



**FREE eBook**

# LEARNING sitecore

Free unaffiliated eBook created from  
**Stack Overflow contributors.**

**#sitecore**

# Table of Contents

About.....	1
<b>Chapter 1: Getting started with sitecore.....</b>	<b>2</b>
Remarks.....	2
Versions.....	2
Examples.....	3
Installation or Setup.....	3
Using SIM manager for installation.....	4
<b>Chapter 2: Diagnostics: Asserts.....</b>	<b>5</b>
Syntax.....	5
Examples.....	5
Ensure two values are equal.....	5
Ensure a value is true or false.....	6
ResultNotNull.....	6
Required Object.....	6
Null/Empty Checks.....	6
IsNotNull.....	6
IsNotNullOrEmpty.....	6
IsNull.....	7
Argument Checks.....	7
ArgumentCondition.....	7
ArgumentNotNull.....	7
ArgumentNotNullOrEmpty.....	7
Ensure item is in editing mode.....	7
Security Asserts.....	8
CanRunApplication.....	8
HasAccess.....	8
<b>Chapter 3: Engagement Automation.....</b>	<b>9</b>
Remarks.....	9
Examples.....	9
Get the contact by username.....	9

Change the automation state of the contact.....	9
Enroll contact in the engagement plan programmatically.....	9
<b>Chapter 4: Glass Mapper.....</b>	<b>10</b>
Introduction.....	10
Remarks.....	10
Examples.....	10
The easiest way to map Sitecore data to code.....	10
<b>Chapter 5: Index configuration.....</b>	<b>11</b>
Examples.....	11
Field Configuration.....	11
StorageType.....	11
IndexType.....	11
VectorType.....	12
Boost.....	13
<b>Chapter 6: Items.....</b>	<b>14</b>
Syntax.....	14
Remarks.....	14
Examples.....	14
Get Item by ID.....	14
Get Specific Version of Item.....	14
Get Item by Path.....	14
Get Raw Value of Field on a Sitecore Item.....	14
Publish Sitecore item programmatically.....	15
<b>Chapter 7: LinkManager.....</b>	<b>16</b>
Syntax.....	16
Examples.....	16
Obtaining a url for an Item.....	16
<b>Chapter 8: Scaling.....</b>	<b>17</b>
Remarks.....	17
Examples.....	17
Implement remote events in load-balanced environment.....	17

<b>Chapter 9: Search</b> .....	<b>19</b>
Remarks.....	19
Examples.....	19
Configuration.....	19
Create an inbound filter for the search.....	21
Create an outbound filter for the search.....	21
Delete all previous versions of the item in the index when adding new version.....	22
<b>Chapter 10: Security</b> .....	<b>24</b>
Remarks.....	24
Examples.....	24
Disable permission checking when accessing an item.....	24
Impersonate a different user when accessing an item.....	24
<b>Chapter 11: Sitecore query syntax</b> .....	<b>25</b>
Remarks.....	25
Examples.....	27
Select by item path.....	27
Relative query.....	27
Item attributes query.....	27
Site-specific query.....	27
<b>Chapter 12: Templates</b> .....	<b>29</b>
Examples.....	29
Get Sitecore Template Item by ID.....	29
Get Sitecore Template Item by name.....	29
<b>Chapter 13: Unicorn</b> .....	<b>30</b>
Introduction.....	30
Remarks.....	30
Examples.....	30
Initial Setup.....	30
Manual Installation/Install from Source.....	30
Data Provider Architecture.....	31
<b>Chapter 14: Web Forms For Marketeers (WFFM)</b> .....	<b>32</b>
Introduction.....	32

Examples.....	32
Submit the WFFM form programmatically.....	32
<b>Chapter 15: Workflow.....</b>	<b>33</b>
Examples.....	33
Set Item State and Executes actions in that state.....	33
Executing workflow command to change the workflow state.....	33
<b>Chapter 16: Workflow.....</b>	<b>35</b>
Remarks.....	35
Examples.....	35
Assign workflow to an item.....	35
<b>Credits.....</b>	<b>36</b>

---

# About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: [sitecore](#)

It is an unofficial and free sitecore ebook created for educational purposes. All the content is extracted from [Stack Overflow Documentation](#), which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official sitecore.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to [info@zzzprojects.com](mailto:info@zzzprojects.com)

---

# Chapter 1: Getting started with sitecore

## Remarks

This section provides an overview of what sitecore is, and why a developer might want to use it.

It should also mention any large subjects within sitecore, and link out to the related topics. Since the Documentation for sitecore is new, you may need to create initial versions of those related topics.

## Versions

Version	Release Date
8.1	2015-01-01
8.0	2015-01-01
7.5	2014-01-01
7.2	2014-01-01
7.1	2013-01-01
7.0	2013-01-01
6.6	2012-01-01
6.5	2011-01-01
6.4	2010-01-01
6.3	2010-01-01
6.2	2009-01-01
6.1	2009-01-01
6.0	2008-01-01
5.3	2006-01-01
5.2	2006-01-01
5.1	2005-01-01
4.3	2004-01-01

# Examples

## Installation or Setup

### Executable File

Download the .exe from <http://dev.sitecore.net/> and double-click to start. This .exe will do everything for you - attach databases, modify host file, and set folder permissions. The only downside is that it leaves an entry in the registry under [HKEY\_LOCAL\_MACHINE\SOFTWARE\Sitecore CMS] and makes your instance available under 'Add/Remove Programs', which is misleading. Sitecore is just an ASP.NET application, not a Desktop application.

### Manual Install

You can choose to install and set up Sitecore manually by downloading the site root zip. You are responsible for attaching databases and ensuring that permissions are set up properly. You may wish to use this method in environments that require custom installation steps (databases are on another server, custom permissions needed). Follow the installation guide (available on <http://dev.sitecore.net/>; make sure you are reading the guide for your specific version) and - if you are installing on production - the Security Hardening Guide.

Once you become a certified Sitecore developer, you will be able to download the latest version of Sitecore and all associated modules from <http://dev.sitecore.net/>. On dev.sitecore.net, there are always two different formats available for download - the complete site root or an executable file. The following is a list of ways that you can install Sitecore.

### Step by Step

1. Download the Sitecore version you will from [the Sitecore website](#) - it's okay if you prefer the automatic installation method, but the following steps covers the manual process
2. Unzip the archives to a folder on your computer. You'll see the following subfolders:
  - **Data** - This is where Sitecore keeps a number of archives related to the currently installed instance, such as logs, indexes, packages, and cache files
  - **Databases** - Database files from SQL Server and Oracle
  - **Website** - Everything that will be accessible from the web browser.
3. Open IIS and create a new website linking to the folder "Website"
4. Set the Application Pool of your website to use the .NET Framework to version 4.5
5. Adjust your DataFolder path (at the web.config) - for Dev you can copy the Data folder to the Website folder
6. Copy the license.xml file to the Data folder
7. At the Website folder properties, uncheck the "read only" checkbox. In Dev you can and give full permissions to the Application Pool user. In production you must follow the instructions contained ([here](#)), under item "4.2 Configuring Folder and Registry Permissions".
8. Install all databases to the "Database" folder and edit the configurations in the ConnectionStrings.config file, located in the Website/App\_Config folder

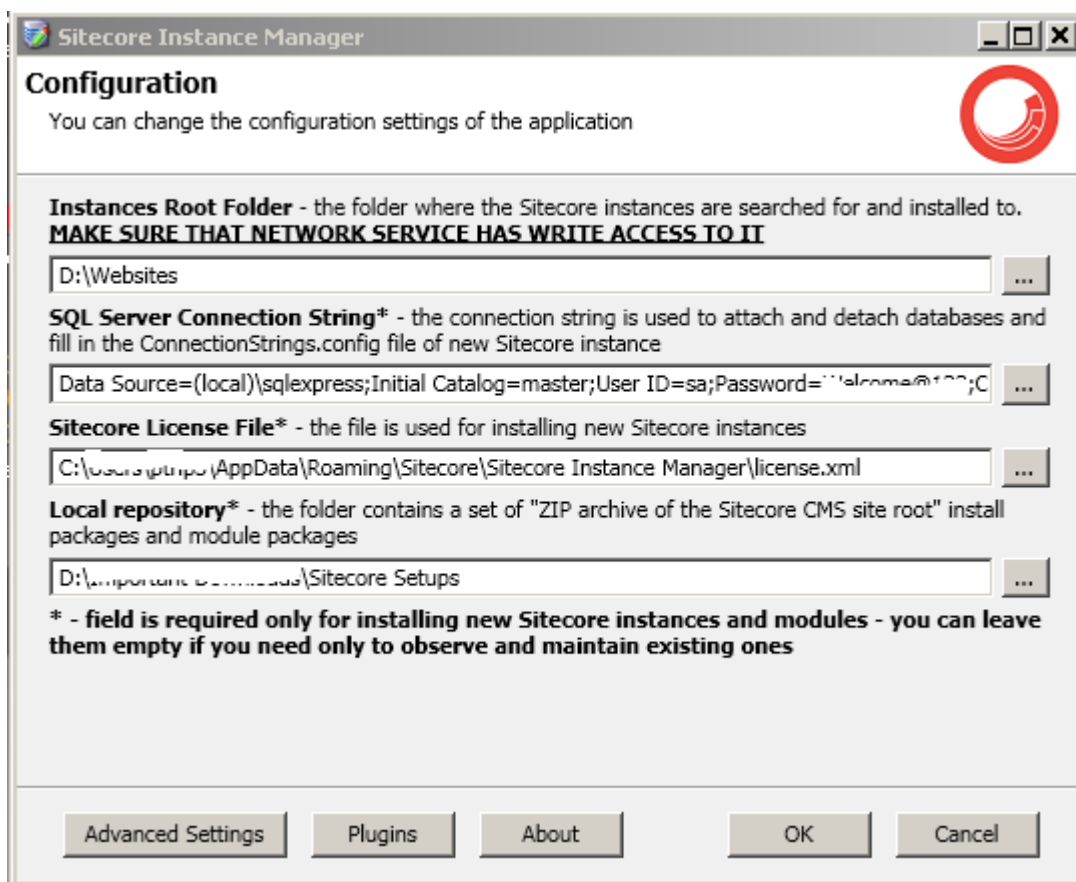


9. Still at the ConnectionStrings.config, setup connection informations for all MongoDB entries (Those with connectionString="mongodb:...")
10. If your site is on localhost, point your browser to <http://localhost/sitecore>. If you see a login screen, you have successfully installed Sitecore.

Source: <http://sitecore-community.github.io/docs/sitecore-basics/installing-sitecore/>

## Using SIM manager for installation

Sitecore Instance Manager is open-source tool which is used for managing the local park of Sitecore instances. You can install, locate, maintain, reinstal or delete Sitecore products. It also helps you install your sitecore instance with any sitecore packages, modules and only thing you need to do is set the folder for local repository where all the installations files are kept. See the below screenshot for settings.



It also has API and plugin engine so you can extend it for any of your need.

For Installation click [here](#)

For More details about SIM please click [here](#)

Read Getting started with sitecore online: <https://riptutorial.com/sitecore/topic/1007/getting-started-with-sitecore>

---

# Chapter 2: Diagnostics: Asserts

## Syntax

- `Assert.ArgumentCondition(bool condition, string argumentName, string message)`
- `Assert.ArgumentNotNull(object argument, string argumentName)`
- `Assert.ArgumentNotNull(object argument, Func<string> getArgumentName)`
- `Assert.ArgumentNotNullOrEmpty(ID argument, string argumentName)`
- `Assert.ArgumentNotNullOrEmpty(string argument, string argumentName)`
- `Assert.ArgumentNotNullOrEmpty(string argument, Func<string> getArgumentName)`
- `Assert.AreEqual(int value1, int value2, string message)`
- `Assert.AreEqual(int value1, int value2, string format, params object[] args)`
- `Assert.AreEqual(string value1, string value2, string message)`
- `Assert.AreEqual(string value1, string value2, string format, params object[] args)`
- `Assert.AreEqual(bool value1, bool value2, string message)`
- `Assert.AreEqual(bool value1, bool value2, string format, params object[] args)`
- `Assert.CanRunApplication(string application)`
- `Assert.HasAccess(bool accessAllowed, string message)`
- `Assert.HasAccess(bool accessAllowed, string format, params object[] args)`
- `Assert.IsEditing(Item item)`
- `Assert.IsFalse(bool condition, string message)`
- `Assert.IsFalse(bool condition, Func<string> getMessage)`
- `Assert.IsTrue(bool condition, string format, params object[] args)`
- `Assert.IsNotNull(object value, string message)`
- `Assert.IsNotNull(object value, string format, params object[] args)`
- `Assert.IsNotNull(object value, Type type)`
- `Assert.IsNotNull(object value, Type type, string format, params object[] args)`
- `Assert.IsNotNullOrEmpty(string value, string message)`
- `Assert.IsNotNullOrEmpty(string value, string format, params object[] args)`
- `Assert.IsNull(object value, string message)`
- `Assert.IsNull(object value, string format, params object[] args)`
- `Assert.IsTrue(bool condition, string message)`
- `Assert.IsTrue(bool condition, Func<string> getMessage)`
- `Assert.IsTrue(bool condition, string format, params object[] args)`
- `Assert.Required(object obj, string message)`
- `Assert.Required(object obj, string format, params object[] args)`
- `Assert.ResultNotNull<T>(T result, string message)`
- `Assert.ResultNotNull<T>(T result)`

## Examples

### Ensure two values are equal

Compares two values for equality. It can compare strings, integers, and Boolean values only.

```
Assert.AreEqual(documentElement.LocalName, "xamlControls", "Xaml files must have a root node named 'xamlControls'.");
```

## Ensure a value is true or false

To assert that a value is either true or false,:

```
Assert.IsFalse(Settings.DoBadThings, "Bad things should not happen, disable DoBadThings.");  
Assert.IsTrue(magicNumber =< 42, "The magic number is greater than 42!");
```

You can also pass formatting parameters for the exception message

```
Assert.IsFalse(myValue > 5, "The value should not be greater than 5, it's currently {0}",  
myValue);
```

## ResultNotNull

`ResultNotNull()` checks to see if the object passed in is not null. If the object and message are not null it will then simply return the object that was passed in, otherwise it will throw

`InvalidOperationException`.

```
return Assert.ResultNotNull(this.Database.GetItem(this.ItemRootId), string.Concat("Root item  
not found. ID: ", this.ItemRootId));
```

## Required Object

This checks to see if the given object is null, and then throws `RequiredObjectIsNullException` if it is.

```
Assert.Required(parameter, "parameter is required.");
```

## Null/Empty Checks

### IsNotNull

This is a very simple and popular method to use to check if an item is not null. It simply checks the object that is passed in to see if it is null.

```
Assert.IsNotNull(database, type, "Name: {0}", item);
```

### IsNotNullOrEmpty

This is the same as `IsNotNull` above, but works on string values instead of objects.

```
Assert.IsNotNullOrEmpty(propertyName, "user");
```

# IsNull

This is simply an inverse of the `IsNotNull()` method. This method asserts that the object is null.

```
Assert.IsNull(this.StandardValues, "A Standard Values item has already been created for this template ");
```

## Argument Checks

### ArgumentCondition

This method checks to see if the argument specified is true. It also takes in the name of the argument that is logged if the condition fails.

```
Assert.ArgumentCondition(pageIndex >= 0, "pageIndex", "Value must be greater than or equal to zero.");
```

### ArgumentNotNull

This method ensures that the argument passed is not null. There are two signatures for this method, the first takes in an object and a parameter name and does a simple null check.

```
Assert.ArgumentNotNull(item, "item");
```

### ArgumentNotNullOrEmpty

This is similar to the `ArgumentNotNull` method, but will also check to see if the object is empty. There are three variants of this method. The first variant takes in a Sitecore ID and an argument name, and checks to see if the ID is null.

```
var nullId = new new ID("{00000000-0000-0000-0000-000000000000}");  
  
// Both of these calls will result in an exception  
Assert.ArgumentNotNullOrEmpty((ID)null, "null");  
Assert.ArgumentNotNullOrEmpty(nullId, nameof(nullId));
```

The second method adds a check to see if the given string is null or empty.

```
// Both of these calls will result in an exception  
Assert.ArgumentNotNullOrEmpty((string)null, "null");  
Assert.ArgumentNotNullOrEmpty("", nameof(emptyString));
```

## Ensure item is in editing mode

Checks to see if the passed `Item` is in Editing mode. If not, it throws an `EditingNotAllowedException`.

```
Assert.IsEditing(Sitecore.Context.Item);
```

## Security Asserts

### CanRunApplication

To check to see if the user has permission to run the given application. If not, `AccessDeniedException` is thrown.

```
Assert.CanRunApplication("WebEdit");
```

### HasAccess

`HasAccess` will check if the given parameter is true, otherwise an `AccessDeniedException` will be thrown.

```
Assert.HasAccess(Context.User.IsAdministrator, "Only administrators can create new domains");
```

Read Diagnostics: Asserts online: <https://riptutorial.com/sitecore/topic/1297/diagnostics--asserts>

---

# Chapter 3: Engagement Automation

## Remarks

Sitecore automation gives marketer a possibility to create marketing workflows which will put user through different states on the website.

The example of automation usage could be registration workflow (registered, confirmed, logged in) or purchase workflow (new order, added products, payment details, purchase complete).

## Examples

### Get the contact by username

```
ContactManager contactManager = Factory.CreateObject("tracking/contactManager", true) as ContactManager;

Contact contact = contactManager.LoadContactReadOnly(userName);
return contact;
```

### Change the automation state of the contact

This method doesn't require initialization of the tracker, which is handy if the state should be changed outside of the site context (for example in the shell).

```
var stateManager = AutomationStateManager.Create(contact);
automationStateManager.MoveToEngagementState(stateItem.ParentID, stateId);
stateManager.SaveChanges(AutomationManager.Provider);
```

### Enroll contact in the engagement plan programmatically

```
AutomationMetadataProvider automationMetadataProvider =
Assert.ResultNotNull(Factory.CreateObject("automation/metadataProvider", true) as
AutomationMetadataProvider);
var context = AutomationManager.Provider.GetAutomationContext(ID.Parse(contact.ContactId));

context.Enroll(fromStateItem.ParentID, ID.Parse(fromState),
automationMetadataProvider.CalculateWakeUpDateTime(ID.Parse(fromState), DateTime.UtcNow),
null);
AutomationManager.Provider.SaveAutomationContext(context);
```

Read Engagement Automation online: <https://riptutorial.com/sitecore/topic/6879/engagement-automation>

---

# Chapter 4: Glass Mapper

## Introduction

Glass.Mapper is the awesome Open Source mapping framework that allows you to focus on solving your business problems. It does the hard work of converting data from your CMS to something your code can work with.

Using Glass.Mapper and your favourite CMS you can map data to strongly typed models in your C# code. Your models do not require any special mark-up to work with Glass.Mapper and almost no configuration to get started.

## Remarks

Visit this url for tutorials <http://glass.lu/Mapper/Sc/Tutorials>

## Examples

### The easiest way to map Sitecore data to code.

Glass.Mapper.Sc allows you to move your data from Sitecore and into your code seamlessly using strongly typed objects.

The framework allows you to map data on to c# classes and interfaces without any additional mark-up. As the data is mapped to your target objects it is converted to the target type. Take a look at this simple example:

```
public class Demo
{
    public virtual Guid Id { get; set; }

    public virtual string Title { get; set; }

    public virtual DateTime Date { get; set; }

    public virtual string Url { get; set; }
}

public void DoWork(
    ISitecoreContext sitecoreContext)
{
    var model =
        sitecoreContext.GetCurrentItem<Demo>();

    var url = model.Url;
}
```

Read Glass Mapper online: <https://riptutorial.com/sitecore/topic/8845/glass-mapper>

# Chapter 5: Index configuration

## Examples

### Field Configuration

When adding custom fields to a Lucene index you can add new fields into the index using the following configuration:

```
<configuration ref="contentSearch/indexConfigurations/defaultLuceneIndexConfiguration">
  <indexAllfields>false</indexAllfields>
  <fieldNames hint="raw:AddFieldByFieldName">
    <field fieldName="title" storageType="YES" indexType="TOKENIZED" vectorType="NO"
    boost="1f" type="System.String"

settingType="Sitecore.ContentSearch.LuceneProvider.LuceneSearchFieldConfiguration,
Sitecore.ContentSearch.LuceneProvider"/>
  </fieldNames>
</configuration>
```

A field has a couple of possible properties:

- storageType
- indexType
- vectorType
- boost

These fields directly relate to the content of the

`Sitecore.ContentSearch.LuceneProvider.LuceneSearchFieldConfiguration` class. If we reflect the values out of this class we can see their possible values, etc.

## StorageType

```
/// <summary>Specifies whether and how a field should be stored. </summary>
public enum Store
{
    /// <summary>Store the original field value in the index. This is useful for short texts
    /// like a document's title which should be displayed with the results. The
    /// value is stored in its original form, i.e. no analyzer is used before it is
    /// stored.
    /// </summary>
    YES,
    /// <summary>Do not store the field value in the index. </summary>
    NO
}
```

## IndexType

```
/// <summary>Specifies whether and how a field should be indexed. </summary>
```



```

public enum Index
{
    /// <summary>Do not index the field value. This field can thus not be searched,
    /// but one can still access its contents provided it is
    /// <see cref="T:Lucene.Net.Documents.Field.Store">stored</see>.
    /// </summary>
    NO,
    /// <summary>Index the tokens produced by running the field's
    /// value through an Analyzer. This is useful for
    /// common text.
    /// </summary>
    ANALYZED,
    /// <summary>Index the field's value without using an Analyzer, so it can be searched.
    /// As no analyzer is used the value will be stored as a single term. This is
    /// useful for unique Ids like product numbers.
    /// </summary>
    NOT_ANALYZED,
    /// <summary>Expert: Index the field's value without an Analyzer,
    /// and also disable the storing of norms. Note that you
    /// can also separately enable/disable norms by calling
    /// <see cref="!:AbstractField.SetOmitNorms" />. No norms means that
    /// index-time field and document boosting and field
    /// length normalization are disabled. The benefit is
    /// less memory usage as norms take up one byte of RAM
    /// per indexed field for every document in the index,
    /// during searching. Note that once you index a given
    /// field <i>with</i> norms enabled, disabling norms will
    /// have no effect. In other words, for this to have the
    /// above described effect on a field, all instances of
    /// that field must be indexed with NOT_ANALYZED_NO_NORMS
    /// from the beginning.
    /// </summary>
    NOT_ANALYZED_NO_NORMS,
    /// <summary>Expert: Index the tokens produced by running the
    /// field's value through an Analyzer, and also
    /// separately disable the storing of norms. See
    /// <see cref="F:Lucene.Net.Documents.Field.Index.NOT_ANALYZED_NO_NORMS" /> for what norms
are
    /// and why you may want to disable them.
    /// </summary>
    ANALYZED_NO_NORMS
}

```

## VectorType

```

/// <summary>Specifies whether and how a field should have term vectors. </summary>
public enum TermVector
{
    /// <summary>Do not store term vectors. </summary>
    NO,
    /// <summary>Store the term vectors of each document. A term vector is a list
    /// of the document's terms and their number of occurrences in that document.
    /// </summary>
    YES,
    /// <summary> Store the term vector + token position information
    ///
    /// </summary>
    /// <seealso cref="F:Lucene.Net.Documents.Field.TermVector.YES">
    /// </seealso>
}

```

```
WITH_POSITIONS,  
/// <summary> Store the term vector + Token offset information  
///  
/// </summary>  
/// <seealso cref="F:Lucene.Net.Documents.Field.TermVector.YES">  
/// </seealso>  
WITH_OFFSETS,  
/// <summary> Store the term vector + Token position and offset information  
///  
/// </summary>  
/// <seealso cref="F:Lucene.Net.Documents.Field.TermVector.YES">  
/// </seealso>  
/// <seealso cref="F:Lucene.Net.Documents.Field.TermVector.WITH_POSITIONS">  
/// </seealso>  
/// <seealso cref="F:Lucene.Net.Documents.Field.TermVector.WITH_OFFSETS">  
/// </seealso>  
WITH_POSITIONS_OFFSETS  
}
```

## Boost

Add's a [boost value](#) to the Lucene index for this item

Read Index configuration online: <https://riptutorial.com/sitecore/topic/7634/index-configuration>

---

# Chapter 6: Items

## Syntax

- Database.GetItem(ID itemId)
- Database.GetItem(ID itemId, Language language)
- Database.GetItem(ID itemId, Language language, Version version)
- Database.GetItem(string path)
- Database.GetItem(string path, Language language)
- Database.GetItem(string path, Language language, Version version)

## Remarks

Most of the above examples make use of `Sitecore.Context.Database` to fetch items. Keep in mind that most Sitecore implementations have multiple content databases, so using the correct database to fetch your item is important.

## Examples

### Get Item by ID

To fetch the latest version of an item in the current language:

```
Sitecore.Context.Database.GetItem(new ID("{11111111-1111-1111-1111-111111111111}"));
```

### Get Specific Version of Item

If you need to get a specific language or version of an item, you can use these overloads of `GetItem()`

```
Sitecore.Context.Database.GetItem("/sitecore/content/Sitecore", Language.Current, new Version(5));
```

### Get Item by Path

To fetch the latest version of an item in the current language:

```
Sitecore.Context.Database.GetItem("/sitecore/content/Sitecore")
```

### Get Raw Value of Field on a Sitecore Item

To get the raw value of a field on the Context Item:

```
Sitecore.Context.Item["Field Name"];
```

To get the raw value of a field on a given item, `item`:

```
item["Field Name"];
```

## Publish Sitecore item programmatically

When publishing Sitecore item programmatically the developer should keep in mind that Sitecore could be configured for several publishing targets, as well that several languages could be defined for item.

```
ID targetDatabaseFieldId = ID.Parse("{39ECFD90-55D2-49D8-B513-99D15573DE41}");

var publishingDatabases =
    PublishManager.GetPublishingTargets(mediaItem.Database)
        .Select(i => i[targetDatabaseFieldId]) //Get Target Database
value
        .Where(i => i != null)
        .Select(i => Database.GetDatabase(i))
        .ToArray();

PublishManager.PublishItem(mediaItem, publishingDatabases,
    LanguageManager.GetLanguages(StaticSettings.WebDatabase).ToArray(), false, false);
```

Read Items online: <https://riptutorial.com/sitecore/topic/2035/items>

---

# Chapter 7: LinkManager

## Syntax

- `public static string GetItemUrl(Item item)`
- `public static string GetItemUrl(Item item, UrlOptions options);`

## Examples

### Obtaining a url for an Item

Given a simple sitecore item:

```
Item item;
```

The item itself does not contain its url. To obtain a url for an item you need to make a call to the static `class Sitecore.Links.LinkManager`

```
string url = LinkManager.GetItemUrl(item);
```

an overload of this accepts a `UrlOptions` class:

```
UrlOptions options = new UrlOptions  
{  
    AddAspxExtension = false  
    ....  
};  
string url = LinkManager.GetItemUrl(item, options);
```

Read `LinkManager` online: <https://riptutorial.com/sitecore/topic/7664/linkmanager>

---

# Chapter 8: Scaling

## Remarks

Sitecore out of the box supports load-balancing for multiple server. Typical configuration is Content Management (CM) and Content Delivery (CD) server, however multiple CM and CD servers are supported too.

## Examples

### Implement remote events in load-balanced environment

If Sitecore is setup in a CM-CD environment there could be a need to fire events on CD server when CM events are fired.

The example could be firing `publish:end:remote` on CD when content editors done publish on CM.

In order to make sure that events are firing the following steps are required to be done:

1. Check that event queues are enabled in `web.config`

```
<!-- ENABLE EVENT QUEUES
    If enabled, Sitecore sends local events to the event queue available to remote
instances,
    and handles events in the queue from remote instances.
    Default value: true
-->
<setting name="EnableEventQueues" value="true" />
```

2. Add `ScalabilitySettings.config` to each instance. Set `InstanceName` for each server and `PublishingInstance` to the CM server instance name.

The example of `ScalabilitySettings.config` could be found in the `App_Config/Include` folder.

```
<!-- INSTANCE NAME
    Unique name for Sitecore instance.
    Default value: (machine name and IIS site name)
-->
<setting name="InstanceName">
  <patch:attribute name="value">BAYERUATCD</patch:attribute>
</setting>
<!-- PUBLISHING INSTANCE
    Assigns the instance name of dedicated Sitecore installation for publishing
operations.
    When empty, all publishing operations are performed on the local installation of
Sitecore.
    Default vaue: (empty)
-->
<setting name="Publishing.PublishingInstance">
  <patch:attribute name="value">BAYERUATCM</patch:attribute>
```

```
</setting>
<!-- COUNTERS INSTANCE NAME
      Instance name for performance counters.
      Default value: (value of InstanceName setting)
-->
<setting name="Counters.InstanceName">
  <patch:attribute name="value">BAYERUATCD</patch:attribute>
</setting>
<!-- SECURITY CACHE EXPIRATION
      Sets the absolute expiration on the cached security data.
      A value of 00:00:00 disables automatic expiration of security caches.
-->
```

Read Scaling online: <https://riptutorial.com/sitecore/topic/5043/scaling>

---

# Chapter 9: Search

## Remarks

Sitecore search is built on top of Lucene, providing the ability to create very fast searching capabilities for your site. Instead of querying against a centralised database (such as Sitecore's SQL DB), it queries Lucene index files which are stored on the physical file system of the web-server. Sitecore provides a layer of abstraction over the Lucene.NET API including a LINQ provider which makes writing Lucene queries a simple & familiar process for .NET developers. Sitecore ships with some standard indexes configured which you can extend or define your own. You can also opt to use SOLR; a centralised & scalable platform built on top of Lucene.

## Examples

### Configuration

Sitecore ships with a set of standard indexes pre-configured which you can extend, or you can define your own. Of the pre-configured, `sitecore_master_index` & `sitecore_web_index` are of most interest for your site search. These are the predefined indexes for all of your Sitecore items in the tree of your master & web databases respectively, and are configured to store all of the standard fields of a Sitecore item that will be common among all templates.

You can look at this configuration of the standard web index at this location: `<Your Site>\App_Config\Include\Sitecore.ContentSearch.Lucene.Index.Web.config`

The main areas of importance of an index's configuration are:

- *The Field Configuration* - What fields should be stored in the index, and how they should be stored.
- *The Strategy* - How and when should the index be updated.
- *The Crawler* - The location where the index can get it's Sitecore data

### Field Configuration

Looking in the `sitecore_web_index` config, you can see the following reference: `<configuration ref="contentSearch/indexConfigurations/defaultLuceneIndexConfiguration" />`. This refers to a shared index configuration file found here: `<Your Site>\App_Config\Include\Sitecore.ContentSearch.Lucene.DefaultIndexConfiguration.config`. Here you can see all of the fields that are included in the standard config.

There are basically two ways to defined a field: either the field is sourced directly from a Sitecore item field, or it's a computed field. A computed field allows you write some code to do some calculations and store the result in the field. This code will get executed when the index is built/updated *not* when the index is queried. This is particularly useful if the field needs to store aggregated data, such as counts etc.



Within the `<fieldMap>` element you'll see the elements `<fieldNames hint="raw:AddFieldByFieldName">` & `<fields hint="raw:AddComputedIndexField">`, which contain the directly sourced fields and the computed fields respectively.

## Strategy

The strategy of your index determines when your index is updated. There are the following options to choose from:

- **OnPublishEndAsynchronousStrategy (onPublishEndAsync)** - When an item is published, the index will be updated asynchronously.
- **SynchronousStrategy (syncMaster)** - When an item is saved, the index will be updated instantly and synchronously.
- **IntervalAsynchronousStrategy (intervalAsyncCore/intervalAsyncMaster)** - Periodically check for item updates & update the index asynchronously
- **ManualStrategy** - No automatic index updates. Indexes will only be updated manually (through the control panel, or programmatically)
- **RebuildAfterFullPublishStrategy (rebuildAfterFullPublish)** - After a publish, the index will be rebuilt in full
- **RemoteRebuildStrategy (remoteRebuild)** - This strategy is for multiple instances of Sitecore. For example, if a rebuild is called for from the content management server, then the remote content delivery servers will subscribed to this event and rebuild their own indexes.

By default, the master index is configured as `syncMaster`. This is because if you're in the experience editor saving items and rendering on the page displays the results of an index, you'll want to see changes you've made to items immediately in the results. The web index is configured as 'onPublishEndAsync', this is because the indexes of your web database only need updating when items are published from the master database to the web.

You can also combine multiple strategies. For example, if you have separate Sitecore instances for your content management (CM) & content delivery (CD), it would make sense to combine the `onPublishEndAsync` with `remoteRebuild`, so that CD indexes get updated when items are published as well as get rebuilt when a user triggers a rebuild from the control panel of the CM server.

You can choose your strategy using the following config:

```
<strategies hint="list:AddStrategy">
  <strategy ref="contentSearch/indexConfigurations/indexUpdateStrategies/onPublishEndAsync"
/>
</strategies>
```

## The Crawler

This allows you to specify the location of the Sitecore data you want to index. The web index has the following default configuration:

```
<locations hint="list:AddCrawler">
  <crawler type="Sitecore.ContentSearch.SitecoreItemCrawler, Sitecore.ContentSearch">
```

```
<Database>web</Database>
<Root>/sitecore</Root>
</crawler>
</locations>
```

The two important bits are the `<Database>` and `<Root>` elements. The `<Root>` element allows you to specify the starting position in your Sitecore tree that the index should index. In reality, you're likely to have a 'Home' node under the content node that you'd point this at so that it only indexes actual content/pages rather than your templates etc.

## Create an inbound filter for the search

Inbound filter is applied when item is added to the search index and allows to specify whether item is included to the index or not.

Examples when inbound filter can be used - do not include standard values and previous versions of the item in the index.

Inbound filters are set in configuration:

```
<indexing.filterIndex.inbound>
  <processor type="Sitecore.ContentSearch.Pipelines.IndexingFilters.ApplyInboundIndexFilter,
Sitecore.ContentSearch"></processor>
</indexing.filterIndex.inbound>
```

Code implementation:

```
public class ApplyInboundIndexVersionFilter : InboundIndexFilterProcessor
{
    public override void Process(InboundIndexFilterArgs args)
    {
        var item = args.IndexableToIndex as SitecoreIndexableItem;

        if (!item.Item.Versions.IsLatestVersion())
        {
            args.IsExcluded = true;
        }
    }
}
```

More examples and information could be found on <http://www.sitecore.net/learn/blogs/technical-blogs/sitecore-7-development-team/posts/2013/04/sitecore-7-inbound-and-outbound-filter-pipelines.aspx>

## Create an outbound filter for the search

Outbound filter can be used to filter the search results.

One of the examples of outbound filter usage is removing items which user doesn't have access to from search results.

Outbound filters are set in the configuration:

```
<indexing.filterIndex.outbound>
  <processor
type="Sitecore.ContentSearch.Pipelines.IndexingFilters.ApplyOutboundSecurityFilter,
Sitecore.ContentSearch"></processor>
</indexing.filterIndex.outbound>
```

Example of outbound filter implementation:

```
public class ApplyOutboundIndexWorkflowFilter : OutboundIndexFilterProcessor
{
    public override void Process(OutboundIndexFilterArgs args)
    {
        //You can use args.IsExcluded to remove items from the search results here
    }
}
```

More examples and information could be found on <http://www.sitecore.net/learn/blogs/technical-blogs/sitecore-7-development-team/posts/2013/04/sitecore-7-inbound-and-outbound-filter-pipelines.aspx>

## Delete all previous versions of the item in the index when adding new version

By default Sitecore adds all versions of the item to the sitecore\_master\_index. The drawback of that is that if users are using workflows and adding lots of versions all of them will be added to the search results in the content editor.

Configuration:

```
<event name="item:versionAdded" >
    <handler type="FilterPatch.Library.ContentSearch.EventHandler, AssemblyName"
method="Execute" />
</event>
```

## Handler implementation

```
public class EventHandler
{
    public void Execute(object sender, EventArgs eventArgs)
    {
        var item = Event.ExtractParameter(eventArgs, 0) as Item;

        //If item has less than 2 versions - then skip
        if(item.Versions.Count < 2)
        {
            return;
        }

        var indexableItem = new SitecoreIndexableItem(item);

        var index = ContentSearchManager.GetIndex(indexableItem);

        using (var context = index.CreateDeleteContext())
        {
            foreach(var version in item.Versions.GetVersions(true))
```

```
        {
            if(!version.Versions.IsLatestVersion())
            {
                var indexableItemVersion = new SitecoreIndexableItem(version);
                context.Delete(indexableItemVersion.UniqueId);
            }
        }
        context.Commit();
    }
}
```

Read Search online: <https://riptutorial.com/sitecore/topic/2440/search>

---

# Chapter 10: Security

## Remarks

Sitecore offers two ways to access items that the context user doesn't have permissions to access. The preferred way is to use the `UserSwitcher` class to temporarily change the user that will be used to access the item. The reason that this is preferred is because you can still have permissions in effect for the user account that is being utilised.

The alternative is to use the `SecurityDisabler` class. This performs the action without any security constraints.

It is recommended to only use these classes for the operations that require elevated permissions. The best way to ensure this is by utilising the 'using' keyword in C#; this will ensure that the `UserSwitcher` / `SecurityDisabler` is correctly disposed.

## Examples

### Disable permission checking when accessing an item

```
using (new Sitecore.SecurityModel.SecurityDisabler())
{
    var item = Sitecore.Context.Database.GetItem("/sitecore/content/home");
}
```

### Impersonate a different user when accessing an item

```
var user = Sitecore.Security.Accounts.User.FromName("sitecore/testname", false);

using (new Sitecore.Security.Accounts.UserSwitcher(user))
{
    var item = Sitecore.Context.Database.GetItem("/sitecore/content/home");
}
```

Read Security online: <https://riptutorial.com/sitecore/topic/2586/security>

---

# Chapter 11: Sitecore query syntax

## Remarks

### Template Names vs. Template IDs vs. Item Names in Queries:

I strongly recommend that you *use Template IDs and **not** Template Names or Item Names* in your queries. This will ensure that your queries will still work, even when templates and/or items are renamed.

The one exception to this is when working with OOTB templates, while querying an OOTB structure, e.g. `/sitecore/content` or `/sitecore/system/Marketing Control Panel`. In these situations, the loss of readability is often greater than the risk of queries breaking, since these templates are far less likely to be renamed.

Note that template names were used in my examples, above, for sake of readability. Those queries should not be used in production, unless the template names are replaced with template IDs.

### Reference Sheet:

I noticed that the Sitecore Query Cheat Sheet is no longer available for download on the web (all of the Sitecore-hosted links now redirect to 404 pages). Fortunately, I had a copy on my machine, and have added a screenshot, below:

Item Attributes		Common Standard Template Fields	
@@name	Case sensitive	@__Display name	Display name
@@key	ToLower(@@name)	@__Updated	Update date/time
@@templateid	Template ID	@__Updated by	Case sensitive username
@@templatename	Case sensitive	@__Created	Creation date/time
@@templatekey	ToLower(@@templatename)	@__Created by	Case sensitive username
@@id	Item ID	@__Lock	Lock owner
@@masterid	Branch template used	@__Workflow state	Workflow state ID
<b>Supported Functions</b>		@__Publish	Item publish date
contains('does this string', 'this string')		@__Unpublish	Item unpub. Date
startswith('does this string', 'this string')		@__Valid to	Version publish date
endswith('does this string', 'this string')		@__Valid from	Version unpublish date
not(condition)		@__Never publish	Prevent item publication
position()		@__Renderings	Layout details
<b>Axes</b>			
ancestor	Ancestors of item		
ancestor-or-self	Item and its ancestors		
child (/*)	Children of item		
descendant (//*)	Descendants of item		
descendant-or-self	Item and its descendants		
following	Siblings sorted after item		
parent	Parent of item		
preceding	Siblings sorted before item		
self (.)	Item		
[index]	Position index: *[5] gets the fifth child		
<b>Operators</b>			
and	And	/*/content/**[@@templateid='<ID>' and key='home']	
or	Or	../*[@@templateid='<ID>' or key='home']	
xor	Exclusive or	Equivalent to and not()	
	Catenation	/*/content/** sitecore /media library/**	
+	Addition		
-	Subtraction		
*	Multiplication		
div	Division		
mod	Modulus (remainder)	/*/content/**[@IntField mod 2 = 1]	
=	Equal comparison		
!=	Not equal comparison		
<	Less than		
<=	Less than or equal to		
>	Greater than		
>=	Greater than or equal to		
false	Evaluates to false		
true	Evaluates to true	Equivalent to not(false)	
<b>Prototypes</b>			
Item selected in field	/*/content/**[contains(@ChecklistMultilistTreelistOrTreelistExField, '<ItemID>']		
Escape Item/Field Names	/*/content/##item-name#[@#field-name#='value']		
Date Comparison	/*/content/**[@__Updated > 'yyyymmddTHHmss']		
<b>APIs</b>			
Sitecore.Data.Database	SelectItems()		
Sitecore.Data.Items.Item.Axes	SelectSingleItem()		
Sitecore.Data.Query.Query	Execute()		
<b>Legend</b>			
Red text	Escape with hash characters (#) when value appears in item name or field name		

# Examples

## Select by item path

### Query:

```
query:/sitecore/content/home/foo/bar
```

### Result

```
bar
```

## Relative query

### Current Item:

```
bar (path: /sitecore/content/home/foo/bar)
```

### Query:

```
query:./child/grandchild
```

### Result:

```
grandchild (path: /sitecore/content/home/foo/bar/child/grandchild)
```

## Item attributes query

### Query:

```
query:/sitecore/content/[@@templatename='Homepage']
```

### Result:

```
home (name: home, path: /sitecore/content/home, template name: Homepage)
```

## Site-specific query

### Tree structure:

```
/sitecore
  /content
    /foo-site
      /home
        /my-account
    /bar-site
```



```
    /home
      /my-account
/baz-site
  /home
    /my-account
```

- The template of each site item (`foo-site`, `bar-site`, `baz-site`) is named `Site Node`.
- The template of each home item (`home`, `home`, `home`) is named `Homepage`
- The template of each user account item (`my-account`, `my-account`, `my-account`) is named `User Account Page`

### Current Item:

The current item could be the `home` item or any page underneath the `home` item for any of the given sites, and this query will still work (provided that there are no items with the `Homepage` template underneath the `home` items that are an ancestor of the current item).

### Query:

```
query:./ancestor-or-self::*[@@templatename='Homepage']/*[@@templatename='my-account']
```

### Result:

If querying from the `home` item or one of its descendants in the `foo-site` site:

```
/sitecore/content/foo-site/home/my-account
```

If querying from the `home` item or one of its descendants in the `bar-site` site:

```
/sitecore/content/bar-site/home/my-account
```

If querying from the `home` item or one of its descendants in the `baz-site` site:

```
/sitecore/content/baz-site/home/my-account
```

Read Sitecore query syntax online: <https://riptutorial.com/sitecore/topic/7212/sitecore-query-syntax>

---

# Chapter 12: Templates

## Examples

### Get Sitecore Template Item by ID

To get template item by template Id.

```
TemplateItem templateItem = Sitecore.Context.Database.GetTemplate(new ID("{11111111-1111-1111-1111-111111111111}"));
```

### Get Sitecore Template Item by name

Input the name of the template to get the template by using GetTemplate Method.

```
TemplateItem templateItem = Sitecore.Context.Database.GetTemplate("NameOfTheTemplate");
```

Read Templates online: <https://riptutorial.com/sitecore/topic/2448/templates>

---

# Chapter 13: Unicorn

## Introduction

Unicorn is a utility for Sitecore that solves the issue of moving templates, renderings, and other database items between Sitecore instances. This becomes problematic when developers have their own local instances - packages are error-prone and tend to be forgotten on the way to production. Unicorn solves this issue by writing serialized copies of Sitecore items to disk along with the code - this way, a copy of the necessary database items for a given codebase accompanies it in source control.

## Remarks

Useful Information can be found here:

- [Unicorn on Github](#)
- [Founder of Unicorn - Kam Figy's Blog](#)
- [Sitecore: Getting Started with Unicorn!](#)

## Examples

### Initial Setup

- You'll need [Sitecore 6.6](#) or later (including [Sitecore 8.x](#)). Note that for Sitecore 6.6 compatibility you must have .NET 4.5 installed.
- Install Unicorn. This is as simple as adding the [Unicorn NuGet package](#) to your project.

PM> Install-Package Unicorn

- When you install the NuGet package, a [README file](#) will come up in Visual Studio with help to get you started.

### Manual Installation/Install from Source

- Clone the repository
- Place a copy of your Sitecore.Kernel.dll assembly in /lib/sitecore/v7 (for v7/v8)
- Build the project for your Sitecore version using Visual Studio 2012 or later
- Copy Unicorn.dll, Rainbow.dll, Rainbow.Storage.Sc.dll, Rainbow.Storage.Yaml.dll and Kamsar.WebConsole.dll to your main project in whatever fashion you wish (project reference, as binary references, etc)
- Copy Standard Config Files\*.config to the App\_Config\Include\Unicorn folder
- Configure to your liking; the [setup README file](#) is a good starting point.
- Hit \$yoursite/unicorn.aspx to perform initial serialization of your configured predicate

## Data Provider Architecture

There are two components to the Unicorn data provider: the database-specific implementation, and the Unicorn implementation.

The Unicorn implementation is an individual configuration of Unicorn dependencies that get automatic serialization. For example, if you were serializing two presets you'd need two instances of `UnicornDataProvider` - one for each `IPredicate` implementation.

The database specific implementation is a subclass of the original Sitecore data provider that provides a container for one or more `UnicornDataProvider` instances. Out of the box, a `UnicornSqlServerDataProvider` is provided. You could roll your own if you're on Oracle. This provider is effectively an unblockable event handler that allows Unicorn to trap item changes even if the evil `EventDisabler` class is being used.

If you want to wire multiple Unicorn data providers to your database, you create a class that derives from `UnicornSqlServerDataProvider`. In this class you can select to:

- Create a constructor that injects your provider(s) using the base constructor:

```
public MyDataProvider(string connectionString) :
base(connectionString, new UnicornDataProvider(), new
UnicornDataProvider(), ...)
```

- Create a constructor that injects your provider(s) using code (this is better if you have to construct dependencies, etc that don't fit well in a base call):

```
public MyDataProvider(string connectionString) : base(connectionString, null)
{
    AddUnicornDataProvider(new UnicornDataProvider());
    // ...
}
```

Read Unicorn online: <https://riptutorial.com/sitecore/topic/8852/unicorn>

---

# Chapter 14: Web Forms For Marketeers (WFFM)

## Introduction

Web Forms For Marketeers is the popular Sitecore module that allows to create forms in Sitecore and customize and extend their behavior using save actions.

More information about the module can be found in [https://dev.sitecore.net/Downloads/Web\\_Forms\\_For\\_Marketers.aspx](https://dev.sitecore.net/Downloads/Web_Forms_For_Marketers.aspx)

## Examples

### Submit the WFFM form programmatically

This example shows how to submit the WFFM form in code.

```
var controlResults = new List<ControlResult>();
controlResults.Add(new ControlResult(Pdf_Request_Form.Name.ItemID.ToString(), "Name", name,
string.Empty));
controlResults.Add(new ControlResult(Pdf_Request_Form.Email.ItemID.ToString(), "Email", email,
string.Empty));
FormDataHandler.ProcessData(Pdf_Request_Form.ItemID, controlResults.ToArray(), new
IActionDefinition[] {}, DependenciesManager.ActionExecutor);
```

Read [Web Forms For Marketeers \(WFFM\)](https://riptutorial.com/sitecore/topic/9758/web-forms-for-marketeers--wffm-) online: <https://riptutorial.com/sitecore/topic/9758/web-forms-for-marketeers--wffm->

# Chapter 15: Workflow

## Examples

### Set Item State and Executes actions in that state

```
public void MoveToStateAndExecuteActions(Item item, ID workflowStateId)
{
    Sitecore.Workflows.IWorkflowProvider workflowProvider = Item.Database.WorkflowProvider;
    Sitecore.Workflows.IWorkflow workflow = workflowProvider.GetWorkflow(item);

    // if item is in any workflow
    if (workflow != null)
    {
        using (new Sitecore.Data.Items.EditContext(item))
        {
            // update item's state to the new one
            item[Sitecore.FieldIDs.WorkflowState] = workflowStateId.ToString();
        }

        Item stateItem = ItemManager.GetItem(workflowStateId,
            Language.Current, Sitecore.Data.Version.Latest, item.Database,
            SecurityCheck.Disable);

        // if there are any actions for the new state
        if (!stateItem.HasChildren)
            return;

        WorkflowPipelineArgs workflowPipelineArgs = new WorkflowPipelineArgs(item, null,
            null);

        // start executing the actions
        Pipeline pipeline = Pipeline.Start(stateItem, workflowPipelineArgs);
        if (pipeline == null)
            return;
        WorkflowCounters.ActionsExecuted.IncrementBy(pipeline.Processors.Count);
    }
}
```

Thanks to [this thread](#)

### Executing workflow command to change the workflow state

Thanks to [this great post](#)

If we want to mimic the Sitecore UI behavior and execute the command which will change the workflow state, we need to use `WorkflowProvider` to get an instance of the workflow assigned to the given item and call `Execute` method with a chosen command ID. This will fire all the actions which are defined under the command item node, change the state of the item and fire all the auto-actions defined below the new state item node:

```
public static WorkflowResult ExecuteCommand(Item item, string commandName, string comment)
```

```
{
    IWorkflow workflow = item.Database.WorkflowProvider.GetWorkflow(item);

    if (workflow == null)
    {
        return new WorkflowResult(false, "No workflow assigned to item");
    }

    WorkflowCommand command = workflow.GetCommands(item[FieldIDs.WorkflowState])
        .FirstOrDefault(c => c.DisplayName == commandName);

    if (command == null)
    {
        return new WorkflowResult(false, "Workflow command not found");
    }

    return workflow.Execute(command.CommandID, item, comment, false, new object[0]);
}
```

Read Workflow online: <https://riptutorial.com/sitecore/topic/4235/workflow>

---

# Chapter 16: Workflow

## Remarks

Workflows provide a flexible and controllable way of content creation, maintenance, and review. Workflow contains a list of states and commands.

## Examples

### Assign workflow to an item

Information about the item workflow is stored in the "Workflow" field.

```
var workflow = Factory.GetDatabase("master").WorkflowProvider.GetWorkflow(workflowId);
workflow.Start(item);

var workflowId = item.Fields["__Default workflow"].Value;
var workflow = Factory.GetDatabase("master").WorkflowProvider.GetWorkflow(workflowId);
workflow.Start(item);
```

Read Workflow online: <https://riptutorial.com/sitecore/topic/6315/workflow>



# Credits

S. No	Chapters	Contributors
1	Getting started with sitecore	<a href="#">Adrian Iorgu</a> , <a href="#">Community</a> , <a href="#">Jack Spektor</a> , <a href="#">Liam</a> , <a href="#">Paritosh Tripathi</a> , <a href="#">Rodrigo Peplau</a>
2	Diagnostics: Asserts	<a href="#">James de la Bastide</a> , <a href="#">JohnD</a>
3	Engagement Automation	<a href="#">Jack Spektor</a>
4	Glass Mapper	<a href="#">Shriroop</a>
5	Index configuration	<a href="#">Liam</a>
6	Items	<a href="#">Jack Spektor</a> , <a href="#">JohnD</a> , <a href="#">Zachary Kniebel</a>
7	LinkManager	<a href="#">Liam</a>
8	Scaling	<a href="#">Jack Spektor</a> , <a href="#">Liam</a>
9	Search	<a href="#">David Masters</a> , <a href="#">Jack Spektor</a> , <a href="#">Laurel</a>
10	Security	<a href="#">Jack Spektor</a> , <a href="#">Laurel</a> , <a href="#">SamMullins</a>
11	Sitecore query syntax	<a href="#">Zachary Kniebel</a>
12	Templates	<a href="#">Dheeraj Palagiri</a> , <a href="#">Wesley Lomax</a>
13	Unicorn	<a href="#">Shriroop</a>
14	Web Forms For Marketeers (WFFM)	<a href="#">Jack Spektor</a>
15	Workflow	<a href="#">Rodrigo Peplau</a>