecore/templates/System/Proxy* template. The Proxy template includes fields which configure how the proxy works.

These fields include:

- **Source item** — the original item
- **Target item** — the item that will act as the parent for the virtual items
- **Proxy type** — the drop list where you specify the proxy type. By default the list includes “Entire sub-tree” and “Root item only”.
- **Source database** — the database containing the source item. Leave this field blank if the source is in the current database.

2.4.2 Required Configuration

You must enable proxy items in the Sitecore `web.config` file.

Each database can have its own proxy settings, which are included as sub-elements in the database element (for example, as sub-elements under the `<database id="master"...>` element).

Proxies are *not* enabled by default. Enabling proxies has a negative effect on the performance of the entire site. You should therefore only enable proxies only if the site will use them.

Sitecore includes the following `web.config` configuration settings:

- **proxiesEnabled** — indicates whether proxies are enabled for this database. The possible values are true or false.

  For example:

  ```
  <proxiesEnabled>true</proxiesEnabled>
  ```

- **proxyDataProvider** — link to a type deriving from ProxyDataProvider. This is the class that reads the physical data (for example, from SQL Server).

  For example:

  ```
  <proxyDataProvider ref="proxyDataProviders/main"
  param1="$(id)"></proxyDataProvider>
  ```

- **publishVirtualItems** — indicates whether Sitecore should publish virtual items as if they were normal items. The possible values are true or false.

  If this setting is set to True, Sitecore will copy virtual items to the publishing target (for example, the Web database). Once they have been copied to the publishing target, the items act as normal items (the connection between the virtual item and its source will no longer exist).

  If this setting is set to false, Sitecore only copies the proxy definitions to the publishing target. Assuming that proxies are enabled on the publishing target database, Sitecore will then generate virtual items based on the proxy definition in that database. This is the preferred approach.

  For example:

  ```
  <publishVirtualItems>false</publishVirtualItems>
  ```

It is very important that the proxy settings for the master database and publishing target are compatible. For instance, in a standard Sitecore installation, there are two valid cases:

Only proxy definitions published.

```
<!-- master -->
<database id="master"...
  ...
  <proxiesEnabled>true</proxiesEnabled>
  <publishVirtualItems>false</publishVirtualItems>
 ...
</database>
<!-- web -->
<database id="web"...
  ...
  <proxiesEnabled>true</proxiesEnabled>
 ...
</database>
```

Virtual items published.

```
<!-- master -->
<database id="master"...
  ...
  <proxiesEnabled>true</proxiesEnabled>
  <publishVirtualItems>true</publishVirtualItems>
 ...
</database>
<!-- web -->
<database id="web"...
  ...
  <proxiesEnabled>false</proxiesEnabled>
 ...
</database>
```

2.5 Aliases

Sitecore determines the default URL for an item based on its path; for instance the default URL for */sitecore/content/home/hr/jobs* is */hr/jobs.aspx*. In some cases it is useful to have shorter URLs which map to longer paths, for instance */jobs.aspx* may be preferable for marketing materials (email campaigns, print advertisements, and so on.). Sitecore supports alternate URLs through a feature known as aliases.

Aliases are based on the */System/Alias* template and defined under */sitecore/system/Aliases*.

Each alias name must be unique; if a single Sitecore instance is hosting multiple sites under a single document root they cannot both use the same alias (for instance /jobs). The alias template contains a single field in the data section which allows you to select the target item.

It is generally advisable to prevent search engines from indexing multiple URLs for a single content item, therefore aliases are intended for marketing materials only. Sitecore features for generating URLs such as the sc:link and sc:path XSL extension functions as well as links generated by the HTML editors use the default URL for each item; aliases should not be hard-coded into renderings, include files, or any other type of presentation component.

2.6 Link Management

Sitecore stores all internal links as GUIDs. All GUID links are resolved at runtime to their text representations when generating a Web page. This means that the links remain valid even if you move the target item (because the GUID remains the same). This also ensures optimal performance for link database operations.

The links are resolved using a pipeline. Likewise, a reverse link (link -> guid) pipeline is also available for incoming HTTP requests. The link pipelines support multilingual links. Sitecore can automatically generate appropriate URLs for an item represented in English, Danish, and so on. This feature optimizes search engine indexing.

Furthermore, the default link generation is aware of sub-sites and can substitute a sub-site's domain name in multi-site installations.

Default validators are provided which continuously checks for unique tree item names but which also checks for the uniqueness of display names whenever they are used to generate a link.

In IIS7, links can have any extension or no extension. For example, the following are all valid IIS 7 links:

- http://mysite.com/home/products.aspx
- http://mysite.com/home/products.html
- http://mysite.com/home/products

Chapter 3

Publishing

This chapter describes how Sitecore copies “work in progress” content to the “live” Web site.

This chapter contains the following sections:

- Overview
- Publishing Types
- Publication Restrictions
- Publishing Targets
- Live Mode

3.1 Overview

By default, when content is modified, it must be published before it will appear on the Web site. Content is edited in the Master database (which contains “work in progress”) while Sitecore generates requested Web pages using content in the Web data (which contains “live” content). Publishing copies the latest “publishable” version of “publishable” items from the Master database to one or more publishing target databases, by default the Web database.

You may publish individual items, a list items which are known to have changed (known as “incremental publishing” the entire site), cycle through the entire database to publish items that Sitecore can detect have changed (known as “smart publishing” the entire site), or cycle through the entire database and publish all publishable content regardless of whether it has changed or not (known as “republish” the entire site).

When changes are made to any items that are not associated with a workflow, Sitecore adds these items to an internal publishing queue (the “list of items which are known to have changed”). When items associated with a workflow reach a final workflow state they are added to the publishing queue as well.

Incremental Publishing publishes all the items in the queue, copying the appropriate versions of the appropriate items from the Master database to the Web database.

Note
An item will only be published if all of its ancestors have been published – */Company/AboutUs* cannot be found in the Web database unless */Company* also exists in the Web database.

Note
Publishing clears various caches. This process can have an impact on performance. Therefore, you may want to consider restricting the Publish operation to appropriate users.

3.2 Publishing Types

Sitecore supports four different types of publishing:

- Publishing an Individual Item
- Incremental Publishing of the Entire Site
- Smart Publishing of the Entire Site
- Republishing of the Entire Site

3.2.1 Publishing an Individual Item

Sitecore supports the publication of an individual item, with or without the descendants underneath the item. Publication of an individual item can be triggered either directly by the user in the Content Editor or automatically using Workflow actions.

3.2.2 Incremental Publishing of the Entire Site

Incremental publishing publishes all the items in the publishing queue, which is the list of items known to have been modified. Incremental publishing is the quickest publishing option and requires the fewest resources of the three options which publish the entire site.

3.2.3 Smart Publish of the Entire Site

Smart publishing starts at the root item and iterates through the content tree comparing the item revision fields stored in the Master database and the selected publishing targets. Publishable items that have revision fields which do not match are copied from the Master database to the publishing targets. Smart publishing automatically removes items in the Web database which are no longer publishable. Smart publishing takes longer and requires more resources than incremental publishing.

3.2.4 Republish of the Entire Site

Republishing overwrites the entire contents of the target database with the publishable items in the master database. Republishing is the most expensive type of publishing, because it must perform a write operation for all items, and write operations are more expensive than read operations.

Because of the expense, you should only use republish when the databases appear to be in an inconsistent state, for example, after a publishing operation has failed due to a network outage.

3.3 Publication Restrictions

Sitecore supports a number of features to restrict the unwanted publication of content items. The main two features include publishing restrictions and workflows. For more information about Workflows, please refer to the Workflow Reference guide.

By default, all items that are not in a workflow are considered “publishable”. Sitecore supports a number of restrictions which can disable the publication of specific items or specific item versions.

The restrictions include:

- Making an item “un-publishable” by turning off the “publishable” toggle associated with the item in the item’s publishing restrictions. This affects every version of the item in every language.
- Making an item publishable starting at a specific data and/or until a specific date. This makes every version of the item in every language un-publishable before the start date and after the end date.
- Making a specific version of an item “un-publishable” by turning off the “publishable” toggle associated with the item version in the item’s publishing restrictions.
- Making specific versions of an item publishable starting at a specific data and/or until a specific date. This makes the specific versions of the item in a specific language un-publishable before the start date and after the end date. If other versions of the item are publishable, however, the publishable version with the highest version number will be published during a publish operation.

The publishing restrictions are stored with the item in Publish and Lifetime sections defined on the standard template.

3.3.1 Publish Section Fields

- **Publish**

  This field defines the start date and time when the item becomes publishable. If blank, the item is publishable until the date specified in the **Unpublish** field.

- **Unpublish**

  This field defines the date and time after which the item cannot be published. If blank, the item will not cease to be publishable once it becomes publishable.

- **Publishing Targets**

  This field defines the list of acceptable publishing targets for this item. If the site only has one publishing target defined, all items are automatically publishable to that target, whether selected or not.

- **Never Publish**

  This checkbox defines whether the item (all versions and languages) can be published.

3.3.2 Lifetime Section Fields

- **Valid from**

  This field defines the start date and time when the version becomes publishable. If blank, version is publishable until the date specified in the Valid to field.

- **Valid to**

  This field defines the date and time after which the version cannot be published. If blank, the version will not cease to be publishable once it becomes publishable.

- **Hide versions**

  This checkbox defines whether the current version can be published.

3.3.3 Preview Mode

Sitecore provides the ability to preview the Web site without publishing. Preview is offered both as a stand-alone application and a tab in the Content Editor. The Preview application supports the ability to specify a preview date, which allows you to view the site in the past or future, based on the publishing restrictions as they are currently set.

3.3.4 Scheduled Publication

Publishing restrictions indicate which versions and items may be published at specific dates and times, but restrictions do not automatically invoke the publish operation. Sitecore does support, however, periodic automatic publication.

The <*scheduling*> section of the `web.config` file configures when Sitecore will perform automatic publishing operations. The amount of time that elapses before Sitecore checks for any scheduled tasks to be executed is defined in the <*frequency*> section of the `web.config` file.

For example:

```
<scheduling>
  <!-- Time between checking for scheduled tasks waiting to execute -->
  <frequency>01:00:00</frequency>
```

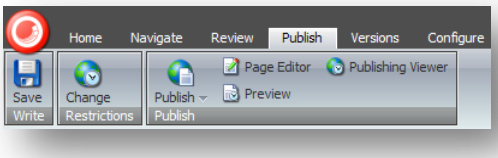
3.3.5 Publishing triggered by Workflow

Sitecore contains an auto-publish action which allows it to publish an item when it reaches a specific workflow step. The auto-publish action respects publishing restrictions and the item is not published if the item is not publishable due to publishing restrictions.

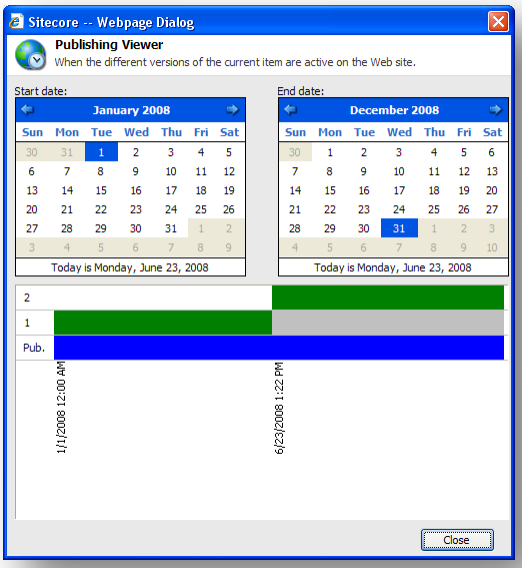
3.3.6 Publishing Viewer

Sitecore provides an overview of publishing restrictions for the various versions of an item in the current language in the Publishing Viewer.

In the Content Editor, in the **Publish** tab, in the **Publish** group, click Publishing Viewer:



The **Publishing Viewer** dialog box contains a visual representation of the item's publishing restrictions as well as the creation date of the various versions. At the top of the window, is a calendar that displays the start date as well as the end date for the view. In the lower half is a visualization of creation date of the item in blue. Green bars indicate when specific versions were created and/or publishable.



The view can be changed to a longer or shorter period of time by changing the start and end dates in the two calendars at the top of the dialog box. They do not affect the actual time settings for the item itself.

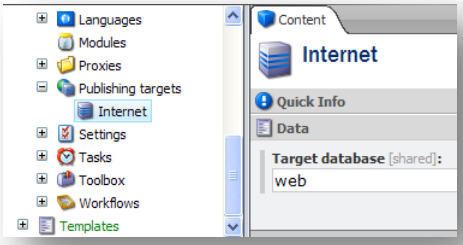
In the bottom of the dialog box, key dates are inserted, such as the start date for a new version of the item. When a version is published it is displayed in the topmost row. The exact date and time for each version can be seen by placing the mouse over the specific color bar.

The bar coloring code is:

- Green — the version is active and valid for publish.
- Gray — the version is inactive. The version is typically superseded by a newer version.
- Blue — the period where the item is valid. It is not necessarily published.

3.4 Publishing Targets

Every publishing target is associated with a particular database. For instance, the Internet target is associated with the Web database. Content is published to the Internet target regardless of whether or not that target is selected in the **Publishing** section of the content item.



Publishing targets are defined under */sitecore/system/Publishing targets*.

Publishing targets are relevant to the Staging module. The Web database of each runtime server is typically configured as a publishing target.

3.5 Live Mode

Sitecore supports running a Web site directly from the master database, referred to as “running in live mode”. Running in live mode eliminates the need to publish content and is similar to viewing a site in the Preview client. A Web site configured to run live mode acts in most ways exactly in the same way as a default Web site. Live mode respects all publishing restrictions and workflows in the same way that a default Web site supports these features.

3.5.1 Live Mode Configuration

Three specific attributes in the `<site>` definition in the `web.config` configure live mode.

- Database — By default, the database attribute specifies a publishing target, such as the Web database. For live mode, the database attribute specifies the Master database.
- filterItems — By default, the filterItem attribute is not specified and defaults to false. For live mode, the filterItems attribute is set to true. This configures the site to respect publishing restrictions retrieving items.
- enableWorkflow — By default, the enableWorkflow attribute is not specified and defaults to false. For live mode, the enableWorkflow attribute is set to true. This configures the site to respect workflows when retrieving items.

The default *WebRoot/AppConfig/Include* folder contains a sample `web.config` include file which converts a Web site to run in live mode.

If a site leverages a specific workflow provider, the Master database definition in the `web.config` should contain an appropriate workflow provider definition item.

3.5.2 Rendering Implications

When running in live mode, renderings may items which exist but which have no version that have completed a workflow. In these cases, the item will not have any content, but will appear as a blank item, especially in navigation controls or other lists of items.

Modify menu renderings (and other renderings that show lists of items) so that they do not show empty items. You can check for empty items by checking the following condition:

```
sc:fld('__created',.)!=''
```

3.5.3 Caching Implications

Running in live mode has caching implications. Sitecore normally clears caches during publish operations, since a publish operation may change content which could result in “stale” information in the cache. Live mode eliminates publishing, which means it also eliminates automatic cache clearing.

To resolve this issue, Web sites running in live mode either avoid caching, or automatically trigger a clear cache operation when users change content stored in cached renderings.