Sitecore CMS 7.0 or later

Sitecore Search Scaling Guide

Administrator's guide to scaling with Sitecore search and item buckets.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2015 Sitecore. All rights reserved.

Table of Contents

Chapter 1 Introduction ... 3
1.1 Item Buckets – Conceptual Overview ... 4
1.1.1 Overview ... 4
1.1.2 Fundamental Concepts ... 4
1.1.3 Terminology ... 7
1.2 Creating Item Buckets ... 9
1.2.1 Making Content Items Bucketable ... 10
1.2.2 Hiding Items in an Item Bucket ... 11
1.2.3 Making Templates Bucketable ... 11
1.3 Synchronizing Item Buckets ... 13
1.3.1 Locking Parent/Child Relationships ... 13
1.4 Managing Item Buckets ... 15
1.4.1 Building the Search Indexes ... 15
1.4.2 Item Bucket Settings ... 15
Chapter 2 Configuring Scalability in Sitecore ... 16
2.1 Configuring Cache ... 17
2.1.1 Custom Cache Settings ... 17
2.2 Multiple Search Indexes ... 18
2.3 Configuring Scalability Settings ... 21
2.3.1 Sitecore Buckets Scaling Config ... 22
2.3.2 Creating a Custom Crawler ... 26
Custom Crawler Configuration ... 27
2.3.3 Creating Multiple Search Indexes (Sharding) ... 28
Chapter 3 Extending Scalability with Solr ... 29
3.1 Benefits of Using Solr ... 30
3.2 Configuring Solr to work with Sitecore ... 32
3.2.1 Preparing Solr ... 32
3.2.2 Creating a Solr Core ... 32
3.2.3 Generating an XML Schema for Solr ... 34
3.2.4 Enabling Solr Term Support ... 36
3.2.5 Verifying that Solr is Running Correctly ... 36
3.3 Configuring an IOC Container ... 38
3.3.1 Selecting the Correct Support DLL files ... 38
3.4 Configuring Sitecore to work with Solr ... 40
3.4.1 Enabling the Solr Config File ... 40
3.4.2 Solr Specific Settings ... 41
3.4.3 Specifying an IOC Container ... 42
3.5 Re-building the Search Indexes ... 44
3.6 Troubleshooting Solr ... 46
Chapter 4 Language Support in Solr ... 47
4.1 Solr Schema-less Fields ... 48
4.2 Multiple Language Support ... 49
4.2.1 Storing Fields for Items in Multiple Languages ... 49
4.2.2 Retrieving a Specific Language Version Using Linq To Sitecore ... 49
Chapter 5 Appendix ... 51
5.1 Tips and Tricks ... 52

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2015 Sitecore. All rights reserved.

Chapter 1

Introduction

This document is designed for Sitecore administrators and developers and describes how to set up, configure, and tune Sitecore CMS and item buckets for performance and scalability.

The document contains the following chapters:

- **Chapter 1 — Introduction**
  This chapter is an introduction to Sitecore CMS and Item buckets.
- **Chapter 2 — Configuring Scalability in Sitecore**
  This chapter explains some of the basic scaling strategies you can follow and settings you can use to improve the scalability of your Sitecore solution.
- **Chapter 3 — Extending Scalability with Solr**
  This chapter explains how to install and configure Solr to make your Sitecore CMS installation more scalable.
- **Chapter 4 — Language Support in Solr**
  This chapter explains how to configure the Solr provider to support different languages in Sitecore.
- **Chapter 5 — Appendix**
  This chapter contains some tips and tricks as well as a list of known issues.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2015 Sitecore. All rights reserved.

1.1 Item Buckets – Conceptual Overview

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

1.1.1 Overview

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

Benefits of using item buckets:

- You can search for content items in each item bucket. You can even search for non-bucket items.
- You can use the Search API with bucketable content items.
- All the bucketable content items in an item bucket are automatically organized into a logical format.
- Content items that are stored under other content items can act as embedded items.
- A single repository can contain millions of bucketable content items without slowing down the UI or overloading the content tree.

Important
Converting content items into item buckets removes the parent to child relationship between items in an item bucket. This conversion can also cause your website to not work as designed. For more information about coding and item buckets, see the *Developer's Guide to Item Buckets and Search*.

You do not 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.

1.1.2 Fundamental Concepts

This section explains some of the key concepts related to item buckets.

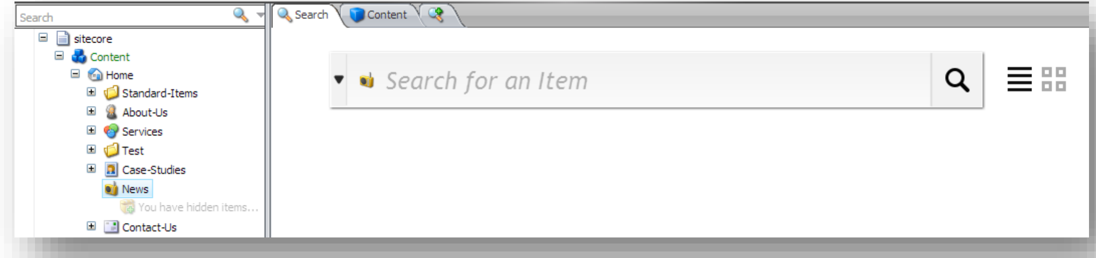
Item Buckets Definition

An item bucket is a repository in the content tree that stores content items. The difference between an item bucket and a normal repository is that an item bucket can store thousands of content items without displaying them in the content tree.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2015 Sitecore. All rights reserved.

Search Functionality

An item bucket contains a Search tab that lets you find the items that are stored in the container so you no longer have to navigate for items in the content tree. If you have thousands of content items this is a more efficient way of finding items.



Content Items

Item Buckets can contain both normal (standard) content items and bucketable content items:

- **Standard content items**

  Visible in the content tree and maintains their parent child relationship. A normal content item is based on a template that does not support item buckets.

- **Bucketable content items**

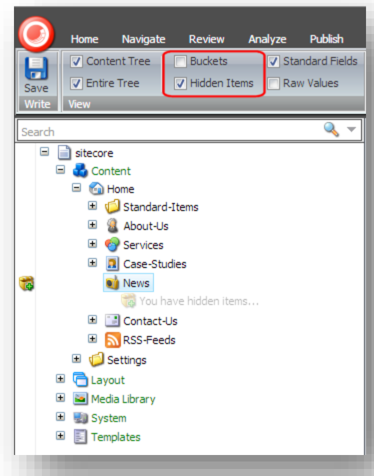
  Not visible in the content tree and does not maintain their parent child relationship. A bucketable item is based on a template that supports item buckets. To make an item bucketable, select the checkbox on the template, standard values.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2015 Sitecore. All rights reserved.

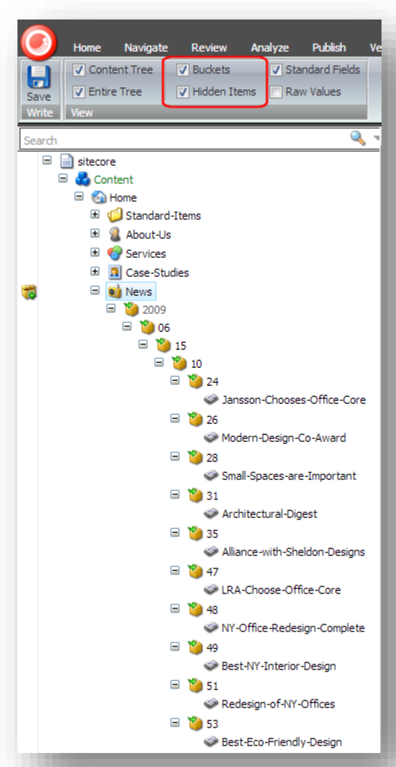
Folder Structure

The content items in an item bucket are automatically organized into folders and the parent to child relationship between the content items is completely removed.

Item bucket with the folder structure hidden

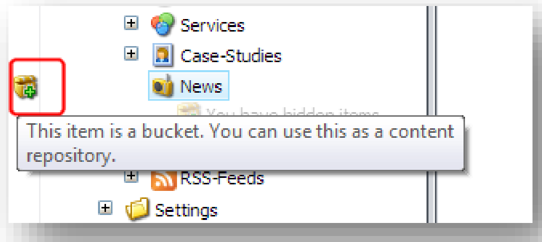


Item bucket showing folder structure



Note
By default the item bucket folder structure is kept hidden from content authors.

You can mark all the item buckets with an icon in the Quick Action Bar. This allows content authors to see which containers are item buckets and which are normal containers.



Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2015 Sitecore. All rights reserved.

Use Semantics Instead of the Content Tree

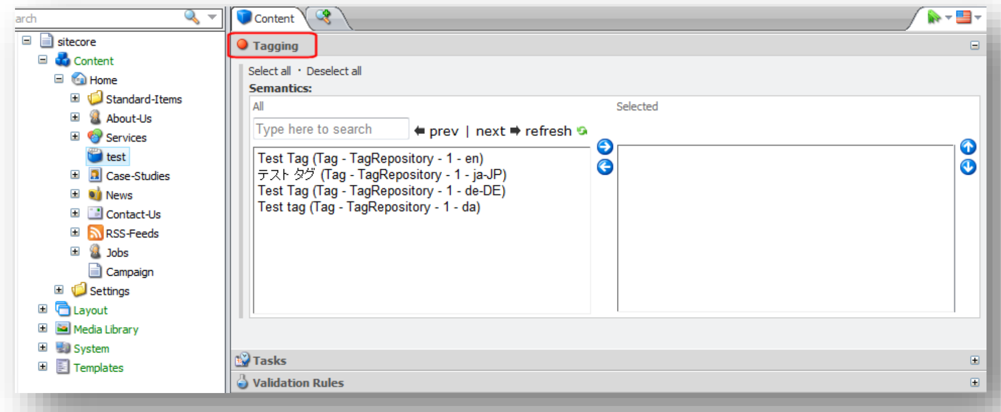
To decide if you should convert an item into an item bucket, and in-turn, hide all its descendants, you should ask yourself if you care about the structure of the data that is stored in the item bucket.

For example, if you have a *product repository*, a *media repository*, or a *tag repository* within the content tree, you might want to place all the content items in the appropriate folder. When you want to work with a particular product, media item, or tag, you can search for it and open it.

The item buckets system uses semantics to make connections, not the content tree hierarchy. For example, for a selection of products, you would traditionally create categories in the content tree and place the individual product items in these categories. With item buckets, you can place all the products in one repository and tag each product with the category that it belongs to and find them using search.

To view semantics on a content item:

1. In the Content Editor, enable standard fields. To do this in the ribbon, select the **View** tab and then select the **Standard Fields** check box.
2. Select a content item, such as '*test*'.
3. In the **Content** tab, scroll down and select the **Tagging** group.



Improved Scalability

Item buckets help you to manage the problem of storing large numbers of items in the Sitecore content tree, by retrieving them quickly using search, and thus enabling you to work with them in a speedy and efficient manner.

1.1.3 Terminology

This section contains a list of some key terms and definitions used with item buckets.

Item Bucket

An item bucket is a repository in the content tree that can store multiple content items. The difference between an item bucket and a normal repository is that an item bucket can store thousands of content items without displaying them in the content tree.

Bucketable

A bucketable item is a Sitecore content item that has been converted to make it compatible with item buckets.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2015 Sitecore. All rights reserved.

Structured Item

Structured items are normal Sitecore content items or containers that appear in the content tree.

Unstructured Item

Sitecore items that have been made bucketable are known as unstructured items.

Facet

Facets are the categories of search results that appear to the left of item bucket search results.

Synchronize

After making items in an item bucket bucketable you need to synchronize them to update the structure of an item bucket.

Semantics

Item buckets use semantics such as keyword tagging to enable you to easily find items, so you do not need to navigate for these items in the content tree.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2015 Sitecore. All rights reserved.

1.2 Creating Item Buckets

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

Note
Before you start to work with item buckets, we recommend that you look through the Appendix of tips and tricks at the end of this document.

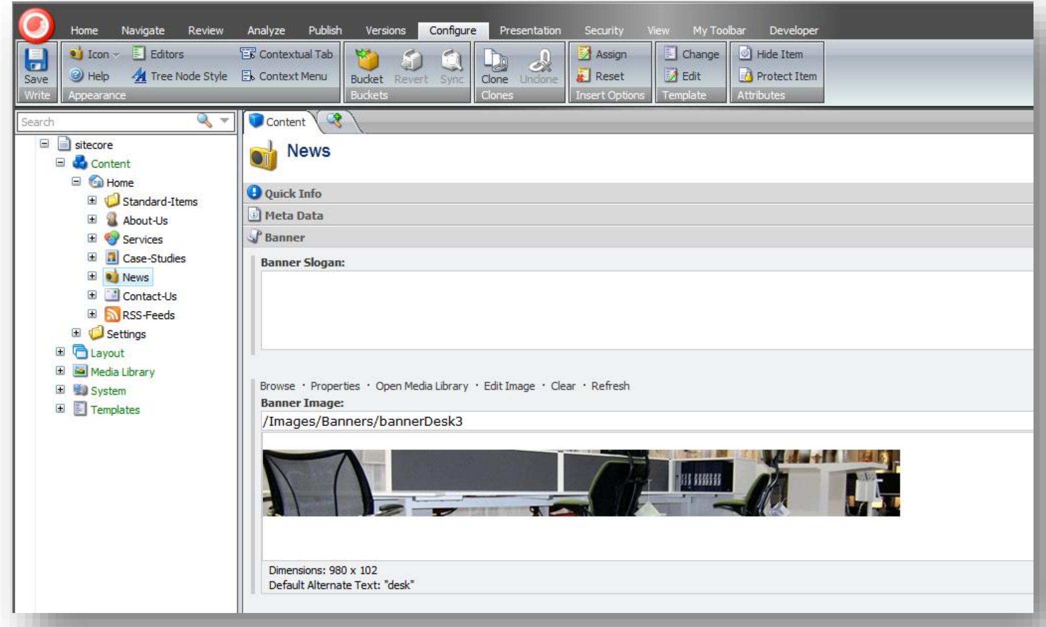
You can create an item bucket from a new content item or convert an existing item into an item bucket.

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. If the content item contains thousands or even millions of items, it can take some time to organize the content items after converting the item into a bucket. A progress bar displays a tally of the items that are being processed.

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.

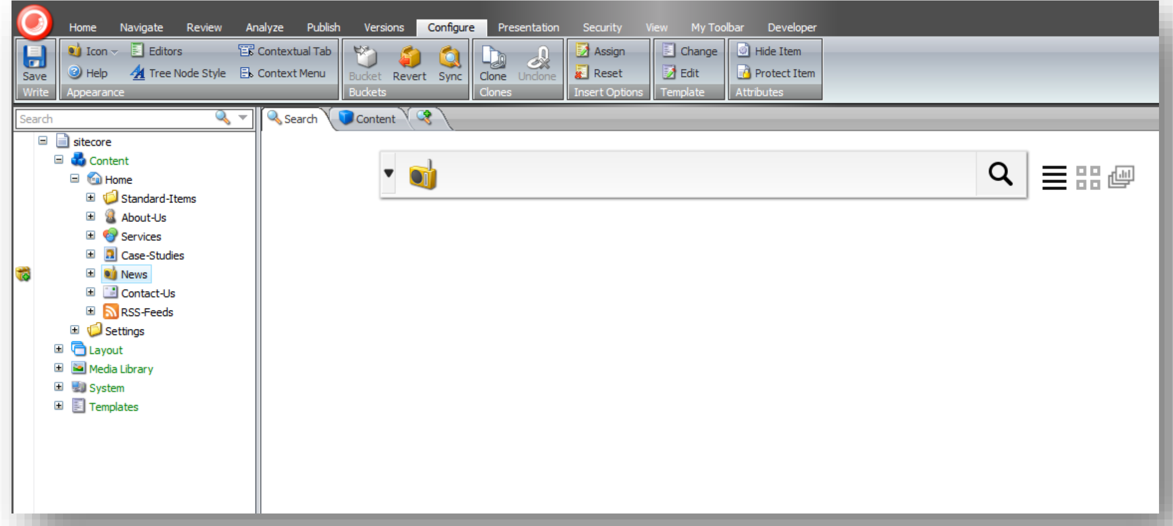
   Alternatively, select an existing content item that can expand over time to contain countless sub-items.

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.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2015 Sitecore. All rights reserved.

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



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

For more information about searching in item buckets, see the *Content Author's Cookbook*.

An icon appears in the Quick Action Bar to the left of the content tree indicating that this item is now an item bucket.

To display the item buckets icon, right-click the **Quick Action Bar,** and then select **Item Bucket**.

1.2.1 Making Content Items Bucketable

When you convert an item to an item bucket, you must ensure that the content items 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 that it is based on bucketable.

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

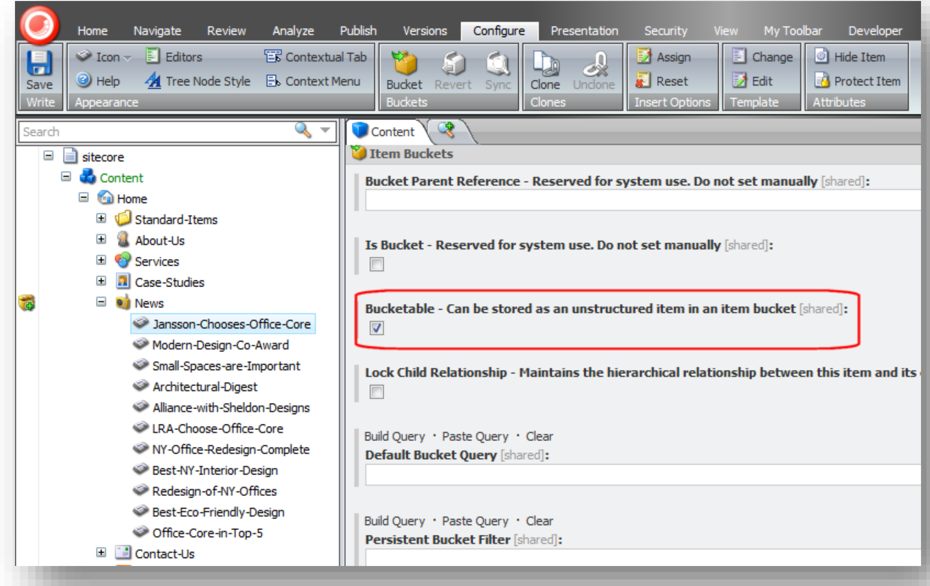
If the content items are based on a template that is not bucketable, the system does not automatically structure and hide the content items for you. Instead, the content items are treated like normal items in the content tree.

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.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2015 Sitecore. All rights reserved.

3. In the right hand pane, click the **Content** tab and scroll down and expand the **Item Buckets** section.



4. Select the **Bucketable** check box.
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 Item Buckets.

1.2.2 Hiding Items in an Item Bucket

Normal (unbucketable) content items that are stored in an item bucket are not hidden in the content tree by default. Only bucketable items can be hidden in an item bucket.

To hide content items stored in an item bucket:

1. Open the Sitecore Desktop and open the **Content Editor**.
2. In the **Content Editor**, on the **View** tab, in the **View** group, clear the **Hidden Items** check box.

We recommend that you clear the **Hidden Items** check box if you are using item buckets. This prevents the system from unnecessarily loading all the items in the content tree. You can still work with the hidden content items when you need to.

1.2.3 Making Templates Bucketable

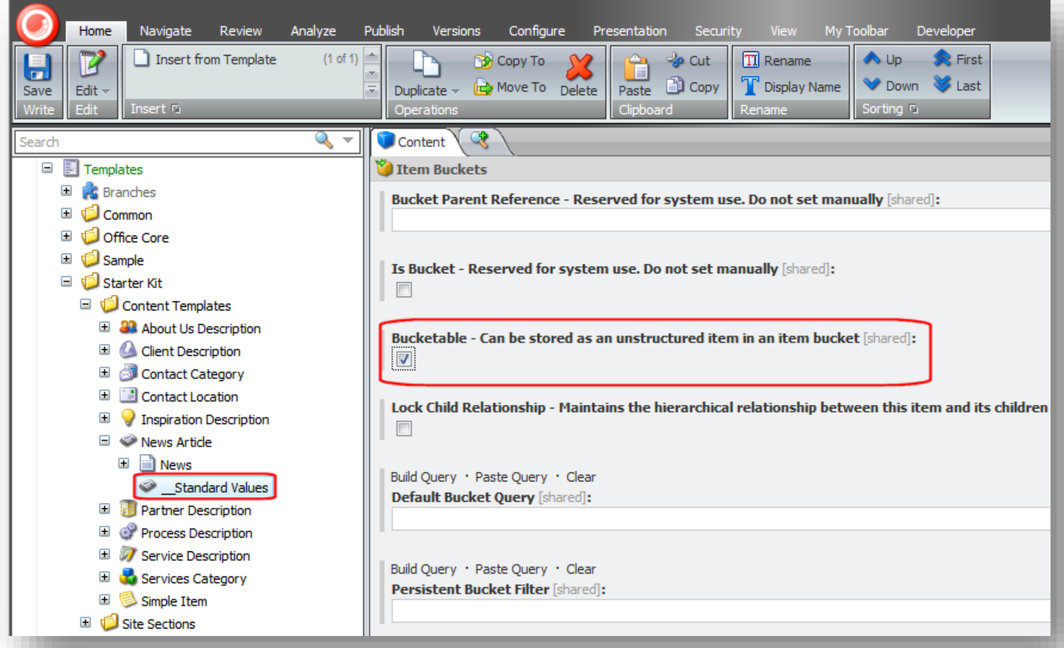
If you have a large number of similar content items that you want to hide in an item bucket, it makes more sense to make the template that they are based on bucketable.

To make a template bucketable:

1. In the **Content Editor**, on the **View** tab, in the **View** group, select the **Standard Fields** check box.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2015 Sitecore. All rights reserved.

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. Select the _**Standard Values** item to make all the content items that use this template bucketable.
6. In the **Template Manager,** in the right hand pane, click the **Content** tab.
7. Scroll down and expand the **Item Bucket** section.



8. Select the **Bucketable** check box.
9. Save your changes.

After you have made the template bucketable, you must synchronize the item bucket to update its structure.

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

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2015 Sitecore. All rights reserved.

1.3 Synchronizing Item Buckets

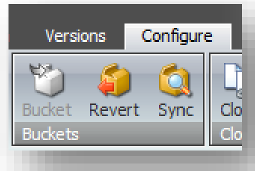
When you create an item bucket, you can store both bucketable, unstructured content items and normal, structured content items in it. If you decide to convert some of the normal content items into bucketable items or make the templates that they are based on bucketable, you must synchronize the item bucket to update its structure.

You must synchronize an item bucket when:

- You convert a content item into an item bucket and it contains bucketable items.
- You convert a content item into an item bucket and it contains content items whose template you have made bucketable.

To synchronize an item bucket:

1. In the **Content Editor**, 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 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 search for all of these content items in the item bucket.

1.3.1 Locking Parent/Child Relationships

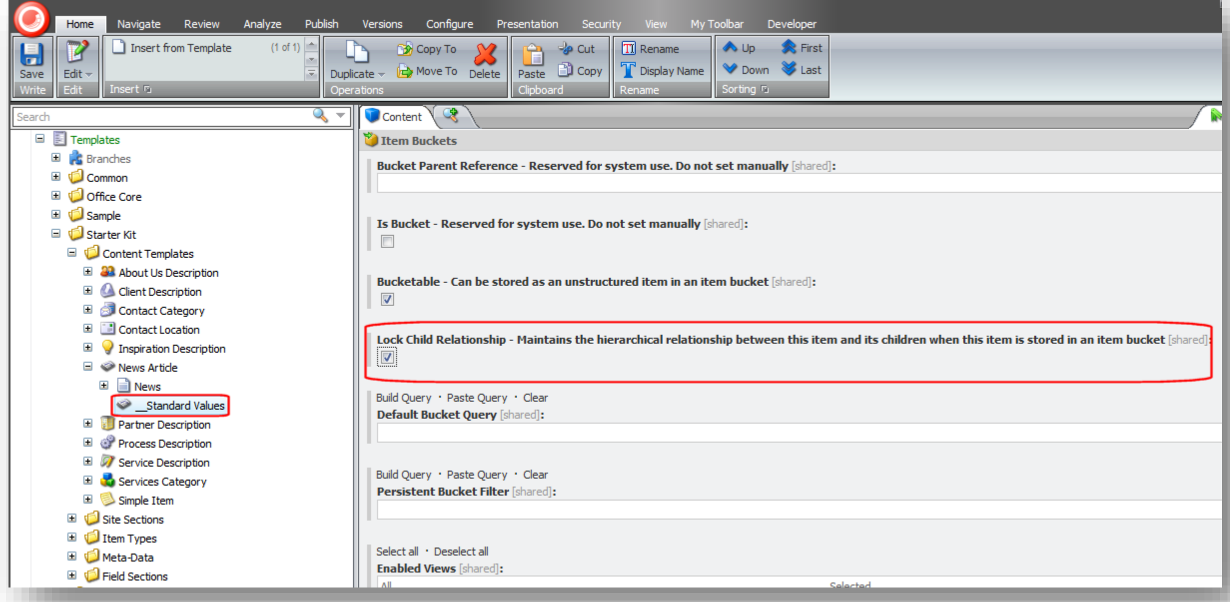
In some cases, you may want to lock the relationship between a parent item and its child items even though both are stored in an item bucket. You might need to ensure that the child items are always stored below the parent item, for example you might want to lock the parent to child relationship between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

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2015 Sitecore. All rights reserved.

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



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

Note
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.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2015 Sitecore. All rights reserved.

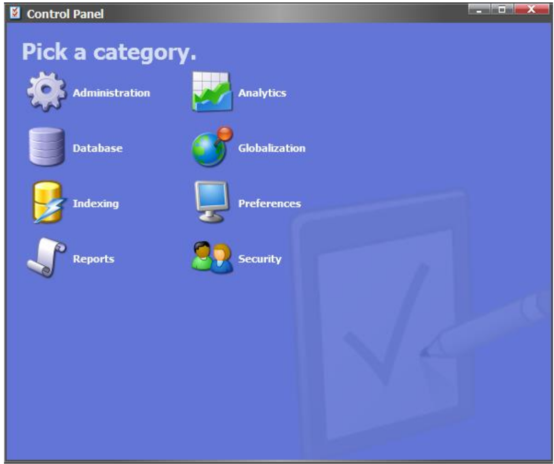
1.4 Managing Item Buckets

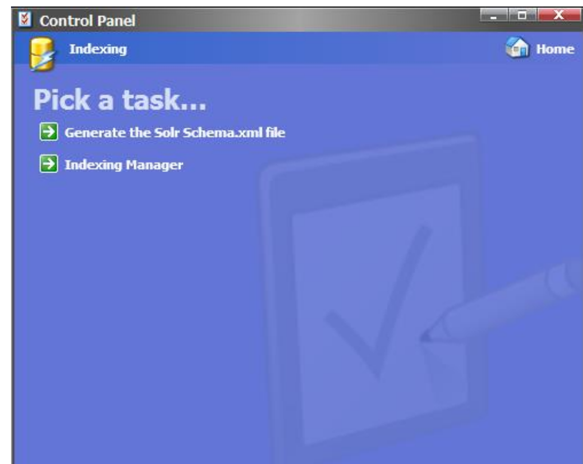
There are a number of settings and tools that you can use to configure the way item buckets work on your installation.

For example, you can re-build item bucket indexes, specify search settings, set up default search queries and, create facets and more.

1.4.1 Building the Search Indexes

You can build the item buckets search indexes from the Control Panel.





Use the Item Buckets section in the control panel to:

- Generate a Solr schema.xml file.
- Rebuild the item bucket indexes.

1.4.2 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:

- Defining the facets that are available on your website.
- Specifying the way that an item opens when you click it in the search results.
- Configuring the Search Box dropdown to add or remove search options.
- Adding or removing Quick Actions to your search results.

You can also specify which field is 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 either *Multilist* or *Multilist with Search*. Before you decide which kind of field type you would like to use for your tags, you should think about how many tags you are going to need.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2015 Sitecore. All rights reserved.

Chapter 2

Configuring Scalability in Sitecore

This chapter describes how to scale big data implementations of Sitecore CMS that use item buckets, and Sitecore search and indexing.

If you intend to store thousands or millions of items in Sitecore then you need to make your solution more scalable. Read this chapter to learn more about strategies such as using multiple indexes, sharding, and performing remote querying.

This chapter contains the following sections:

- Configuring Cache
- Multiple Search Indexes
- Configuring Scalability Settings

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2015 Sitecore. All rights reserved.

2.1 Configuring Cache

You can configure Sitecore cache settings to improve performance and scalability. The standard cache settings are in the `web.config` file found in the Website root folder.

Some preliminary work has been done to preconfigure the cache levels for a site that contains 100,000 items or more. You may need to adjust these settings depending on the number of items in your content tree and the type of Sitecore implementation you have.

It is tempting to assign more and more cache as your item count grows, however this is not an effective scaling strategy. The data cache and item cache handle whatever you assign to them but performance is not always better. For example, you could specify 800MB of cache but this means you have the additional overhead of looking up a much a bigger cache. Creating a bigger cache does not necessarily mean you achieve better performance or improve scalability.

Note
In the web.config, the *EnableEvents* setting is set to *true* by default. This setting makes scaling configuration simpler by enabling the event queue for publishing items out of the box. If you have hundreds or thousands of records the event queue must be cleaned up regularly to avoid performance issues. Reducing the execution interval of the cleanup agent can help you to achieve this. In previous versions of Sitecore this was set to false. It is now set to true by default because new Index Strategies require that the Events system is enabled.

2.1.1 Custom Cache Settings

Sitecore database cache settings contained in the `web.config` file:

```
<!-- core -->
<database id="core" singleInstance="true" type="Sitecore.Data.Database, Sitecore.Kernel">
        <cacheSizes hint="setting">
          <data>20MB</data>
          <items>10MB</items>
          <paths>500KB</paths>
          <itempaths>10MB</itempaths>
          <standardValues>500KB</standardValues>
        </cacheSizes>
      </database>

<!-- master -->
<database id="master" singleInstance="true" type="Sitecore.Data.Database, Sitecore.Kernel">
        <cacheSizes hint="setting">
          <data>20MB</data>
          <items>10MB</items>
          <paths>500KB</paths>
          <itempaths>10MB</itempaths>
          <standardValues>500KB</standardValues>
        </cacheSizes>
      </database>

<!-- web -->
<database id="web" singleInstance="true" type="Sitecore.Data.Database, Sitecore.Kernel">
        <cacheSizes hint="setting">
          <data>20MB</data>
          <items>10MB</items>
          <paths>4MB</paths>
          <itempaths>10MB</itempaths>
          <standardValues>4MB</standardValues>
        </cacheSizes>
      </database>
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2015 Sitecore. All rights reserved.

2.2 Multiple Search Indexes

If you want to use item buckets to support millions of content items on your website, you need to make your search more scalable, so the best approach may be to create multiple search indexes.

Choosing an appropriate index strategy is something that should be discussed during the design phase of a project, such as whether to shard or not? One index satisfies the needs of most Sitecore solutions but multiple indexes offer more scaling possibilities.

For example, you could split the different parts of your website into the following indexes:

- *Index 1* – Content
- *Index 2* – System
- *Index 3* – Media library

Advantages of Using Multiple Search Indexes

- Spread the load by storing each index on a different server – This makes indexing quicker.
- Better performance – If you have millions of items search is quicker.
- Smaller indexes on file.
- You can set very specific configuration settings for different parts of the content tree.

If you require an even more scalable index and greater querying power, we recommend using Solr as your provider. For more information, see Chapter 3, Extending Scalability with Solr.

Configuring Multiple Search Indexes

To make it easier to create multiple search indexes, Sitecore includes all the configuration files you need to create a sharded or non sharded setup. Enable whichever configuration file that you want to use. If the default configuration files are not sharded enough, it is quite easy to append your own by adding the appropriate configuration.

Use the code sample and table below, which outlines what elements you need to add.

```
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/">
  <sitecore>
    <contentSearch>
      <configuration type="Sitecore.ContentSearch.LuceneProvider.LuceneSearchConfiguration,
       Sitecore.ContentSearch.LuceneProvider">
        <indexes hint="list:AddIndex">
          <index id="sitecore_core_index"
           type="Sitecore.ContentSearch.LuceneProvider.LuceneIndex,
            Sitecore.ContentSearch.LuceneProvider">
            <param desc="name">$(id)</param>
            <param desc="folder">$(id)</param>
            <!-- This initializes index property store. Id has to be set to the index id -->
            <param desc="propertyStore" ref="contentSearch/databasePropertyStore"
             param1="$(id)" />
            <strategies hint="list:AddStrategy">
              <!-- NOTE: order of these is controls the execution order -->
              <strategy ref="contentSearch/indexUpdateStrategies/intervalAsyncCore" />
            </strategies>
            <commitPolicy hint="raw:SetCommitPolicy">
              <policy type="Sitecore.ContentSearch.TimeIntervalCommitPolicy,
               Sitecore.ContentSearch" />
            </commitPolicy>
            <commitPolicyExecutor hint="raw:SetCommitPolicyExecutor">
              <policyExecutor type="Sitecore.ContentSearch.CommitPolicyExecutor,
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2015 Sitecore. All rights reserved.

```
              Sitecore.ContentSearch" />
            </commitPolicyExecutor>
            <locations hint="list:AddCrawler">
              <crawler type="Sitecore.ContentSearch.LuceneProvider.Crawlers.DefaultCrawler,
               Sitecore.ContentSearch.LuceneProvider">
                <Database>core</Database>
                <Root>/sitecore</Root>
              </crawler>
            </locations>
          </index>
        </indexes>
      </configuration>
    </contentSearch>
  </sitecore>
</configuration>
```

Name	Description	Example
`<Root>`	Root node of the index. Specify the root node of the content tree to be included in the index. The crawler indexes content below this location.	`<Root>/sitecore/media library</Root>`
`<name>`	Name of the search index.	`<param desc="name">$(id)</param>`
`<Database>`	Database name.	`<Database>core</Database>`
`<strategies>`	List of index strategies to run.	`<strategies hint="list:AddStrategy"> <strategy ref="contentSearch/indexUpdateStrategies/intervalAsyncCore" /> </strategies>`
`<CommitPolicy>`	Controls when the index commits what it has either in memory or in temporary files to disk. This can be time based or document count based.	`<commitPolicy hint="raw:SetCommitPolicy"> <policy type="Sitecore.ContentSearch.TimeIntervalCommitPolicy,Sitecore.ContentSearch"/> </commitPolicy>`
`<commitPolicyExecutor>`	The class that executes the commit.	`<commitPolicyExecutor hint="raw:SetCommitPolicyExecutor"> <policyExecutor type="Sitecore.ContentSearch.CommitPolicyExecutor, Sitecore.ContentSearch" /> </commitPolicyExecutor>`

Index Context Switcher

If you decide to use the sharded approach to indexing, Sitecore uses the `<Root>` element in relation to the `Context.Item` to determine which index to use. This index switching is done for you.

It is important to note that the more specific your `<Root>` is, the higher it needs to be listed in the configuration file. The Index Context Switcher uses the indexes in the order they are listed.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2015 Sitecore. All rights reserved.

For example, if you have an index <Root> of /sitecore/content/Home, it should be located below the index for a <Root> of /sitecore/content/Home/Flights.

```
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/">
  <sitecore>
    <contentSearch>
      <configuration type="Sitecore.ContentSearch.LuceneProvider.LuceneSearchConfiguration,
       Sitecore.ContentSearch.LuceneProvider">
        <indexes hint="list:AddIndex">
          <index id="sitecore_core_index"
           type="Sitecore.ContentSearch.LuceneProvider.LuceneIndex,
            Sitecore.ContentSearch.LuceneProvider">
            <param desc="name">$(id)</param>
            <param desc="folder">$(id)</param>
            <!-- This initializes index property store. Id has to be set to the index id -->
            <param desc="propertyStore" ref="contentSearch/databasePropertyStore"
             param1="$(id)" />
            <strategies hint="list:AddStrategy">
              <!-- NOTE: order of these is controls the execution order -->
              <strategy ref="contentSearch/indexUpdateStrategies/intervalAsyncCore" />
            </strategies>
            <commitPolicy hint="raw:SetCommitPolicy">
              <policy type="Sitecore.ContentSearch.TimeIntervalCommitPolicy,
               Sitecore.ContentSearch" />
            </commitPolicy>
            <commitPolicyExecutor hint="raw:SetCommitPolicyExecutor">
              <policyExecutor type="Sitecore.ContentSearch.CommitPolicyExecutor,
               Sitecore.ContentSearch" />
            </commitPolicyExecutor>
            <locations hint="list:AddCrawler">
              <crawler type="Sitecore.ContentSearch.LuceneProvider.Crawlers.DefaultCrawler,
               Sitecore.ContentSearch.LuceneProvider">
                <Database>core</Database>
                <Root>/sitecore</Root>
              </crawler>
            </locations>
          </index>
        </indexes>
      </configuration>
    </contentSearch>
  </sitecore>
</configuration>
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2015 Sitecore. All rights reserved.

2.3 Configuring Scalability Settings

When you install Sitecore with item buckets, by default several configuration files are added to the `App_Config\Include` folder. This section explains the customizable settings related to scalability contained in the following config files:

- Sitecore.Buckets.config
- Sitecore.ContentSearch.Lucene.DefaultIndexConfiguration.config
- Web.config
- Sitecore.Buckets.Scaling.config
- Custom Crawler
- Sharding config files

The `Sitecore.Buckets.config` file contains the following settings which you can use to improve the scalability of your Sitecore solution:

Setting Name and Description	Example
BucketTriggerCount If you enable the AutoBucket events, this setting specifies the maximum number of children that an item can have before it is automatically converted into an item bucket. Example: When an item has 100 children, Sitecore asks if you want to automatically convert the parent item into an item bucket.	`<setting name="BucketConfiguration.BucketTriggerCount" value="100"/>`
BucketTemplateId To change the template of a folder item that contains all the hidden bucketable items, you must change this setting to point to the GUID of the new folder item. We recommend you use the default value.	`<setting name="BucketConfiguration.BucketTemplateId" value="{ADB6CA4F-03EF-4F47-B9AC-9CE2BA53FF97}" />`
SecuredItems This setting determines what happens to results that are returned when a user does not have permission to access them. The options are "*hide*" and "*blur*".	`<setting name="BucketConfiguration.SecuredItems" value="hide"/>`

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2015 Sitecore. All rights reserved.

The `Sitecore.ContentSearch.DefaultIndexConfiguration.config` file contains the following setting:

Setting Name and Description	Example
ContentSearch.LuceneQueryClauseCount This setting allows you to move the clause count for Lucene up and down depending on the size of your queries. Increasing this value increases memory consumption. Only increase it if you need to run very large queries.	`<setting name="ContentSearch.LuceneQueryClauseCount" value="1024"/>`

The Sitecore `web.config` file contains the following setting:

Setting Name and Description	Example
Indexing.UpdateInterval This interval determines how often the Web database index is updated. Set an index update interval to record when unstructured items are created, deleted or modified in the Web database. If you do not set an update interval these events are not automatically included in the index. This setting is particularly important when you have 100,000 or more unstructured items in your index.	`<setting name="Indexing.UpdateInterval" value="00:05:00"/>`

Note
In case of configuring Scaled Environment, on Solr only one CD instance should use the *OnPublishEndAsync* or *intervalAsync* strategy. Other instances should use manual strategy or no strategy at all. At the same time, only one CM instance should be configured for using the *intervalAsync* strategy, all the others CM instances should use manual strategy or no strategy at all.

2.3.1 Sitecore Buckets Scaling Config

Enable the `Sitecore.Buckets.Scaling.config` file if you want to use a dedicated query server. This section describes the purpose of each setting with some examples.

For example, you could designate two servers to handle search indexing:

- *Server 1 – Remote search indexing server* - for building and rebuilding the search indexes of an authoring server on a dedicated remote server.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2015 Sitecore. All rights reserved.

- *Server 2 – Dedicated search query server* – to send the search queries of an authoring server to a dedicated query server.

Using a Remote Indexing Server

The default bucket indexes are written in the default data directory in the file system. The default data directory is specified in the `web.config` file. Rebuilding the indexes can be quite slow and can demand a lot of resources on the web server. It is therefore a good idea to rebuild your indexes on a remote server computer.

Advantages	**Disadvantages**
The remote server can have an SSD drive that makes indexing a lot faster.	While the index is copied back to the local data directory, there is a short time when the index is locked for reading and writing.
Only the performance of the remote server used for re-indexing is affected, not the computer that is delivering the content.	

In the `Sitecore.Buckets.Scaling.config`, find the `RemoteIndexLocation` setting, and enter a network path that has full read and write access.

```
  <setting name="RemoteIndexLocation" value="c:\remote"/>
```

You can then specify your index:

```
configuration xmlns:patch="http://www.sitecore.net/xmlconfig/">
  <sitecore>
    <contentSearch>
      <configuration type="Sitecore.ContentSearch.LuceneProvider.LuceneSearchConfiguration,
         Sitecore.ContentSearch.LuceneProvider">
        <indexes hint="list:AddIndex">
          <index id="sitecore_master_index"
            type="Sitecore.ContentSearch.LuceneProvider.LuceneIndex,
            Sitecore.ContentSearch.LuceneProvider">
            <param desc="name">$(id)</param>
            <param desc="folder">$(id)</param>
      <!-- This initializes index property store. Id has to be set to the index id -->
       <param desc="propertyStore" ref="contentSearch/databasePropertyStore" param1="$(id)" />
            <strategies hint="list:AddStrategy">
              <!-- NOTE: order of these controls is the execution order -->
              <strategy ref="contentSearch/indexUpdateStrategies/syncMaster" />
            </strategies>
            <locations hint="list:AddCrawler">
               <crawler type="Sitecore.ContentSearch.LuceneProvider.Crawlers.DefaultCrawler,
                  Sitecore.ContentSearch.LuceneProvider">
                <Database>master</Database>
               <Root>/sitecore</Root>
              </crawler>
            </locations>
          </index>
        </indexes>
      </configuration>
    </contentSearch>
  </sitecore>
</configuration>
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2015 Sitecore. All rights reserved.

Using a Dedicated Query Server

You can specify a dedicated query server that runs all the search queries that are generated in the UI. This may be useful if you have many content authors and you do not want the overhead of running queries that might affect the performance of the rest of the system.

Use the *QueryServer* setting, to route all the queries to a dedicated authoring server.

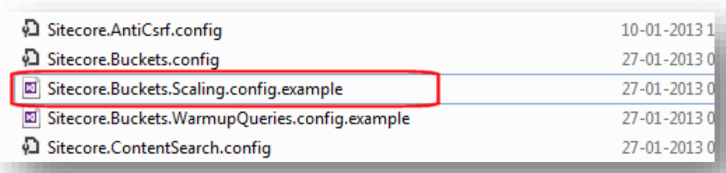
You can use several query servers if necessary with some authoring servers using *QUERYSERVER 1* and other authoring servers using *QUERYSERVER 2*. Although this is possible with the Lucene provider we recommend that you use SOLR if you want to implement distribution of search queries or indexing.

Note
You can shard both Lucene and Solr search indexes across multiple remote servers. The settings in this section can apply equally to both Lucene and Solr implementations.

Enabling the Sitecore.Buckets.Scaling.config

The `Sitecore.Buckets.Scaling.config` file is in your website *Include* folder:
wwwroot\<sitename>\Website\App_Config\Include

To start defining your scalability settings, enable the `Sitecore.Buckets.Scaling.config` file by removing the `.example` extension from the file name.



Configuring Scalability Settings

For each setting listed there is a description and an example of usage.

Events

Name and Description	Example
Remote Indexing Events Every time a change is made to a Sitecore item an event is added to the event queue and sent to a remote server for indexing. These events allow developers to implement code that will run when the indexing process starts and/or ends on remote servers.	<event name="item:indexing:remotestarting"/> <event name="item:indexing:remotefinished"/>

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2015 Sitecore. All rights reserved.

Name and Description	Example
Query Server Address This enables you to specify a dedicated query server for your content management server. In the value parameter, enter a URL address for the remote server.	`<setting name="Scaling.QueryServer" value="http://queryserver"/>`
Remote Index Rebuild Location This enables you to specify a specific location on the indexing server where the indexes will be rebuilt. In the value parameter, enter the path to the folder you want to specify.	`<setting name="Scaling.RemoteIndexLocation" value="c:\remote"/>`
Remote Index Drop Location Use this setting if you want to spread indexes across multiple locations. This is a read only drop location that the built indexes get moved to so other servers can copy these indexes into their local index directory. In the value parameter, enter the path to the drop location.	`<setting name="Scaling.NetworkDropPoint" value="\\SERVERNAME\c$\inetpub\wwwroot\sitename\Website\Data\indexes"/>`
Indexing Server Service Location This is the method that is called from the local server to the remote indexing server to start the remote rebuild. It is a web service method enabling you to specify how the indexes should be sent to the remote server. For example, by specifying a folder or location.	`<setting name="Scaling.RemoteIndexingServer" value="http://localhost/sitecore/shell/Applications/Buckets/Rebuild.asmx/Build"/>`
Search Server Copy Service This setting indicates the Web Service method called on the Initiator when a remote server has finished rebuilding the indexes, and before the remove server copies the index back to the Initiator. The Web Service method disables the History Engine, to suspend history updates as the index is copied back from the remote server to the Initiator.	`<setting name="Scaling.RemoteIndexingReceipt" value="http://localhost/sitecore/shell/Applications/Buckets/Rebuild.asmx/Reciept"/>`

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2015 Sitecore. All rights reserved.

Name and Description	Example
Service to Re-enable Indexing Engine This setting indicates the Web Service method called on the Initiator when a remote server has finished copying the index back to the Initiator. The Web Service method re-enables the History Engine, which would have been disabled before the copy operation began.	`<setting name="Scaling.RemoteIndexingReceiptEnable" value="http://localhost/sitecore/shell/Applications/Buckets/Rebuild.asmx/EnableIndexing"/>`

2.3.2 Creating a Custom Crawler

If you want to create a custom crawler to index a specific part of your website, such as the Media Library, you can adapt the sample code below.

You also need to create your own config file to implement this crawler.

This example is taken from the `Sitecore.ContentSearch.Lucene.Index.Master.config.`

```
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/">
  <sitecore>
    <contentSearch>
      <configuration type="Sitecore.ContentSearch.LuceneProvider.LuceneSearchConfiguration,
       Sitecore.ContentSearch.LuceneProvider">
        <indexes hint="list:AddIndex">
          <index id="sitecore_master_index"
           type="Sitecore.ContentSearch.LuceneProvider.LuceneIndex,
            Sitecore.ContentSearch.LuceneProvider">
            <param desc="name">$(id)</param>
            <param desc="folder">$(id)</param>
            <!-- This initializes index property store. Id has to be set to the index id -->
            <param desc="propertyStore" ref="contentSearch/databasePropertyStore"
             param1="$(id)" />
            <strategies hint="list:AddStrategy">
              <!-- NOTE: order of these is controls the execution order -->
              <strategy ref="contentSearch/indexUpdateStrategies/syncMaster" />
            </strategies>
            <commitPolicy hint="raw:SetCommitPolicy">
              <policy type="Sitecore.ContentSearch.TimeIntervalCommitPolicy,
               Sitecore.ContentSearch" />
            </commitPolicy>
            <commitPolicyExecutor hint="raw:SetCommitPolicyExecutor">
              <policyExecutor type="Sitecore.ContentSearch.CommitPolicyExecutor,
               Sitecore.ContentSearch" />
            </commitPolicyExecutor>
            <locations hint="list:AddCrawler">
              <crawler type="Sitecore.ContentSearch.LuceneProvider.Crawlers.DefaultCrawler,
               Sitecore.ContentSearch.LuceneProvider">
                <Database>master</Database>
                <Root>/sitecore</Root>
              </crawler>
            </locations>
          </index>
        </indexes>
      </configuration>
    </contentSearch>
  </sitecore>
</configuration>
```

This example declares a new index called *sitecore_master_index*. It then uses a custom crawler to tokenize and list the field types. This enables you to search within list items.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2015 Sitecore. All rights reserved.

Custom Crawler Configuration

In Lucene and Solr, you can customize the custom crawler to specify exactly which fields it should include or exclude when indexing.

Specify these settings in the context of where you make the changes, for example in the `Sitecore.ContentSearch.Lucene.DefaultIndexConfiguration.config` file.

Index All Fields

By default `IndexAllFields` is set to *true* and automatically includes all fields in the search index. This also means that any new fields added to your templates are automatically included in the index.

There are two different ways you can use this setting:

1. Set to *true* and add all the fields you would like to exclude from the index to the `ExcludeField` list.

```
<exclude hint="list:ExcludeField">
            <__display_name>{B5E02AD9-D56F-4C41-A065-A133DB87BDEB}</__display_name>
            <__Base_template>{12C33F3F-86C5-43A5-AEB4-5598CEC45116}</__Base_template>
            <__Created>{25BED78C-4957-4165-998A-CA1B52F67497}</__Created>
            <__Created_by>{5DD74568-4D4B-44C1-B513-0AF5F4CDA34F}</__Created_by>
            <__DefaultWorkflow>{CA9B9F52-4FB0-4F87-A79F-24DEA62CDA65}</__DefaultWorkflow>
```

2. Set to false and add all fields you would like to include in the index to the `IncludeField` list.

```
<include hint="list:IncludeField">
            <fieldId>{8CDC337E-A112-42FB-BBB4-4143751E123F}</fieldId>
          </include>
```

ExcludeTemplate

When the index is running you can exclude templates of a certain type. To exclude a template you must specify the GUID of the template you want to exclude.

```
<exclude hint="list:ExcludeTemplate">
    <BucketFolderTemplateId>{ADB6CA4F-03EF-4F47-B9AC-9CE2BA53FF97}</BucketFolderTemplateId>
</exclude>
```

Include Field

When the index is running you can include certain fields by supplying their ID. This is useful if `IndexAllFields` is set to *false*.

```
<include hint="list:IncludeField">
    <fieldId>{8CDC337E-A112-42FB-BBB4-4143751E123F}</fieldId>
</include>
```

Exclude Field

You can also exclude fields by specifying their ID. The default index has many of the system fields excluded already.

```
<exclude hint="list:ExcludeField">
     <__DefaultWorkflow>{CA9B9F52-4FB0-4F87-A79F-24DEA62CDA65}</__DefaultWorkflow>
     <__Lock>{001DD393-96C5-490B-924A-B0F25CD9EFD8}</__Lock>
</exclude>
```

Remove Special Fields

Special fields are standard fields that Sitecore indexes. If you do not want to search by these fields you can exclude them from the index by specifying them individually.

```
<fields hint="raw:RemoveSpecialFields">
     <remove type="both">AllTemplates</remove>
     <remove type="both">Created</remove>
     <remove type="both">DisplayName</remove>
     <remove type="both">Editor</remove>
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2015 Sitecore. All rights reserved.

```
        <remove type="both">Hidden</remove>
        <remove type="both">Icon</remove>
        <remove type="both">Links</remove>
        <remove type="both">Updated</remove>
   </fields>
```

Add Computed Index Fields

This section lets you configure computed fields which will be included into the search indexes.

```
<fields hint="raw:AddComputedIndexField">
     <field fieldName="_content" returnType="string">
       Sitecore.ContentSearch.ComputedFields.MediaItemContentExtractor,Sitecore.ContentSearch
     </field>
     <field fieldName="calculateddimension"
       returnType="stringCollection">
       Sitecore.ContentSearch.ComputedFields.CalculatedDimension,Sitecore.ContentSearch
     </field>
</fields>
```

2.3.3 Creating Multiple Search Indexes (Sharding)

The data from each of the three Sitecore databases (*master*, *web*, *core*), is by default, stored in a single Lucene search index. As your search indexes grow, if they get larger than normal, you could implement a sharding strategy to store the data from each database in its own separate search index.

If you want to split you searches indexes up by database, you can use the following config files to configure each index:

- `Sitecore.ContentSearch.Lucene.Indexes.Sharded.Core.config`
- `Sitecore.ContentSearch.Lucene.Indexes.Sharded.Master.config`
- `Sitecore.ContentSearch.Lucene.Indexes.Sharded.Web.config`

You can find each of these files in the following location:

`wwwroot\<site name> \Website\App_Config\Include`

If you are using Sitecore with item buckets and have thousands or millions of items, sharding is one strategy you could use if you want to continue using Lucene. If your search indexes continue to grow and become too large for this strategy, we recommend that you switch to using Solr.

Note
If you follow the sharded approach, you should turn off the other Lucene config files as leaving these enabled will create redundant indexes.

For more information on using Solr, see Chapter 3, Extending Scalability with Solr.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2015 Sitecore. All rights reserved.

Chapter 3

Extending Scalability with Solr

Solr is an open source enterprise search platform from the Apache Lucene project. Using Solr integration for search is the same as using a file-based Lucene index but instead of storing its information in file-based Lucene indexes they are stored remotely on the Solr server. This is a particularly effective strategy in a distributed environment.

Solr is optimized for performance and handling large numbers of documents.

This chapter contains the following sections:

- Benefits of Using Solr
- Configuring Solr to work with Sitecore
- Configuring an IOC Container
- Configuring Sitecore to work with Solr
- Re-building the Search Indexes
- Troubleshooting

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2015 Sitecore. All rights reserved.

3.1 Benefits of Using Solr

Solr is an open source enterprise search platform from the Apache Lucene project that is available as a pre-configured Sitecore module.

When you use Solr integration for Item buckets the experience should be exactly the same as using Lucene except that instead of storing information in file-based Lucene indexes it is stored on a Solr server.

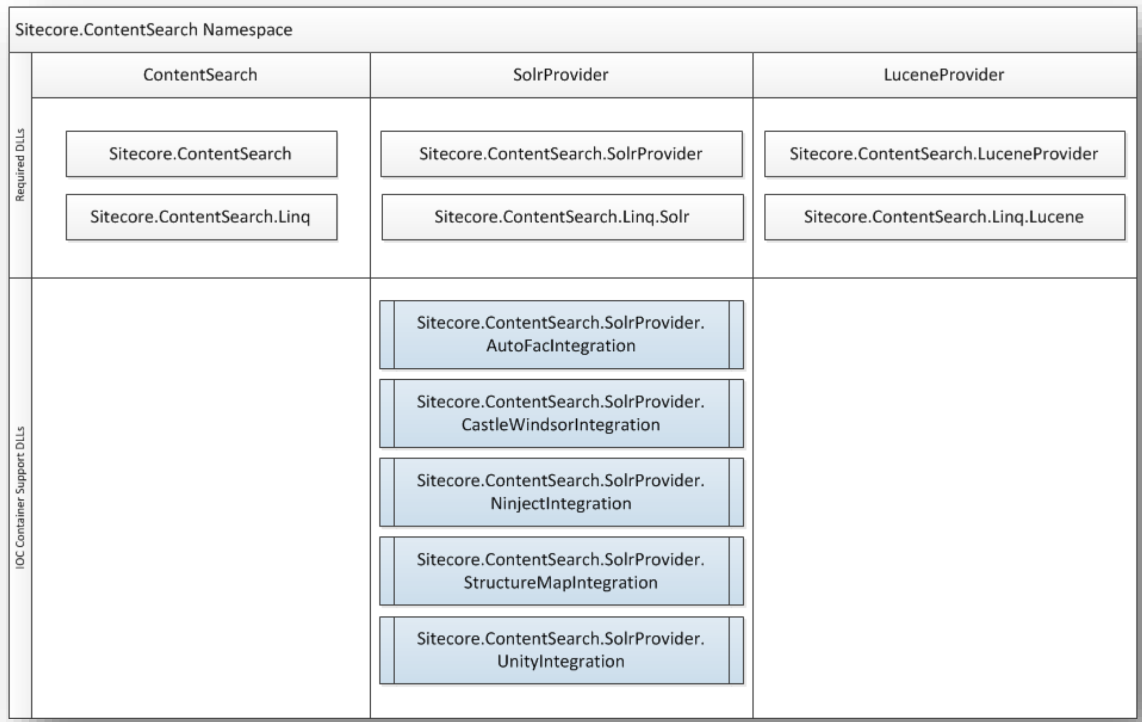
Benefits of using Solr:

- It is heavily optimized for search performance with a powerful query cache.
- It can handle a large number of documents.
- There is detailed configuration support for language indexing.
- You can shard indexes across multiple servers.
- With Solr Cloud you can also split an index across multiple physical locations.

The Sitecore.ContentSearch namespace

The `Sitecore.ContentSearch` namespace works from a provider based model. This means that any search operations, such as indexing or searching, get passed down to whichever provider is enabled. This pluggable design means that you can add support for more search providers over time.

Lucene is still the default search provider but the `Sitecore.ContentSearch` namespace now also works separately, not being so tightly coupled with Lucene.



Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2015 Sitecore. All rights reserved.

Summary of setup steps

In this chapter learn how to configure Solr provider support for Sitecore.

There are four key parts to the Solr setup:

1. Configure a Solr instance to use with Sitecore.
2. Configure an IOC container.
3. Configure Sitecore to work with the Solr provider.
4. Re-index all content items.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2015 Sitecore. All rights reserved.

3.2 Configuring Solr to work with Sitecore

Follow the steps in this section to configure a Solr instance to use with Sitecore.

Important
Your Sitecore installation comes with a Sitecore Solr Support zip package which includes several DLLs you can use depending on which IOC container you choose. The distribution also includes a *Sitecore.ContentSearch.Solr.Indexes.config* file and a ReadMe file.

3.2.1 Preparing Solr

At time of writing the most current release of Solr is version 4.1.0

Version 4.1.0 (stable): http://www.apache.org/dyn/closer.cgi/lucene/solr/4.1.0

There are several different ways to install Solr, for example in Jetty or Tomcat / Linux or Windows, your system administrator can help you choose the best approach. There are also many configuration options depending on your environment, size of installation or the amount of documents you need to index.

Before proceeding any further, please ensure that your Solr instance starts up with no errors and you are able to access the administration screens.

Note
This document does not explain all the configuration options available in Solr, it only documents configuration settings specific to Sitecore search and indexing with item buckets.

Recommended Solr reading:

http://www.packtpub.com/apache-solr-3-enterprise-search-server/book

http://www.packtpub.com/solr-3-1-enterprise-search-server-cookbook/book

http://www.packtpub.com/apache-solr-4-cookbook/book

3.2.2 Creating a Solr Core

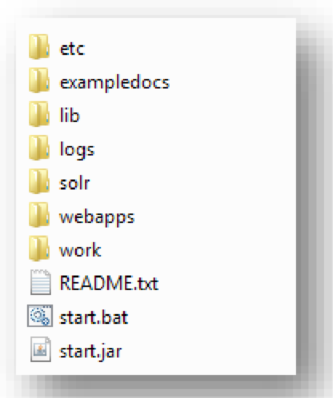
You need to create a Solr core to store all your index data. For example, the data collected from Sitecore content items. We recommend to create separate cores for each Sitecore index. Some methods are executed in terms of the context core disregarding the current index, which can lead to inconsistent and unexpected results.

To create a Solr core:

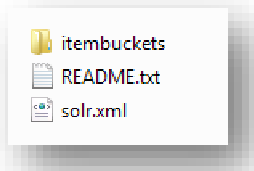
1. In Solr, navigate to the **solr** directory.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2015 Sitecore. All rights reserved.

Your root Solr installation should have a directory structure that looks similar to this:



2. Create an **itembuckets** folder in the Solr root. In this example no other cores are present.



Note
The name of the core does not have to be *itembuckets*. If you give it a different name, ensure that you update the index references in the file `Sitecore.ContentSearch.Solr.Indexes.config` (as explained in the next step).

3. Open the `solr.xml` file and update the core reference by entering the name of the home directory for the Solr core. In this example, the core is called *item buckets*.

```
<cores
    adminPath="/admin/cores"
    defaultCoreName="itembuckets"
    host="${host:}"
    hostPort="${jetty.port:}"
    hostContext="${hostContext:}"
    zkClientTimeout="${zkClientTimeout:15000}">
  <core name="itembuckets" instanceDir="itembuckets" />
</cores>
```

The **itembuckets** folder is now the home directory for this Solr core.

4. In the **itembuckets** folder, create the following three folders or sub directories:

Directory	Description
`conf/`	This directory is mandatory and must contain your `solrconfig.xml` and `schema.xml`. Any other optional configuration files are also stored here. You can find an example `solrconfig.xml` included in your Solr distribution.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2015 Sitecore. All rights reserved.

Directory	Description
`data/`	This directory is the default location for your index, and is used by the replication scripts for dealing with snapshots. You can override this location in the `conf/solrconfig.xml`. Solr creates this directory if it does not already exist.
`lib/`	This directory is optional. If it exists, Solr loads any Jars found in this directory and uses them to resolve any *plugins* specified in the `solrconfig.xml` or `schema.xml`. For example, *Analyzers* or *Request Handlers*. Alternatively you can use the `<lib>` syntax in `conf/solrconfig.xml` to direct Solr to your plugins. See the example `conf/solrconfig.xml` file for details.

To make the Social Connected work correctly with Solr, you must create a new core in Solr named *Social*. If you give the new core a different name, you must specify this name in the `Sitecore.Social.Solr.Index.Master.config` and `Sitecore.Social.Solr.Index.Web.config` files, in the **Parameters** section.

3.2.3 Generating an XML Schema for Solr

The main difference between the default Lucene instance and Solr is that Solr needs a defined xml schema when working with documents.

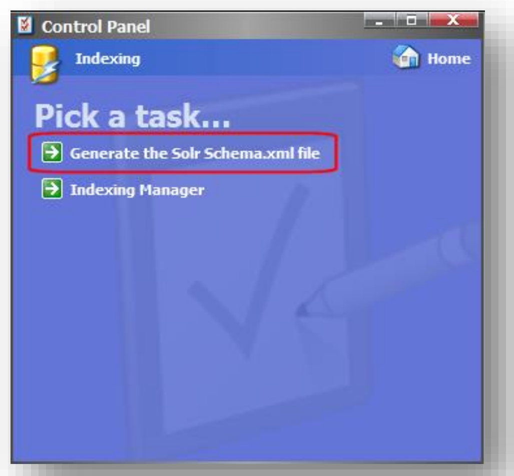
You can modify an existing schema using the Sitecore *Build Solr Schema* tool. This tool automatically generates a basic schema and ensures all the fields that Sitecore needs are present. You can add your own fields to this schema as long as you do not change the system index fields.

To generate a new Solr `schema.xml` file:

1. In the Sitecore Desktop, open the Control Panel. In the Control Panel, click **Indexing**.

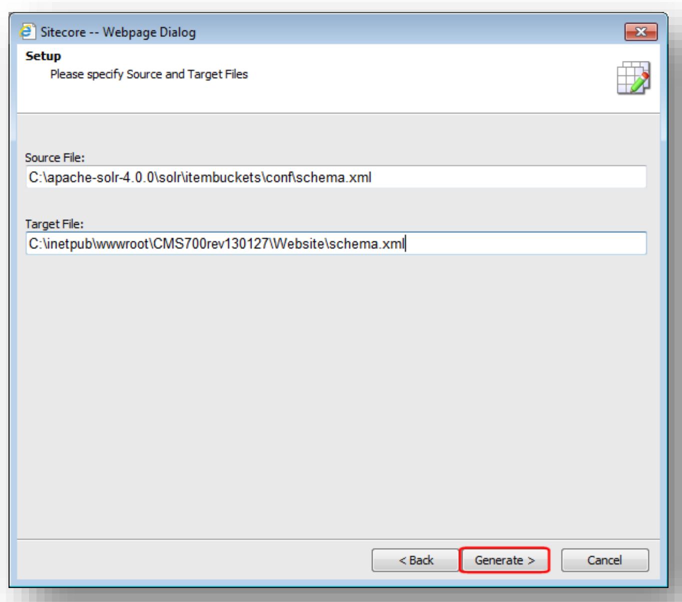
2. Then click **Generate the Solr Schema.xml file**.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2015 Sitecore. All rights reserved.



3. In the **Build Schema** window, in the **Source** field, enter the path to the Solr schema that you want to use, for example: `C:\apache-solr-4.0.0\solr\itembuckets\conf\schema.xml`

   In the **Target File** field, enter the destination for the new schema file, for example the website root folder: `C:\inetpub\wwwroot\<sitename>\Website\schema.xml`



4. Click **Generate**.

When you click Generate this updates the `schema.xml` and adds all the necessary fields that Sitecore needs.

When the tool has finished, copy the file to the *conf* directory in the `itembuckets/conf` folder. It must also be renamed to `schema.xml`.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2015 Sitecore. All rights reserved.

Note
If you have any other field definitions, copy fields or dynamics fields configured in your schema they are overwritten by the schema generator. To preserve these fields, copy your original schema and merge it with the newly generated schema afterwards.

3.2.4 Enabling Solr Term Support

When you implement Solr with Sitecore you need to enable term support in the Solr search handler.

The term functionality is built into Solr but is disabled by default. To power the dropdowns in the UI you must enable the terms component.

Note
In Lucene term support is enabled by default.

To enable Solr term support:

1. In your Solr installation, open the `solrconfig.xml` file. This file is located in the **conf** folder along with the `schema.xml` file. The path to this file is something like this:

   `apache-solr-4.0.0\solr\itembuckets\conf`

2. In the `solrconfig.xml` file, locate the `requestHandler` node:

```
<requestHandler name="/select" class="solr.SearchHandler">
```

3. Inside the `requestHandler` node, you need to add the following line:

```
<bool name="terms">true</bool>
```

   Add this line under the "`defaults`" node:

```
<lst name="defaults">
   <str name="echoParams">explicit</str>
   <int name="rows">10</int>
   <str name="df">text</str>
   <bool name="terms">true</bool>
</lst>
```

4. Also add the following node after the "`defaults`" element.

```
<arr name="last-components">
   <str>terms</str>
</arr>
```

5. Save your changes.

3.2.5 Verifying that Solr is Running Correctly

After generating a new `schema.xml` file and updating the `solrconfig.xml` file you need to verify that Solr runs correctly.

To check Solr starts up correctly:

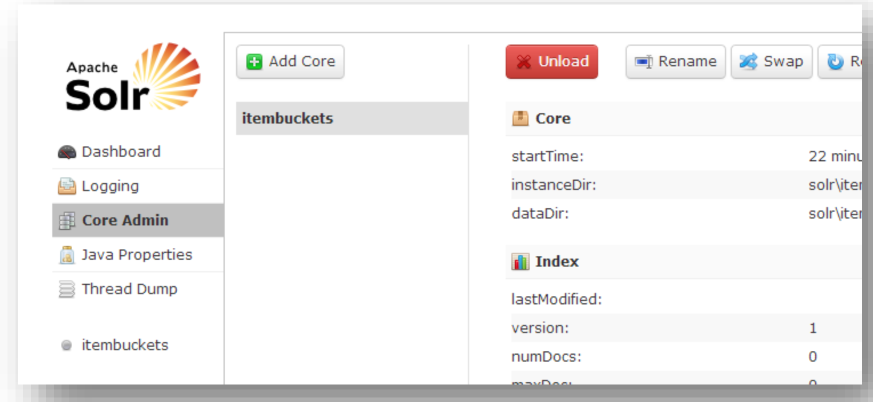
1. Ensure Solr is not running. A full restart is needed so that it loads the new configuration.
2. Start up Solr.

   Note
   How you start up Solr is dependent on the type of installation you have.

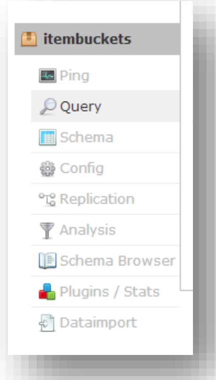
3. Check the output log files. If there are no errors, then open the Solr administration page.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2015 Sitecore. All rights reserved.

To open the administration page, enter the following URL in your browser: `http://localhost:8983/solr/admin`.



4. Select the **CoreAdmin** tab and then click **itembuckets** in the left hand navigation.



5. If you can see the Solr administration page and do not get any errors in the log files, then Solr is running correctly.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2015 Sitecore. All rights reserved.

3.3 Configuring an IOC Container

The Sitecore Solr provider makes use of an IOC (Inversion of Control) container so that all the elements inside it are swappable without the need for re-compilation.

Sitecore supports five of the most commonly used open source IOC containers, all of which are currently available on the NuGet website.

Versions currently supported:

- *Castle Windsor v3.1.0.0*
- *AutoFac v2.5.2*
- *Ninject v3.0.0*
- *StructureMap v2.6.2*
- Unity v2.1.505

Note
As these projects are open source they are often updated. You may have to request a specific version of the container from NuGet using the –Version switch in the NuGet command line.

3.3.1 Selecting the Correct Support DLL files

When you have chosen a suitable IOC container ensure that you include the correct support DLLs in the bin directory alongside the DLLs installed for the container.

Depending on your container, ensure that you also copy the following DLL files into the bin folder:

IOC container	DLL files
Castle Windsor	• SolrNet.dll • Microsoft.Practices.ServiceLocation.dll • Castle.Facilities.SolrNetIntegration.dll • Sitecore.ContentSearch.SolrProvider.CastleWindsorIntegration.dll
AutoFac	• SolrNet.dll • Microsoft.Practices.ServiceLocation.dll • AutofacContrib.CommonServiceLocator.dll • AutofacContrib.SolrNet.dll • Sitecore.ContentSearch.SolrProvider.AutoFacIntegration.dll
Ninject	• SolrNet.dll • Microsoft.Practices.ServiceLocation.dll • CommonServiceLocator.NinjectAdapter.dll • Ninject.Integration.SolrNet.dll • Sitecore.ContentSearch.SolrProvider.NinjectIntegration.dll

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2015 Sitecore. All rights reserved.

IOC container	DLL files
StructureMap	• SolrNet.dll • Microsoft.Practices.ServiceLocation.dll • StructureMap.SolrNetIntegration.dll • Sitecore.ContentSearch.SolrProvider.StructureMapIntegration.dll
Unity	• SolrNet.dll • Microsoft.Practices.ServiceLocation.dll • Unity.SolrNetIntegration.dll • Sitecore.ContentSearch.SolrProvider.UnityIntegration.dll

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2015 Sitecore. All rights reserved.

3.4 Configuring Sitecore to work with Solr

Follow the steps in this section to configure Sitecore to work with Solr.

Important
Before you can enable the Solr config file you must copy the *Sitecore.ContentSearch.Solr.Indexes.config* file from your *Sitecore Solr Support zip* package to your website Include folder:
wwwroot\<sitename>\App_Config\Include\

3.4.1 Enabling the Solr Config File

Your website *Include* folder contains several configuration files. Lucene search is enabled by default. If you want to use Solr you must disable the Lucene search config files and enable the Solr config file. This enables Solr integration and gives you access to all the Solr specific configuration settings.

To switch configuration files so that Solr is enabled and Lucene is disabled:

1. Navigate to the website **Include** folder: wwwroot\<sitename>\App_Config\Include\

   Disable the following Lucene configuration files by adding .example to the file name extension.

Sitecore.Analytics.config	15-10-2012 19:42
Sitecore.Analytics.ExcludeRobots.config	15-10-2012 19:42
Sitecore.AntiCsrf.config	10-01-2013 14:21
Sitecore.Buckets.config	27-01-2013 09:36
Sitecore.Buckets.Scaling.config	27-01-2013 09:36
Sitecore.Buckets.WarmupQueries.config.example	27-01-2013 09:36
Sitecore.ContentSearch.config	27-01-2013 09:36
Sitecore.ContentSearch.Lucene.DefaultIndexConfiguration.config	27-01-2013 09:36
Sitecore.ContentSearch.Lucene.Index.Core.config	27-01-2013 09:36
Sitecore.ContentSearch.Lucene.Index.Master.config	27-01-2013 09:36
Sitecore.ContentSearch.Lucene.Index.Web.config	27-01-2013 09:36
Sitecore.ContentSearch.Lucene.Indexes.Sharded.Core.config	27-01-2013 09:36
Sitecore.ContentSearch.Lucene.Indexes.Sharded.Master.config	27-01-2013 09:36
Sitecore.ContentSearch.Lucene.Indexes.Sharded.Web.config	27-01-2013 09:36
Sitecore.ContentSearch.Solr.Indexes.config.example	01-02-2013 16:45
Sitecore.ContentSearch.VerboseLogging.config.example	27-01-2013 09:36
Sitecore.Mvc.config.disabled	10-01-2013 09:55
Sitecore.MvcAnalytics.config.disabled	10-01-2013 16:07
Sitecore.MvcExperienceEditor.config.disabled	10-01-2013 14:39
Sitecore.MvcSimulator.config.disabled	10-01-2013 16:28

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2015 Sitecore. All rights reserved.

2. Enable the `Sitecore.ContentSearch.Solr.Indexes.config` file by removing `.example` from the file name.

Sitecore.ContentSearch.Lucene.DefaultIndexConfiguration.config.example	27-01-2013 09:36
Sitecore.ContentSearch.Lucene.Index.Core.config.example	27-01-2013 09:36
Sitecore.ContentSearch.Lucene.Index.Master.config.example	27-01-2013 09:36
Sitecore.ContentSearch.Lucene.Index.Web.config.example	27-01-2013 09:36
Sitecore.ContentSearch.Lucene.Indexes.Sharded.Core.config.example	27-01-2013 09:36
Sitecore.ContentSearch.Lucene.Indexes.Sharded.Master.config.example	27-01-2013 09:36
Sitecore.ContentSearch.Lucene.Indexes.Sharded.Web.config.example	27-01-2013 09:36
Sitecore.ContentSearch.Solr.Indexes.config	01-02-2013 16:45
Sitecore.ContentSearch.VerboseLogging.config.example	27-01-2013 09:36
Sitecore.Mvc.config.disabled	10-01-2013 09:55

Note
It is important to remove, or rename, the Lucene configuration include files so that they do not get loaded. If these files are not removed you get an error message. For more information on handling this error message, see the Troubleshooting section.

3.4.2 Solr Specific Settings

The following Solr specific settings can be found in the `Sitecore.ContentSearch.Solr.Indexes.config` file.

Specifying a Solr Service Address

This setting tells Sitecore where the Solr server is located. Sitecore appends the core name so only the base address needs to be supplied.

```
<setting name="ContentSearch.Solr.ServiceBaseAddress" value="http://localhost:8983/solr" />
```

Enabling a Search Provider

This setting tells Sitecore that Solr is enabled and so attempts to connect to the Solr server the next time the index is accessed. If it cannot connect you get an error. To disable, set this back to `Lucene`, which was the default setting.

```
<setting name="ContentSearch.Provider" value="Solr" /> (Default: "Lucene")
```

Maximum Number of Search Results

This is a global setting found in the `Sitecore.ContentSearch.config` file.

This setting contains the maximum number of documents to retrieve on a single request if a limit has not been specified in the query, for example, `Take(10)`. It is important to remember, for performance reasons, when querying how many results will be returned from the query being run and to handle them correctly, for example by using paging.

```
<setting name="ContentSearch.SearchMaxResults" value="500" />
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2015 Sitecore. All rights reserved.

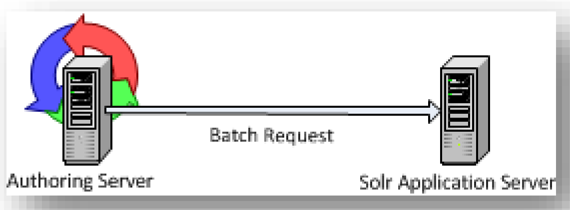
Enabling Batch Mode

When an item is indexed the composed document is saved to the search index. When the default Lucene provider is enabled then each write is being flushed to a file on the local disk. When a document is written using the Solr provider the update has to travel over a network.

When an index is rebuilt a large number of document updates are created, this could result in a lot of network traffic which is not very efficient. Therefore using batch can help to optimize the update process as your indexes grow in size.

```
<setting name="ContentSearch.Update.BatchModeEnabled" value="true" />
<setting name="ContentSearch.Update.BatchSize" value="500" />
```

Batch mode (enabled by default) takes these document updates and only flushes to the Solr server when the batch has reached a certain size.



As your index grows you may want to increase this batch size to gain the most out of this process.

3.4.3 Specifying an IOC Container

The final part is to update the `global.asax` file so that the Solr provider is loaded on application start. You can do this by instructing your application to inherit from one of the application classes provided; specific configuration is dependent on your IOC container choice.

For example, to update the `Global.asax` file to use `Castle Windsor`:

1. In your website root folder, locate the `Global.asax` file:

   `wwwroot\<sitename>\Website`

2. Open the `Global.asax` file and in the first line, replace the following:

   ```
   Inherits="Sitecore.Web.Application"
   ```

   With

   ```
   Inherits="Sitecore.ContentSearch.SolrProvider.CastleWindsorIntegration.WindsorApplication
   "
   ```

This registers the IOC (inversion of control) components for `Castle Windsor` enabling Solr integration to work correctly.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2015 Sitecore. All rights reserved.

Alternatively, replace the `Sitecore.Web.Application` reference with a reference to one of the following IOC containers:

IOC Container	Class
Castle Windsor	`<%@Application Language='C#' Inherits="Sitecore.ContentSearch.SolrProvider.CastleWindsorIntegration.WindsorApplication" %>`
Autofac	`<%@Application Language='C#' Inherits="Sitecore.ContentSearch.SolrProvider.AutoFacIntegration.AutoFacApplication" %>`
Ninject	`<%@Application Language='C#' ' Inherits="Sitecore.ContentSearch.SolrProvider.NinjectIntegration.NinjectApplication" %>`
StructureMap	`<%@Application Language='C#' ' Inherits="Sitecore.ContentSearch.SolrProvider.StructureMapIntegration.StructureMapApplication" %>`
Unity	`<%@Application Language='C#' Inherits="Sitecore.ContentSearch.SolrProvider.UnityIntegration.UnityApplication" %>`

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2015 Sitecore. All rights reserved.

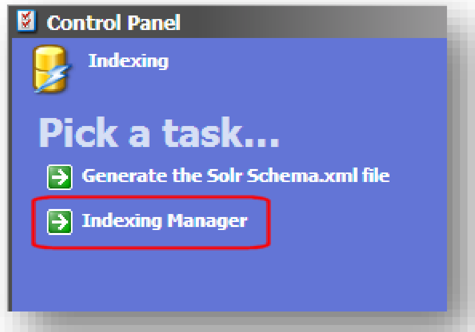
3.5 Re-building the Search Indexes

Before you can start using the Solr indexing system you need to re-index all your Sitecore content.

To rebuild the Sitecore search indexes:

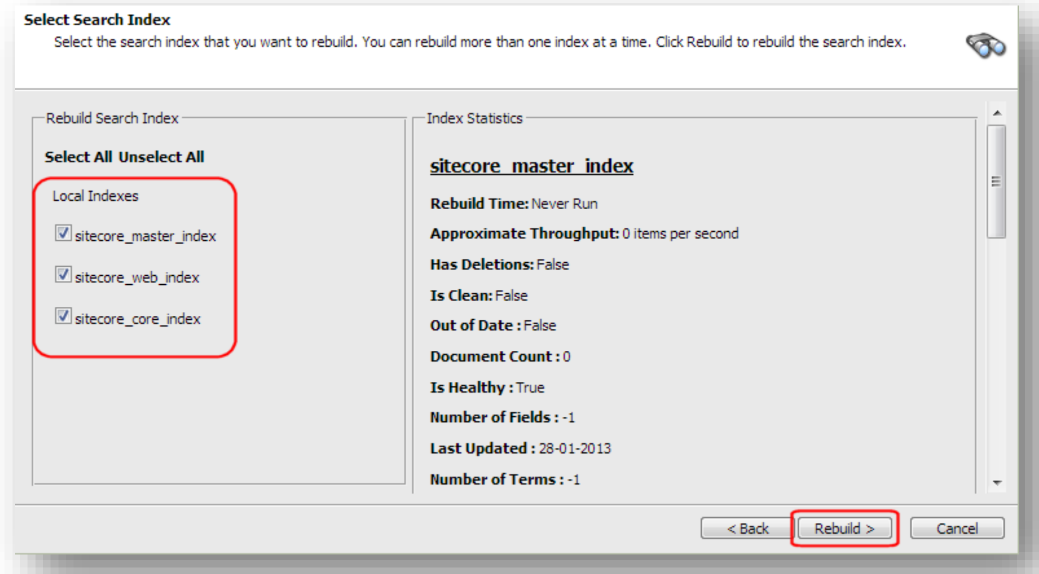
1. Log into the Sitecore Desktop.
2. Click **Start** and then click **Control Panel**.
3. In **Control Panel**, select **Indexing**.

4. Then select **Indexing Manager**.



Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2015 Sitecore. All rights reserved.

5. In the **Select Search Index** window, select all local indexes.



6. Click **Rebuild** and then click **Finish** when the Wizard has completed.

When the Wizard has finished, try to perform a search in the Sitecore Content Editor using the item buckets search functionality.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2015 Sitecore. All rights reserved.

3.6 Troubleshooting Solr

Q: I see the error *'Connection error to search provider [Solr]: Unable to connect to..'*

A: This means that Sitecore is unable to communicate with the Solr server. Check the network address in the configuration file. Also check that the Solr application server is accessible over the network and all the relevant ports are open.

Q: I see the error *'Solr configuration conflict. Do you still have other provider indexes enabled?'*

A: You have not removed all the existing Lucene configuration files from the App_Config/Include directory of your Sitecore installation. The Solr provider is unable to start if these are still present.

Q: I am already using a container. I don't want two active containers in my application, so how do I access the container instantiated by Sitecore?

A: The supplied application (where relavant) exposes a public property called *Container* which can be used to retreive the current container instance.

Q: Can I have more than one provider active at the same time?

A: No, currently you can only have one provider active per Sitecore instance.

Q: Should I use the Solr provider instead of the default Lucene provider?

A: The Lucene provider supplies identical support and excellent performance so the choice of which to use should come down to issues of scale and support. If the amount of content items becomes vast and each item contains a lot of data then you may want to look at moving your search index into Solr. Solr is designed for large numbers of items, allows the index to be sharded and scaled efficiently and also includes advanced features such as query and http caching which support the enterprise customer.

Solr also requires a separate Solr instance, whereas Lucene is local and file based. If you use Lucene you always have to assess the management and up-keep of this instance. Solr requires very little manual attention, other than the initial configuration and log/disk space management but it is another factor to consider.

Q: I search for an item field in the Sitecore search interface and I see *'field not in schema'* errors in my Solr log. It works correctly in Lucene, so what is going on?

A: The most important difference between the Lucene and Solr providers is that Solr requires a schema for all fields. The Solr Provider tries to work round this limitation by using dynamic fields to map fields that are not in the schema. You can observe this mapping in the *fieldMap* section of the *SolrProvider.config* file. If, for example, you have a field called *title* which is a rich text field type then, using the field mapping logic, it stores the field as *title_t*. This translation is handled for you in Linq requests but through the Sitecore search UI, if you want users to be able to search a certain field without its dynamic extension, you must add this field to the *schema.xml* file.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2015 Sitecore. All rights reserved.

Chapter 4

Language Support in Solr

This chapter explains how to configure the Solr provider to support different languages in Sitecore. It includes the following sections:

- Solr Schema-less Fields
- Multiple Language Support

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2015 Sitecore. All rights reserved.

4.1 Solr Schema-less Fields

When using Sitecore with Solr, items in different languages are stored separately as fields with different culture extensions.

Solr and Lucene process fields from Sitecore differently. Solr requires a schema to know how to process each field and Lucene does not. In Solr, the schema file (schema.xml) must be updated for each field you want to add to the index. This allows a great deal of flexibility in how you analyze and process the information that goes into these fields. In Sitecore it is easy to add/remove or rename template fields so it becomes very inconvenient if you have to re-generate your Solr schema.xml file every time you add a new field.

To overcome this problem Sitecore uses Solr's dynamic fields. These are a special type of index field that do not require the Solr schema to be updated every time you make a change. When Sitecore stores a new field name, the field type is mapped to a dynamic field, so there is no need to re-generate the Solr schema more than once.

Example: Defining a simple schema-less field

A Sitecore template has a field called *title* which is a *text/rich text* field.

This field is not defined in the schema so when it is indexed it instead uses the **_t* dynamic field that has been set up in the Solr `schema.xml`.

How to define a dynamic field in a Solr schema file:

```
<dynamicField name="*_t" type="text_general" indexed="true" stored="true" />
```

The field is therefore stored in the Solr index as *title_t*.

Note
Sitecore handles the adding and removing of these extensions when they are queried using *Linq To Sitecore* but if you run into problems it is important that you have a more technical understanding of how this works. The next section, Multiple Language Support explains this in more detail.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2015 Sitecore. All rights reserved.

4.2 Multiple Language Support

This section explains how Sitecore stores fields if an item has more than one language version.

4.2.1 Storing Fields for Items in Multiple Languages

Language processing is based on the default language defined in the Sitecore `web.config` file:

```
        <setting name="DefaultLanguage" value="en" />
```

Any field item in a language that does not match the default language is stored with an additional culture extension. For example, an extension of *_fr* is added to create the French language context.

Storing items with different languages as separate fields allows us to use language specific analyzers. Solr comes with several very powerful analyzers for many commonly used languages. The example in this section uses the type *text_fr* which maps to a field specifically tailored for processing French text. Including French specific stop words and stemming.

```
<!-- French -->
<fieldType name="text_fr" class="solr.TextField" positionIncrementGap="100">
  <analyzer>
    <tokenizer class="solr.StandardTokenizerFactory" />
    <!-- removes l', etc -->
    <filter class="solr.ElisionFilterFactory" ignoreCase="true"
     articles="lang/contractions_fr.txt" />
    <filter class="solr.LowerCaseFilterFactory" />
    <filter class="solr.StopFilterFactory" ignoreCase="true" words="lang/stopwords_fr.txt"
     format="snowball" enablePositionIncrements="true" />
    <filter class="solr.FrenchLightStemFilterFactory" />
    <!-- less aggressive: <filter class="solr.FrenchMinimalStemFilterFactory"/> -->
    <!-- more aggressive: <filter class="solr.SnowballPorterFilterFactory"
     language="French"/> -->
  </analyzer>
</fieldType>
```

Example: Defining a Simple Schema-less Language Version

Again using the template field *title*, in this example it is used in an item that has both *English* (default) and *French* language versions.
The default English version is stored using the default **_t* extension. So in this example, the field is stored as *title_t*.

The French version, is stored using the **_t_fr* extension as it does not match the default language.

```
        <dynamicField name="*_t_fr" type="text_fr" indexed="true" stored="true" />
```

This means the French version of this field is stored as *title_t_fr* in the Solr index.

Note
If the default language in the web.config is French (fr) then the fields are stored as *title_t* and *title_t_en* respectively.

4.2.2 Retrieving a Specific Language Version Using Linq To Sitecore

To retrieve a language specific version of an item you need to use the `IExecutionContext` object, and the `CultureExecutionContext` implementation.

When you request an object from the search context that you want to query, you can specify a `CultureExecutionContext` for a specific language.

For example, you could choose *French*:

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2015 Sitecore. All rights reserved.

```
var queryable = ctx.GetQueryable<MultiLanguage>(
    new CultureExecutionContext(CultureInfo.GetCultureInfo("fr")));
```

This means that any queries issued using this object return the language specific versions of those fields (if they exist) without the need to specifically name the field.

So an object property such as:

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

would map to *title_t* with no `CultureExecutionContext` specified and *title_t_fr* with a French `CultureExecutionContext` specified, like the example above.

Example: Updating the default language analyzer in Solr

If your default language is not English you need to change the default text extension analyzer (*_t*) to match the language you want as the default.

For example, to change the default language from *English* to *French*:

1. Update the Sitecore `web.config` setting by changing the `DefaultLanguage` setting from *en* to *fr*.

```
<setting name="DefaultLanguage" value="fr" />
```

2. Open the `Solr schema.xml` file and change the default analyzer from `text_general` analyzer (which is generic for all languages) to `text_fr` (French language). You need to change both *_t* dynamic field and the *text* regular field.

   Text general:

```
<dynamicField name="*_t" type="text_general" indexed="true" stored="true" />
<field name="text" type="text_general" indexed="true" stored="false"
multiValued="true" />
```

   Text French:

```
<dynamicField name="*_t" type="text_fr" indexed="true" stored="true" />
<field name="text" type="text_fr" indexed="true" stored="false"
multiValued="true" />
```

   This means that Solr now analyzes the *_t* dynamic field using the French language analyzer, which gives better search results for text submitted in French.

3. Restart Solr

4. Re-index all your search index data.

Note
After changing the default language it is important to remember to restart Solr and re-build all your search indexes.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2015 Sitecore. All rights reserved.

Chapter 5

Appendix

This chapter contains additional information that may be useful for extending or modifying your Sitecore CMS and item buckets solution.

This chapter contains the following sections:

- Tips and Tricks

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2015 Sitecore. All rights reserved.

5.1 Tips and Tricks

Standard Web.Config Tweaks

- Change the `web.config` setting of `Indexing.UpdateInterval` to 30 seconds or lower depending upon performance.
- 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

- Keep the number of facets to a minimum. If you have more than 100 facets, you might see a small degradation of performance.
- 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 do not do this, the *PublishingQueue*, *History,* and *EventQueue* tables become very large, slowing down processing, and your Sitecore installation may not start.

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

Content Author Tips

- You can use wildcards to search for IDs.

  For example, if you know the first 4 characters of the GUID of an item, enter, for example, *id:c728** and click search to find the corresponding item.

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.
- Do not run processes on the index to create a backup. This is an expensive operation and the index is likely to be out of date when the backup is complete.
- It is very important that you set up a SQL Maintenance task that rebuilds your indexes. When you create a lot of content, index fragmentation increases, especially with the bulk importation of content.

  The hotspots are the *Items*, *Versioned*, *Unversioned*, *Shared*, *Blobs,* and *Links* tables. To be on the safe side, you should set rebuilds for every table. If you do not do this, the CMS gets sluggish.

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

```
-- Show fragmentation for all tables
EXEC sp_MSforeachtable @command1="print '?' DBCC 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
```

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2015 Sitecore. All rights reserved.

```
WHILE @@FETCH_STATUS = 0

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

CLOSE TableCursor
DEALLOCATE TableCursor
```

Importing Data Tweaks

- If you import a lot of content, do it in batches of, for example, one thousand items, and then bucket or re-sync the bucket to avoid overloading the process with items.

Bucket Config Tweaks

- You can tweak your index so that you only include the things that you really want in the index. This decreases rebuild time and improves search time.
- Consider rebuilding your indexes on a computer that has a solid state disk (SSD). Incremental updates do not have to be performed on SSD but they benefit from this approach. If you have one dedicated server that rebuilds indexes and deploys them to an environment, ensure that this server has an SSD. Indexes do not take up much space, so it is fine to use a small SSD — for example 64GB.
- Do not 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 display facets, enable debug mode in the `Sitecore.ContentSearch.config` file. To enable debug mode change the following setting: `<setting name="ContentSearch.EnableSearchDebug" value="false" />`. In Debug mode all the queries are logged, 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 the search tips if you do not need them. To do this open the `Sitecore.Buckets.config` file and change the following setting: `<setting name="BucketConfiguration.EnableSearchTips" value="true"/>`.
- Disable all the dropdowns that you are not using in the `/sitecore/system/Settings/Buckets/Settings/Search Box Dropdown` item. The most expensive lookups are *recently modified* and *recently created*.
- Add all the items in `/sitecore/system/Settings/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.
- Ensure that the **Buckets** option is not selected.

Sitecore® is a registered trademark. All other brand and product names are the property of their respective holders. The contents of this document are the property of Sitecore. Copyright © 2001-2015 Sitecore. All rights reserved.