Sitecore CMS 6

Data Definition Reference

A Conceptual Overview for CMS Administrators, Architects, and Developers

Table of Contents

Chapter 1

Introduction

This document describes the concepts that architects, developers, and CMS administrators need to understand when designing, implementing, and maintaining the information infrastructure associated with a Sitecore Web site.

This document contains the following chapters:

Chapter 1 — Introduction
A brief description of this document and its intended audience.

Chapter 2 — The Data Template
An in-depth description of data templates, used to define the fields associated with content items.

Chapter 3 — The Standard Template
An in-depth description of the Sitecore Standard template, which defines fields required by all items.

Chapter 4 — The Template Field
An in-depth description of the data template field definition item and related features.

Chapter 5 — Field and Item Validation
An introduction to field and item validation features.

Chapter 6 — The Branch Template
An introduction to branch templates, used to create multiple items simultaneously, as a copy of a provided group of items.

Chapter 7 — The Command Template
An introduction to command templates, used to initiate a Sitecore command during item creation.

1.1 Features

This document contains a detailed description of the components and concepts used when defining the information architecture of a Web site. In this document you will find details describing the function and purpose of the following:

Templates — Sitecore users create items using one of three template types: data templates, branch templates, and command templates. Data Templates form the framework around which items are built. They define fields used to control how data is entered and can inherit from other templates to enable reuse.

Fields — these are the areas within which the data entered into the system is controlled. Through these fields, grouped into **Sections**, we can organize and control the amount and type of data entered into the system. Fields are organized by their field type.

Field Types — these are the different ways through which data is entered or selected for fields. There are numerous flexible ways of entering data into the system and it is the field types which control the type of data that can be entered or selected in the fields.

Data is validated throughout the system. The different types of validation are discussed and we will describe how validation covers the entire data definition area within the system to control the validity and authenticity of data definition structure.

You can use standard values to ensure that default data is automatically inserted into newly created items. Standard values are default values that populate null fields of existing items or fill data into newly created items.

Branch templates — allow you to create a set of items rather than a single item at a time.

Command Templates — allow you to insert of items according to logic rather than predefined structures.

1.2 Common Terminology

This section defines some of the common data definition concepts.

1.2.1 Template

Sitecore uses the term “template” to refer to the components within Sitecore that are used to create new content. The terms: “data template”, “branch template”, and “command template” refer to specific template types. Together these templates and template types control the schema of data entered into the system as items.

1.2.2 Data Infrastructure

The term “data infrastructure” is used to describe how the data is structured prior to content creation, as opposed to data entered by an end-user, which is “content”. This document as a whole discusses data infrastructure.

1.2.3 Data Representation

Sitecore data representation is:

- Flexible — with Sitecore you can define data structures through an easy-to-use browser-based user interface.
- Abstract — you can also change data structure definitions without impacting data.
- Hierarchical — the data is structured using implicit and explicit relations.
- XML-oriented — the system also provides XML representations of data.
- Object-oriented — Sitecore also provides hierarchical, object-oriented representations of data.

1.2.4 Data Repositories

The Sitecore data repositories provide a variety of data services. Data can be represented in a limitless variety of languages, and then controlled using strict versioning and security. The flow of data through the system can be set using a variety of custom rules developed into a workflow. Finally, publication of data to the “live” Web site can also be influenced by a variety of settings that strictly monitor and control how data is published.

1.3 General Information

The following sections describe general information that pertains to data definition.

1.3.1 Naming conventions

When naming templates, fields and field sections, try to use simple, relevant, and easy to understand names. By default, Sitecore displays the names you provide to both technical and non-technical users. Choose names that business users, such as content authors, will easily understand and relate to.

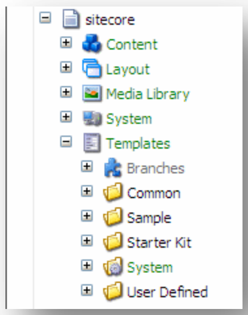
Avoid using a naming convention that recommends prefixes or suffixes that indicate the type of element being named, such as “xyz template” or “abc field”.

Sitecore supports the concept of a “display name” and “field title” which it displays in place of the item or field’s true name, for cases where confusing names cannot be avoided.

Program code references templates and fields by name, so avoid special characters in names.

1.3.2 Template Storage Locations

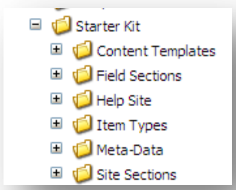
The various types of templates in Sitecore are stored as items under `sitecore/Templates` in the master database.



As you can see in the previous image, Sitecore comes with several folders created as defaults for the storage of the various templates. These are “Branches”, “Common”, “Sample”, “System” and “User Defined”.

Note
We recommend that you do not delete or rename any of these folders as this may have impact on other areas of the system.

Templates can, however, be stored anywhere in the template area, even in custom created folders. Organize templates into folders relevant to the Web site that they are created for. The following image shows an example, the template folders in the Starter Kit folder.



Chapter 2

The Data Template

This chapter describes the structure and purpose of data templates and standard values.

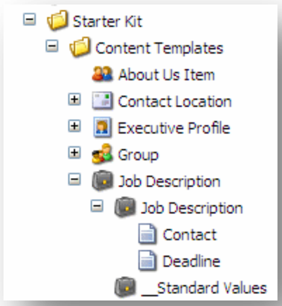
This chapter contains the following sections:

- Understanding Data Templates
- Defining Standard Values

2.1 Understanding Data Templates

A data template defines the sections and fields that make up an item. In object-oriented programming terms, a data template is most similar to a class defining a number of attributes. In relational database terms, data templates function most like a table defining a number of columns.

Each data template consists of a number of sections, which in turn contain a number of fields. The system represents data templates as hierarchies of definition items; the root item defines the template, each child defines a field section and each grandchild defines a field.



In this example, you can see the *Job Description* data template opened with its field section *Job Description* immediately below it and underneath this field section you can see the fields *Contact* and *Deadline*.

Each data template is based on one or more other data templates, which themselves may be based on other data templates. Sitecore combines the sections and fields in the data template and all base templates into a superset of sections and fields. Most data templates should be directly or indirectly based on the Sitecore Standard template, which defines fields and sections required by all items.

For more information about base templates, see section Base Templates under section 2.1.2, Data Template Sections.

For more information about the Sitecore standard template, see Chapter 3, The Standard Template.

As soon as a field is added to a data template, that field appears in all the items based, either directly or indirectly, on that data template.

2.1.1 Data Template Fields

A data template field defines the user interface control and other properties that influence how the field behaves in the Content Editor and Page Editor. For more information about fields, see Chapter 4, The Template Field.

Note
When defining field names, ensure that they are unique even between field sections. Both XSLT and .NET code use field names alone, without reference to sections, to extract content from fields.

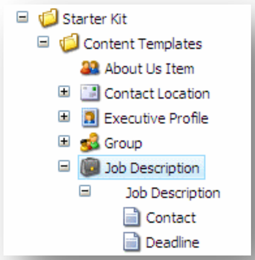
2.1.2 Data Template Sections

Data template sections organize the fields into template into related groups. Fields are always stored within a section. Sections, however, are only reflected in the Content Editor. Programmers do not need indicate a section when extracting content from a field.

Use sections to group fields logically, making them easier for content authors to find and use.

Consider creating data templates with a single section and related fields to use as one of multiple base templates. This makes it easy to build up a new data template with multiple standard sections of fields.

Section definition items are stored under the corresponding data template definition item. In the following image, you can see the data template *Job Description* and immediately under it is the **Job Description** section, which, in turn, contains the **Contact** and **Deadline** fields.



When you edit an item associated with a data template, each section appears as a collapsible group of fields in the Content Editor.

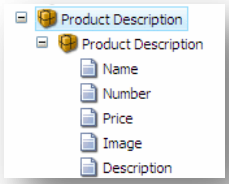
The **Section Name** field of the section definition is used as a label for the section in the Content Editor. Section names can be localized.

Base Templates

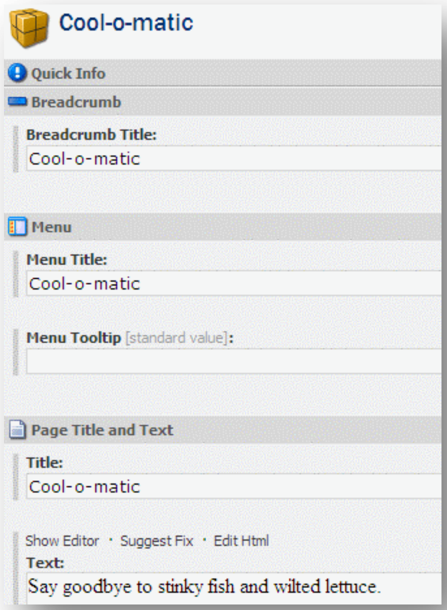
A data template inherits the sections and fields defined on its base templates. You can see the base templates associated with a data template in the **Inheritance** tab in the **Template Manager** or **Content Editor**.

If a set of fields or sections are common to many templates then these can be gathered together in one data template, then that template can be added to other templates as desired.

As an example, consider the “*Product Description*” template from the Starter Kit. The template defines a single section called **Product Description** and five fields: **Name**, **Number**, **Price**, **Image**, and **Description**.



However, an item created from this template contains additional sections and fields.



Cool-o-matic

Quick Info

Breadcrumb

Breadcrumb Title:
Cool-o-matic

Menu

Menu Title:
Cool-o-matic

Menu Tooltip [standard value]:

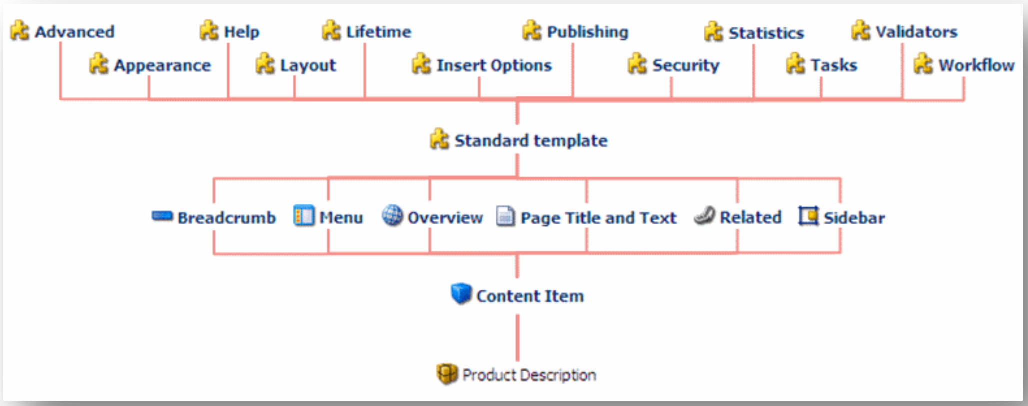
Page Title and Text

Title:
Cool-o-matic

Show Editor · Suggest Fix · Edit Html

Text:
Say goodbye to stinky fish and wilted lettuce.

The item inherits these fields and sections from the *Product Description* data template's base templates. In the following image, you can see the *Product Description*'s base templates.



Multiple Inheritance

A data template can be based on any number of data templates, not just one. Occasionally more than one inherited template will contain the same field or field section. In this case, the UI will merge these fields or field sections to prevent duplication.

Circular Inheritance

If a data template is based upon itself, either directly or indirectly, this is referred to as circular inheritance. Circular inheritance causes severe issues.

The symptoms include:

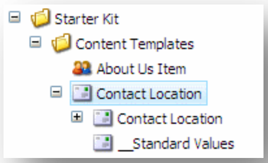
- The system becomes unresponsive, especially when working with data templates.
- ASP.NET raises application errors.
- Log entries indicate circular template inheritance detected.

Circular template inheritance often occurs when developers modify the base templates associated with the Standard template or its base templates. Avoid making modifications to the default templates provided with the system. Aside from the risk of circular inheritance, modifications to templates under the `/sitecore/templates/system` branch could complicate the process of upgrading Sitecore or raise other challenges.

If any template does not explicitly inherit from another template, that template implicitly inherits from Sitecore's Standard template. The Standard template inherits from a number of templates defined under `/sitecore/templates/System/Templates/Sections`, each defining a section of the Standard template.

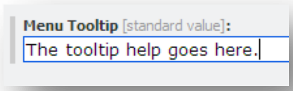
2.2 Defining Standard Values

Data templates can optionally have an associated __Standard Values item. The system stores the standard values item as a child of the data template definition item (as shown). The template definition item contains a **__Standard Values** field; a link to the standard values item.



The standard values item, like any item based on the data template, contains all the fields defined in the data template itself, plus any fields inherited from base templates. The standard values item contains default field values for items based on the data template. Standard values are used when a field value for an item is set to NULL.

When viewing an item in the Content Editor, any field that contains a standard value will be indicated by the token [standard value] immediately next to the field title (as shown).



Standard values can be inherited from any base template, not just the base template that an item is directly based on.

Tip
To reduce administration and centrally manage system settings, it is best to define the following in template standard values rather than in individual items:

- Layout settings
- Initial workflow
- Insert Options

Note
If a standard value for a field is set on standard values items for multiple templates an item is based on, only the value from the first template in the inheritance list is used. The values from the other templates are ignored.

2.2.1 $name Token

Sitecore supports the $name token in standard values. Sitecore replaces the $name token with the name of the item during creation. This replacement, however, only takes place during item creation. The item name then becomes the field contents, which override the $name standard value.

If a user renames an item, the $name token will *not* be re-evaluated. Thus, fields that had been assigned a value using the $name will continue to show the original item name even after the item has been renamed.

If the field value that has been set using the $name token is reset, the field will then show the value "$name" and the user must set the field value to something more appropriate.

2.2.2 Blank vs. NULL Field Values

When a content author clears a field, such as a text field, this will often leave the field with blank contents, which is not the same as NULL contents. By default, such a field will display a blank value, not the standard value defined for the field.

Field definition items, however, contain a **Reset Blank** checkbox. If this is selected, Sitecore replaces blank values with NULL. Thus, when **Reset Blank** is enabled for a field, and the user blanks out the contents of that field, the Content Editor displays the standard value for the field.

Chapter 3

The Standard Template

This chapter contains details about the Sitecore Standard template.

This chapter contains the following sections:

- Understanding the Standard Template
- The Structure of the Standard Template

3.1 Understanding the Standard Template

The Standard template is the Sitecore default base template shared by most other data templates. This is done either explicitly through inheritance, or implicitly if not specifically excluded from inheritance (by specifying the null GUID as a base template).

The Standard template defines sections such as security and workflow relevant to all types of items. It defines how Sitecore should manage an item, such as when it should be published, which workflow it is in, which users should be allowed to access it, and so on.

Most templates should be based “eventually” on the Standard template, either directly or indirectly.

The **Quick Info** field section, which is always the first section of a template shown to Administrator and Developer users, is a field section in the Standard template. This field section contains item location details, the templates location in the content structure and various GUID’s related to storage.

Most fields in the Standard template represent system values shared with all versions in all languages.

3.2 The Structure of the Standard Template

The following sections contain details of all the fields in the Standard template.

3.2.1 Advanced Section

The **Advanced** section is used to associate a data template with its standard values item.

- **__Standard values** — this field contains a link to the Standard Values item.

3.2.2 Appearance Section

The **Appearance** section is used primarily to hold the data that specifies how the Content Editor displays the item.

- **Context Menu** — this field contains a link to menu displayed when a user right clicks on an item in the Content Editor's content tree.
- **Display Name** — this field overrides the item name in the Content Editor.
- **Editor** — this field defines a custom editor for an item.
- **Editors** — this field controls which editing tabs are shown in the Content Editor.
- **Hidden** — this field controls whether or not the item is visible in Content Editor. You can manage this by selecting the **Hidden Items** checkbox on the **View** tab of the main ribbon in the Content Editor.
- **Icon** — this field controls the icon shown in the content tree and in the item header in the Content Editor.
- **Read Only** — this field controls whether or not the item is *Read Only*.
- **Ribbon** — this field controls the method of overriding the Content Editor default ribbon.
- **Sortorder** — this field is used by the system to specify the order in which the fields are displayed.
- **Subitems Sorting** — this field is used by the system to sort the order of subitems. Options include "Created", "Default", "Display Name", "Logical", "Reverse" and "Updated".

3.2.3 Help Section

The **Help** section is used to hold the help information.

- **Help link** — this field contains the link to the detailed help for this item.
- **Long description** — this field contains the fly-over help shown when hovering the mouse over an item in the content tree.
- **Short description** — this field contains the description shown in the item header of the Content Editor.

3.2.4 Insert Options Section

The **Insert Options** section is used to hold a set of Insert Rules and Options.

- **Insert Rules** — this field contains the insert rules.

- **Insert Options** — this field contains the insert options.

3.2.5 Layout Section

The **Layout** section is used primarily to associate the data template with its renderings.

- **Renderings** — this field contains the renderings used to display this item.
- **Renderers** — this field contains the renderers used to display this item.

3.2.6 Lifetime Section

The **Lifetime** section is used to hold the publication restrictions for individual versions of this item.

- **Valid from** — this field contains the date from which this version is valid for publication.
- **Valid to** — this field contains the date to which this version is valid for publication.
- **Hide version** — this field controls whether to hide this version from publication.

3.2.7 Publishing Section

The **Publish** section is used to hold the publishing restriction information.

- **Publish** — this field contains the date from which this item is valid for publication.
- **Unpublish** — this field contains the date to which this version is valid for publication.
- **Publishing targets** — this field contains the valid Publishing targets for the item.
- **Never publish** — this field controls whether or not an item should be published. *Never published* overrides *Publish* and *Unpublish*.

3.2.8 Security Section

The **Security** section is used to hold the security settings.

- **Owner** — this field contains the name of the user who is the current owner of the item. For use with the virtual Creator-Owner role.
- **Security** — this field contains the security access rights for the item.

3.2.9 Statistics Section

The **Statistics** section is used to hold the basic creation, revision and update statistics.

- **Created** — this field contains the date and time the item was created.
- **Created by** — this field contains the name of the user who created the item.
- **Revision** — this field contains the revision number, stored as a GUID text string.
- **Updated** — this field contains the last date and time that the item was updated.
- **Updated by** — this field contains the name of the user who performed the last update.

3.2.10 Tasks Section

The **Tasks** section is used to hold reminder information for associated tasks and to store an archive date.

- **Archive date** — this field contains the date the item will be archived.
- **Reminder date** — this field contains the date when a reminder e-mail will be sent to the reminder recipients.
- **Reminder recipients** — this field contains the e-mail address of the recipients of the reminder. This can be one or more e-mail addresses separated by a semi colon (“;”).
- **Reminder text** — this field contains the text of the reminder e-mail.

3.2.11 Validation Rules Section

The **Validation Rules** section is used to hold the validation rules.

- **Quick Action Bar Validation Rules** — this field contains the item validation rules displayed in the Quick Action Bar.
- **Validation Button Validation Rules** — this field contains the item validation rules used by the Validation button in the **Proofing** group of the **Review** tab in the Content Tree.
- **Validation Bar Validation Rules** — this field contains the item validation rules displayed in the Validation Bar on the right of the Content Editor.
- **Workflow Validation Rules** — this field contains the item validation rules that are used in workflow validation.

3.2.12 Workflow Section

The **Workflow** section is used to hold the workflow status information.

- **Workflow** — this field contains the workflow state the item is in.
- **State** — this field contains the current workflow state the item is in.
- **Lock** — this field contains information about whether or not the item is locked, who locked it and what date/time it was locked.
- **Default workflow** — this field contains information about the default workflow for items created from this template.

Chapter 4

The Template Field

This chapter describes the details of the data template field definition items and the various field types associated with template fields.

This chapter also describes the query syntax that should be used in the *Source* property of a field definition.

This chapter contains the following sections:

- Understanding the Template Field Definition Item
- Understanding Field Types
- Using Sitecore Query

4.1 Understanding the Template Field Definition Item

Data template fields define the user interface controls and other properties that influence how the field behaves in the Content Editor and Page Editor. It is primarily used to control the structure of a field and can provide automatic error checking when you enter data in the field.

The template field is equivalent to a property in object-oriented programming, or a column in a relational database.

Every field in a template must exist within a section, which organizes fields into logical and reusable groups. For more information about Template Sections, see section 2.1.2, Data Template Sections.

4.1.1 The Template Field Template

Field definition items are based on the *Template field* template. Each template field definition item defines many field properties.

These field properties include:

Field Name

This is the name assigned to the field when it is created, used as the label for the field unless the **Title** field is filled in.

Tip
To make code easier to read when referencing fields by name, construct field names like variable names, without special characters.

Note
Field names *should* be unique. If fields are defined on a single data template with the same name then a validation error will occur when the data template is saved. But fields may also be inherited from base templates, thus, an item could potential contain multiple fields with the same name. If this occurs, Sitecore will display both fields in the Content Editor, but programmers, who use the field name to retrieve contents when using the API and XSLT Renderings, may get unexpected results.

Field Type

The field type specifies which user interface control the Content Editor will display to accept input for this field and also controls the storage format for that field. For more information about field types, see section 4.2, Understanding Field Types.

Title

The title is displayed above the field in the Content Editor, unless the title is blank, in which case the Content Editor displays the field name.

Source

The source property provides information that influences the user interface control associated with the field in the Content Editor. The behavior of the source field depends on the field’s type.

Some examples include:

- For list field types, such as Droplink, the source property indicates a location in the content tree that includes the items included in the list displayed by the field.

- For image and file types, the source property indicates the start folder displayed in the media library dialog.

Note
If the path begins with a tilde (~) character, the window opens with the desired folder selected, but allows the user to access the entire tree.

- For rich text field definitions, the item specified in the source property specifies a HTML editor profile controlling the features provided by the editing interface.
- For selection fields, the field source property may specify a Sitecore query using the `query:` prefix before the actual query statement as shown in the following example.

```
query:/sitecore/content/Home/Employees/*[ startswith(@EmployeeName, 'A')]
```

For more information about Query syntax, see section 4.3, Using Sitecore Query.

Source Parameters

Multiple parameters may be specified in the *Source* property to support complex fields by separating the parameters using the ampersand (“&”) character as shown in this example.

```
DataSource=/sitecore/content/home& IncludeTemplatesForSelection=section,sitemap
```

Various fields support the following parameters:

- **DataSource** — the field data source item, equivalent to using a path as the field source property as described previously.
- **DatabaseName** — the name of the database containing the data source item.
- **IncludeTemplatesForSelection** — the user can select items associated with this comma-separated list of template names.
- **ExcludeTemplatesForSelection** — the user cannot select items associated with this comma-separated list of template names.
- **IncludeTemplatesForDisplay** — the user can navigate items associated with this comma-separated list of template names.
- **ExcludeTemplatesForDisplay** — the user cannot see items associated with a comma-separated list of template names.
- **IncludeItemsForDisplay** — the user can see items with this comma-separated list of IDs.
- **ExcludeItemsForDisplay** — the user cannot see items with this comma-separated list of IDs.
- **AllowMultipleSelection** — the user can select more than one item.

Blob

This field is for internal Sitecore use only and should not be used.

Shared

When this checkbox is selected, the field has the same value for every numbered version in all supported languages. When the *Shared* property is set, changes to the field value in any language or numbered version of the item will be reflected in all the other language versions and numbered versions.

Shared values should only be considered in the following cases:

- Old values of the field are irrelevant.

- Workflow restrictions do not apply to the field value.
- Values are very large (versioning consumes storage).

Note
If both the **Shared** and **Unversioned** checkboxes are selected, the field acts as a shared field.

Note
Shared fields *do not* support workflow restrictions.

Unversioned

When this checkbox is selected, the field has the same value for every numbered version within a language, but may have different values between languages. Unversioned fields are similar to shared fields, but the system can maintain different field values for different languages.

Note
If both the **Shared** and **Unversioned** checkboxes are selected, the field acts as a shared field.

Note
Unversioned fields *do not* support workflow restrictions.

Default Value

This field is for internal Sitecore use only and should not be used.

Validation

This field can contain a regular expression against which the content of the field is validated when the field is saved. In order to save the field, its value must match this regular expression.

Validation Text

The system displays this message when you try to save the field and its contents do *not* match the expression in the **Validation** field.

Field Security

This field opens the **Assign Security** window for the Field Definition item. The Field Read and Field Write access rights apply.

Reset Blank

If the **Reset Blank** checkbox is selected, it resets the field value to NULL when the item is saved with this field set to blank. NULL fields reflect the standard value for the field. For more details about blank and NULL fields, see section 2.2.2, Blank vs. NULL Field Values.

Exclude from Text Search

If this checkbox is selected, this field is not included in the search indexes.

4.2 Understanding Field Types

Each field has an associated field type. The field type controls what type of user interface component is used to enter data into the field and in what format that data is stored.

All field values are stored as text values. Some field types, such as single-line text, store a simple text value; other field types, such as Multilist, store the GUIDs of the selected Sitecore items separated by pipe characters (“|”); whereas complex field types, such as image, store an XML element.

4.2.1 Field Types Definition

The Field Type control organizes the list of field types displayed to make it easier for users to find the type they need and to understand how a specific type works.

The field type categories include:

- Simple Types
- List Types
- Link Types
- Developer Types
- Deprecated Types
- System Types.

4.2.2 Simple Types

The Simple field types are used by fields storing simple text-based content.

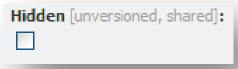
The simple field types include:

- Checkbox
- Date
- Datetime
- File
- Image
- Multi-Line Text
- Password
- Rich Text
- Single-Line Text

Checkbox

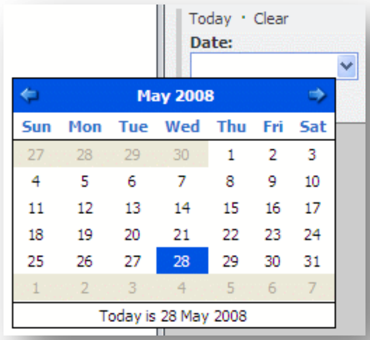
This field type displays a toggle button. If the user selects the checkbox, Sitecore stores a “1” in the field value. If the user does not select the checkbox, Sitecore stores a blank value.

The following image shows the **Hidden** checkbox from the **Appearance** section of the **Standard** template.



Date

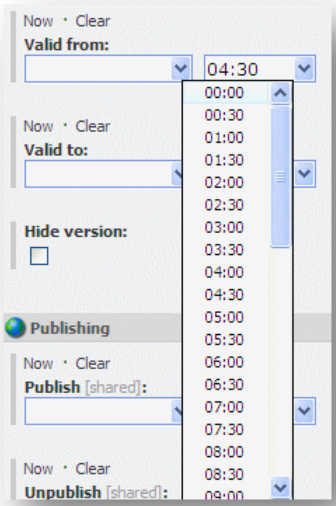
This field type is used to store dates. Dates for this field can either be entered as a text string or by using a calendar picker. Sitecore stores the date as a text string with the format "yyyymmdd".



Datetime

This field type stores both a date and a time. Datetime fields can be entered as text strings or by using the calendar picker. The time is also entered in a separate field by using a drop-down list and is broken into 30 minute segments. Alternatively the time can be entered manually. Sitecore stores the content as a text string with the format "yyyymmddThhmmss".

The following image shows the **Valid from** datetime field from the **Lifetime** section of the standard template. You can also see the drop-down list of times that you can select.

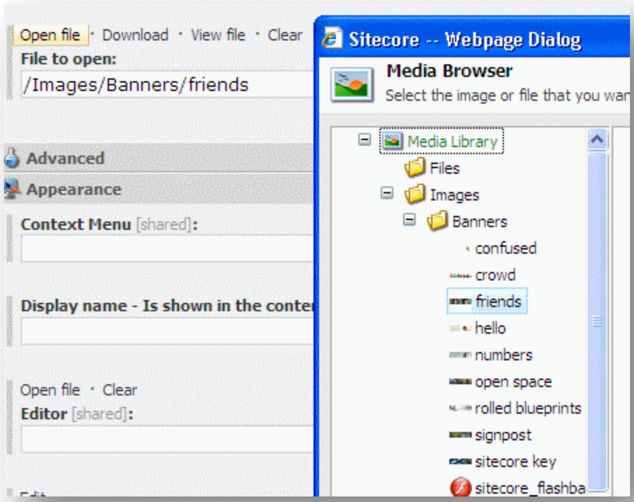


File

This field type stores links to files in the media library. Users can enter the path to the file manually or click Open file to open the **Media Browser** and select the desired file from the media library.

The source property for this field type indicates the selected media library path in the Media Browser, which defaults to the Media Library root if the source is not provided. If the source property starts with a tilde (~), the Media Browser will show the entire Media Library content tree, otherwise the Media Browser will only show the path specified and sub-folders.

The following image shows a file field with the path populated. On the right, you can also see the **Media Browser** opened with the item highlighted.



File to open: /Images/Banners/friends

Media Browser
Select the image or file that you war

Image

This field type links to images in the media library. Users can enter the path to the image manually or click Open file to open the **Media Browser** and select the desired file from the media library.

The source property for this field type indicates the selected media library path in the Media Browser, which defaults to the Media Library root if the source is not provided. If the source property starts with a tilde (~), the Media Browser will show the entire Media Library content tree, otherwise the Media Browser will only show the path specified and sub-folders.

The following image shows the **Image** field from the **Home-Products** item in the Starter Kit. You can see that, once an image is selected it displays a small version of the image along with the image dimensions and the default alternate text.

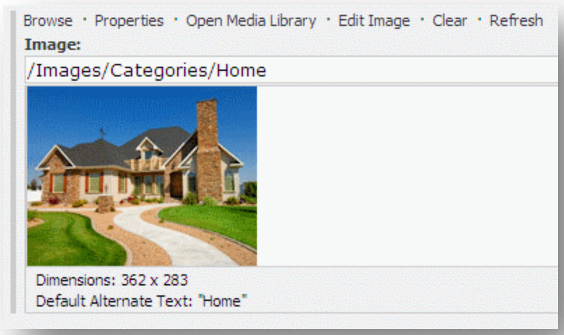

Browse · Properties · Open Media Library · Edit Image · Clear · Refresh

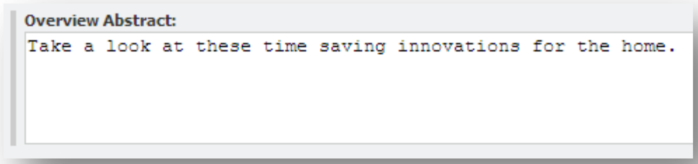
Image:
/Images/Categories/Home

Dimensions: 362 x 283
Default Alternate Text: "Home"

Multi-Line Text

This field type is used to store simple multi-line text entries. This field has no validation and there is no Rich Text support for fields of this type.

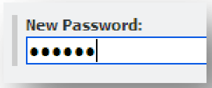
The following image shows the **Overview Abstract** multi-line text field from the **Home Products** item in the Starter Kit.



Overview Abstract:
Take a look at these time saving innovations for the home.

Password

This field type masks the provided text. The text provided, however, is stored normally. Sitecore does not hash or otherwise mask the provided text in the database.

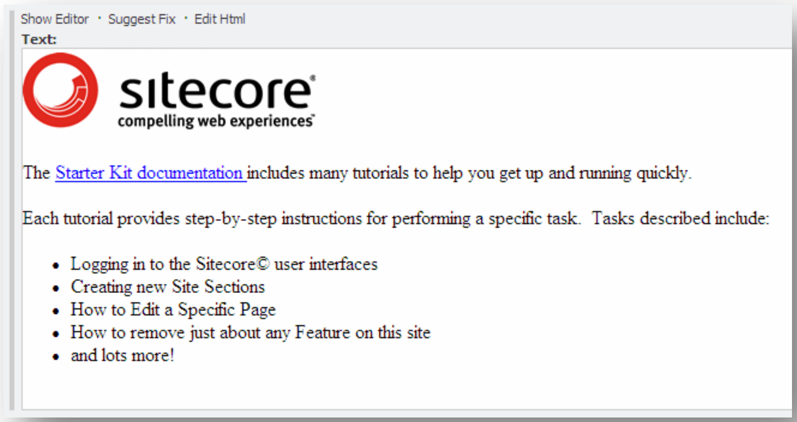


New Password:
••••••

Rich Text

This field type stores HTML text. The field displays the contents as a browser would display the source HTML, although the field actually stores character encoded HTML. The Show Editor button provides access to a Rich Text Editor, while the Edit Html button provides access to the stored HTML. For further information about Rich Text fields, see the Content Reference manual.

The following image shows a rich text field which includes an internal link and an image:



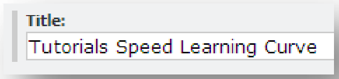
The Starter Kit documentation includes many tutorials to help you get up and running quickly.

Each tutorial provides step-by-step instructions for performing a specific task. Tasks described include:

- Logging in to the Sitecore© user interfaces
- Creating new Site Sections
- How to Edit a Specific Page
- How to remove just about any Feature on this site
- and lots more!

Single-Line Text

This field type requests and stores a single line of text.



Tutorials Speed Learning Curve

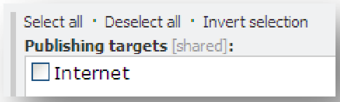
4.2.3 List Types

List Type fields are used to select and store one or more options from various types of lists. The options provided in a list field are specified using the Source property. The list options are usually the sub-items underneath the specified item path.

Checklist

This field type displays the source item sub-items as a list of checkboxes which accept multiple selections. The field stores a pipe (|) separated list of GUIDs corresponding to the selected checkboxes.

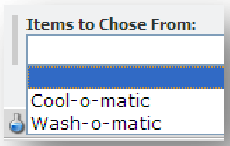
The following image shows the **Publishing targets** checklist field in the **Publishing** section of the Standard Template:


Select all · Deselect all · Invert selection
Publishing targets [shared]:

☐ Internet

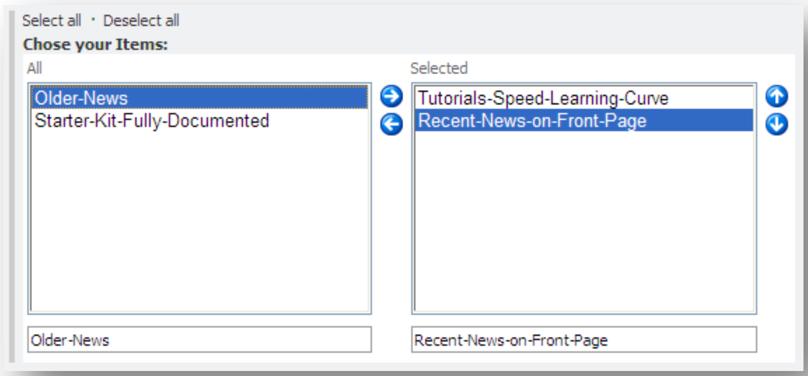
Droplist

This field type displays the source item sub-items drop-down list that allows users to select a single item. The field stores the name of the selected item.



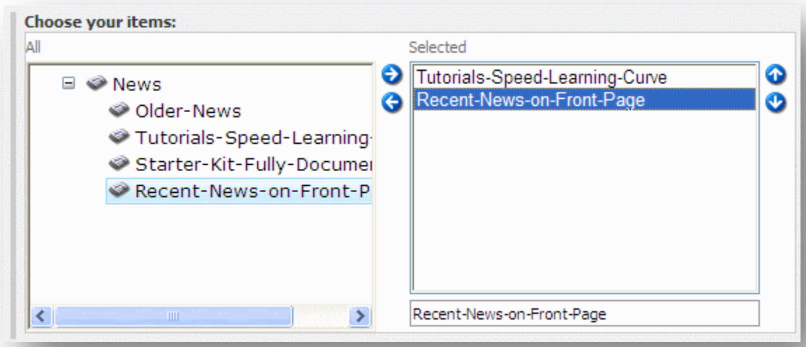
Multilist

Fields of this type allow users to select multiple values from a list of items. The field stores a pipe (|) separated list of GUIDs corresponding to the selected items.


Select all · Deselect all
Chose your Items:
All
Older-News
Starter-Kit-Fully-Documented
Selected
Tutorials-Speed-Learning-Curve
Recent-News-on-Front-Page
Older-News
Recent-News-on-Front-Page

Treelist

Fields of this type allow users to select multiple items from a directory tree structure. The field stores a pipe (|) separated list of GUIDs corresponding to the selected items.

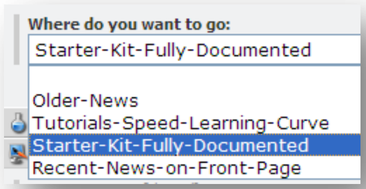

Choose your items:
All
News
Older-News
Tutorials-Speed-Learning
Starter-Kit-Fully-Documei
Recent-News-on-Front-P
Selected
Tutorials-Speed-Learning-Curve
Recent-News-on-Front-Page
Recent-News-on-Front-Page

4.2.4 Link Types

Link field types store a reference to another item.

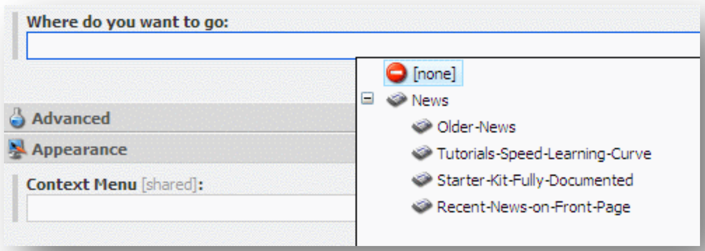
Droplink

This field type displays the source item sub-items drop-down list that allows users to select a single item. The field stores the GUID of the selected item.


Where do you want to go:
Starter-Kit-Fully-Documented
Older-News
Tutorials-Speed-Learning-Curve
Starter-Kit-Fully-Documented
Recent-News-on-Front-Page

Droptree

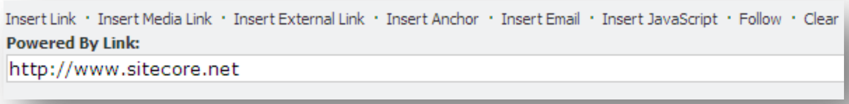
This field type displays the source item sub-items drop-down content tree that allows users to select a single item. The field stores the GUID of the selected item.



General Link

Fields of this type can be used for internal links, external links, media links, anchors and JavaScript links.

The following image shows a General Link field that is populated with an external link:


Insert Link · Insert Media Link · Insert External Link · Insert Anchor · Insert Email · Insert JavaScript · Follow · Clear
Powered By Link:
http://www.sitecore.net

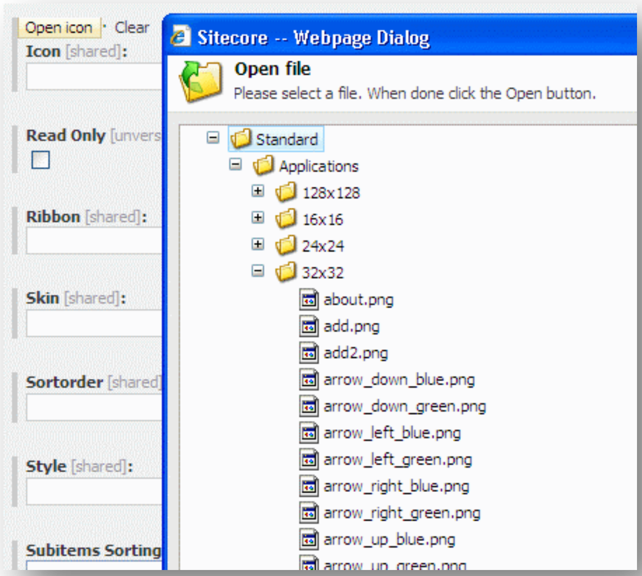
4.2.5 Developer Types

This section groups the fields which are primarily used to define the look and presentation of an item in the Content Editor and the functionality tied to it. It is generally used by Sitecore developers and not intended for typical Web site developers.

Icon

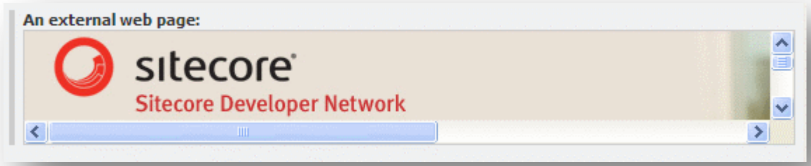
This field type is used for the ‘themed’ icons that are provided by Sitecore.

The following image shows an **Icon** field with the **Open file** window open:



Iframe

This field type is used to display a Web page in a field. The *Source* property is used to set the URL of the Web page that you want to link to.



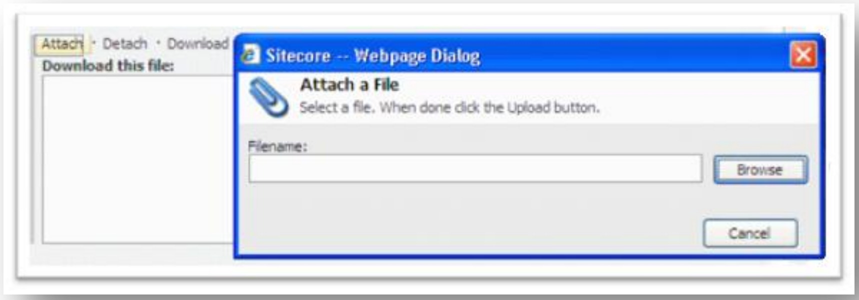
4.2.6 System Types

This section contains fields intended for Sitecore's internal use only. These field types are not intended for custom Web sites.

Attachment

This field type is used by media items to store a file or image in the database.

The following image shows an Attachment field with the **Attach a File** window open:


Attach · Detach · Download
Download this file:
Sitecore -- Webpage Dialog
Attach a File
Select a file. When done click the Upload button.
Filename:
Browse
Cancel

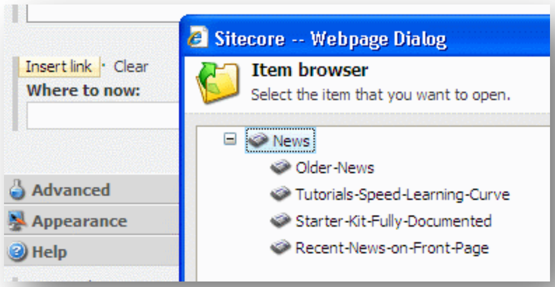
Field Source

This field type contains the Source property in field definition items.

Internal Link

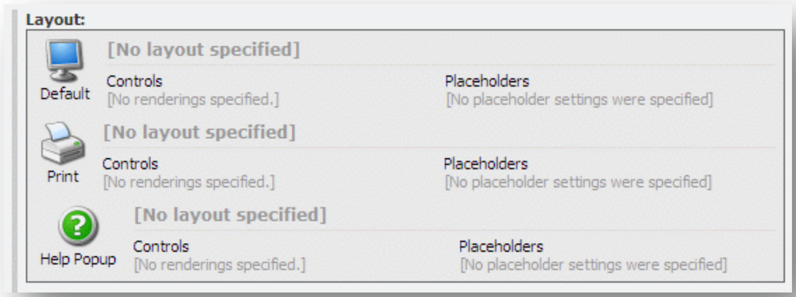
This field type is used for links to internal items. Only internal Sitecore items can be selected as target items for fields of this type. The *Source* property of the field specifies the root folder displayed in the related item browser.

In the following image, you can see an Internal Link field with the **Item Browser** window open:


Insert link · Clear
Where to now:
Advanced
Appearance
Help
Sitecore -- Webpage Dialog
Item browser
Select the item that you want to open.
News
Older-News
Tutorials-Speed-Learning-Curve
Starter-Kit-Fully-Documented
Recent-News-on-Front-Page

Layout

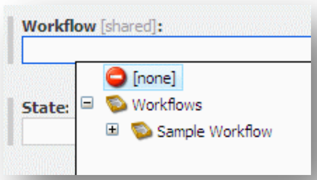
This field type contains the **Layout Details** associated with an item.


Layout:
Default
[No layout specified]
Controls
[No renderings specified.]
Placeholders
[No placeholder settings were specified]
Print
[No layout specified]
Controls
[No renderings specified.]
Placeholders
[No placeholder settings were specified]
Help Popup
[No layout specified]
Controls
[No renderings specified.]
Placeholders
[No placeholder settings were specified]

Reference

This field type displays the source item sub-items drop-down content tree that allows users to select a single item. The field stores the GUID of the selected item.

In the following image, you can see the **Workflow** field from the **Workflow** section of the Standard Template:



The *Source* property is set to /`sitecore/system/Workflows` to produce the tree.

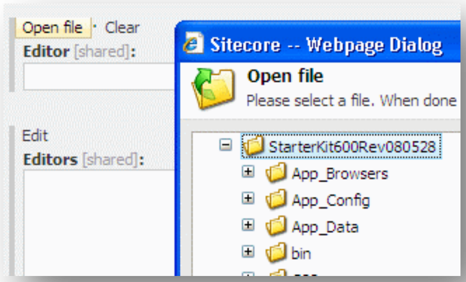
Security

This field type stores security settings for an item.

Server File

This field type is used for links to local server files.

The following image shows the **Editor** field in the **Appearance** tab of the Standard Template, with the **Open file** window open:



Deprecated Types

These field types existed in previous versions of Sitecore, but have been superseded by new field types and available for backwards compatibility.

The following table lists the deprecated field types and the corresponding new field types:

Sitecore 5.3 or earlier	Sitecore 6
html	Rich Text
link	General link
lookup	Droplink
memo	Multi-Line text

Sitecore 5.3 or earlier	Sitecore 6
reference	Droplink
server file	None
text	Single-Line Text
tree	Droptree
tree list	Treelist
valuelookup	Droplist

4.3 Using Sitecore Query

Sitecore query provides query string syntax that filters and retrieves items from the Sitecore content tree using a simplified version of XPath syntax. Developers use Sitecore query strings in .NET code and in the Source field of field definition items (requires the "query:" prefix at the start of the Source field).

4.3.1 General Syntax

The Sitecore query syntax leverages the concept of a context item and uses the following symbols to reference related items:

Symbol	Meaning
`/`	the root of the content tree or a parent-child relationship
`text`	match by item name
`#`	escape text containing dashes (-) . For example: #meta-data#
`*`	wildcard, match items of any name
`..`	the parent of the context item
`[ ]`	search criteria related to fields and XML element attributes
`@`	a field defined in the item's base template
`@@`	an XML element attribute, all Sitecore items are treated as "item" elements which contain the following attributes: • name The item's name • key The item's name in all lower-case characters • id The item's GUID • tid The item's base template's GUID • mid The branch template used to create the item, if any • sortorder The item's sort order • template The item's base template's name • parentid The item's parent's GUID

Combining these symbols references specific items or groups of items. For example:

Sitecore Query String	Result set
`/*`	The root of the content tree.
`/sitecore/content/home`	The Sitecore Home item.
`/sitecore/content/home/*[startswith(@title, 'P')]`	Immediate subitems under the home item that contain a Title field that starts with "P".
`*[@__hidden='1']`	All hidden subitems underneath the context item.

Sitecore Query String	Result set
`query: /*/content/#meta-data#/colors/*[@show='1']`	A Source field which selects all subitems under the Color item that have the “Show” checkbox field checked.
`./*[@@tid="{A87A00B1-E6DB-45AB-8B54-636FEC3B5523}"]`	Subitems under the context item based on the Folder template.

4.3.2 Axes

The axis component of a query determines the direction of the node selection in relation to the context node. An axis can be thought of as a directional query.

The following table lists some common axes:

Axes	Description
ancestor	Returns all the ancestors of the context item (same as XPath)
ancestor-or-self	Returns the context item and all the ancestors of the context item (same as XPath)
child	(/*) returns all the children of the context item (same as XPath)
descendant	(//*) returns all the descendants of the context item; a descendant is a child or a child of a child and so on (same as XPath)
descendant-or-self	Returns the context item and all the descendants of the context item (same as XPath)
following	Returns all the following siblings of the context node (same as following-sibling in XPath)
parent	(..) returns the parent item of the context item (same as XPath)
preceding	Returns all the preceding siblings of the context item (same as preceding-sibling in XPath)
self	(.) returns the context item (same as XPath)
[int index]	Returns the child item with the mentioned index

4.3.3 Operators

The following list of operators can be used in Sitecore query expressions:

Operator	Description	Example	Return Value
|	Contracts two item-sets	//*/Products|//*/Shapes	Returns children of Products and Shapes items

Operator	Description	Example	Return Value
+	Addition	6 + 4	10
-	Subtraction	6 - 4	2
*	Multiplication	6 * 4	24
div	Division	8 div 4	2
=	Equal	position()=3	true if position() is 3, false otherwise
!=	Not equal	position()!=3	False if position() is 3, true otherwise
<	Less than	position()<4	true if position() is less than 4
<=	Less than or equal to	position()<=4	true if position() is less than or equal to 4
>	Greater than	position()>4	true if position() is greater than 4
>=	Greater than or equal to	position()>=4	true if position() is greater than or equal to 4
or	or	position()=3 or position()=4	true if position() is 3 or 4, false otherwise
and	and	position()>3 and position()<7	true if position() is between 3 and 7, false otherwise
mod	Modulus (division remainder)	5 mod 2	1

4.3.4 Internal Architecture

Sitecore processes queries using the fasted technology possible. This could be either the SQL database, if the data provider supports the requested query, or in the Sitecore data manager. Note that the SQL database provides the best performance, but, unfortunately does not provide support for all queries.

The SQL database supports queries such as “/`sitecore/content/home`” which resolves an item by path, and “`//home`” which locates a set of items by name.

How does this work in practice? Imagine that you have a large site that uses an SQL database and you're trying to find all the content items named *needle*. The following API code will retrieve this result set:

```
Item content = Sitecore.Context.Database.Items["/sitecore/content"];
Item[] needles = content.Axes.SelectItems("//needle");
```

The SQL Server data provider supports this kind of query, so the query is resolved directly in the database and runs fairly quickly even though the database contains a large number of content items.

Adding additional search criteria, however, can change how the Sitecore performs the search. In the previous example, imagine that the items include a checkbox field called **IsHidden** and we only want needles that are not hidden. The following API code will retrieve this result set:

```
Item content = Sitecore.Context.Database.Items["/sitecore/content"];
Item[] needles = content.Axes.SelectItems("//needle[@IsHidden != '1']");
```

The SQL Server data provider does not support predicates (the portion of the search string enclosed in square brackets: [@IsHidden != ‘1’]). Therefore, Sitecore resolves this query in the data manager using the query API. To do this, the all the items in the query scope (all descendants of `/sitecore/content` in this example) are loaded so that the predicate can be evaluated against each item and the matching items returned. Unfortunately, the performance penalty associated with loading this many items can be quite dramatic for large sets of items.

Based on this understanding, we can see that a more optimal approach would be to first find all items named needle, then searching through the result set in memory to find the items that are not hidden.

Chapter 5

Field and Item Validation

This chapter provides details of the various ways that validation can be used on fields and items to control the authenticity of entered data.

This chapter contains the following sections:

- Understanding Validation Rules and Options
- Default Validation Rules
- Validation Options

5.1 Understanding Validation Rules and Options

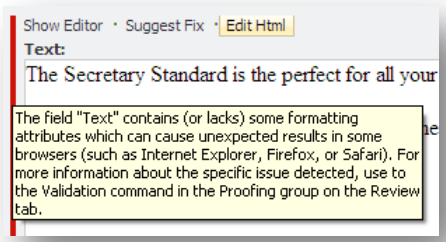
Validation rules and options validate field values using field validation rules resolved on-the-fly based on settings specified in the `web.config` file. The sections holding validation settings including direct expression settings, pipelines and event handlers as well as simpler settings like the ability to turn off the **Validation** bar in the Content Editor.

These validation rules validate a single field value in a single language version of an item. They can also validate each item using any number of item validation rules, which validate global values such as the item name, as well as conditions involving multiple field values.

When validating items and fields the validation rules use the same basic technology, and the main difference between the item and field validation rules are:

- Field validation rules validate values in one or more fields in a single language version. Whereas item validation rules validate item data shared to all versions, such as the item name, as well as conditions involving multiple field values.
- The user interface displays its field validation rules from /`sitecore/system/Settings/Validation Rules/Field Rules`, and its item validation rules from `/sitecore/system/Settings/Validation Rules/Item Rules`.
- In templates, the template field definitions define the field validation rules, whereas the template standard values and individual items define the item validation rules.

Any field validation issues appear to the left of the field value in the Content Editor as colored bars. The colors displayed are gray for no errors, yellow for warnings and red for errors. In the following image, you can see a field validation warning (the red bar); with a tooltip explanation displayed that contains more details about the error.



The Secretary Standard is the perfect for all your

The field "Text" contains (or lacks) some formatting attributes which can cause unexpected results in some browsers (such as Internet Explorer, Firefox, or Safari). For more information about the specific issue detected, use to the Validation command in the Proofing group on the Review tab.

5.2 Default Validation Rules

The following tables list the preinstalled item and field validation rules and a brief description of the validation that they perform.

5.2.1 Item Validation Rules

Item Validation Rule	Validates
Broken Links	Checks all languages and all versions for broken links in one or more fields.
Duplicate Name	Checks that the item name is unique among siblings.
Full Page XHtml	Validates the XHTML generated when a visitor requests an item.
Media Size Too Big	Checks whether a Media Library item exceeds a specific size.
Url Characters	Checks if an item name contains characters that must be escaped when rendering URLs, which negatively impacts search engine indexing.

5.2.2 Field Validation Rules

Field Validation Rule	Validates
Broken Links	Checks if a field contains broken links.
Is Email	Checks if a field contains an email address.
Is Integer	Checks if a field contains an integer.
Is XHtml	Checks if a field contains XHTML.
Max Length 40	Checks if a field contains a value of 40 or less characters.
Rating 1 to 9	Checks if a field contains a value between 1 and 9.
Required	Checks if a field contains a value.
Spellcheck	Checks spelling using the RAD Editor spell check validation, also used in the Rich Text editor.
W3C XHtml Validation	Validates the field HTML using the W3C validation service.

5.2.3 System Field Validation Rules

System Field Validation Rule	Validates
Alt Required	Checks that the alt text is filled in.
Extension May Not Start with a Dot	Checks that the media file extension does not start with a dot.

Extern Link Target	Checks for external links, i.e. the links to other sites.
Image Has Alt Text	Checks the alt text on an image.
Image Has Alt Text from Media Library	Checks if the media item has default alt text. The default alt text (from the media library) will be used.
Image Has Valid Path	Checks that a given path exists.
Image Size	Checks the size for the images referenced through image fields
Rich Text Image Size	Checks the image dimensions for the images included in the rich text fields, i.e. if the image is too big to look good in the site design.

5.3 Validation Options

Validation options can be chosen in the **Validation Rules** section of the data template standard values and template field definitions. Sitecore comes with a set of predefined validation options which are listed in the following table with a brief description of the validation that they perform.

Validation Option	**Controls**
Quick Action Bar	Validation issues appear in the **Quick Action** bar on the left in Content Editor
Validation Button	Validation issues appear when the user chooses the Validation command from the **Proofing** group on the **Review** tab, and when the user invokes a transition to a workflow state which includes the workflow validation action.
Validation Bar	Validation issues appear in the **Validation** bar on the right in Content Editor
Workflow Validation Rules	Validation issues appear in the user interface when a user chooses a workflow command associated with the workflow validation action. The user cannot complete the workflow action without resolving all validation errors.

Chapter 6

The Branch Template

This chapter describes the structure and purpose of branch templates.

This chapter contains the following sections:

- Understanding Branch Templates
- Using Branch Templates

6.1 Understanding Branch Templates

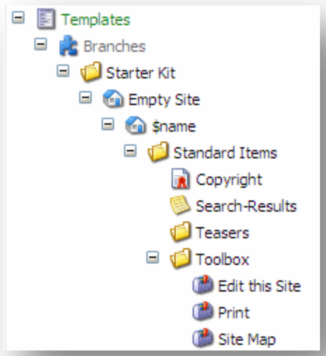
A branch template consists of a definition item and a collection of one or more subitems, which may also contain subitems, down any depth. All items defined underneath the branch template definition item may be based on any number of data templates, as desired.

Developers may assign branch templates in an item's insert options, similar to the way developers assign data templates as insert options.

When a user chooses a branch template from the insert options, Sitecore creates a copy the entire collection of descendants defined underneath the branch template definition item, and places that collection as a group of subitems underneath the currently selected item.

Sitecore copies all field contents defined in the branch template descendants, but replaces the $name token assigned to fields and item names with the name provided by the user during the insert operation.

The following image shows the Empty Site branch template from the Sitecore Starter Kit. The only immediate subitem of the branch template definition item is called *$name*. When you create an item based on this branch template, Sitecore requests a name and will replace the $name token with the given name for the created item. Below the *$name* item you can see a variety of folders and items, which will be created below the *$name* root to create an entire subset of content below the newly created item.



6.2 Using Branch Templates

You can use branch templates to help users to create multiple items using a reusable, predefined structure. Branch templates can also be used to copy initial field values into the created item (rather than inherit them from the standard values). Note that this includes both field values for the Standard template fields (which are assigned via the ribbon controls, for example, the item's icon or access right assignments) and fields defined in custom data templates (such as a product number).

Branch templates can also be useful when you want content authors to be able to create multiple items at once (either siblings or descendants). Multiple content sub-trees can be created using branch templates.

6.2.1 What happens?

When the user invokes the branch template, the system carries out several actions:

- It copies the descendants of the branch template definition item, including all field values, to create new items.
- It performs token substitution on the new items, replacing *$name* and other tokens in both item names and field values with the name entered by the user when invoking the branch template.
- In branch templates consisting of one child item with zero or more descendants, the name of that child is typically *$name* so the user can specify the name of the root item to create. For item names in branch templates, only *$name* is supported
- For any field not overriding its standard value in the branch template, the corresponding field in the new items contains its standard value, unless the standard value contains a token such as *$name*, which Sitecore also replaces with the given item name.

Note
Field values in branch templates could easily result in data duplication. Therefore template standard values are generally preferable to field values in branch templates. You should also remember that field values in branch templates are not inherited like template standard values, but are copied. Therefore, changes made to a branch after creating items using that branch are not reflected in the items previously created.

Chapter 7

The Command Template

This chapter describes the structure and purpose of command templates.

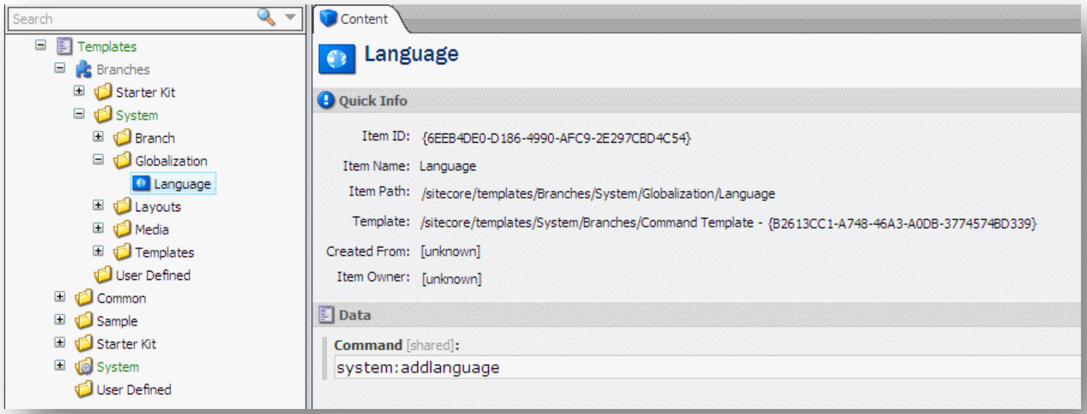
This chapter contains the following sections:

- Understanding Command Templates
- Using Command Templates

7.1 Understanding Command Templates

Command templates define a class and method to be called during an insert operation. Unlike data templates and branch templates, which consist of predefined structures, command templates reference Sitecore UI commands to invoke wizards or other logic used to create new items.

In the following image, you can see the *Language* command template that is used when you create new languages. In the right-hand pane you can see the **Command** field which contains the name (*system:addlanguage* in this case) of the command that is called when the command template is invoked.



Item ID: {6EEB4DE0-D186-4990-AFC9-2E297CBD4C54}
Item Name: Language
Item Path: /sitecore/templates/Branches/System/Globalization/Language
Template: /sitecore/templates/System/Branches/Command Template - {B2613CC1-A748-46A3-A0DB-3774574BD339}
Created From: [unknown]
Item Owner: [unknown]

Command [shared]:
system:addlanguage

7.2 Using Command Templates

You can use command templates to insert items according to logic rather than predefined structures. Command templates can also be used to insert multiple items when a user invokes an insert option, but command templates provide more flexibility than branch templates.

A command template can be assigned as an insert option to items and standard values. The command template insert option will appear identical to data template and branch template insert options. The only difference is that command template insert option will trigger a Sitecore UI command.

Command templates will typically invoke a wizard application that collects information from a user and then programmatically creates an appropriate set of items.

One example of a Sitecore defined command template involves creating a template:

- The insert options for `/Sitecore/System/Languages` include the `/Sitecore/Templates/Branches/System/Globalization/Language` command template.
- This invokes the command used by the control panel to create a new language.
- The value of the **Command** field in each command template corresponds to an entry in the `/App_Config/Commands.config` file and makes the system to invoke methods in the specified class when the user invokes the command template.