



Sitecore 7.2

Developer's Guide to Item Buckets and Search

A developer's guide to working with item buckets, search, and indexing in Sitecore

Table of Contents

| | | |
|-----------|------------------------------------------------------------------|----|
| Chapter 1 | Introduction..... | 4 |
| 1.1 | Introduction..... | 5 |
| 1.1.1 | Backwards Compatibility..... | 5 |
| 1.2 | Fundamental Concepts..... | 6 |
| 1.2.1 | Item Bucket..... | 6 |
| 1.2.2 | Why Use an Item Bucket?..... | 6 |
| | Viewing Hidden Items..... | 7 |
| Chapter 2 | Configuring Item Buckets..... | 8 |
| 2.1 | Creating an Item Bucket..... | 9 |
| 2.1.1 | Item bucket Icon in the Quick Actions Bar..... | 10 |
| 2.2 | Making Content Items Bucketable..... | 11 |
| 2.2.1 | Making a Template Bucketable..... | 12 |
| | Changing a Bucketable Template to a Non-Bucketable Template..... | 13 |
| 2.2.2 | Changing the Bucketable Settings..... | 13 |
| 2.2.3 | Synchronizing an Item Bucket..... | 14 |
| 2.2.4 | Locking Parent/Child Relationships..... | 14 |
| 2.3 | Managing Item Buckets..... | 16 |
| 2.3.1 | Building the Search Indexes..... | 16 |
| 2.3.2 | Clearing the HTML Cache..... | 16 |
| 2.3.3 | Item Bucket Settings..... | 17 |
| Chapter 3 | Searching..... | 19 |
| 3.1 | Configuring Search..... | 20 |
| 3.1.1 | Specifying which Fields are Displayed in the Search Results..... | 20 |
| 3.1.2 | Specifying a Search Result Image and Search Result Text..... | 20 |
| 3.1.3 | Displaying Media Library Images in Search Results..... | 21 |
| 3.1.4 | Viewing the Search Results..... | 22 |
| | Different Ways to Display Search Results..... | 22 |
| 3.1.5 | Exclude Current Item from Search..... | 23 |
| 3.2 | Using Facets to Refine your Search..... | 24 |
| 3.2.1 | Language Search..... | 25 |
| 3.2.2 | Complex Searches..... | 25 |
| | Searching within a Range..... | 25 |
| | Combining, <i>and</i> , <i>or</i> , and <i>not</i> | 26 |
| 3.2.3 | Opening Items in the Search Results..... | 26 |
| 3.3 | Using Search Filters..... | 29 |
| 3.3.1 | Auto-Organizing..... | 31 |
| 3.3.2 | Paging Results..... | 31 |
| 3.3.3 | Predefined Search Options..... | 32 |
| 3.3.4 | Default Search Query..... | 33 |
| 3.3.5 | Persistent Search Query..... | 34 |
| 3.4 | Security and Item Buckets..... | 35 |
| | Locking..... | 35 |
| 3.4.1 | Identification and Authentication Modifications..... | 35 |
| | Keyboard Shortcuts..... | 36 |
| 3.5 | Using a Custom Class to Create a Query..... | 37 |
| 3.6 | Using Item Buckets with the Data Source of a Control..... | 38 |
| | Tips..... | 38 |
| Chapter 4 | Sitecore DMS and Item Buckets..... | 40 |
| 4.1 | Personalization and MV Tests..... | 41 |
| 4.1.1 | Setting the Data Source..... | 42 |
| 4.2 | Inserting and Managing Links..... | 43 |
| 4.2.1 | Inserting a Link in the Rich Text Editor..... | 43 |
| | Inserting a General Link with Search..... | 44 |
| 4.3 | Tagging Associations across Many Items..... | 45 |

| | | |
|-----------|-------------------------------------------------------|----|
| 4.3.1 | Creating a Tag..... | 45 |
| Chapter 5 | Developing with Item Buckets | 46 |
| 5.1 | New Field Types | 47 |
| | Multilist with Search Field..... | 47 |
| | General Link with Search Field | 48 |
| | Treelist with Search Field..... | 48 |
| 5.2 | Creating a Tag Repository | 49 |
| 5.3 | LINQ to Sitecore..... | 50 |
| | Complex Searches | 53 |
| | Adding a New Linq Provider..... | 53 |
| 5.4 | Adding a New Search Provider..... | 55 |
| | New Logging Classes | 56 |
| | Query Warm-up..... | 56 |
| 5.4.1 | Pipelines..... | 57 |
| 5.4.2 | Miscellaneous..... | 59 |
| 5.5 | Linq to Provider | 60 |
| 5.5.1 | Accessing the Linq to Sitecore API | 60 |
| 5.5.2 | Custom Search Type / Object Mapping | 60 |
| 5.5.3 | Supported IQueryable methods | 62 |
| 5.5.4 | IQueryable Extensions | 64 |
| | Filtering..... | 64 |
| | Facets..... | 64 |
| | Other | 65 |
| 5.6 | Searching | 67 |
| 5.6.1 | Searching in the Default Language..... | 67 |
| 5.6.2 | Searching and Facets | 67 |
| 5.6.3 | Using a Field as a Tag Repository | 67 |
| 5.6.4 | Including and Excluding Search Filters | 68 |
| 5.6.5 | Editing Search Filters | 68 |
| 5.6.6 | In-Memory Index | 68 |
| 5.6.7 | Applying Quick Actions to Search Results | 69 |
| | Add New Quick Actions..... | 70 |
| 5.6.8 | Showing Dynamic Fields in Search Results | 71 |
| 5.6.9 | Adding New Filters and Setting up Alias Filters | 71 |
| | Alias Filters..... | 72 |
| 5.6.10 | Creating a New Search Facet | 72 |
| 5.6.11 | Default Bucket Queries | 72 |
| 5.6.12 | A Persistent Bucket Filter | 72 |
| 5.6.13 | Default Queries and Filters | 72 |
| 5.7 | Rule-based Boosting..... | 74 |
| 5.7.1 | Creating a New Boosting Rule Condition..... | 75 |
| 5.7.2 | Implementing Rule-Based Boosting for Fields..... | 75 |
| 5.8 | Multiple Index Support | 77 |
| 5.9 | Adding a New View | 78 |
| Chapter 6 | Configuration and Tuning..... | 79 |
| 6.1 | Configuration Files | 80 |
| 6.2 | Scaling Test Tool | 81 |
| 6.3 | Index Analyzer..... | 82 |
| 6.4 | Scaling with Placeholders | 83 |
| Chapter 7 | Appendix | 84 |
| 7.1 | Tips and Tricks | 85 |
| 7.2 | Default Fields in Lucene..... | 88 |

Chapter 1

Introduction

This document describes how to set up, configure, and tune search and indexing in Sitecore 7.2.

The document contains the following chapters:

- **Chapter 1 — Introduction**
This chapter is an introduction to search and item buckets and describes the fundamental concepts.
- **Chapter 2 — Configuring Item Buckets**
This chapter describes how to set up and configure Item Buckets.
- **Chapter 3 — Searching**
This chapter contains practical advice on configuring and using the module from an administrator's perspective.
- **Chapter 4 — Sitecore DMS and Item Buckets**
This chapter describes how to use Item Buckets when setting up, for example, MV tests.
- **Chapter 5 — Developing with Item Buckets**
This chapter describes how to develop with Item Buckets, and how to use the API.
- **Chapter 6 — Configuration and Tuning**
This chapter describes the configuration files, and some tools for tuning performance.
- **Chapter 7 — Appendix**
This chapter contains advice about upgrading existing solutions to using Items Buckets, as well as some other tips and tricks.

1.1 Introduction

Item Buckets is a system that lets you store millions of content items in one container. You can convert individual items in the content tree into item buckets that can contain any number of subitems. These subitems are not listed in the content tree and do not have a direct parent-child relationship with the item bucket item.

You can search in each item bucket to find the content items that you are interested in.

Item buckets allow content authors to:

- Hide content items in the content tree.
- Use the search functionality to retrieve content items from the item buckets.
- Use the search functionality to set the value of fields in content items.
- Alter the parent-child relationship of content items.

Important

Converting content items into item buckets removes the hierarchy of that content and can cause your code not to work as intended if the code depends on the hierarchical structure. For more information about coding and item buckets, see the chapter *Developing with Item Buckets*.

You don't have to use the item buckets functionality when you install Sitecore. The buckets system only starts to work when you create the first item bucket.

In an item bucket, you can create a hybrid structure that consists of content items that are hidden in the item bucket and content items that are structured in the normal way.

You can also define a sub-structure within an item bucket.

1.1.1 Backwards Compatibility

The old Sitecore search API — `Sitecore.Search` — has been retained for backwards compatibility.

We recommend that you use the new search API — `Sitecore.ContentSearch`.

1.2 Fundamental Concepts

This section explains some of the basic concepts used in item buckets.

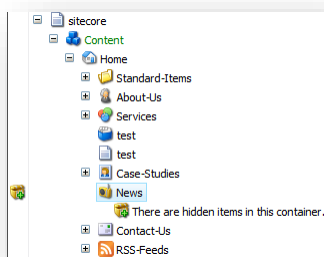
1.2.1 Item Bucket

An item bucket is a repository in the content tree that can store other content items. The difference between an item bucket and a regular container in the content tree is that an item bucket can store a theoretically unlimited amount of items without displaying them in the content tree.

The items in an item bucket are by default hidden in the content tree, which is why you no longer can use the content tree to navigate to and select the items. Instead you can search for and open the items in an item bucket from the Sitecore search engine available from every item in the Content Editor or from the ribbon in the Page Editor.

Furthermore, the parent to child relationship between the content items in an item bucket is completely removed and instead the items are automatically organized into folders. By default, the items are organized according to the date and time of when the item was created, but this can be configured to use different behavior.

To provide the content author with a helpful overview of which containers are item buckets and which are normal containers, an item bucket icon can be enabled in the Quick Action Bar to the left of the content tree in the Content Editor. Additionally, if you expand an item bucket in the content tree you will also see a visible cue, that the container is an item bucket. If the items in an item bucket are hidden, a small notification tells you that there are hidden items in the container.



Using item bucket has many advantages, including:

- Automatically organizing all the content items in an item bucket in a logical format, so that the performance of the search engine increases.
- A single item bucket can contain millions of content items without slowing down the UI or congesting the content tree.
- You can have as many buckets as you want. This is useful if you want to split up your buckets into logical containers for example one for products and one for articles.

1.2.2 Why Use an Item Bucket?

An item bucket addresses the problem of managing large numbers of items within the content tree, retrieving them, and working with them in a speedy and efficient manner. To decide if you should turn an item into an item bucket, and in-turn, hide all its descendants, you must ask yourself if you care about the structure of the data that is stored in the item bucket.

For example, if you have items within the content tree that contain a large number of sub items such as products, media, or tags, it may be an advantage to turn these items into item buckets and thereby remove the need for hierarchically managing the content.

Important

Note that a connection between two or more items does not necessarily need to be hierarchically.

Then when you want to work with a particular item placed in an item bucket, you can search for it and open it. The advanced Sitecore search functionality allows you to search among all items in Sitecore using for example free text, search filters, or facets, that makes it easier for you to find exactly what you need.

Viewing Hidden Items

Even when the bucketable items are hidden in the item bucket you have the option to view the items anyway by selecting the **Bucket** check box in the **Content Editor**, on the **View** tab, in the **View** group.

However, we recommend that you clear the **Buckets** check box when working with item buckets. This will prevent the system from unnecessarily loading all the items in the content tree. Remember that you can use the search tab to find and work with the hidden content items.

Chapter 2

Configuring Item Buckets

This chapter describes how to convert a content item into an item bucket that can contain thousands of items.

This chapter contains the following sections:

- Creating an Item Bucket
- Making Content Items Bucketable
- Managing Item Buckets

2.1 Creating an Item Bucket

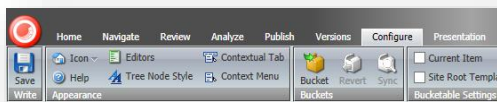
Content items that are stored in item buckets are just like any other content items — you can create, edit, and delete them.

When you convert a content item that already exists into an item bucket, the item bucket organizes and hides all its descendants if they are based on templates that are bucketable or if the item itself is set as bucketable. Depending on how many items it contains, it can take a considerable amount of time to organize the items after converting the item into an item bucket. A progress bar appears displaying a running tally of the items being processed. During the bucketing process it is possible to cancel the construction of the item bucket, in case you regret before the organizing of the items are complete.

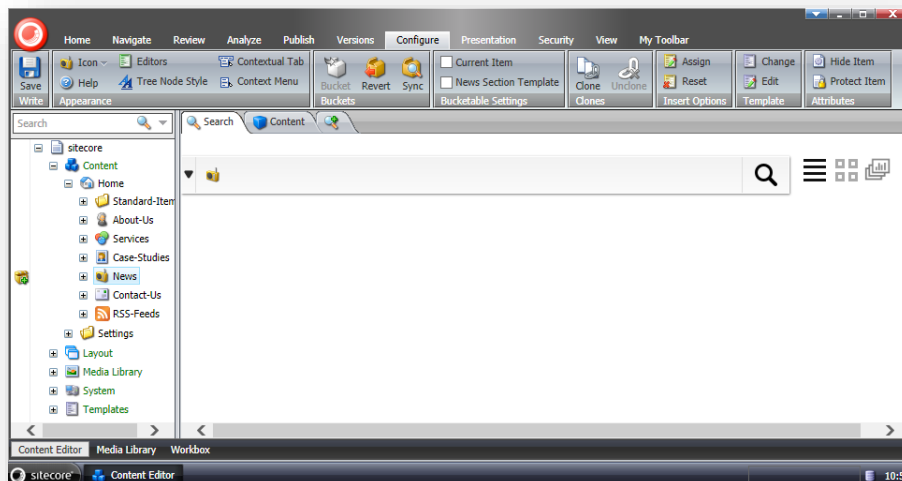
You can create an item bucket from a new content item or convert an existing item structure into an item bucket.

To create an item bucket:

1. In the **Content Editor**, in the content tree, create a content item, for example a folder, and give it a suitable name.
2. In the content tree, select the content item and then on the **Home** tab, click **Edit** to lock the item.
3. Click the **Configure** tab and then in the **Buckets** group, click **Bucket** to convert the new item into an item bucket.



When you convert a content item into an item bucket, a new **Search** tab appears in the right-hand pane.



You use this tab to search for content items in the item bucket.

2.1.1 Item bucket Icon in the Quick Actions Bar

In the Quick Action Bar you can enable an item buckets icon, to indicate which content items in the content tree are item buckets.

To display the item buckets icon, right-click the quick action bar left of the content tree, and select **Item Buckets**.

2.2 Making Content Items Bucketable

When you set up an item bucket, you must ensure that the content items that you want to store in the item bucket are bucketable.

To make a content item bucketable, you can:

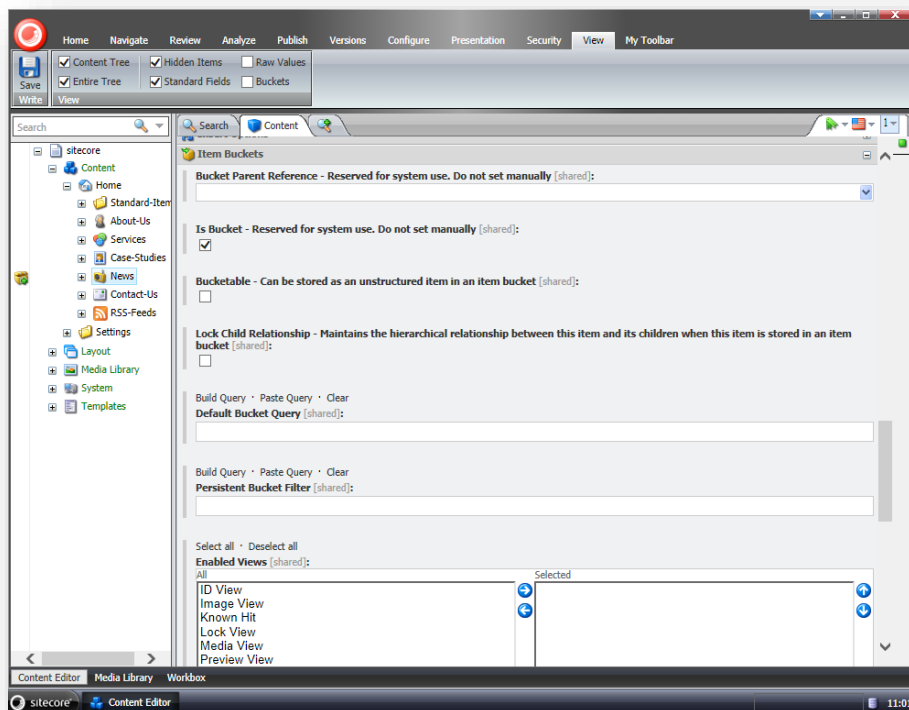
- Make the individual content item bucketable.
- Make the template standard values that it is based on bucketable.

Content items that are bucketable are hidden and searchable when they are stored in an item bucket.

If the content items are based on a template that is not bucketable, the system will not automatically structure and hide the content items for you. Instead, the content items are treated like normal items in the content tree.

To make a content item bucketable:

1. In the **Content Editor**, on the **View** tab, in the **View** group, select the **Standard Fields** check box.
2. Select the content item that you want to make bucketable.
3. In the right hand pane, click the **Content** tab and scroll down and expand the **Item Buckets** section.



4. Select the **Bucketable** check mark.
5. Save your changes.

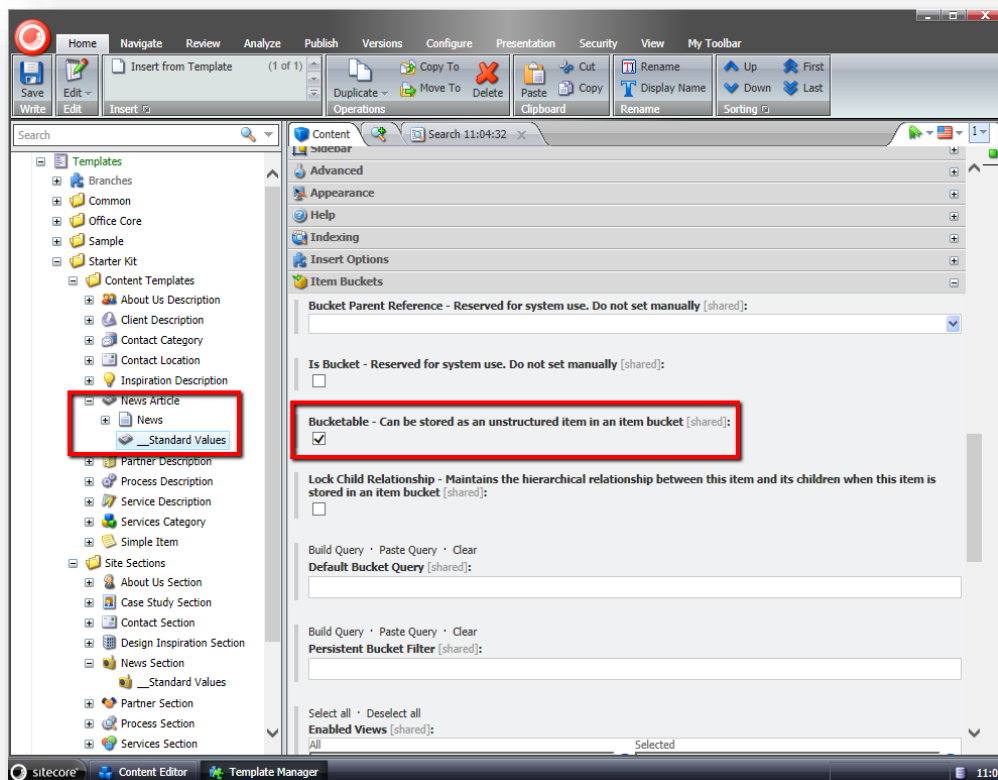
After you have made the content item bucketable, you must synchronize the item bucket and update its structure. For more information about synchronizing an item bucket, see the section *Synchronizing an Item Bucket*.

2.2.1 Making a Template Bucketable

If you have a large number of similar content items that you want to hide in an item bucket, it makes more sense to make the template that they are based on bucketable.

To make a template bucketable:

1. In the Content Editor, on the **View** tab, in the **View** group, select the **Standard Fields** check box.
2. Select one of the content items that you want to make bucketable.
3. In the right-hand pane, on the **Content** tab, expand the **Quick Info** section.
4. Click the template link and the template that this content item is based on opens in the **Template Manager**.
5. In the **Template Manager**, in the content tree, expand the template in question and select the **_Standard Values** item.
6. In the right hand pane, click the **Content** tab.
7. Scroll down and expand the **Item Buckets** section.



8. Select the check box for **Bucketable - Can be stored as an unstructured item in an item bucket**.
9. Save your changes.

After you have made the template bucketable, you must synchronize every item bucket that contains content items that are based on this bucketable container. This updates their structure and hides the bucketable items.

10. In the **Content Editor**, select the item bucket folder that contains items based on this template.

11. On the **Configure** tab, in the **Buckets** group, click **Sync**.

If you create any content items based on this template in another folder that is not an item bucket, these items are treated like normal content items and are displayed in the content tree.

Changing a Bucketable Template to a Non-Bucketable Template

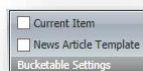
If you change the template of an item in an item bucket from bucketable to non-bucketable, synchronizing the item bucket will not make the item visible in the item bucket. To achieve this, you must revert the item bucket to a normal container and then convert it into an item bucket again.

2.2.2 Changing the Bucketable Settings

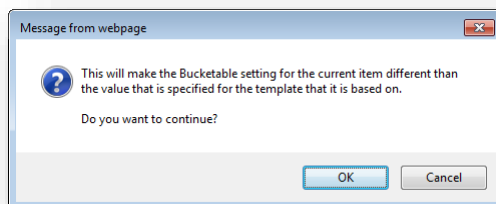
You can change the setting of the Bucketable field of an item and the template that the item is based on.

To change the Bucketable setting for the current item:

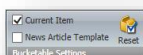
1. In the Content Editor, on the **Configure** tab, in the **Bucketable Settings** group, select the **Current Item** check box:



2. Sitecore shows a confirmation dialog if this makes the setting different from the Bucketable setting of template that the item is based on:



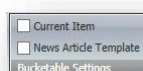
If an item has a different Bucketable setting than the template of the item, the **Bucketable Settings** group has a **Reset** button:



When you click this button, the Bucketable setting of the item is reset so that it is the same as the setting in the template of the item.

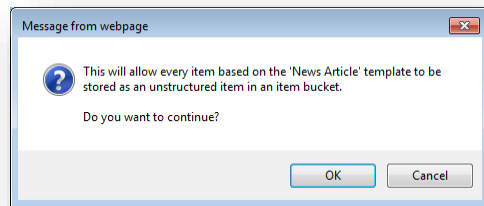
To change the Bucketable setting for the template of the current item:

1. In the Content Editor, on the **Configure** tab, in the **Bucketable Settings** group, select the **News Article Template** check box:



“News Article Template” is the actual name of the template, and it will probably be different for other items than the item in this example.

2. Sitecore shows a confirmation dialog before making changes:



When you select this setting for the template, the setting of the current item is also changed.

2.2.3 Synchronizing an Item Bucket

When you create an item bucket, you can store both bucketable unstructured content items and normal structured content items in it. If you decide to convert some of the normal content items into bucketable items or make the templates that they are based on bucketable, you should always synchronize the item bucket to make sure all the items are organized correctly.

You should synchronize an item bucket when you make items bucketable.

To synchronize an item bucket:

1. In the content tree, select the item bucket whose structure you want to update.
2. On the **Configure** tab, in the **Buckets** group, click **Sync**.

The structure of the contents in the item bucket is now updated:

- The bucketable items are organized and hidden.
- All the content items that are based on bucketable templates are organized and hidden.
- The normal content items remain visible.

You can use the Sitecore search engine to search for all the content items in the item bucket.

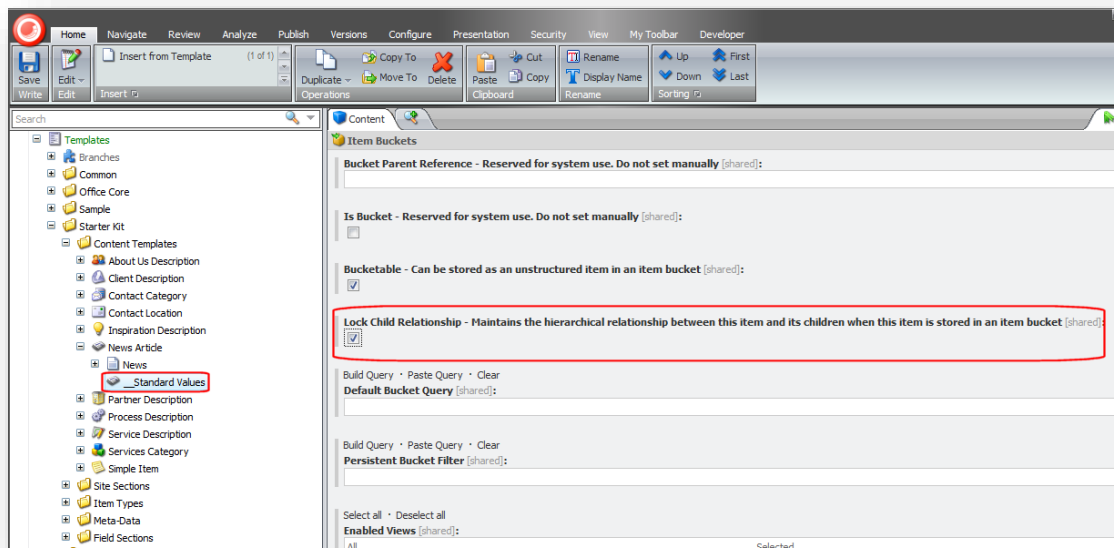
2.2.4 Locking Parent/Child Relationships

In some cases, you may want to lock the relationship between a parent item and its child items even though both are stored in an item bucket. You might need to ensure that the child items are always stored below the parent item, for example, you might want to lock the parent to child relationship between news articles and comments.

To lock the parent to child relationship:

1. In the **Template Manager**, navigate to the template for the parent item. In this case it would be the news article template.
2. Expand the template in the Content Tree, and select the *_Standard Values* item.

3. In the right-hand pane, scroll down to the **Item Buckets** section.



4. Select the **Lock Child Relationship** check box.

If you create a content item that is a child of a content item based on this template, it is not automatically structured in the item bucket. Instead it retains its relationship with the parent item. For example, comments will always be children of the news article that they refer to.

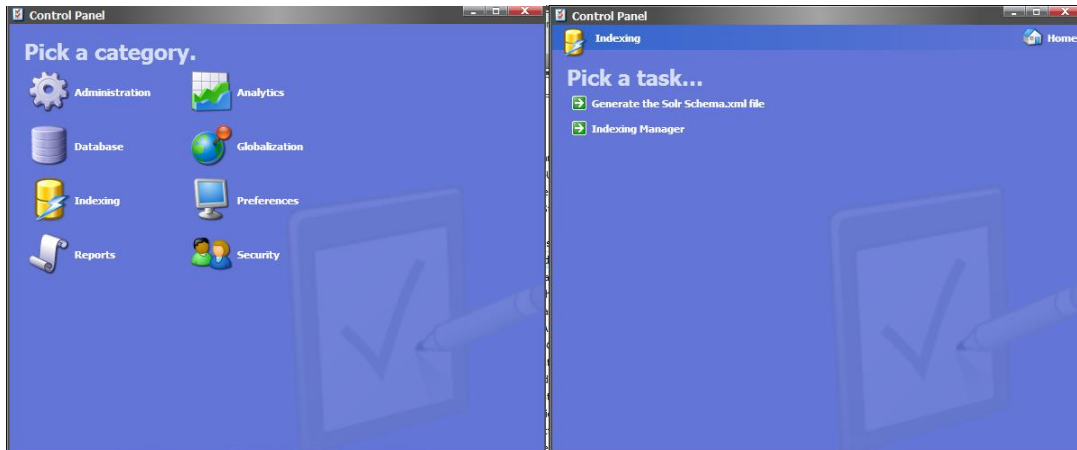
2.3 Managing Item Buckets

There are a number of settings and tools that you can use to configure the way item buckets works on your installation.

These can build the item buckets indexes, specify a number of search settings, set up default search queries, and create facets among other things.

2.3.1 Building the Search Indexes

You can build the search indexes from the Control Panel.



You use the Indexing section in the control panel to:

- Generate a Solr schema.xml file — if you are using SOLR as your provider.
- Rebuild search indexes — only supporting the indexes in Sitecore.ContentSearch.

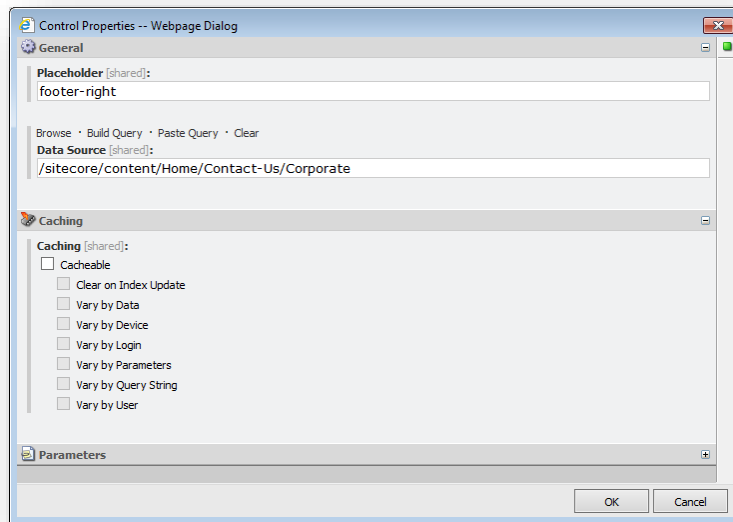
2.3.2 Clearing the HTML Cache

It is possible to have Sitecore clear the HTML cache for presentation components (such as renderings or sublayouts) each time the search index has been updated. This is useful for components that retrieve items or render information from a search index.

This is how to enable it:

1. Select an item in the Content Editor.
2. On the Presentation tab, click **Details**.

3. Select a control in the **Layout Details** dialog box and click it:



4. In the **Caching** section, select the **Clear on Index Update** option.

2.3.3 Item Bucket Settings

There are a number of settings that you use to configure how search works with item buckets. These settings are stored at `/sitecore/system/Settings/Buckets`.

You can use these settings to define various features including:

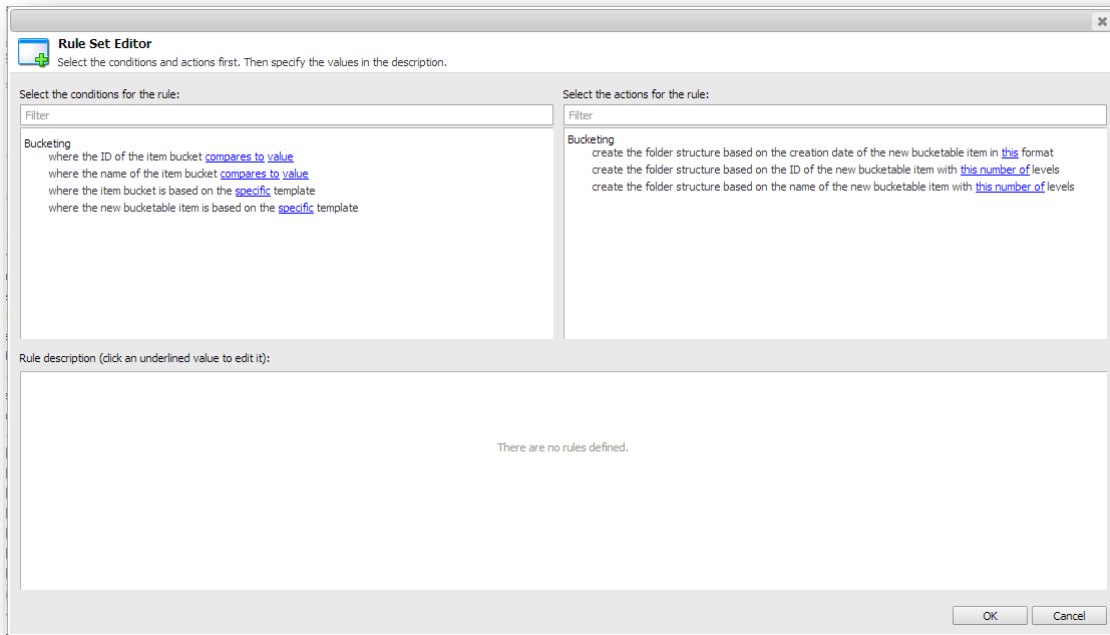
- Changing the keyboard shortcut for searching.
- Defining the facets that are available on your website.
- Specifying the way that an item opens when you click it in the search results.
- Specifying the number of items displayed on a search results page.
- Specify how the folder structure is created.

You can also specify which fields are used when you perform a tag search. To specify which fields are used for tag searches, in the *Item Buckets Settings* item, you must ensure that the **Tag Parent** field points to the item in the content tree that contains all your tags — the tag repository.

You can then create a field called *Tags* in any template and set the type to multi-list.

By default, the items are organized according to the date and time of when the item was created. You can specify a different behavior in the *Item Buckets Settings*. Click **Edit Rule** in the **Rules for**

Resolving the Bucket Folder Path file:



Chapter 3

Searching

Once you start using item buckets, you will soon have buckets that contain hundreds if not thousands of content items. This underlines the need for search functionality that can help content authors and developers find the individual content items that they need to edit and update.

To this end, Sitecore has implemented a new search interface.

This chapter describes how to search for content items.

This chapter contains the following sections:

- Configuring Search
- Using Facets to Refine your Search
- Using Search Filters
- Security and Item Buckets
- Using a Custom Class to Create a Query
- Using Item Buckets with the Data Source of a Control

3.1 Configuring Search

After you have created an item bucket and specified which content items can be stored in it, you can configure how search will work.

You can specify many aspects of how search works including which:

- Facets you can use to filter your search results.
- Fields are displayed in the search results.
- Image is displayed with each content item in the search results.
- Text is displayed with each content item in the search results.

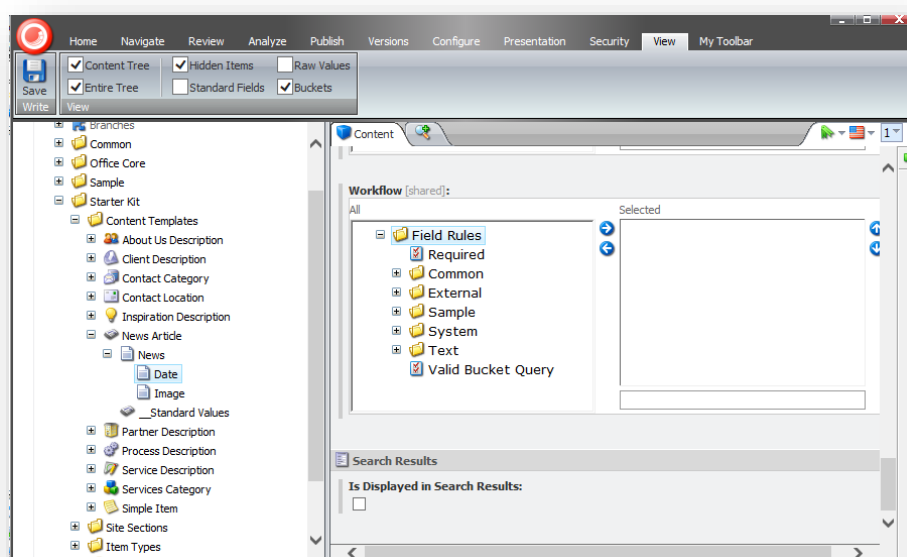
You can also specify a default query that is run every time you open a search pane.

3.1.1 Specifying which Fields are Displayed in the Search Results

You can specify which fields are displayed in the search results when you search an item bucket.

To specify that a field should be displayed in the search results:

1. Open the template in the **Template Manager**.
2. Click the **View** tab and then in the **View** group, select the **Standard Fields** check box.
3. In the content tree, expand the template and select the field that you want to be displayed in the search results.
4. On the **Content** tab, scroll down to the **Search Results** section.



5. In the **Search Results** section, select the **Is Displayed in Search Results** check box.

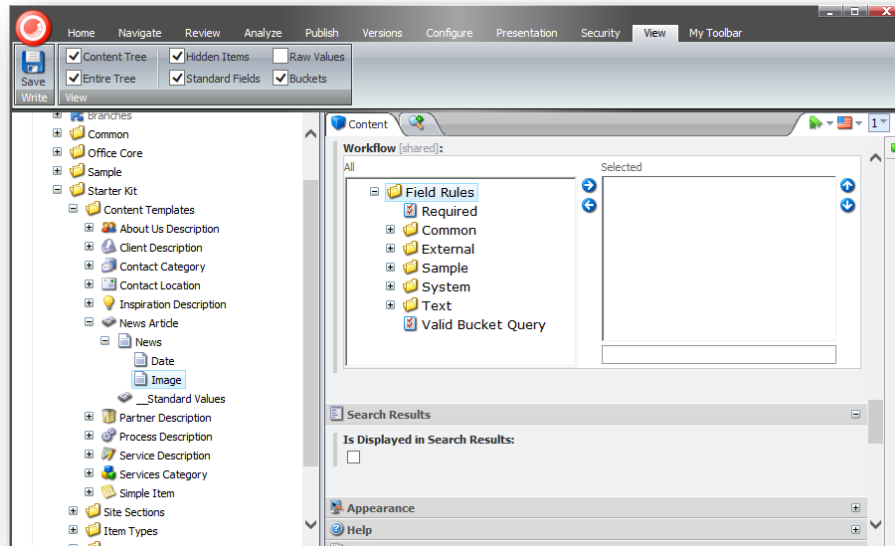
3.1.2 Specifying a Search Result Image and Search Result Text

When you search in an item bucket, Sitecore displays an image with each content item that is listed in the search results. The **Search** tab displays the template icon by default.

You can specify which of the image fields in the template should be shown in the search results.

To specify which image field should be displayed in the search result:

1. Open the template in the **Template Manager**.
2. Click the **View** tab and then in the **View** group, select the **Standard Fields** check box.
3. In the content tree, navigate to the template and expand it.
4. Select the image field that you want to appear in the search results.
5. On the **Content** tab, scroll down to the **Search Results** section.



6. In the **Item Buckets** section, select the **Is Displayed in Search Results** check box.

The image that is displayed in this field is shown in the search results.

You can also use this check box to specify which text field should be displayed in the search results.

These values are cached, so that this search does not have to be run again every time you run a search. It is therefore required that you clear the cache after you have made changes for this by resetting the cache in `/sitecore/admin/cache.aspx`.

3.1.3 Displaying Media Library Images in Search Results

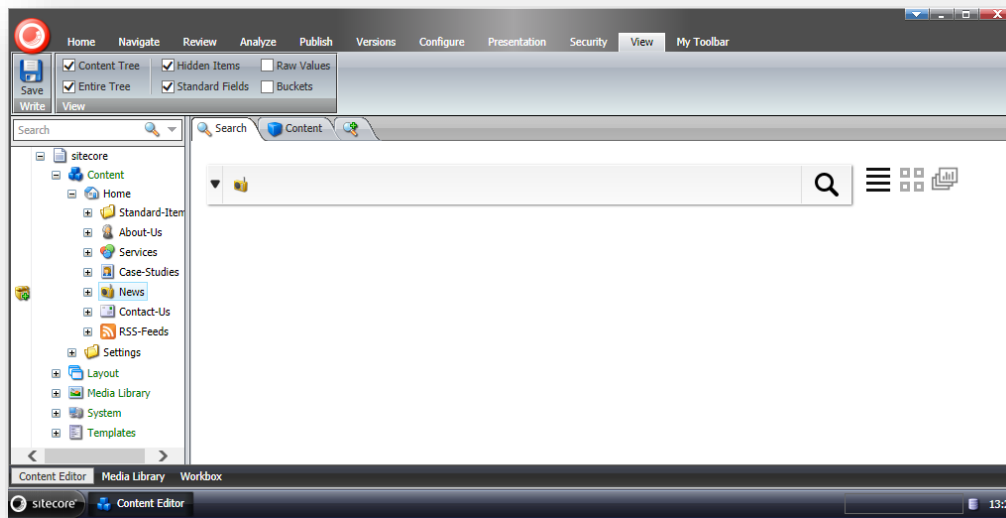
The entire content tree can work with item buckets, including the Media Library. If you need to search for media items and want the images to appear in the search results, you must make sure that the **Is Displayed in Search Results** check box is selected in:

- `/sitecore/templates/System/Media/Versioned/File/Media/Blob` - for Versioned media item
- `/sitecore/templates/System/Media/Unversioned/File/Media/Blob` - for Unversioned media item

After you make these changes, you must use `/sitecore/admin/cache.aspx` to clear the cache.

3.1.4 Viewing the Search Results

When you convert a content item into an item bucket, it automatically inherits the new search interface.



To search for an item in the item bucket, enter a search term that you think the item contains, then press ENTER and a list of search results is displayed. You can return all the content items by typing "*" and pressing ENTER.

You can use the buttons on the right to specify how you want the results displayed.

Different Ways to Display Search Results

Sitecore 7 comes with several different views that you can use to display search results.

The three default views are:

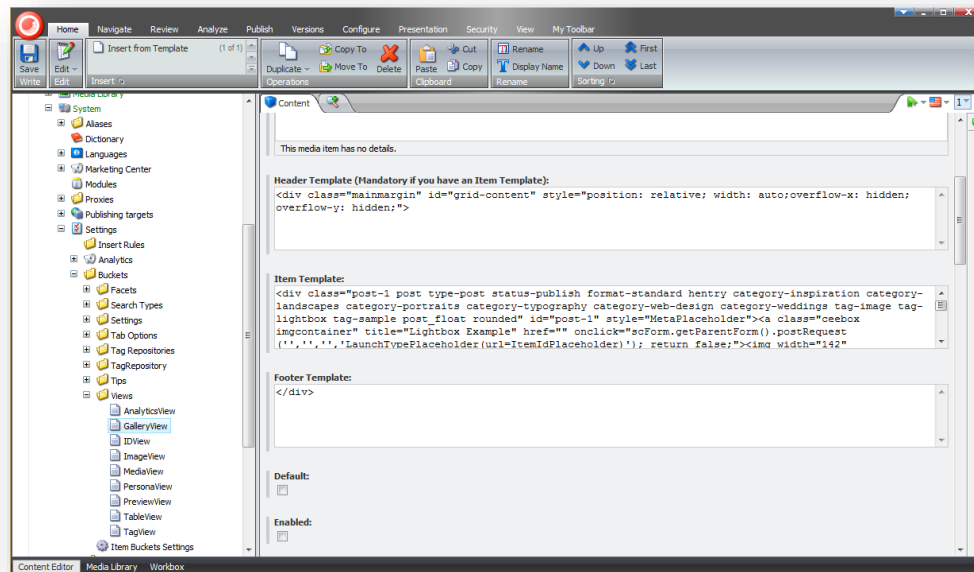
- List — The default view. Items are shown in a plain list.
- Grid — Item information is shown in rectangular cells laid out in rows containing several columns of cells.
- Image — Item images are shown in a grid. Obviously, if an item is an image, this image is shown. Other items are represented by their icons.

You can also use the following views, but you must enable them first:

- ID — The ID View is useful to developers and administrator. It can be used as a quick way of assigning multiple values to a list. When you click the ID view for a search, you get a list of item IDs (GUIDs) that you can copy and paste into a multilist field, for example.
- Lock — Similar to the Grid View, but also shows whether the items in the result set are locked or not.
- Media — Similar to image, but for media such as videos.
- Preview — Show a preview of each item in the result – that is: what this item looks like in context on the website.
- Table — Show the items in the result as a table.
- Tag — This view orders items in the result by tags.

To enable another view:

1. In the **Content Editor**, navigate to `/sitecore/system/settings/Buckets/Views` and select the view that you want to enable.



2. In the **View Details** section, select the **Enabled** check box.
3. If you want this to be the default view, select the **Default** check box. The default view must be enabled.
4. You can now select views for an item by selecting the item in the **Content Editor**, and then scroll down to **Item Buckets** section on the **Content** tab. Here you can select the views that should be enabled for this item.

3.1.5 Exclude Current Item from Search

When you search within an item bucket, the item you search *from* — the item that is selected when the search starts — is included in the search results by default — when it matches the search.

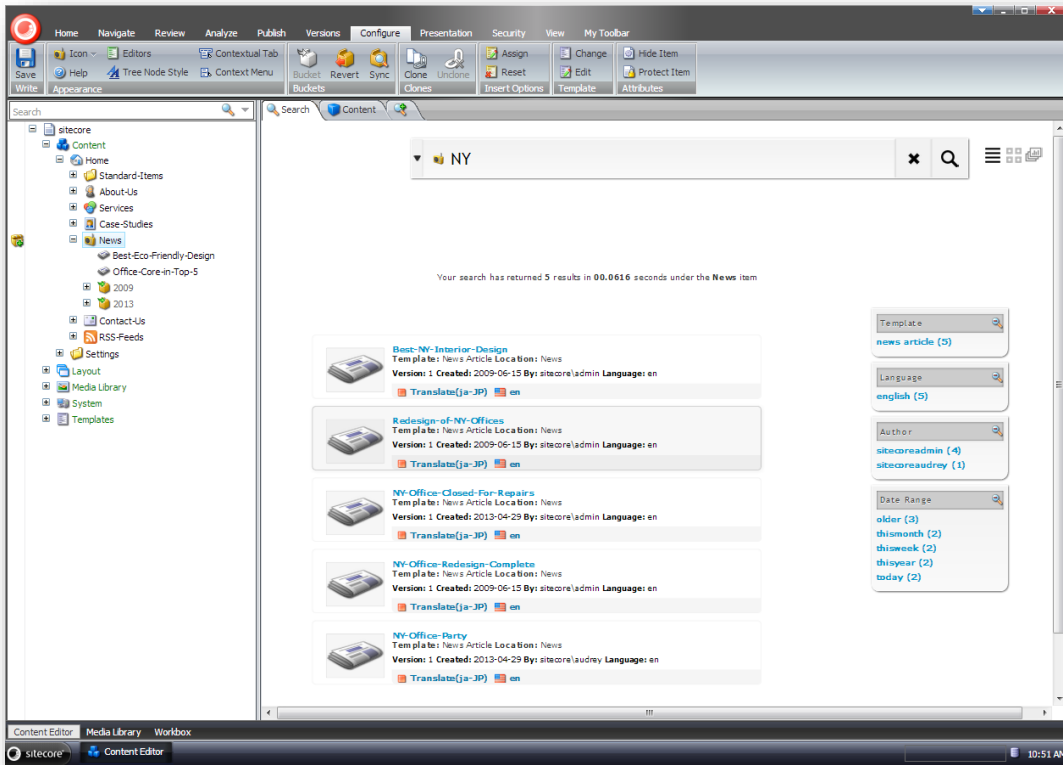
To exclude the current item from the search results, in `Sitecore.Buckets.config` file, set the `BucketConfiguration.ExcludeContextItemFromResult` setting to `true`.

3.2 Using Facets to Refine your Search

After you have run a search, you can use facets to filter the results into a smaller and more concise list.

A facet is a characteristic or a way in which a content item can be viewed or classified. For example, you can classify a content item in terms of the language that it is in, the date on which it was created, or the template that is based on.

When you run a search query, a facet search is also performed. The facets are listed on the right of the search pane. You can use these facets to filter the search results.



To filter your query, click one of the facet links. For example, in the previous image, the facets show that for the five results returned by this search, they were all based on the *news article* template, they are all in English, four were created by *sitecoreadmin* and one by *sitecoreaudrey*, and they were created at a number of different times.

Sitecore uses the following facets:

| Facet | Description |
|------------------------|------------------------------------------------------------------------------------|
| Author | Groups the results according to the authors who created the items. |
| Author Template | Groups the results according to a combination of author and template. |
| Bucket | Groups the results according to the buckets that they are stored in. |
| Creation Date & Author | Groups the results according to the date items were created and who authored them. |

| Facet | Description |
|-------------------|------------------------------------------------------------------------------------|
| Date Range | Groups the results in three categories — created within a day, a week, or a month. |
| File Size | Groups the results according to the size of the file. |
| File Type | Groups the results according to file type. |
| Image Dimensions | Groups the results according to the dimensions of the images they contain. |
| In Workflow | Groups the results according to the workflow they are in. |
| Language | Groups the results according to language. |
| Language Template | Groups the results according to the languages the template are in. |
| Location | Searches all the bucket locations to see which buckets the results are stored in. |
| Owner | Groups the results according to their owner. |
| Tags | Groups the results according to their tags. |
| Template | Groups the results according to templates. |
| Template Author | Groups the results according to a combination of template and author. |

3.2.1 Language Search

The search functionality supports many different languages including Chinese, Arabic, and non-UTF based characters.

3.2.2 Complex Searches

Sitecore supports complex searches such as wildcard, replacement, and exact text searches.

Examples that are supported:

- Tim*
- *Tim
- *Tim*
- T*m
- T?m
- ?im
- Ti?e
- Ti*e
- “Tim Tim”

To run one of these searches, enter the text in the search box.

If you want to search

Searching within a Range

Sitecore also supports range searches.

Examples that are supported:

- price:[400 TO 500] – use square brackets to include the endpoints in the range.
- price:{ 400 TO 500} – use curly brackets to exclude the endpoints from the range.
- price:[400 TO 500} – square and curly brackets can be mixed. In this example, 400 is included in the range, while 500 is excluded.
- title:[Algeria TO Bahrain]

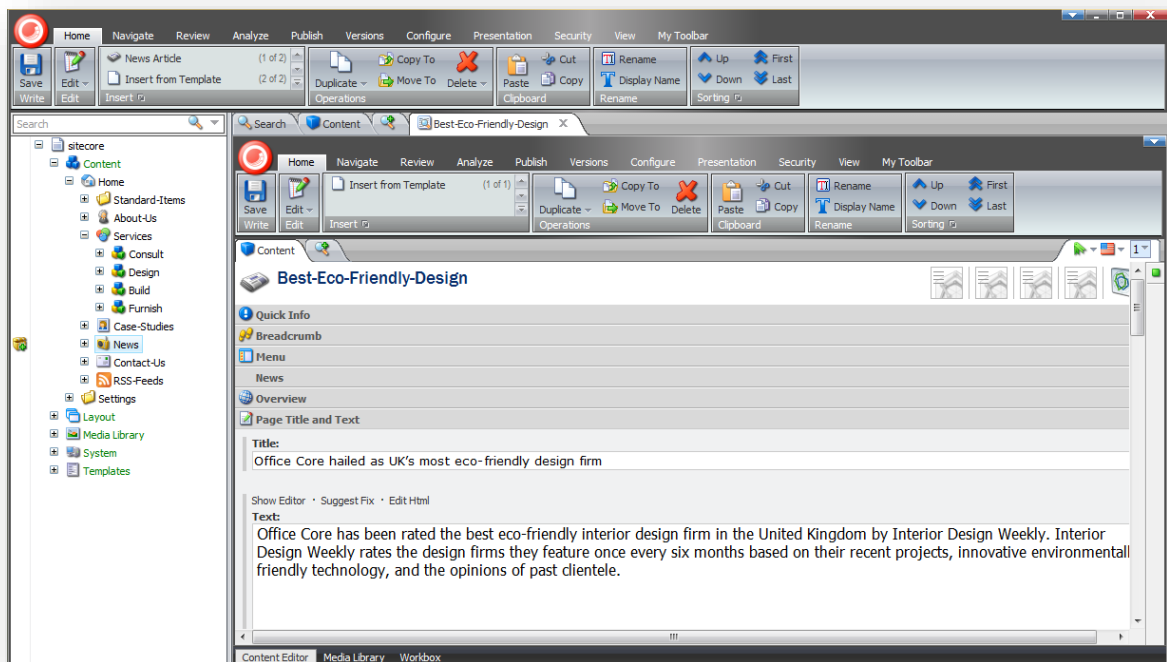
Combining, *and*, *or*, and *not*

Sitecore also supports complex text searches.

For any of the filters you use, you can click on the icon for that filter and it will toggle between that filter using MUST, MUST NOT or SHOULD logic.


3.2.3 Opening Items in the Search Results

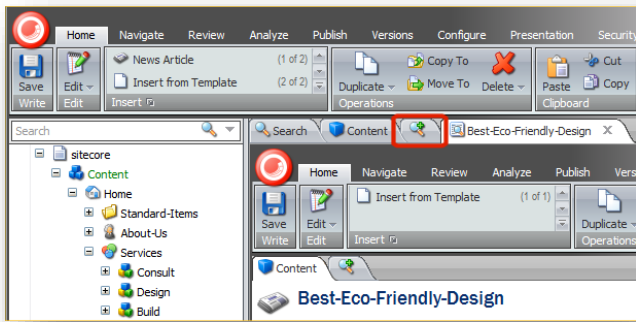
To open an item that appears in the search results, click the image, the title, or any of the links in the results and the item opens in a new tab in the Content Editor.



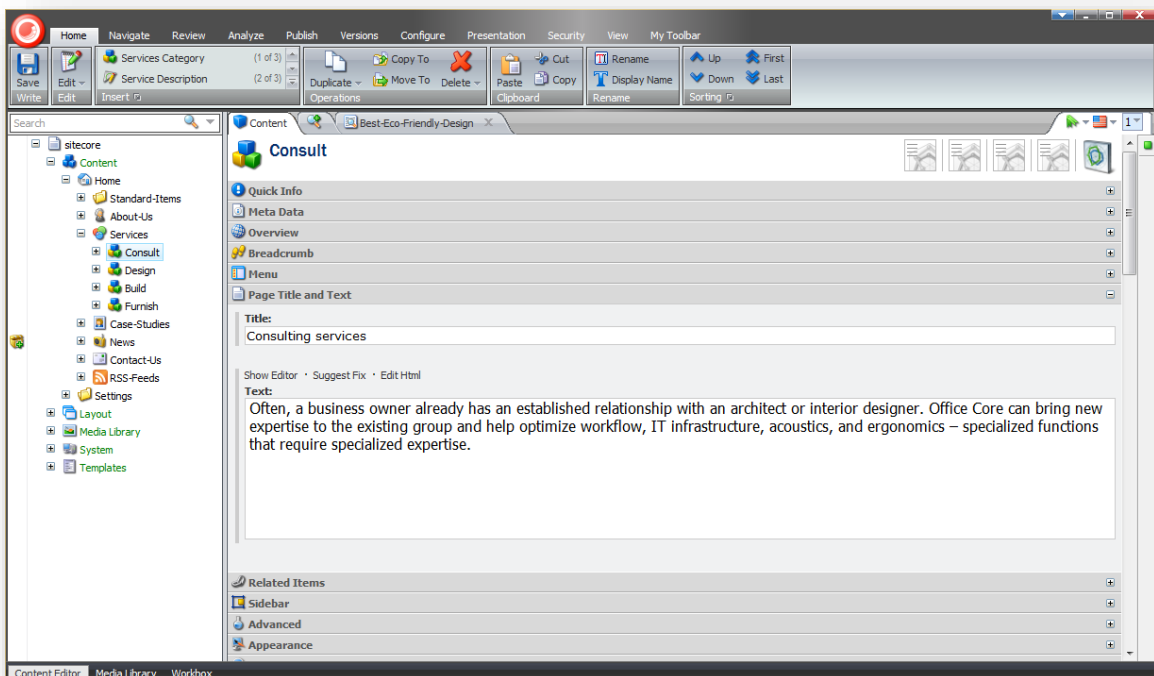
You can edit this item in the new tab. You can open as many extra tabs as you need.

This allows you to open multiple content items at the same time. These content items can come from multiple searches. You can also have multiple search tabs open at the same time. Be aware that search tabs and content tabs are reloaded when you select a different item in the content tree.

To open another search tab, in the editing pane, click the  tab.

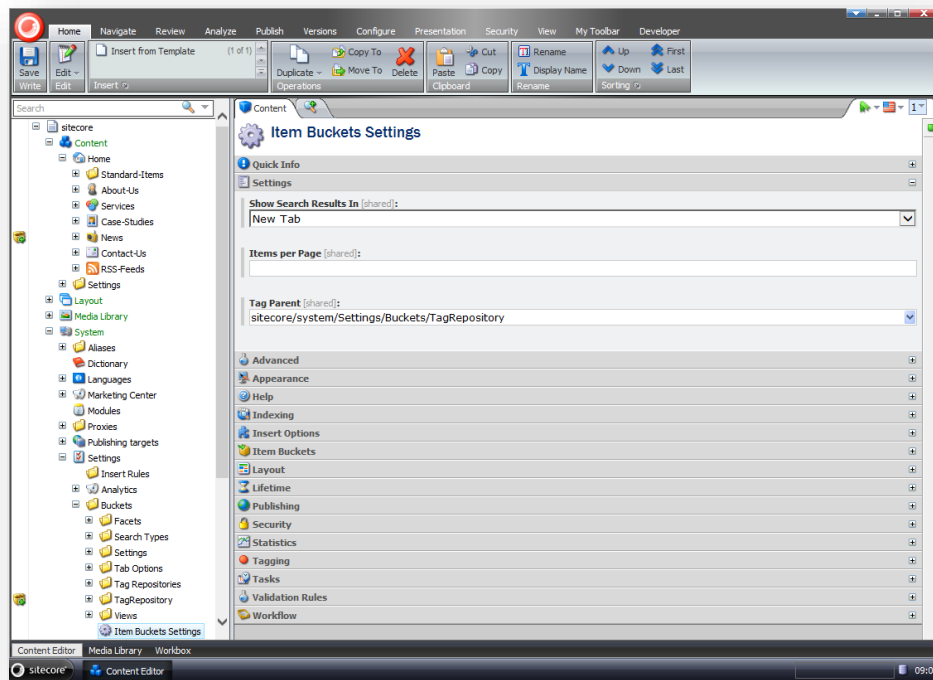


If you select another item in the content tree and open it, all the open tabs persist — but the search results are lost.



When you click a content item in the search results, it opens in a new tab in the Content Editor by default. You can configure the way that search results open in the *Item Bucket Settings* item —

/sitecore/System/Settings/Buckets/Item Buckets Settings



The **Show Search Results In** field contains the following options:

| Option | Description |
|----------------------|-------------------------------------------------------------------------------|
| New Content Editor | The content item opens in a new instance of the Content Editor. |
| New Tab | The content item opens in a new tab. |
| New Tab Not Selected | The content item opens in a new tab. However this tab is not the current tab. |

3.3 Using Search Filters

You can insert filters into a search string to narrow down the results.

To use a search filter, enter the reserved filter keyword and Sitecore either auto-suggests a filter or prompts you to enter a date or a text.

The following filters are supported:

Template Filter



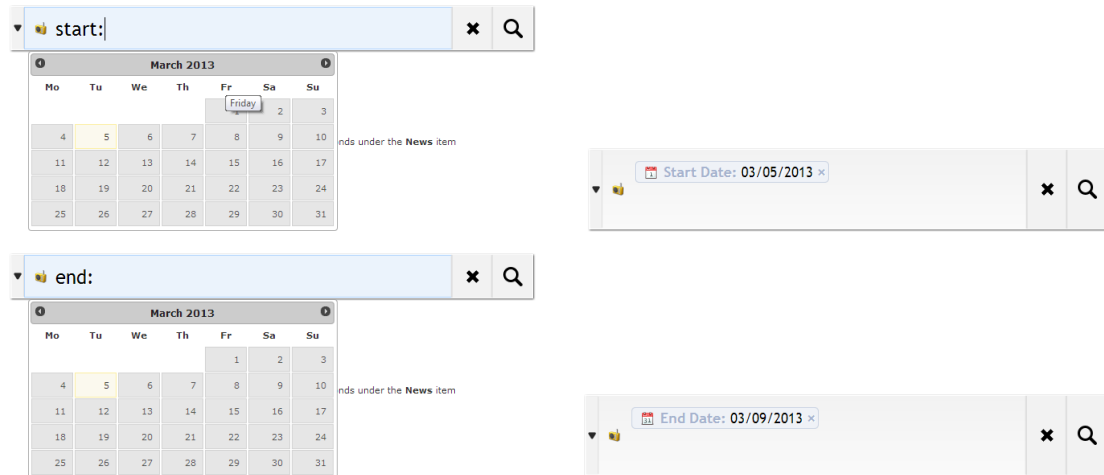
Version Filter



Language Filter



Start and End Date Filter



When you work with the calendar, you can use the following keyboard shortcuts:

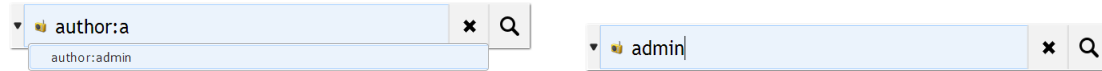
| Keyboard Shortcut | Description |
|---------------------|-----------------------------------------------|
| PAGE UP & PAGE DOWN | Previous month, next month |
| CTRL+PAGE UP/DOWN | Previous year, next year |
| CTRL+HOME | Current month or open when closed |
| CTRL+LEFT/RIGHT | Previous day, next day |
| CTRL+UP/DOWN | Previous week, next week |
| ENTER | Accept the selected date |
| CTRL+END | Close and remove the date |
| ESC | Close the calendar without making a selection |

Note
Depending on browser, browser version, and operating system, these shortcuts may not always be available.

File Type Filter

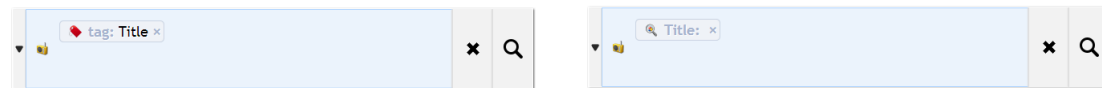


Author Filter

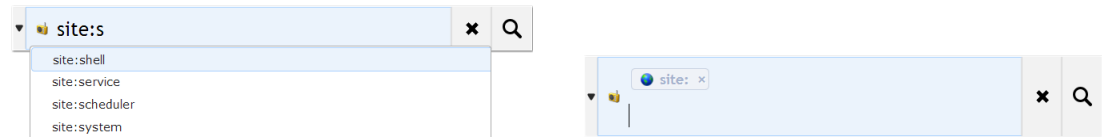


You must enter at least 2 characters before the system makes any suggestions.

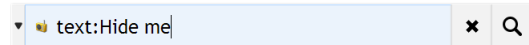
Tag Filter



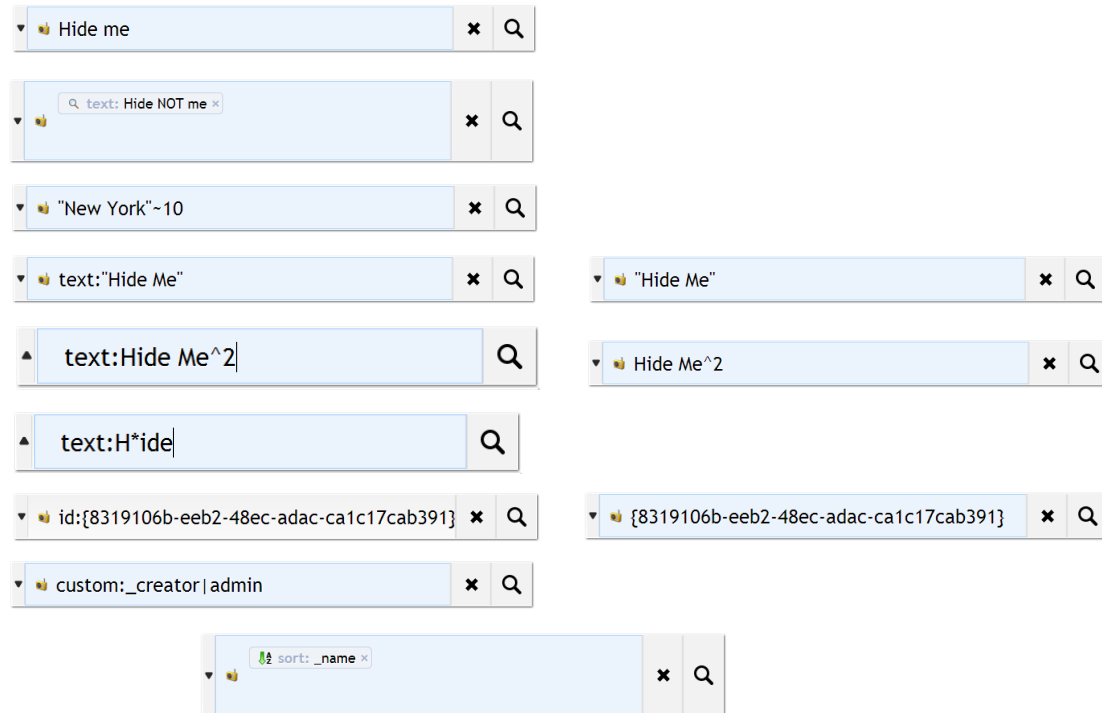
Site Filter



Advanced Text

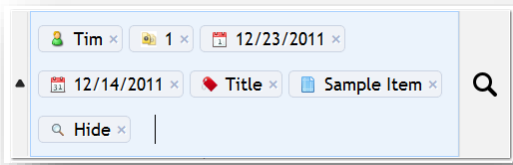


When the *text:* prefix is used, the words (if more than one) are split into individual search terms. When *text:* is not used and there is more than one word, they are turned into *one* search term — as if they were inside a pair of quotation marks.



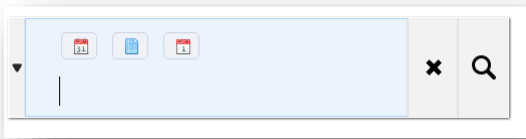
Combining Filters

You can combine various filters by entering them consecutively — they are ANDed:



3.3.1 Auto-Organizing

If you enter too many terms in the drop-down box, the search filters are shrunk automatically.



3.3.2 Paging Results

The search results are displayed in lists of 20 items per page by default, and the number of pages is displayed at the bottom of the tab.

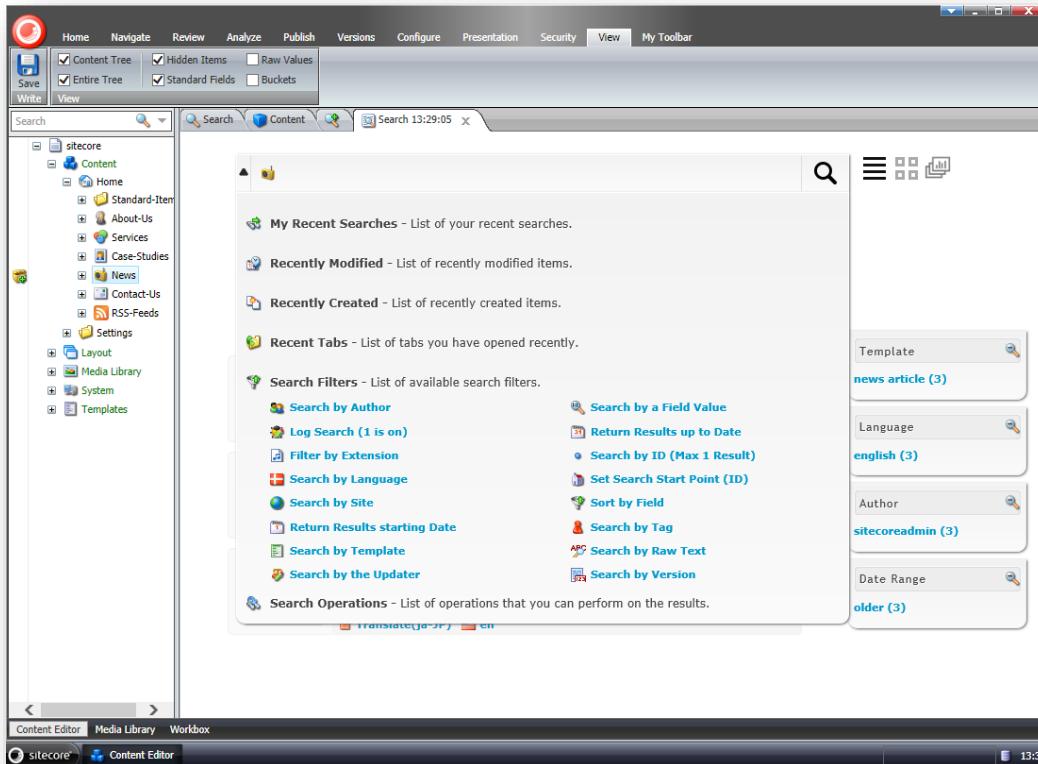
To specify the number of items shown per page, in the `Sitecore.Buckets.config` file, change the value of the `BucketConfiguration.DefaultNumberOfResultsPerPage` setting.

If more than 10 pages are listed, you can move on to the next 10.



3.3.3 Predefined Search Options

When you click the drop-down arrow on the left hand side of the search field, a list of categories appears. You can expand each category to see a more detailed list of search options.



You can add more search options to this list. The search options are stored at:
 /sitecore/system/Settings/Buckets/Settings/Search Box Dropdown.

The search categories are:

| Category | Description |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| My Recent Searches | A list of your recent searches. |
| Recently Modified | A list of the items that have been modified recently. |
| Recently Created | A list of the items that have been created recently. |
| Recent Tabs | A list of the tabs that you have opened recently. |
| Search Filters | A list of filters that you can use in your searches. These filters are stored in /sitecore/system/Settings/Buckets/Search Types/. It is recommended that you store custom search filters in a subfolder called User Defined. |
| Search Operations | A list of operations that you can perform on the search results. This is a powerful feature that lets you run any operation on the search results. You can add more operations to this list. |

The available search operations are:

| Operation | Description |
|------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| Add Tag | Add a single tag to many items at once. |
| Search and Replace | Replace all results text, link etc. values. |
| Clone Results to | Clone all items that were found (a dialog will open) |
| Delete Results | Delete all items that were found (you will <i>not</i> be asked to confirm, so be careful) |
| Publish Items | Publish the items that were found |
| Apply Campaigns Goals Event to All Items | (Only if DMS installed) |
| Apply Presentation to All Items | Apply Presentation to many items |
| Copy to Datasource Query | Copy this search that gave this result in a way so that it can be used as a datasource query |
| Apply Security Rule | Apply a security setting to many items. |
| Copy Results To | Copy all items found (a dialog will open) |
| Move Results To | Move all items found (a dialog will open) |
| Serialize Items | This will save the search results in a file in a subfolder called <code>serialization</code> under the Data folder in the root of the web site. |
| Apply Profile Score to All Items | Add a profile score to many items |

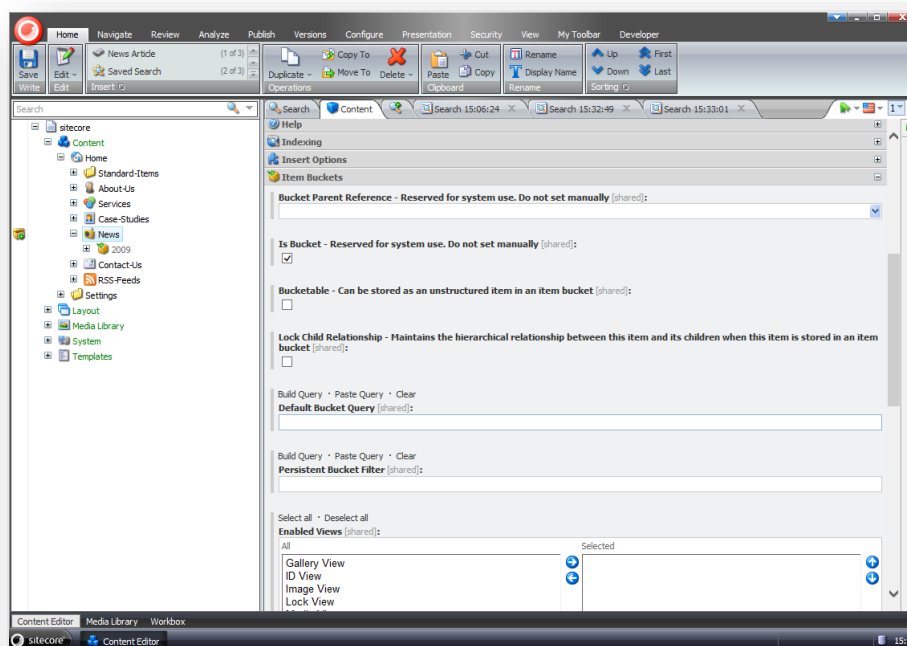
3.3.4 Default Search Query

On every item bucket, you can specify a default query that runs when you open the **Search** tab and displays a list of items that match that query.

To define a default search query:

1. In the content tree, locate the item bucket that you want to define a default query for:
2. On the **Content** tab, expand the **Item Buckets** section.

- In the **Default Bucket Query** field, enter a query, for example `text:new`.



This query finds all the content items in this item bucket that contain the word `new` and displays them on the **Search** tab every time you open it.

On the **Search** tab, you can remove this default query if you want to perform a different search. The default query reappears the next time you open the **Search** tab for this item bucket.

3.3.5 Persistent Search Query

You can also add a query to the search field that you cannot remove and is always a part of your search. This query *cannot* be deleted in the **Search** tab.

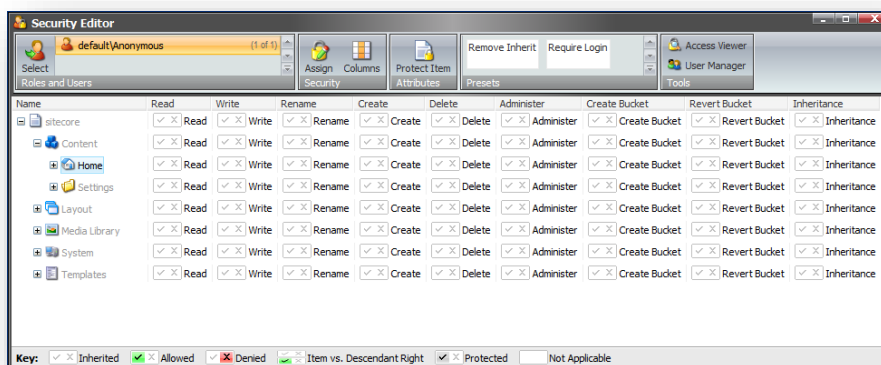
3.4 Security and Item Buckets

If you need to restrict the ability of a user or role to convert a content item into an item bucket or to convert an item bucket back into a normal item, you can use the Security Editor to change their access rights.

The Sitecore security system contains two access rights to support item buckets:

- Create Bucket
- Revert Bucket

If these security settings are not immediately available in the **Security Editor**, in the **Security** group, click **Columns** and then in the **Columns** dialog box, select *Create Bucket* and *Revert Bucket*.



Tip

We recommend that you use the Sitecore security system to prevent particular users from performing certain operations. For example, you do not want someone to accidentally delete a lot of items.

Locking

You must lock a content item before you can convert it into an item bucket or turn an item bucket into an ordinary content item.

To minimize the possibility of accidentally creating an item bucket or making a template bucketable, all users but administrators *must* place a lock on the item before they can place it in a bucket.

3.4.1 Identification and Authentication Modifications

Item buckets drastically improves the way that identification and authentication — IA — is managed on a website. Items that are stored in a bucket no longer maintain a child to parent relationship and the item bucket is simply a pool of items.

Item buckets support all the normal IA operations including:

- Copy To
- Copy From
- Move To
- Move From
- Clone To
- Clone From

- Delete

If you copy or move a content item that is based on a bucketable template into an item bucket, the item is placed in the bucket and automatically organized in the bucket structure.

If you copy or move a content item that is not based on a bucketable template into an item bucket, it is placed in the bucket and is treated like a normal content item.

You can also:

- Drag a copy into an item bucket.
- Drag a copy out of an item bucket.
- Drag an item to move it into an item bucket.
- Drag an item to move it out of an item bucket.

If you delete an item bucket, a message appears informing you that when you delete an item bucket you also delete the content items that are stored in it. You can restore these content items from the recycle bin by restoring the item bucket. If you want to see the content items that are in the item bucket, revert the bucket.

Note

After you restore an item bucket from the Recycle Bin, you must rebuild the search index.

Keyboard Shortcuts

Here is a list of the keyboard shortcuts that you can use with item buckets. Note, however, that there are many possible combinations of browsers, browser versions, and operating systems, and all shortcuts may not always work.

| Shortcut | Description |
|------------------|-------------------------------------------------------------------------------------------------------------------|
| CTRL + SHIFT + B | Converts a content item into an item bucket. You must lock the item first. |
| CTRL + SHIFT + D | Converts an item bucket into a content item. You must lock the item first. |
| CTRL + X | Clears the text box. |
| CTRL + SHIFT + S | Opens a new search tab on the selected content item. This shortcut does not convert the item into an item bucket. |
| CTRL + SHIFT + A | Selects the closest item bucket ancestor of the current item and adds a search tab to it. |
| Space Bar | Scrolls down the results when the search box is not selected. |
| CTRL + Space Bar | Displays the dropdown options when the search box is selected. |
| ESC | This hides the drop-down list if it is shown. |
| CTRL + B | Shifts the focus to the text box if it is outside the textbox. |
| SHIFT + Number | Runs the search in a particular view. 1 will run the first, 2, the second, and so on. |
| 1-9 | The numeric characters move focus to the corresponding page of the search results. |

3.5 Using a Custom Class to Create a Query

To quickly query a field, you can use a compiled class that returns an `Item[]` as the source of your fields. Start your query with the word `code` and then enter the `.class, assembly namespace` as the source field.

You do this by implementing the `IDataSource` interface.

```
code:Sitecore.Namespace.Class, Assembly
```

3.6 Using Item Buckets with the Data Source of a Control

If all the items in a bucket are hidden and cannot be selected, you can choose the data source for a control by specifying a search query. This section describes the syntax for setting the data source to run a query in an item bucket.

Types of queries:

- Template
- Version
- Language
- Creation Start and End Date
- File Type
- Author
- Tag (Facet)
- Site
- Advanced Text
- ID
- Custom

You must place the queries in a semicolon separated list. For example, to search for all the Nicam products, you could specify:

```
+text:Nicam;template:<Product Template ID>;
```

This is passed to your control as a string and you can then use `UIFilters` helper class to create a list of items. Please refer to the Sample Datasource Sublayout in the Layouts folder to see how to use this.

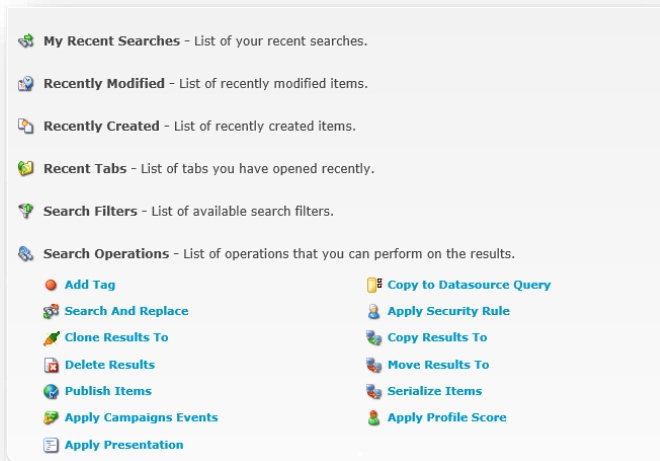
Tips

Sitecore runs the query in the context of its location. For example, if you run a query on a control, Sitecore starts the query from the context item and works through all its descendants. If you need to perform a global search or search in another part of the content tree, you can pass the location parameter to the query for the data source.

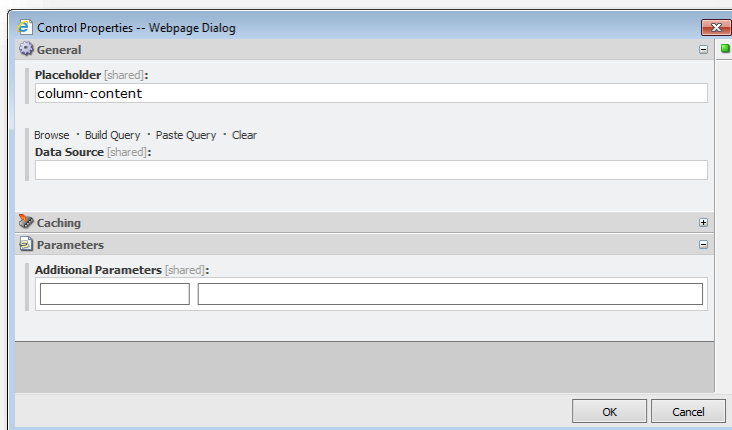
For example, the following query looks for all the items that are tagged *Nicam* and that exist under the `/sitecore/content/home` node — the ID of the home item is shown:

```
tag:{TagId};location:{110D559F-DEA5-42EA-9C1C-8A5DF7E70EF9};
```

To paste queries, search for content in the normal way. Once you have a filter, click the drop down menu and in the **Search Operations** section, click **Copy to Datasource Query**.



When you configure your presentation data source, you can paste the query into a **Data Source** field.



Chapter 4

Sitecore DMS and Item Buckets

As you work with Sitecore, you may need to link to an item that is stored in an item bucket. For example, you may need to insert a link to a bucketable item or use a bucketable item as a variation when you are creating an MV test. To help you locate items in these situations, Sitecore provides search functionality in the appropriate dialog boxes in the Page Editor and in the Content Editor.

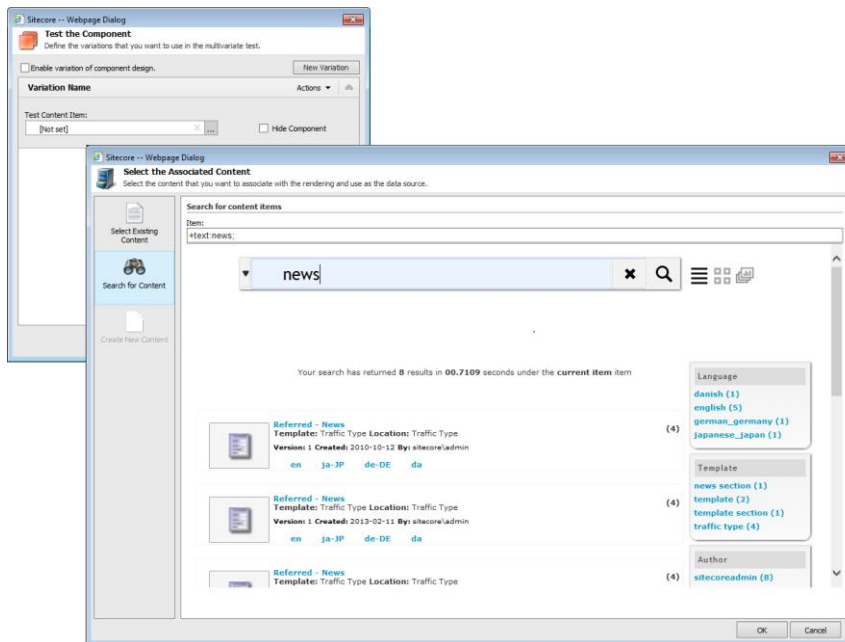
This chapter contains the following sections:

- Personalization and MV Tests
- Inserting and Managing Links
- Tagging Associations across Many Items

4.1 Personalization and MV Tests

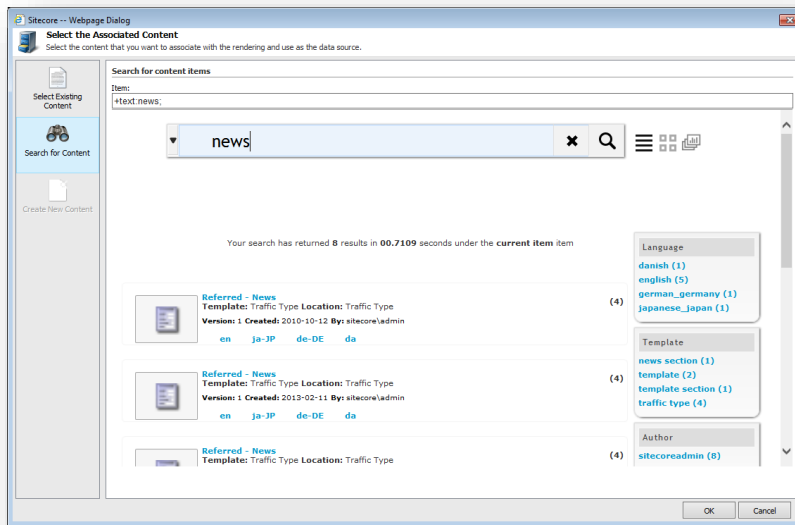
When you set up a personalization rule or create an MV test, you must specify the data source for each variant in the test.

For example, when you create an MV test, you must select the content items that should be used as variants in the test. Selecting a normal content item is straightforward — you browse the content tree and select the item in question. However, this is not so simple when you want to use a content item that is stored and hidden in an item bucket. A search tab has therefore been added to the **Select the Associated Content** dialog box to help you find the items that you need.



4.1.1 Setting the Data Source

When you specify the data source for a control, you can also use the new search functionality to either set an individual item or a list of items as the data source.



4.2 Inserting and Managing Links

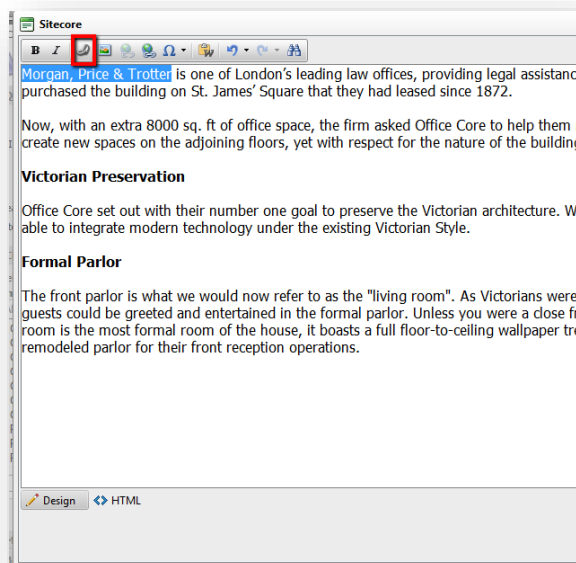
This section describes how to insert and manage links to items that are stored in item buckets.

4.2.1 Inserting a Link in the Rich Text Editor

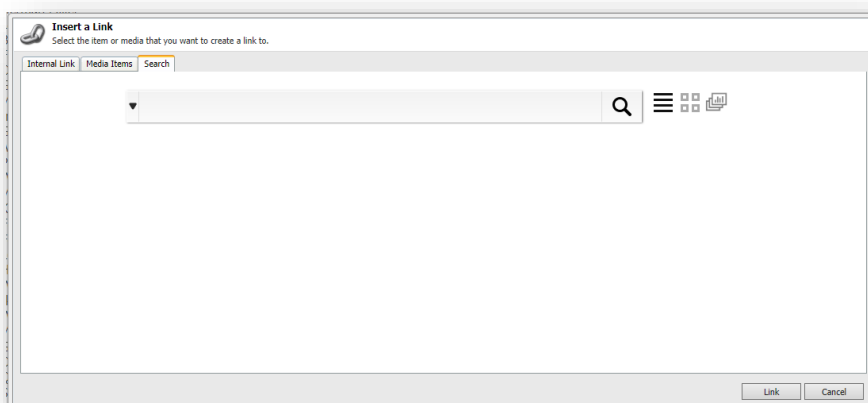
When you edit content items in the Page Editor or in the Content Editor, you often use the Rich Text Editor.

To insert a link into a rich text field:

1. In the **Content Editor**, open the content item that you want to edit.
2. Scroll down to the rich text field that you want to edit and click **Show Editor**.
3. In the **Rich Text Editor**, select the text that you want to use as a link.

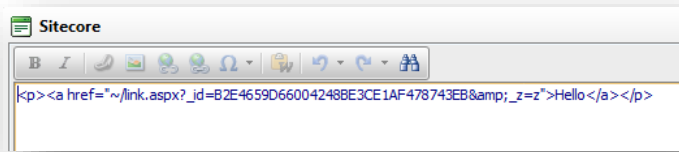


4. Click the **Insert Sitecore Link** button and an **Insert a Link** dialog box appears. This dialog box contains a **Search** tab.



5. Enter the search terms that you want to use.

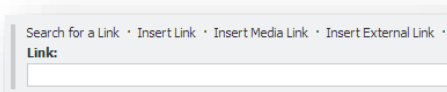
- In the search results, click the item that you want to link to and the link to this item is inserted into the text.



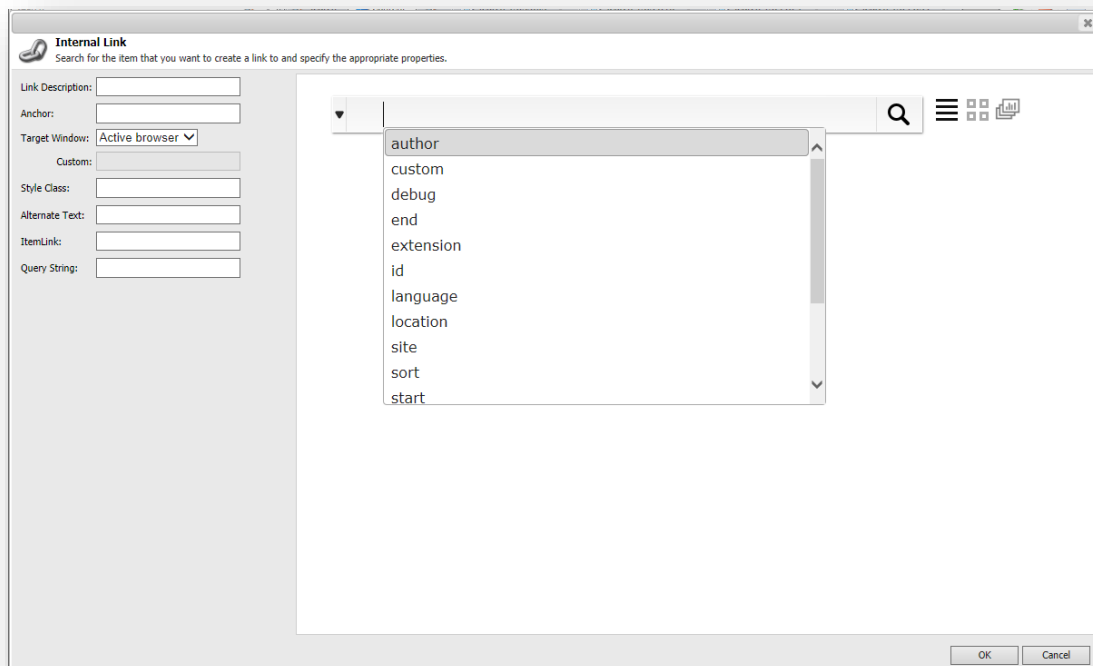
If you want to insert an image that is stored in an item bucket into a rich text field, the **Insert Media Item** dialog box also contains a **Search** tab that you can use to find the image.

Inserting a General Link with Search

The Item Buckets functionality has also introduced some new field types. One of these is the **General Link with Search** field.



To insert a link into a *General Link with Search* field, click **Search for a Link** and the **Internal Link** dialog box opens.



For more information about the *bucket link* field types, see the chapter *Developing with Item Buckets*.

4.3 Tagging Associations across Many Items

Because the parent to child relationship is removed, content items that are stored in an item bucket need a way to connect to other content items. The item buckets functionality supports a semantic tagging system which allows you to tag associations on every item.

To support item tagging, add a field to the template and call it, for example, *tags*. This field should be a *Multilist with Search* field. To comply with best practices, we recommend that you add this field to a base template. If you want to tag every item in the content tree, add this field to the standard template.

For example, if you want to tag media items, set the *tags* field on the *File* template and set the source of the field type to:

```
StartSearchLocation={Tag Repository ID} } (if you are using the sharded approach to indexes then you can add) &IndexName=itembuckets_sitecore
```

This is where semantics come in. You can add any tag to any content item. If you create a tag called *Work in Progress*, you can tag all of the content items that are not yet ready for publication as *Work in Progress*. You can then search for all items that are marked *Work in Progress*.

4.3.1 Creating a Tag

The tags are stored in the `/sitecore/system/settings/buckets/tag_repositories` folder. You can create as many tags in this folder as you want. These tags can be based on any template that you think is appropriate. You can use these items to tag the content items that make up your website.

Sitecore searches on the tag field of an item by default. If you add your own field and would like it to also work with the tag repository, you must add this new field as shown below. You must also go through all the config files and add your field to the indexes so that it is aggregated into the `_tags` field in the index.

```
<fields hint="raw:AddCustomField">
  <field luceneName="_tags" storageType="no" indexType="tokenized">semantics</field>
  <field luceneName="_tags" storageType="no"
indexType="tokenized">mycustomtaggingfield</field>
</fields>
```

Chapter 5

Developing with Item Buckets

This chapter describes how to use develop with item buckets, and how to the Sitecore API to work with item buckets.

This chapter contains the following sections:

- New Field Types
- Creating a Tag Repository
- LINQ to Sitecore
- Adding a New Search Provider
- Linq to Provider
- Searching
- Rule-based Boosting
- Multiple Index Support
- Adding a New View

5.1 New Field Types

There are some new field types available that have been extended so that they can support vast amounts of content without degrading performance.

Sitecore contains a multilist field that can scale to hundreds of items. However, performance starts to degrade when there are more than a hundred items in the field.

Note the following about the parameters in the source field:

StartSearchLocation: This is the location where the search will start from (the place in the content tree).

TemplateFilter: You can specify a Template ID or a pipe delimited list of Template ID and it will filter the result to only those templates.

PageSize: You can set a number, and the result of the search will be returned this many items at a time.

Multilist with Search Field

A Multilist with Search field has no limitation and can scale to thousands of items. We recommend that you use a Multilist with Search field to list items that are stored in an item bucket.

Use this field to attach a search query to a multilist field and display the search results as selectable items. For example, if you want a multilist of all the product items, you can set the source field of the field to `TemplateFilter="Product ID"` and it returns the items in the list.

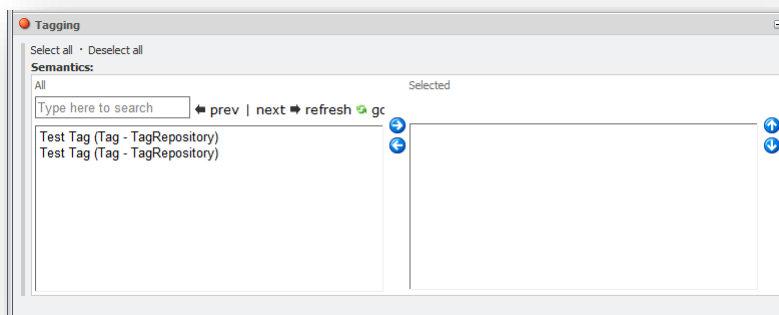
You can use optional filters as in this example:

```
StartSearchLocation=<GUID>&Filter=text:jim
```

You can specify that it is possible for users to change the starting location dynamically. You do this by entering this in the Source field:

```
EnableSetNewStartLocation=True
```

These queries populate the list and you can then select the items from the list. You can also use a search filter to filter the list even more.

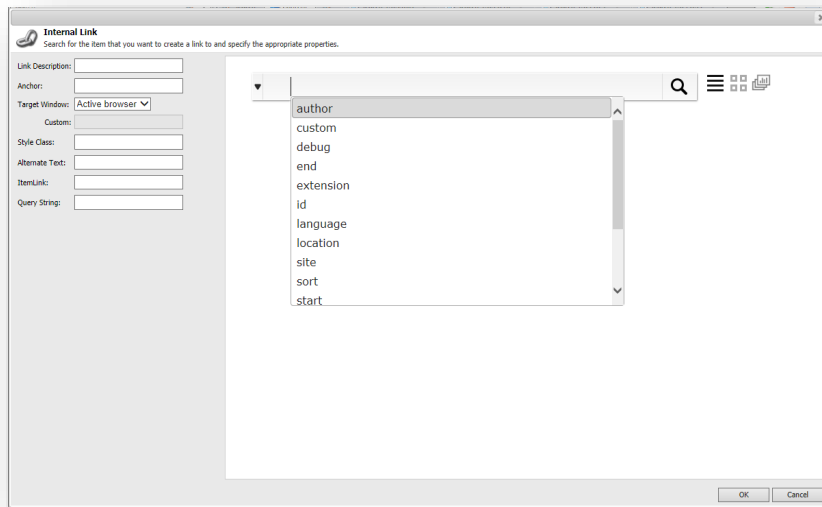


For more information about the search filters that are available, see the section *Using Search Filters*.

You can set a field to sort the list on by adding `SortField=_name` to the Source property of the field. You can add a sorting direction like this: `SortField=_name[asc|desc]`. If you specify a `SortField` without a sorting direction, Sitecore uses `asc` sorting.

General Link with Search Field

Use this field to create a link to an item that is stored in an item bucket.



Treelist with Search Field

Treelist with Search is a new field type that allows you to reference items in much the same way as a multilist field. However, unlike the multilist field, you can search within this field. In addition, you can enter a GUID to determine the start location of the search.

This will override the default filter that has been set on the template. The text field will search for the key item in this field, including any filters applied.

5.2 Creating a Tag Repository

To create a tag repository and to enable the tag filter in searches, you must create a new class in your Visual Studio solution and implement the `ITagRepository` interface.

You must implement methods that return *All Tags*, *Single Tags*, *First Tag*, *Tags by Name*, *Tags by Value*.

You can have as many tag repositories as you need and they can also come from different sources. After you have written your code, you must create a tag repository item in the content tree, under `/sitecore/system/Modules/Item Buckets/Tag Repositories`. You must point to the class you just compiled with the `namespace.class` assembly syntax.

Sitecore comes with a built-in tag repository.

To enable the built-in tag repository on your website:

1. In any template, create a field called **Tags**. This must be a multilist or multilist with search field.
2. Navigate to the `/sitecore/system/Settings/Buckets/Item Buckets Settings` item.
3. In the **Tag Parent** field, point to the parent item that stores all the tags.

The parent item that stores all the tags can also be an item bucket. The parent item that stores all the tags is a suitable candidate for an item bucket.

5.3 LINQ to Sitecore

- Sort by standard string, integer, float, or any type that implements IComparable

The Linq to Items layer does not implement all of IQueryable. The following methods have been implemented:

- All
- Any
- Between — with an extra overload for including or excluding the start and end ranges.
- Boost — makes this part of the query more important than the other parts.
- Cast — you can use this instead of Select.
- Contains
- Count
- ElementAt
- EndsWith
- Equal
- Facets — an extension that fetches the facets of predicates.
- First
- FirstOrDefault
- Last
- LastOrDefault
- Min
- Max
- Match — an extension for running regular expression queries.
- OrderBy
- OrderByDescending
- Select
- Single
- SingleOrDefault
- Skip
- Reverse
- Take
- ToList()
- ToLookUp()
- ToDictionary()
- Page — an extension that does Skip and Take automatically for you.
- StartsWith
- Union

Not supported

- Join
- GroupBy
- GroupByJoin
- Intersect
- Sum
- Average
- Concat
- TakeWhile
- SkipWhile

| Lucene Syntax | Linq to Sitecore | |
|-----------------|-------------------------------------------|-------------------------------------------------------------------------------------|
| Terms & Phrases | "test" or "hello dolly" | c.Where("test") or c.Where("hello dolly") or c.Where("test" "hello dolly") |
| Fields | title:"The Right way" and text:"go" | c.Title == "The Right way" or c.Text == "go" or c.Equals("go") |
| WildCard | *amber* | c.ContactName.Contains ("amber") |
| Prefix | amber* *amber | c.ContactName.StartsWith("amber") or c.ContactName.EndsWith("amber") |
| Fuzzy | roam~ or roam~0.8 | c.ContactName.Like("roam") or c.ContactName.Like("roam", 0.8) |
| Proximity | "jakarta apache"~10 | c.ContactName.Like("jakarta apache", 10) |
| Inclusive Range | mod_date:[20020101 TO 20030101] | c.ModifiedDate.Between("20020101", "20030101", Inclusion.Both) |
| Exclusive Range | title:{Aida TO Carmen} | c.Title.Between("Aida", "Carmen", Inclusion.None) |

| | | |
|-------------|--------------------------------------|----------------------------------------------------------------------------------------------------|
| Boosting | jakarta^4 apache | <code>c.Title.Equals("jakarta").Boost(4) c.Title.Equals("apache")</code> |
| Boolean Or | "jakarta apache" OR jakarta | <code>where c.Title.Equals("jakarta apache") c.Equals("jakarta")</code> |
| Boolean And | "jakarta apache" AND "Apache Lucene" | <code>where c.Equals("jakarta apache") && c.Equals("Apache Lucene")</code> |
| Boolean Not | "jakarta apache" NOT "Apache Lucene" | <code>where c.Equals("jakarta apache") && !c.Equals("Apache Lucene")</code> |
| Grouping | (jakarta OR apache) AND website | <code>where (c.Title == "jakarta" c.Title == "apache") && (c.Title == "website")</code> |

Sitecore supports two search providers:

- Lucene.net
- SOLR — shipped separately

The Linq layer is an abstract layer that converts common queries to something that a search provider understands.

For example, with a query like

```
var query = context.GetQueryable<Product>.Where(item => item.Name == "Sample Item")
```

the Linq layer resolves it to something that SOLR or Lucene.net understands. If you implement a new search provider, this provider can also understand the query. Although the Linq layer converts it to a common query, the implementation of your search provider determines exactly what comes back.

The Linq layer is used internally by Sitecore but can also be used by developers. You can use this layer in your sublayouts.

To start a search, you must set up a search context:

```
using (var context = ContentSearchManager.GetIndex(item).CreateSearchContext())
{
    IQueryable<SearchResultItem> searchQuery =
context.GetQueryable<SearchResultItem>().Where(item => item.Name == "Sitecore")
}
```

This returns the results of a query on your search index and returns it as a `SearchResultItem` type.

You can also use the indexer to run queries:

```
using (var context = ContentSearchManager.GetIndex(item).CreateSearchContext())
{
    IQueryable<SearchResultItem> searchQuery =
context.GetQueryable<SearchResultItem>().Where(item => item["_name"] == "Sitecore")
}
```

This converts the query to something your provider understands. For example, for Lucene it is converted to:

```
_name:sitecore
```

This is something that Lucene understands and can work with.

Complex Searches

```

using (var context = ContentSearchManager.GetIndex(item).CreateSearchContext())
{
    IQueryable<SearchResultItem> searchQuery =
context.GetQueryable<SearchResultItem>().Where(item => item[" name"] == "Sitecore").Where(item
=> item.Title == "Test || item.Body.Contains("CMS"))
}
OR
using (var context = ContentSearchManager.GetIndex(item).CreateSearchContext())
{
    IQueryable<SearchResultItem> searchQuery =
context.GetQueryable<SearchResultItem>().Where(item => item.Name == "Sitecore").Where(item =>
item.Title == "Test || item.Body.Contains("CMS"))
}

```

For Lucene, the Linq layer converts it to:

```
+(+_name:sitecore) +(title:test body:*cms*)
```

For Solr, the Linq layer converts it to:

```
+(+_name:sitecore) +(title:test body:*cms*)
```

There is no great difference; they should both return the same results. The SOLR provider needs to parse the queries differently.

You can use the Predicate Builder Helper that comes with Sitecore 7.0 to make subtle changes to the way the Linq Queries are built. The Linq Layer uses expression trees to evaluate the LINQ to a lower level that any provider can understand.

Adding a New Linq Provider

You can add a new Linq Provider by creating a new project in Visual Studio, and then adding a reference to `Sitecore.ContentSearch.Linq` to the project.

If you use a RESTful layer, you do not need anything else. If not, you must add a reference to your provider to the project, as well.

There are a few classes that you must construct, at a minimum. These classes are listed below. You can implement others as well, to have more control.

QueryObject

```
YourNewQuery : IDumpable
```

The `IDumpable` interface allows you to dump debug information into Visual Studio.

Index

```
YourNewIndex<TItem> : Index<TItem, YourNewQuery>
```

This interface is responsible for constructing your `QueryMapper` and `QueryOptimizer`.

Override `QueryMapper`, `QueryOptimizer`, `Execute`, `FindElements`

QueryMapper

```
YourNewQueryMapper : QueryMapper<YourNewQuery>
```

This is where all the magic happens. This is where you map an `IQueryable` method to the logic that your search provider uses to solve this functionality.

The `Strip` methods in this class pass the query in your Query object to an `IEnumerator` collection of query method objects and prevent the query from being directly implemented.

For example, if you run `Count` on `IQueryable` within the Lucene provider, it cannot just pass `count:10` or `size:10` in the query and expect Lucene to understand. However, it can use the input when it is ready to resolve the Linq Query to Lucene.

```
TopDocs hits = indexSearcher.Search(parser.Parse(query), null, count);
```

The `Visit` methods are used in expression trees to build up a query in Linq and convert it to something else.

An example that uses `StartsWith`:

```
protected virtual Query VisitStartsWith(StartsWithNode node)
{
    var fieldNode = QueryHelper.GetFieldNode(node);
    var valueNode = QueryHelper.GetValueNode<string>(node);

    return new WildcardQuery(new Term(fieldNode.FieldKey, valueNode.Value + "*"));
}
```

The `LuceneProvider` for Linq maps `StartsWith` in `IQueryable` to a `WildcardQuery` in Lucene and produces the following output:

```
Fieldname:fieldvalue*
```

If you were to implement, for example, a search provider for MongoDB, it would look something like this for `EndsWith`:

```
protected virtual Query VisitStartsWith(EndsWithNode node)
{
    var fieldNode = QueryHelper.GetFieldNode(node);
    var valueNode = QueryHelper.GetValueNode<string>(node);
    return Query<TItem>.Find(b => b[fieldNode.FieldKey], "/" + valueNode.Value);
}
```

QueryOptimizer

This is an abstract class that is used to optimize queries.

QueryOptimizerState

This is used to store the state of every query, for example, the default boost of 1 on every query. This can also be used to store things like the default analyzer and so on.

Boosting Items

Every content item contains a **Boost** Field. This field is used to increase the relative importance of items in the content tree. The values start at 1.0 — default — and can go up to, for example, 1.0 or 2.3. After you set this value, save the item and the order of your results will immediately reflect the new boost values.

5.4 Adding a New Search Provider

Sitecore has been designed with flexibility in mind and it is quite easy to add your own search provider.

There are many end points that you must implement, and you need to *translate* back and forth between Sitecore syntax and semantics and the syntax and semantics of the search provider. It is not mandatory to *translate* all the features in Sitecore, only the ones you want to use.

Abstract Document Builder

`Sitecore.ContentSearch.AbstractDocumentBuilder` is the entry point for mapping the configuration.

Computed Fields

In its simplest form, an index takes a text and places it in an index. Sometimes you may need more control over the way that data is stored or, more importantly, what data is stored. Sitecore uses computed fields to perform lookups and complex logic to determine what gets placed in your index.

For example, Sitecore uses computed fields to store the parsed Created Dates of items, whether or not an item has a lock or whether or not an item is a clone. To create your own computed fields, you need to create a new class and implement the `IComputedIndexField` interface.

The `IComputedIndexField` interface is quite simple. It expects a string and an object. The `FieldName` string is the name that the field uses in your index. The object is the value. `ComputeFieldValue` takes the item that is being indexed. It is from here that you can use it to look up other things to index — for example, the presentation data sources of the items.

```
public class IsClone : IComputedIndexField
{
    public object ComputeFieldValue(IIndexable indexable)
    {
        Item item = (Item) (indexable as SitecoreIndexableItem);
        return item.IsClone;
    }

    public string FieldName { get; set; }

    public string ReturnType { get; set; }
}
```

Computed fields are useful for storing data that needs to be calculated or logic for looking values up from an item. They can also be used for faceting and scaling inbuilt fields or for scaling facets.

Converters

In a classic data in and data out situation, it can sometimes be a good idea to convert your data so that it is stored in a certain way and then extracted in another way. For example, if you want to store dates in the index in a certain format — `20121212` — and fetch them from the index as a strongly typed `DateTime`, this is the job of a converter.

To create a new converter for a type, create a class that inherits from `Sitecore.ContentSearch.Convertors.TypeConverter` and override the methods as necessary.

For example, to cast a GUID in the index to a `Sitecore.Item.ID` in code, use the following converter:

```
public class IndexFieldIDValueConverter : TypeConverter
{
    public override bool CanConvertFrom(ITypeDescriptorContext context, Type
sourceType)
    {
        if (sourceType == typeof(ID))
            return true;
    }
}
```

```

        return base.CanConvertFrom(context, sourceType);
    }

    public override bool CanConvertTo(ITypeDescriptorContext context, Type
destinationType)
    {
        if (destinationType == typeof(string))
            return true;

        return base.CanConvertTo(context, destinationType);
    }

    public override object ConvertTo(ITypeDescriptorContext context,
System.Globalization.CultureInfo culture, object value, Type destinationType)
    {
        return ((ID) value).ToShortID().ToString().ToLowerInvariant();
    }
}

```

New Logging Classes

Sitecore comes with a logging framework called Log4net. This is wrapped up in the `Sitecore.Diagnostics` namespace, and you can use it to write to log files. Sitecore 7.0 introduces two new log files for searching and crawling.

To write to the search and crawling log files, use `SearchLog.Log.xxx` and `CrawlingLog.Log.xxx` respectively. They write to separate files.

In the `Sitecore.Buckets.config` file, set the `BucketConfiguration.EnableBucketDebug` setting to true and Sitecore uses these files to log all your searches. All the crawling is also logged by default so that developers can see exactly what is being placed in the index and what is being searched for.

Query Warm-up

When Sitecore starts up for the first time, it runs through a list of pre-defined queries that warm up the index cache of your search provider. The warm-up queries are listed in the `Sitecore.Buckets.WarmupQueries.config.example` file:

```

<configuration>
  <sitecore>
    <search>
      <warmup>
        <query>-
          _template:adb6ca4f03ef4f47b9ac9ce2ba53ff97|+(_path:110d559fdea542ea9c1c8a5df7e70ef9)|+_lastest
version:1|-_id:110d559fdea542ea9c1c8a5df7e70ef9*</query>
        <query>-_template:adb6ca4f03ef4f47b9ac9ce2ba53ff97|+_lastestversion:1</query>
        <query>+_language:en|-
          template:adb6ca4f03ef4f47b9ac9ce2ba53ff97|+( _path:3c1715fe6a134fcf845fde308ba9741d)|+(+is
displayed in search results:1)|+_lastestversion:1|-
          _id:3c1715fe6a134fcf845fde308ba9741d*</query>
        <query>__smallupdateddate:[20121111 TO 20121113]</query>
        <query>  smallcreateddate:[20121111 TO 20121113]</query>
        <query>  smallcreateddate:[20121111 TO 20121113]</query>
      </warmup>
    </search>
  </sitecore>
</configuration>

```

To add new queries, delimit every search term with a pipe “|” symbol and then normalize and escape the queries — that is, prevent *strange characters* from being passed on.

You can use the search log files to identify the most common searches and then place them in the warm-up phase so that the indexes are already warm when the users start to use them.

Field Readers

You can use field readers to convert known Sitecore field types to other values that you would prefer to store in the index. For example, a check box that is selected stores a `1` in the index and a cleared check box stores a blank.

You can use field readers to map both existing and any new field types that you create to different values in the index.

ContentSearchManager

The `ContentSearchManager` class is useful for getting access to everything you can do with your indexes. Use this class when you need to update, delete, search, or simply fetch information from an index.

The `ContentSearchManager` class has many useful properties and also gives you access to the `IQueryable` interface. It also gives you access to all the indexes so that you can rebuild them or get statistics about the health of each index.

5.4.1 Pipelines

Sitecore uses pipelines for searching, crawling, and UI interaction. This makes it very easy for you to plug in your own code when you need to.

`contentSearch.getContextIndex`

You can use the `contentSearch.getContextIndex` pipeline to control where Sitecore starts to search from. Sitecore currently uses `Context.Item` to determine where it should search. You can use this pipeline, for example, to specify that Sitecore should start searching from a certain part of the content tree in certain situations.

`contentSearch.getGlobalSearchFilters`

Searches run through the Linq Layer have some default filters. These filters contain things like ignoring certain templates, setting the path to start the search from and always returning the latest version of the item in the search results. You can add your own global filters here. You have access to the `IQueryable` collection where you are able to add your own filters.

`contentSearch.QueryWarmup`

Run specific queries when the application pool starts for the first time.

`contentSearch.translateQuery`

This pipeline gives you the raw query that is sent from the UI or from a query to the `Search.ashx` handler, and it allows you to manipulate the query before it is consumed by the Linq layer.

For example, Solr supports dynamic queries and this pipeline can be used to change a UI query from, for example, `title:sitecore` to `title_string:sitecore` before the Linq layer converts it to `IQueryable`.

`buckets.createBucket`

When an item bucket is created through the UI or through code, you can hook onto this pipeline to manipulate the bucketing process.

`buckets.removeBucket`

When an item bucket is converted into a normal container through the UI or through code, you can hook onto this pipeline to manipulate the unbucketing process.

`buckets.syncBucket`

When an item bucket is synchronized through the UI or through code, you can hook onto this pipeline to manipulate the synchronization process.

Buckets.isBucket

This pipeline is used to determine whether or not a content item is an item bucket. This pipeline allows you to modify the way in which you determine whether a content item is an item bucket or not.

buckets.isItemContainedWithinBucket

This pipeline is used to determine whether or not a content item is stored in an item bucket.

buckets.isTemplateBucketable

This pipeline is used to determine whether or not an item is bucketable. There are many ways to determine this.

You can modify this behavior so that it uses complex logic to specify that if the item is based on a template that has a very large numbers of items, it should be bucketable.

buckets.addSearchTabToItem

This pipeline is used add a search tab to an item. You can modify this to add many tabs or even have tabs that contain predefined searches that are already run when you open the tab.

buckets.cloneItemIntoBucket

This pipeline is used for cloning items into an item bucket.

buckets.copyItemIntoBucket

This pipeline is used for copying items into an item bucket.

buckets.moveItemIntoBucket

This pipeline is used for moving items into an item bucket.

buckets.getFacets

This pipeline is used to return facets in results.

buckets.dynamicFields

This pipeline is used to fill the SearchResultItem DynamicFields Dictionary with values that can be dynamically displayed in the search results.

This is useful if you are building new views for the UI, as you use this pipeline to dynamically generate values to display.

buckets.fillItem

This pipeline is used internally by Sitecore to add all the built-in fields that can be displayed in the search UI, for example, Template, Author, Created Date, and so on.

This pipeline is similar to the Dynamic Fields pipeline.

buckets.dynamicQuickActions

This pipeline is used to return quick actions. Quick actions are helper links that are displayed in search results. They allow you to quickly perform operations on items in the search results without having to open the item. You can use the QuickActions pipeline to create dynamic QuickActions, for example, an action that allows users to quickly translate an item. It could also be used to show a quick link to the next workflow state of items.

buckets.uiLaunchResult

This pipeline makes it possible to open search results from different data sources in different views. You do this by adding your own processor to the uiLaunchResult pipeline and implement how an item from a specific data source is opened.

The intended use is where you have implemented a crawler that indexes data from external data sources and the crawler saves these index entries with a value such as, for example, “media” in datasource.

You update the “buckets.resolveUIDocumentMapperFactoryRules” pipeline and add your own processor that defines a new rule that makes it possible to search for items with a datasource that is not “sitecore” (for example, “media”). The processor you have implemented in `uiLaunchResult` can manage how these items are opened.

5.4.2 Miscellaneous

LinqHelper

`LinqHelper` gives developers access to methods for converting raw UI queries or raw strings to `IQueryable` objects that make querying more user-friendly.

SearchStringModel

This is the model that is passed between the UI and the Linq Layer to determine which query to run. The model contains three simple properties:

| Property | Description |
|-----------|--------------------------------------------------------------------------------------------|
| Type | The name of the field that you want to use in the search. |
| Value | The value of the field. |
| Operation | Whether the search should use this value, must use this value, or must not use this value. |

When you make requests directly to `Search.ashx` that do not go through the Sitecore search API, you must use this model.

For example, if you are using JavaScript to make an AJAX post to the ASHX handler to return search results, you must use this model to run your searches.

Create an array in JavaScript:

```
var searchQuery = new Array();
searchQuery.push({
    type: "text",
    value: "sitecore",
    operation: "must"
});

function runQuery(o, pageNumber, onSuccessFunction, onErrorFunction) {
    $.ajax({
        type: "GET",
        url: QueryServer +
"/sitecore/shell/Applications/Buckets/Services/Search.ashx?callback=?",
        contentType: "application/json; charset=utf-8",
        dataType: "jsonp",
        cache: false,
        data: {
            selections: searchQuery,
            pageNumber: pageNumber,
            type: "Query",
            pageSize: 20,
            version: "1"
        },
        responseType: "json",
        success: onSuccessFunction,
        error: onErrorFunction
    });
}
```

5.5 Linq to Provider

The Linq search API provides access to search the indexes using standard Linq queries in the same way that other Linq providers works like Linq to SQL, Linq to Objects, and so on.

The search API is using the standard `IQueryable<T>` interface and has support for most of the available operations. For a general introduction to Linq, see <http://msdn.microsoft.com/en-us/library/vstudio/bb397926.aspx>.

Note

There are some operations that are not supported even though they are available through the `IQueryable<T>` interface. If these methods are called, a `NotSupportedException` or `InvalidOperationException` exception is thrown at runtime.

5.5.1 Accessing the Linq to Sitecore API

All searches are performed through `IProviderSearchContext` search context. The search context exposes the method `GetQueryable<T>` method which returns an instance of `IQueryable<T>`.

Example

```
public IEnumerable< MySearchResultItem > PerformSearch()
{
    var index = ContentSearchManager.GetIndex ("[My Index ]");

    using (var context =
index.CreateSearchContext (SearchSecurityOptions.EnableSecurityCheck))

    {
        var queryable = context.GetQueryable<MySearchResultItem>();

        var results = queryable.Where(d => d.Name == "Sitecore");

        return results;
    }
}
```

5.5.2 Custom Search Type / Object Mapping

Because the search API uses the generic `IQueryable<T>` interface to expose the search indexes, it is possible to use custom classes/POCO classes to describe the information in the indexes.

To implement custom search types, the class *must*:

- Have an empty constructor.
- Expose public properties with getters and setters and/or a public indexer to hold the search result data.

The Linq provider automatically maps document fields in the index to properties on the custom search type by the names of the properties. Properties or fields from the index that have not been matched together by name are skipped during the object creation/mapping.

It is also possible to map properties that do not match to fields in the index by decorating the properties with the `IndexField` attribute. You can use this, for example, to expose special Sitecore fields like `_name` as a property called `Name`. A different use case is field names with spaces, because they cannot be mapped directly to a property by name.

Furthermore, you can implement an indexer that is populated with the field name as key and the value for each field in the index document. There is also an `ObjectIndexerKey` that you can use to wrap indexers as different types. This is useful if you only have the string version of a property name but need to use it as an indexer for a property type which is actually something like an int.

Depending on the search provider being used, the indexed and stored data in the index might not be the native types for the value. For Lucene everything is stored and indexed as strings.

Supported Types

The following types are supported for automatic type conversion when mapping index document fields to properties:

- .NET built-in integral types
- .NET built-in floating-point types
- Boolean
- String
- DateTime
- Guid
- Sitecore ID
- Sitecore ShortID
- Sitecore ItemUri
- IEnumerable<T>
- DateTimeOffset
- Language
- Version
- Database
- CultureInfo
- TimeSpan

Custom Search Type Example

```
public class MySearchResultItem
{
    // Fields
    private readonly Dictionary<string, string> fields = new Dictionary<string, string>();

    // Properties

    // Will match the _name field in the index
    [IndexField(" name")]
    public string Name { get; set; }

    // Will match the myid field in the index
    public Guid MyId { get; set; }

    public int MyNumber { get; set; }
    public float MyFloatingPointNumber { get; set; }
    public double MyOtherFloatingPointNumber { get; set; }
    public DateTime MyDate { get; set; }
    public ShortID MyShortID { get; set; }
    public ID SitecoreID { get; set; }

    // Will be set with key and value for each field in the index document
    public string this[string key]
    {
        get
        {
            return this.fields[key.ToLowerInvariant()];
        }
    }
}
```

```
set
{
  this.fields[key.ToLowerInvariant()] = value;
}
}
```

5.5.3 Supported IQueryable methods

Restriction Operators

Where

```
var results = from d in queryable where d.Name == "Sitecore" select d;
```

or

```
var results = queryable.Where(d => d.Name == "Sitecore");
```

Projection Operators

Select

```
var results = from d in queryable select d.Name;
```

or

```
var results = queryable.Select(d => d.Name);
```

Anonymous types

```
results = queryable.Select(d => new { d.Name, d.Id });
```

Unsupported

```
SelectMany
```

Partitioning Operators

Take

```
results = queryable.Take(10);
```

Skip

```
results = queryable.Skip(10);
```

Page

```
results = queryable.Page(2, 100);
```

Ordering Operators

OrderBy

```
results = queryable.OrderBy(d => d.Name);
```

OrderBy Descending

```
results = queryable.OrderByDescending(d => d.Name);
```

ThenBy

```
results = queryable.OrderBy(d => d.Name).ThenBy(d => d.Id);
```

ThenBy Descending

```
results = queryable.OrderBy(d => d.Name).ThenByDescending(d => d.Id);
```

Unsupported

```
Reverse
```

Grouping Operators

Unsupported

```

GroupBy -Simple 1
GroupBy -Simple 2
GroupBy -Simple 3
GroupBy -Nested
GroupBy -Comparer
GroupBy -Comparer, Mapped

```

Set Operators

Unsupported

```

Distinct
Union
Intersect
Except

```

Element Operators

First -Simple

```
results = queryable.First();
```

First -Condition

```
results = queryable.First(d => d.Name == "Sitecore");
```

FirstOrDefault -Simple

```
results = queryable.FirstOrDefault();
```

FirstOrDefault -Condition

```
results = queryable.FirstOrDefault(d => d.Name == "Sitecore");
```

ElementAt

```
results = queryable.ElementAt(10);
```

Last

```

result = queryable.Last();
result = queryable.Last(d => d.Id > 10);

```

LastOrDefault

```

result = queryable.LastOrDefault();
result = queryable.LastOrDefault(d => d.Id > 10);

```

Single

```

result = queryable.Single();
result = queryable.Single(d => d.Id > 10);

```

SingleOrDefault

```

result = queryable.SingleOrDefault();
result = queryable.SingleOrDefault(d => d.Id > 10);

```

Quantifiers

Any -Simple

```
results = queryable.Any();
```

Any -Grouped

```
results = queryable.Any(d => d.Name == "Sitecore");
```

Unsupported

```
All
```

Aggregate Operators

Count -Simple

```
results = queryable.Count();
```

Count -Conditional

```
results = queryable.Count(d => d.Id < 10);
```

Unsupported

```
Sum
Min
Max
Average
Aggregate
```

Join Operators

Unsupported

```
Cross Join
Group Join
Cross Join with Group Join
Left Outer Join
```

5.5.4 IQueryable Extensions

Filtering

Filtering is similar to using `Where` to restrict the result list. Both methods will affect the result in the same result list, but when you use a `Filter` the scoring/ranking of the search hits is not affected by the filters.

Note

To avoid affecting the ranking of the search results, use `Filter` when applying restrictions to search queries in the `GetGlobalFilters` pipeline.

Furthermore, filters can be cached to optimize search performance.

Example:

```
results = queryable.Filter(d => d.Id > 4 && d.Id < 8);
```

Facets

Simple Faceting

```
var results = queryable.FacetOn(d => d.Name);
var facets = results.GetFacets();

foreach (var category in facets.Categories) {
    Console.WriteLine(category.Name);

    foreach (var facetValue in category.Values) {
        Console.WriteLine("{0}: {1}", facetValue.Name, facetValue.Aggregate);
    }
}
```

Pivot Faceting

```
var results = queryable.FacetPivotOn(p => p.FacetOn(d => d.Name).FacetOn(d =>
d.Year));

var facets = results.GetFacets();

foreach (var category in facets.Categories) {
    Console.WriteLine(category.Name);

    foreach (var facetValue in category.Values) {
        Console.WriteLine("{0}: {1}", facetValue.Name, facetValue.Aggregate);
    }
}
```


Boosting

```
results = queryable.Where(d => d.Id == 7.Boost(2f)).Where(d =>
d.Template.Contains("o"));
```

Other

Between

```
results = queryable.Where(item => item.Price.Between(50.0f, 400.0f, Inclusion.Both));
```

```
results = queryable.Where(item => item.Price.Between(2.0f, 12.0f, Inclusion.Both) ||
item.Price.Between(80.0f, 400.0f, Inclusion.Both));
```

```
results = queryable.Where(d => d.Date.Between(new DateTime(2004, 12, 31),
DateTime.Now, Inclusion.Both));
```

```
results = queryable.Where(d => d.Id.Between(1, 4, Inclusion.Both));
```

```
results = queryable.Where(d => d.Id.Between(1, 4, Inclusion.Lower));
```

```
results = queryable.Where(d => d.Id.Between(1, 4, Inclusion.Upper));
```

```
results = queryable.Where(d => d.Id.Between(1, 4, Inclusion.None));
```

string.Contains

```
results = queryable.Where(d => !d.Template.Contains("Hello:));
```

string.CompareTo

```
results = queryable.Where(d => !d.Name.CompareTo("Hello") == 1);
```

Equal

```
results = queryable.Where(d => d.Id.Equal(4));
```

Matches

```
results = queryable.Where(i => i.Template.Matches("^.*$"));
```

MatchWildcard

```
results = queryable.Where(i => i.Template.Where(i =>
i.Template.MatchWildcard("H?li*m")));
```

Like

```
results = queryable.Where(i => i.Template.Like("Citecoar"));
```

string.StartsWith

```
results = queryable.Where(d => !d.Name.StartsWith("Hello"));
```

string.EndsWith

```
results = queryable.Where(d => !d.Name.EndsWith("Hello"));
```

GetResults

```
results = queryable.GetResults().Hits.Where(i =>
i.Document.Name.Contains("o")).Where(hit => hit.Score > 0.6);
```

GetFacets

```
results = queryable.Where(d => d.Id > 0).FacetOn(d => d.Template, 0).GetFacets();
```

5.6 Searching

5.6.1 Searching in the Default Language

When you enter a search query, Sitecore searches every language by default. If you would only like to search in one language, you can add a filter to your search query, for example, `language:da`. If you would like the system to only return results in the context language of the UI, in the `Sitecore.Buckets.config` file, set the `BucketConfiguration.ForceClientLanguageInSearch` setting to `true`. This only enforces this filter in the Sitecore UI, not in your code.

5.6.2 Searching and Facets

There are two ways to add a facet to your searches through the UI:

- Add facets for every possible value.
However, this will not scale for facets that have huge amounts of variants.
- Limit the facets to specific values.

To limit the facet values that search should use, open the facet item — `sitecore/system/Settings/Buckets/Facets/Author` — and in the **Facet Filter** field, enter a reference to the class that implements the `ISimpleFacet` interface. You have to create this class yourself.

For example, if you have a list of colors — Red, Black, Green, Blue, and Yellow.

If you add a facet for `color` and leave the **Facet Filter** field empty, when you search for `cars`, the search results tell you how many of the cars are Red, Black, and so on.

If you create a facet filter you could tell the facet to only return results for Red, Black, and Green.

To do this, create a new class in your project and implement the `ISimpleFacet` interface. This interface simply returns a string. Use the logic in the class to determine the list of values that the facet should search for.

5.6.3 Using a Field as a Tag Repository

Sitecore comes with a tag repository — `/sitecore/system/settings/buckets/tag repository`. This repository can contain any type of item. You use these tags to tag any content item. You can tag a content item with one or more of these tags. This allows you to search for content items by their tags.

Sitecore searches the **Semantics** field of every item by default. If you add your own field and would like it to also act with the inbuilt tag repository, you must add the new field to the appropriate template, and open the index config files, for example,

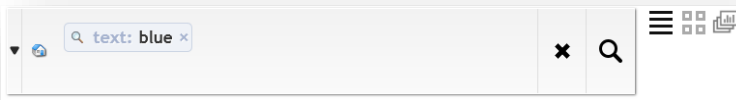
`Sitecore.ContentSearch.Lucene.Index.Master.config` — and add your field to ensure that it is aggregated into the `_semantics` field in the index.

```
<fields hint="raw:AddCustomField">
  <field luceneName=" tags" storageType="no"
indexType="tokenized">semantics</field>
  <field luceneName="_tags" storageType="no"
indexType="tokenized">mycustomtaggingfield</field>
</fields>
```

We recommend that you keep these items hidden in the content tree.




5.6.4 Including and Excluding Search Filters

When you enter a search filter in a search box, you can specify whether or not it should be included in the search query. If you enter a text filter to search for items that contain the key word *blue* — *text:blue*, a magnifying glass icon appears to the left of the search filter by default.



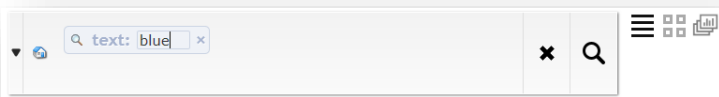
Click on the filter icon to specify whether or not it should be included.

There are three options:

- Must include —  blue x
- Must not include —  blue x
- Should include —  blue x

5.6.5 Editing Search Filters

You can edit the search filter by clicking on the search term (“blue” in the example above):



5.6.6 In-Memory Index

You can use in-memory indexes to solve many different problems. If you need to store text somewhere temporarily and then query it quickly, you could use an `InMemoryLuceneIndex`.

Of the search providers that Sitecore currently supports, Lucene is the only one that supports in-memory indexes.

```
var index = new InMemoryLuceneIndex("products");
index.Analyzer = new StandardAnalyzer(Lucene.Net.Util.Version.LUCENE_30);
index.IndexDocumentMapper = new DefaultLuceneDocumentTypeMapper();

var repository = new TestRepository();
IEnumerable<TestDocument> information = repository.GetTestDocuments();

using (var context = index.CreateUpdateContext())
{
    foreach (TestDocument testDocument in information)
    {
        var document = new Document();
        document.Add(new Field("ID", testDocument.Id, Field.Store.YES,
Field.Index.NOT_ANALYZED));
        document.Add(new Field("Name", testDocument.Name, Field.Store.YES,
Field.Index.ANALYZED));
        document.Add(new Field("Template", testDocument.Template, Field.Store.YES,
Field.Index.ANALYZED));
        document.Add(new Field("TemplateSortable", testDocument.TemplateSortable,
Field.Store.YES, Field.Index.NOT_ANALYZED));
        document.Add(new Field("Body", testDocument.Body, Field.Store.YES,
Field.Index.ANALYZED));
        document.Add(new Field("Date", testDocument.Date, Field.Store.YES,
Field.Index.ANALYZED));
    }
}
```

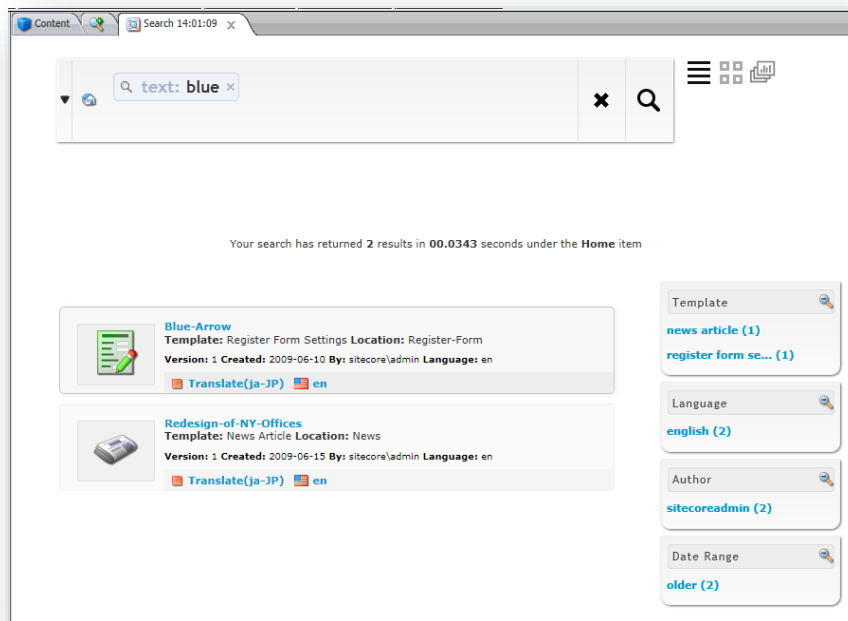
```
context.AddDocument (document) ;  
}  
  
context.Optimize () ;  
context.Commit () ;  
}
```

5.6.7 Applying Quick Actions to Search Results

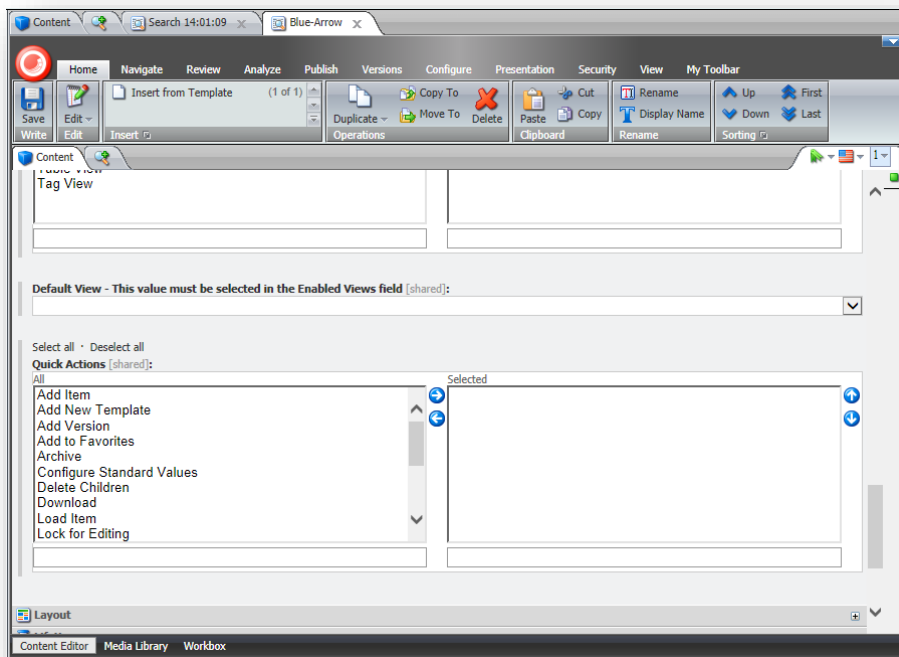
There are a number of quick actions that you can apply to a content item that are available when the item appears in search results.

To add a quick action to a content item:

1. In the Content Editor, search for the item that you want to add the quick action to.



- In the search results, select the content item and it opens in another tab.



- Expand the Item Buckets section and scroll down to the Quick Actions field.
- Select the quick actions that you want to add to this content item.
- Save your changes.

Important

You must click Save in the tab that you edited the content item in.

The next time that this item appears in some search results the quick actions are displayed with the item in the search results.



Add New Quick Actions

You can also add extra quick actions to the list of available actions.

To add a quick action:

- In the content tree, navigate to `/sitecore/System/Settings/Buckets/Settings/Quick Actions`.
- Insert a new quick action and give it a suitable name.

3. On the **Content** tab, in the **Command** group, in the **Command** field, enter the command name, for example: *item:save*.

Alternatively, in the configuration file, use the dynamic `QuickActions` pipeline to allow much more complex and dynamic Quick Actions, for example *Bring back all current workflow commands*.

5.6.8 Showing Dynamic Fields in Search Results

If you want to create a dynamic field that you can display in search results, you must specify this in the `buckets.dynamicFields` pipeline. For example, if you want to display *Facebook likes* for a specific item you can use this pipeline to display this information in the search results.

The `DynamicFields` pipeline aggregates a dictionary of keys and values. To obtain the information you want, you must refer to the relevant key and value in the dictionary. In the Facebook example, the key should be *Flikes* and the value should be the number of likes specified by the Facebook API.

To display a dynamic field in a specific search view:

1. In the content tree, go to `/sitecore/system/Settings/Buckets/Views` and select the search view.
2. On the **Content** tab, in the **View Details** section, select the **Item Template** field.
This field contains the HTML output for the search view.
3. Insert a placeholder that displays the number of Facebook likes in the search view.
4. To insert the placeholder, enter the key followed by `DynamicPlaceholder`.

For example if the key for Facebook likes is *Flikes* in the dictionary, the placeholder should be called `FlikesDynamicPlaceholder`.

When Sitecore displays the search results, it looks at all the dynamic placeholders and replaces them with the value of each specific key.

5.6.9 Adding New Filters and Setting up Alias Filters

To add a new filter to the search UI:

1. In the content tree, navigate to `/sitecore/system/Settings/Buckets/Search Types`.
It is best practice to add the new search filter in the *User Defined* folder.
2. Select the *User Defined* folder, and on the **Home** tab, in the **Insert** group click **Field Search Type**.
3. Give the new search type an appropriate name.

This is the name that users are required to type into the search box to apply the filter.

For example if the search type is called *Date* the user must type in *Date:* in the search box when they want to apply the filter. Remember that the name is case sensitive and you should ensure that case is consistent across all names.

In the new search type, in the **Search Type** section, in the **Control Type** field select the type of control that is most appropriate for your search. Under the item's field value, locate the relevant field type and add it to the selected column.

You may need to create a new field type that matches your new search type. For example a calendar would be best suited to a filter that uses dates.

4. In the **Display Text** field, enter some appropriate text and ensure that the wording is consistent with all the other search filters.

This text is displayed in the drop-down menu when you browse the search filters in the search UI.

5. If you want to apply a custom syntax to the **Control Type** field to create specific outcomes, enter this in the **Web Method** field.

For example, if the **Control Type** field is a calendar, the **Web Method** field can make a request to a web service to tell Sitecore to display a calendar control that only allows you to select a date from the last 2 calendar years.

Alias Filters

Sometimes, you have field names that are long and descriptive and therefore not very easy to type into a search box when you want to search on them.

Alias Filters are a way of setting up an alias for another search field to solve this problem. For example, a field called *Product Price* should probably be shortened to *price* when searching. You may also want to add a slider control to be able to slide between prices or to a particular price. Aliases will help you do this. These could be the control type parameters for this slider:

```
min:0&max:20000&value:40&range:true&start:0&end:2000
```

5.6.10 Creating a New Search Facet

You can use facets to drill down to more specific results in any list of search results. The default facets are displayed in the facets menu on the right side of the search results.

To create a custom facet, navigate to the `/sitecore/system/Settings/Buckets/Facets` item of the content tree. Right click on the *Facets* item and in the context menu, click **Insert, Facet**.

You now have to specify the name of the field in your index, in the parameters field in the content tab. You can apply hierarchical faceting by listing many fields separated by commas. This is useful if you want to facet on, for example, *Clothes Type* first, and then on *Color* so the facets display like this:

- Belts/Black (1)
- Belts/Green (9)
- Belts/Blue (12)
- Jumper/Black (33)

You can create folders your facets, It is easier to get an overview this way, and it also makes it clearer which facets you have created and which facets Sitecore has provided.

5.6.11 Default Bucket Queries

Default bucket queries are run automatically when the search UI is accessed. It is possible for content authors to remove the default query from the search field if they chose.

5.6.12 A Persistent Bucket Filter

Default filters can be set on any item by accessing the content tab and entering a search string in the Default Filter field. A notable difference between default queries and default filters is that default filters *cannot* be removed from the search UI.

5.6.13 Default Queries and Filters

The supported filters are:

- tag
- template
- location

- sort
- custom
- tag
- start
- end

To implement multiple filters, you must insert a semicolon between each filter.

Note

Every filter is case sensitive.

For example, to search for the keyword *pineapple* between a start date of 03/03/2012 and an end date of 04/04/2012, the filter string is:

```
text:pineapple;start=03/03/2012;end=04/04/2012
```

5.7 Rule-based Boosting

Rule-Based Boosting relies on the set of out of the box conditions that Sitecore ships with. You can, however, create custom conditions as described in the in the *Rules Engine Cookbook*, in the section *How to Implement a Condition*.

You can also reuse the conditions from the Sitecore Stuff shared source module that is available on the Sitecore Marketplace (<http://marketplace.sitecore.net/>.) This module will give you the following conditions:

- Item Name.
- Item ID.
- Item Level
- Parent Name.
- Parent Template.
- Item Path.
- Ancestor-or-Self.
- Item is Publishable.
- Item Language.
- Item Version.
- Item Version Count.
- Item Is Hidden.
- Item Is Protected.
- Item Has a Layout.
- Field is Empty.
- Can Read Item.
- Can Write Item.
- Can Delete Item.
- Can Rename Item.
- Can Create Subitems.
- Item Is In a Workflow.
- Item Is In a Workflow State.
- Item Is In a Final Workflow State.
- Item Is Locked.
- Item Is Locked by Me.
- Item Is Locked by User.
- True (actions execute always).
- Call Script.

5.7.1 Creating a New Boosting Rule Condition

It is not necessary to create custom actions in general and for most implementations, since the action related to this functionality, “Adjust Boost by Select”, is sufficient. However, you can implement a custom boosting rule action if you need to.

Here is a basic description of how to implement a custom boosting rule action:

```
public class CustomBoostAction<T> : RuleAction<T> where T : Sitecore.ContentSearch.Boosting.
    RuleBoostingContext
    {
        public float Boost { get; set; }

        public override void Apply([NotNull] T ruleContext)
        {
            Assert.ArgumentNotNull(ruleContext, "ruleContext");

            // the value of the Boost property will be set via the macro
            // if your action is not using the macro approach,
            // you will have to set this property based on some other logic

            // your code goes here:

            // you have to set the ruleContext.Boost before the method returns
            // here is how it is set in the out of the box action:
            // ruleContext.Boost += this.Boost;
        }
    }
}
```

From here, you can use the description in the *How to Implement an Action* section of the *Rules Engine Cookbook*.

5.7.2 Implementing Rule-Based Boosting for Fields

You can implement rule-based boosting for fields.

1. Implement the `indexing.resolveFieldBoost` pipeline processor class:

```
public class RuleBasedFieldBoostResolver : BaseResolveFieldBoostPipelineProcessor
{
    public override void Process(ResolveFieldBoostArgs args)
    {
        Assert.ArgumentNotNull(args, "args");
        Assert.ArgumentNotNull(args.FieldDefinitionItem, "field definition item");

        var fieldItem = args.FieldDefinitionItem;

        var ruleContext = new RuleBoostingContext(fieldItem);
        var ruleItems = this.GetLocalBoostingRules(fieldItem);

        if (ruleItems == null || !ruleItems.Any())
        {
            CrawlingLog.Log.Debug(string.Format("No local rules were resolved for field {0}", fieldItem.Uri));
            return;
        }

        var rules = this.ConvertToBoostingRules<RuleBoostingContext>(ruleItems);

        try
        {
            if (rules != null)
            {
                rules.Run(ruleContext);
            }
        }
        catch (Exception exception)
        {
            CrawlingLog.Log.Error(string.Format("Cannot resolve boost for item {0}.", fieldItem.Uri), exception);
        }
    }
}
```

```

        args.ResolvedBoost += ruleContext.Boost;
    }
}

```

2. Insert the processor *after* StaticFieldBoostResolver:

```

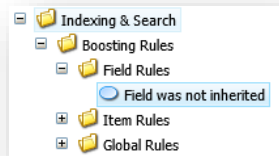
<processor
type="Sitecore.ContentSearch.Pipelines.ResolveBoost.ResolveFieldBoost.SystemFieldFilter,
Sitecore.ContentSearch"/>
<processor
type="Sitecore.ContentSearch.Pipelines.ResolveBoost.ResolveFieldBoost.FieldDefinitionItemResol
ver, Sitecore.ContentSearch"/>
<processor
type="Sitecore.ContentSearch.Pipelines.ResolveBoost.ResolveFieldBoost.StaticFieldBoostResolver
, Sitecore.ContentSearch"/>
<processor
type="Sitecore.ContentSearch.Pipelines.ResolveBoost.ResolveFieldBoost.RuleBasedFieldBoostResol
ver, Sitecore.ContentSearch"/>

```

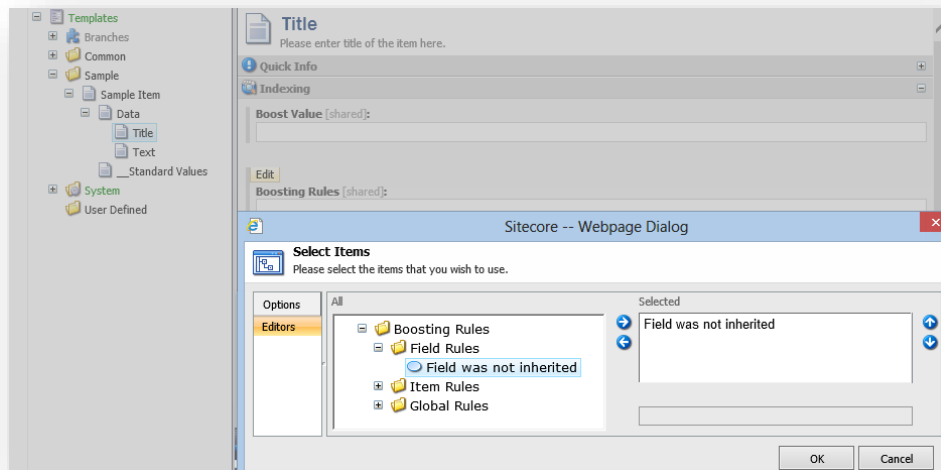
This activates rule-based boosting for fields. You can now create the field boosting rule and associate it with a field.

1. In the content tree, navigate to /sitecore/system/Settings/Indexing and Search/Boosting Rules.

We recommended that you create a designated folder for the field rules like to the one for Item Rules:



2. Locate the field that you want to attach this rule to.
3. Enable *Standard Fields*.
4. In the **Boosting Rules** field, click **Edit** and associate the new rule with the field:



5.8 Multiple Index Support

The introduction of item buckets allows you to use multiple indexes to support the content tree. A practical example would be that you may want to have a separate index for the content section, the system section, and the media library. Having one index will satisfy most requirements but if you expect to have millions of content items, millions of media items, and so on, using multiple indexes is the solution.

For more information about using and configuring multiple indexes, see the *Sitecore Search Scaling Guide*.

5.9 Adding a New View

The search results can be displayed in several different views. The default views are *Grid*, *List*, *Gallery*, and *Image*.

Developers can add new views to the search results to cater for different situations, for example, browsing an image gallery, and having a small image in the results.

To add a new view:

1. Navigate to the `/sitecore/system/Settings/Buckets/Views` item and create a new view item.
2. Set the **Header Template**, **Item Template**, and **Footer Template** fields to the HTML tags that you want to return in the search results.

```

Header Template:
<div class="mainmargin" id="grid-content" style="position: relative; width:
auto;overflow-x: hidden; overflow-y: hidden;">

Item Template:
<div class="post-1 post type-post status-publish format-standard hentry category-
inspiration category-landscapes category-portraits category-typography category-
web-design category-weddings tag-image tag-lightbox tag-sample post_float rounded"
id="post-1" style="MetaPlaceholder"><a class="ceebox imgcontainer" title="Lightbox
Example" href=""

Footer Template:
</div>

```

You can use the following placeholder names to display the values of the items. These are considered built-in Placeholders that will always be available to your views. For more fields you can use the `DynamicFields` pipeline to achieve this.

| Placeholder | Description |
|-----------------------|-----------------------------------------------------------------------------------|
| MetaPlaceholder | The CSS style that you want to use when the results are displayed. |
| LaunchTypePlaceholder | Whether it will launch the result in a new tab or in a new Content Editor window. |
| ItemIdPlaceholder | The item ID. |
| ImagePathPlaceholder | The path to the image of the item. |
| NamePlaceholder | The name of the item |
| TemplatePlaceholder | The name of the template that the item is based on. |
| BucketPlaceholder | The bucket that this result comes from. |
| ContentPlaceholder | The content of the result. |
| VersionPlaceholder | The version of the content item. |
| CreatedPlaceholder | The date that the content item was created. |
| CreatedByPlaceholder | The person who created this item. |

Chapter 6

Configuration and Tuning

This chapter describes some of the configuration files that are used for item buckets, as well as some tools that can help tune performance.

- Configuration Files
- Scaling Test Tool
- Index Analyzer

6.1 Configuration Files

Item Buckets has a `Sitecore.Buckets.config` file that contains configuration settings. This file is mainly documented by the comments inside the file itself.

Custom Settings

The Sitecore `web.config` file also contains settings that are relevant for search. They all have names that begin with “Indexing.”. This is one example:

```
<setting name="Indexing.UpdateInterval" value="00:00:30"/>
```


6.2 Scaling Test Tool

Sitecore comes with a tool that allows you to test the scalability of your new search providers or the way the indexes have been set up. You can also use this tool to test facets, new search types, new query types, and the way your indexes are cached in large content environments.

To enable this tool:

1. Go to the `Sitecore rev. xxx\Website\sitecore\admin\sqlscripts` folder.
2. Run the `ItemGenerator.sql` script on the databases in your test environment.
3. Create a folder in your data directory called “words” and then place some `.txt` files in here for dummy data.

For example, use some of the freely available books on the net in `.txt` format.

In a browser, open <http://<sitename>/sitecore/admin/FillDB.aspx> and fill in the form and then run. After the process has finished, reset IIS and your newly items will appear in the content tree. The tool will create approximately 120,000 items in 10 seconds.

We recommend that you disable the `FillDB.aspx` page in a production environment. You do this in the config file:

```
<setting name="EnableFillDB" value="false" />
```

6.3 Index Analyzer

Every index uses the default analyzer — StandardAnalyzer. An analyzer is a software component that is used for writing and querying the index as well. The analyzer determines how things are stored and how things are queried.

The StandardAnalyzer:

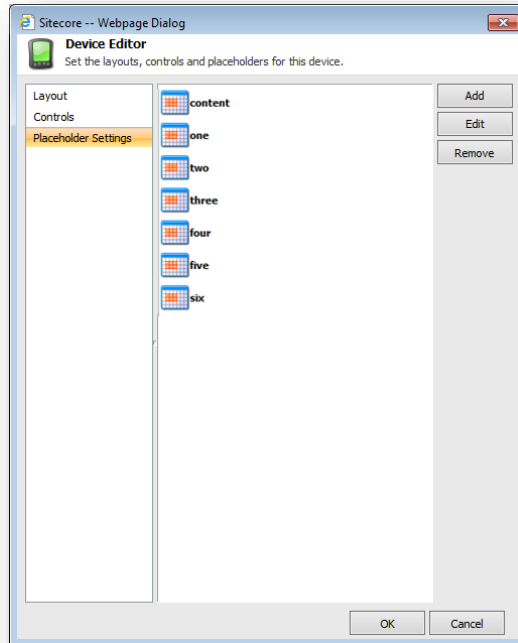
- Ensures that all the search queries and values in the index are in lowercase.
- Splits up bodies of text into small chunks.
- Removes any unnecessary stop words, such as, *the*, *is* and any other words that typically don't add any value to a search query.

6.4 Scaling with Placeholders

Because that you could potentially have many items in an item bucket, you need to make sure that you set placeholder settings for all the placeholders within your site, or you will get a performance decrease in the Page Editor. This will prevent Sitecore from having to search for components that can be added – they are already known.

To do this, assign a placeholder settings item for each placeholder on the page like this:

1. Click the **Presentation** tab, and then click **Details** in the **Layout Group**.
2. Click **Edit** for the layout, and then add the placeholder settings you need:



Chapter 7

Appendix

This chapter describes how various internal processes work and contains information that will help you to extend or modify the module.

This chapter contains the following sections:

- Tips and Tricks

7.1 Tips and Tricks

Publishing

- How does Sitecore publish of millions of items?

The simple answer is: nothing special is done. The rationale is in the answer to this question: “Why would you need to publish 1 million items *all* the time?” The first time you publish from authoring to delivery and if you were to use the Full Publish option, then the answer is: “Yes, this would take some time.” But after that, you only need to run publish added content incrementally.

- If I publish a single item in a bucket, but its bucket folders are not published, what happens?

Sitecore has added an extra pipeline step to the publishing process to detect if an item requires its bucket folders to be published and will add them to the publishing queue as well automatically. This also applies to items that are in workflow. There is no need to add in an extra workflow action for this.

- What publishing should I be doing for Item Buckets?

The answer to this question does not change because of Item Buckets.

Standard Web.Config Tweaks

- You should periodically tweak the cache depending on how many items are in the content tree and how many similar searches have been processed.

Setup Tweaks

- When you import a lot of content programmatically, you must truncate the *PublishingQueue*, *History*, and *Event Queue* tables in the *Master* and *Web* databases and rebuild the indexes on the database tables. If you don't do this, the *PublishingQueue*, *History*, and *EventQueue* tables will get very large, slow down processing, and your Sitecore installation may not start.

After clearing the tables, you must rebuild the index and run a smart publish instead of an incremental publish.

Environment Tweaks

- If possible, disable the inbuilt Windows Search Index as well as any other indexer that is running on the computer that runs the index or on the web server itself. This index uses essential Disk I/O resources that Lucene.net needs.
- Don't run processes on the index to create a backup. The index should not be part of regular backup procedures. Chances are that the backup will be outdated if it is ever needed, so it is a waste of resources.
- It is very important that you set up a SQL Maintenance plan that rebuilds your indexes. When you create a lot of content, index fragmentation will increase, especially with the bulk importation of content.

The hotspots will be the *Items*, *Versioned*, *Unversioned*, *Shared*, *Blobs*, and *Links* tables. To be on the safe side, you should set rebuilds for every table. If you don't do this, performance of the CMS will degrade.

Here is a script for rebuilding all the indexes in your databases.

```
-- Show fragmentation for all tables
EXEC sp_MSforeachtable @command1="print '?' DECC SHOWCONTIG('?)"

--Rebuild all indexes (this method locks the tables while the indexes are rebuilt)

USE [Sitecore_Master] --Change this to your database name
DECLARE @TableName varchar(255)
DECLARE TableCursor CURSOR FOR
```

```
SELECT table name FROM information schema.tables
WHERE table type = 'base table'

OPEN TableCursor
FETCH NEXT FROM TableCursor INTO @TableName
WHILE @@FETCH_STATUS = 0

BEGIN
DBCC DBREINDEX (@TableName, ' ', 90)
FETCH NEXT FROM TableCursor INTO @TableName
END

CLOSE TableCursor
DEALLOCATE TableCursor
```

Importing Data Tweaks

- When you import a lot of content into Sitecore, it is best to use the `BulkUpdateContext` class. When you have imported the content, rebuild the Lucene index.
- If you import a lot of content, do it in batches of, for example, one thousand, and then bucket or re-sync the bucket to avoid overloading the process with items.

Uploading Files to the Media Library

- The media library now supports the indexing of all files that support `IFilter`. For more information about `IFilter`, see <http://en.wikipedia.org/wiki/IFilter>.

Note

Sitecore does not by default provide search in PDF documents on MSSQL databases, and in neither PDF nor Word documents on Oracle databases.

In short, `IFilter` is a generic interface for indexing documents. Sitecore 7 ships configured to use `IFilters` to index text in the binary content of media items. To use this feature, you must install `IFilters` for the types of media items that you want your solution to index. You can use software such as the free `IFilter Explorer` from Citeknet to investigate the `IFilters` installed on your system.

If the system hosting a Sitecore solution does not have an `IFilter` for a given media type, Sitecore can only index the metadata stored in that media item, not its binary content. Additionally, whether search results include media items can depend on the encoding of the format of data contained in those media. For example, `IFilters` may not be able to convert images of text in media items to structured text to parse.

Finally, you must install `IFilters` on the relevant hosts in your production environments (both content management and delivery); having an `IFilter` installed in a development environment will not allow indexing of that data type in your production environments.

Bucket Config Tweaks

- You can tweak your index so that it doesn't index certain things that you don't want in the index. This will decrease rebuild time and improve search time.
- Consider rebuilding your indexes on a computer that has a solid state disk. Incremental updates do not have to be performed on SSD but they will benefit from this as well. If you have one dedicated server that rebuilds indexes and deploys them to an environment, ensure that this server has an SSD. Indexes will not be so big, so a small SSD will suffice — for example 64GB.
- Don't shard too many indexes. Sitecore must context switch between these shards and this slows down search time.
- If you have very large caches, you can see large memory spikes when you run a search. This is normal as a search is filling the `ItemCache` for the results. Be careful of under-optimized

caches — they keep as much of the search results in cache as possible and this may not be optimal.

- If you see a lag in searches or results that are taking a long time to facet, enable Debug mode in the `Sitecore.Buckets.config` file. All queries are logged in debug mode, as well as how long the queries take to run and how many clauses they contain. This can help identify the issue. Wildcard and range queries are probably the main culprits.

- Optimize the out-of-the-box indexes.

Optimization speeds up index rebuilding time and to some small degree, query time as well.

- Disable all the dropdowns that you are not using in the `/sitecore/system/Modules/Item Buckets/Settings/Search Box Dropdown` item. The most expensive lookups are *recently modified* and *recently created*.
- Add all the items in `/Sitecore/System/Modules/Buckets` to your prefetch cache.
- If you have disabled Debug mode but would like to debug a single query, in the search field enter `debug:1`, press tab and then enter the search term. Sitecore only adds that search query to your log file.
- Sitecore 7.0 ships with a slower version of the bucketing process in order to be backwards-compatible. If you do not need this, you should uncomment this from the configuration file:

```
<setting name="FastQueryDescendantsDisabled" value="true" />
```

This will improve unbucketing performance.

7.2 Default Fields in Lucene

The following is a list of the required fields in Lucene:

```
ID = "_id";
Sites = "site";
Database = "_database";
Path = "_path";
Name = "_name";
DisplayName = "_displayname";
Language = "_language";
Creator = "_creator";
UpdatedBy = "parsedupdatedby";
CreatedBy = "parsedcreatedby";
Editor = "_editor";
Created = "_created";
Updated = "_updated";
Hidden = "_hidden";
Template = "_template";
AllTemplates = "_templates";
TemplateName = "_templatename";
Icon = "_icon";
Links = "_links";
Tags = "_tags";
Group = "_group";
LatestVersion = "_latestversion";
Lock = "lock";
Version = "_version";
IsClone = "_isclone";
FullPath = "_fullpath";
IndexName = "_indexname";
UniqueId = "_uniqueid";
DataSource = "_datasource";
Parent = "_parent";
Bucket = "_bucket";
SmallCreateDate = "__smallcreateddate";
SmallUpdatedDate = "__smallupdateddate";
Url = "urllink";
Semantics = "__semantics";
IndexTimestamp = "_indextimestamp";
```



```
HasChildren = "haschildren";  
"__bucketable"  
"__workflow_state"  
"__known_hit"  
"__is_bucket"
```