Sitecore CMS 7.0 or later

Data Definition API Cookbook

A Conceptual Overview for CMS Developers

Table of Contents

Chapter 1 Introduction 3
Chapter 2 Data Templates 4
2.1 How to Access a Data Template 5
2.1.1 How to Access the Data Template Associated with an Item 5
2.1.2 How to Access an Existing Data Template 5
2.1.3 How to Create a Data Template 5
2.1.4 How to Access the Icon of a Data Template 6
2.1.5 How to Access the Base Templates of a Data Template 6
2.2 How to Access a Data Template Section 7
2.2.1 How to Access an Existing Data Template Section 7
2.2.2 How to Add a Section to a Data Template 7
2.2.3 How to Access the Icon of a Data Template Section 7
2.2.4 How to Access the Sort Order of a Data Template Section 8
2.3 How to Access a Data Template Field 9
2.3.1 How to Access an Existing Data Template Field 9
2.3.2 How to Add a Field to a Data Template Section 9
2.3.3 How to Access the Properties of a Data Template Field 9
How to Access the Sort Order of a Data Template Field 10
2.4 How to Access the Standard Values of a Data Template 11
2.4.1 How to Set a Standard Value of a Data Template Field 11
How to Specify a Default Workflow for Items Based on a Data Template 11
How to Specify Default Layout Details for Items Based on a Data Template 12
How to Specify Default Insert Options for Items Based on a Data Template 12
How to Specify a Default Subitems Sorting Rule for Items Based on a Data Template 12
How to Prepare your Data Template to Be Hydrated from the Index 13
Chapter 3 Branch Templates 20
3.1 How to Access an Existing Branch Template 21
3.2 How to Create a Branch Template 22
3.2.1 How to Create a Branch Template Containing a Single Item 22
3.3 How to Add an Item to a Branch Template 23
3.3.1 How to Copy Items to a Branch Template 23
3.4 How to Access Field Values in Items in Branch Templates 24
3.5 How to Control Access to a Branch Template 25
3.5.1 How to Control Who Can Use a Branch Template 25
3.5.2 How to Control Who Can Access Items Created from a Branch Template 25

Chapter 1

Introduction

This Data Definition API Cookbook provides a conceptual overview of Application Programming Interfaces (APIs) that Sitecore developers can use to access and manipulate data template and branch template definitions. For more information about the topics described in this document, see the *Data Definition Reference* and the *Data Definition Cookbook*. For more information about specific Sitecore APIs, see the *Sitecore API Reference*.

This document contains the following chapters:

- Chapter 1 – Introduction
- Chapter 2 – Data Templates
- Chapter 3 – Branch Templates

Chapter 2

Data Templates

This chapter provides information about and examples of APIs that you can use to access and manipulate data template, section, and field definitions.

This chapter contains the following sections:

- How to Access a Data Template
- How to Access a Data Template Section
- How to Access a Data Template Field
- How to Access the Standard Values of a Data Template

2.1 How to Access a Data Template

You can use the `Sitecore.Data.Items.TemplateItem` class to access data template definitions.

2.1.1 How to Access the Data Template Associated with an Item

You can use the `Sitecore.Data.Items.Item.Template` property to access the data template associated with an item. For example, to access the data template associated with the `/Sitecore/Content/Home` item in the Master database:

```
Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
Sitecore.Data.Items.Item home = master.GetItem("/sitecore/content/home");
Sitecore.Data.Items.TemplateItem template = home.Template;
```

For more information about the `Sitecore.Data.Items.Item`, `Sitecore.Data.Database`, and other APIs that are used in this section and throughout this document, see the *Content API Cookbook*.

2.1.2 How to Access an Existing Data Template

You can use the `Sitecore.Data.Database.Templates` collection property to access an existing data template. For the collection key, use the GUID of the data template, or the path to the data template definition item relative to the `/Sitecore/Templates` item, without the leading slash character ("`/`"). For example, to access the `Sample/Sample Item` data template in the Master database (`/Sitecore/Templates/Sample/Sample Item`):

```
Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
Sitecore.Data.Items.TemplateItem sample = master.Templates["sample/sample item"];

if (sample==null)
{
  //TODO: handle case that data template does not exist
}
```

The `Sitecore.Data.Database.Templates` collection returns Null if the specified data template does not exist.

2.1.3 How to Create a Data Template

You can use the `Sitecore.Data.Database.Templates.CreateTemplate()` method to create a data template. You must specify the data template name and the existing parent folder to contain the data template. For example, to ensure the `User Defined/Example Data Template` data template (`/Sitecore/Templates/User Defined/Example Data Template`) exists in the Master database:

```
Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
Sitecore.Data.Items.TemplateItem example =
  master.Templates["user defined/example data template"];

if (example == null)
{
  Sitecore.Data.Items.Item parent = master.GetItem("/sitecore/templates/user defined");
  example = master.Templates.CreateTemplate("Example Data Template", parent);
}
```

2.1.4 How to Access the Icon of a Data Template

You can use the `Sitecore.Data.Items.TemplateItem.InnerItem.Appearance.Icon` property to access the icon of a data template. For example, to set the icon of the `Sample/Sample Item` data template in the Master database to `network/16x16/home.png`:

```
Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
Sitecore.Data.Items.TemplateItem sample = master.Templates["sample/sample item"];
sample.InnerItem.Editing.BeginEdit();
sample.InnerItem.Appearance.Icon = "network/16x16/home.png";
sample.InnerItem.Editing.EndEdit();
```

For more information about data template icons, see the *Client Configuration Reference* and the *Client Configuration Cookbook*.

2.1.5 How to Access the Base Templates of a Data Template

You can use the `Sitecore.Data.Items.TemplateItem.BaseTemplates` property to access the base templates of a data template. You can update the base templates associated with a data template by setting the value of the `Sitecore.FieldIDs.BaseTemplate` field in the data template definition item. For example, to ensure that `User Defined/Example Data Template` in the Master database includes the `Sample/Sample Item` and `Common/Folder` base templates:

```
Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
Sitecore.Data.Items.TemplateItem example =
  master.Templates["user defined/example data template"];
Sitecore.Data.Fields.MultilistField baseTemplates =
  example.InnerItem.Fields[Sitecore.FieldIDs.BaseTemplate];

foreach (string key in new string[] { "sample/sample item",
  Sitecore.TemplateIDs.Folder.ToString() })
{
  Sitecore.Data.Items.TemplateItem template = master.Templates[key];

  if (!baseTemplates.Contains(template.ID.ToString()))
  {
    example.InnerItem.Editing.BeginEdit();
    baseTemplates.Add(template.ID.ToString());
    example.InnerItem.Editing.EndEdit();
  }
}
```

2.2 How to Access a Data Template Section

You can use the `Sitecore.Data.Items.TemplateSectionItem` class to access data template section definitions.

2.2.1 How to Access an Existing Data Template Section

You can use the `Sitecore.Data.Items.TemplateItem.GetSection()` method to access an existing data template section. For example, to access the Data section in the `Sample/Sample Item` data template in the Master database:

```
Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
Sitecore.Data.Items.TemplateItem sample = master.Templates["sample/sample item"];
Sitecore.Data.Items.TemplateSectionItem data = sample.GetSection("data");
```

If the data template definition item contains the specified section, the `Sitecore.Data.Items.TemplateItem.GetSection()` method returns that section. Otherwise, the `Sitecore.Data.Items.TemplateItem.GetSection()` method returns the first matching section encountered while recursively processing the base templates of that data template. If none of the base templates contain the section, then the `Sitecore.Data.Items.TemplateItem.GetSection()` method returns Null.

You can differentiate between sections defined within a data template and sections defined within one of its base templates by comparing the ID of the parent of a data template section definition item to the ID of a data template definition item as demonstrated in the following section How to Add a Section to a Data Template.

2.2.2 How to Add a Section to a Data Template

You can use the `Sitecore.Data.Items.TemplateItem.AddSection()` method to add a section to a data template. For example, to ensure the Data section exists in the `User Defined/Example Data Template` data template in the Master database:

```
Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
Sitecore.Data.Items.TemplateItem example =
  master.Templates["user defined/example data template"];
Sitecore.Data.Items.TemplateSectionItem data = example.GetSection("data");

if (data==null || data.InnerItem.Parent.ID!=example.ID)
{
  data = example.AddSection("Data",false);
}
```

Note
If you do not specify False as the second parameter to the `Sitecore.Data.Items.TemplateItem.AddSection()` method, the method may return a data template section defined in a base template.

2.2.3 How to Access the Icon of a Data Template Section

You can use the `Sitecore.Data.Items.TemplateItem.InnerItem.Appearance.Icon` property to access the icon associated with a data template section. For example, to set the icon of the Data

section in the `Sample/Sample Item` data template in the Master database to `applications/16x16/folder.png`:

```
Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
Sitecore.Data.Items.TemplateItem sample = master.Templates["sample/sample item"];
Sitecore.Data.Items.TemplateSectionItem data = sample.GetSection("data");
data.InnerItem.Editing.BeginEdit();
data.InnerItem.Appearance.Icon = "applications/16x16/folder.png";
data.InnerItem.Editing.EndEdit();
```

For more information about data template section icons, see the *Client Configuration Reference* and the *Client Configuration Cookbook*.

2.2.4 How to Access the Sort Order of a Data Template Section

You can use the `Sitecore.Data.Items.TemplateSectionItem.Sortorder` property to access the sort order of a data template section. The Content Editor sorts the sections within each data template according to the values of the sort order property. If you do not sort the sections, Sitecore sorts them alphabetically by name.

You can update the `Sitecore.FieldIDs.Sortorder` field in the `Sitecore.Data.Items.TemplateSectionItem.InnerItem.Fields` collection property to set the sort order for a data template section. For example, to set the sort order property of the Data section in the `Sample/Sample Item` data template in the Master database to `10`:

```
Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
Sitecore.Data.Items.TemplateItem sample = master.Templates["sample/sample item"];
Sitecore.Data.Items.TemplateSectionItem data = sample.GetSection("data");

if (data.Sortorder!=10)
{
  data.InnerItem.Editing.BeginEdit();
  data.InnerItem.Fields[Sitecore.FieldIDs.Sortorder].Value = 10.ToString();
  data.InnerItem.Editing.EndEdit();
}
```

2.3 How to Access a Data Template Field

You can use the `Sitecore.Data.Items.TemplateFieldItem` class to access data template field definitions.

2.3.1 How to Access an Existing Data Template Field

You can use the `Sitecore.Data.Items.TemplateItem.GetField()` method to access a data template field when the section containing the field is unimportant. For example, to access the Title field in the `User Defined/Example Data Template` data template in the Master database:

```
Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
Sitecore.Data.Items.TemplateItem sample = master.Templates["sample/sample item"];
Sitecore.Data.Items.TemplateFieldItem title = sample.GetField("title");
```

Note
The `Sitecore.Data.Items.TemplateItem.GetField()` method may return a field from a base template.

2.3.2 How to Add a Field to a Data Template Section

You can use the `Sitecore.Data.Items.TemplateSectionItem.AddField()` method to add a field to a data template section. For example, to ensure the Title field exists in the Data section of the `User Defined/Example Data Template` data template in the Master database:

```
Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
Sitecore.Data.Items.TemplateItem example =
  master.Templates["user defined/example data template"];
Sitecore.Data.Items.TemplateSectionItem data = example.GetSection("data");

if (data == null || data.InnerItem.Parent.ID != example.ID)
{
  data = example.AddSection("Data",false);
}

Sitecore.Data.Items.TemplateFieldItem title = data.GetField("title");

if (title == null)
{
  Sitecore.Data.Items.TemplateFieldItem field = data.AddField("Title");
}
```

Note
If the data template section definition item does not contain the specified field, the `Sitecore.Data.Items.TemplateSectionItem.GetField()` method returns Null. The `Sitecore.Data.Items.TemplateSectionItem.GetField()` method does not process base templates.

2.3.3 How to Access the Properties of a Data Template Field

You can use the following properties of the `Sitecore.Data.Items.TemplateFieldItem` class to access the properties of a data template field:

- `Sitecore.Data.Items.TemplateFieldItem.Source`: the configuration source of the field.
- `Sitecore.Data.Items.TemplateFieldItem.Title`: the title of the field.
- `Sitecore.Data.Items.TemplateFieldItem.Type`: the data type of the field.

For more information about data template field properties, see the *Data Definition Reference*, the *Client Configuration Reference*, and the *Client Configuration Cookbook*.

You can access additional properties of a data template using the `Sitecore.Data.Items.TemplateFieldItem.InnerItem.Fields` collection property. For the collection key, specify a member of the `Sitecore.TemplateFieldIDs` class.

For example, to set the data type, source, title, and description properties for the Title field in the `Sample/Sample Item` data template in the Master database:

```
Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
Sitecore.Data.Items.TemplateItem sample = master.Templates["sample/sample item"];
Sitecore.Data.Items.TemplateFieldItem title = sample.GetField("title");
Sitecore.Data.Database core = Sitecore.Configuration.Factory.GetDatabase("core");
Sitecore.Data.Items.Item textFieldType =
  core.GetItem("/sitecore/system/field types/simple types/single-line text");
title.BeginEdit();
title.Type = textFieldType.Name;
title.Source = "//TODO: replace with field source property";
title.Title = "//TODO: replace with field title";
title.InnerItem.Fields[Sitecore.TemplateFieldIDs.Description].Value =
  "//TODO: replace with field description";
title.EndEdit();
```

How to Access the Sort Order of a Data Template Field

You can use the `Sitecore.Data.Items.TemplateFieldItem.Sortorder` property to access the sort order of a data template field. The Content Editor sorts the fields within each section according to the values of the sort order property of the fields. If you do not sort the fields, Sitecore sorts them automatically alphabetically by name. For example, to sort the Title field in the `Sample/Sample Item` data template in the Master database after the Title field, assuming they exist in the same section:

```
Sitecore.Data.Database master =
  Sitecore.Configuration.Factory.GetDatabase("master");
Sitecore.Data.Items.TemplateItem sample = master.Templates["sample/sample item"];
Sitecore.Data.Items.TemplateFieldItem title = sample.GetField("title");
Sitecore.Data.Items.TemplateFieldItem text = sample.GetField("text");
title.BeginEdit();
title.Sortorder = 100;
title.EndEdit();
text.BeginEdit();
text.Sortorder = 200;
text.EndEdit();
```

2.4 How to Access the Standard Values of a Data Template

You can use the `Sitecore.Data.Items.TemplateItem.StandardValues` property to access the standard values of a data template. If the `Sitecore.Data.Items.TemplateItem.StandardValues` property is Null, then standard values do not exist for the data template.

If standard values do not exist for a data template, you can use the `Sitecore.Data.Items.TemplateItem.CreateStandardValues()` method to create the standard values item for that data template. For example, to access the standard values for the `User Defined/Example Data Template` data template in the Master database, creating the standard values item if needed:

```
Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
Sitecore.Data.Items.TemplateItem example =
  master.Templates["user defined/example data template"];
Sitecore.Data.Items.Item standardValues = example.StandardValues;

if (standardValues == null)
{
  standardValues = example.CreateStandardValues();
}
```

2.4.1 How to Set a Standard Value of a Data Template Field

You can use the `Sitecore.Data.Items.Item.Fields` collection property to set the standard value for a data template field. You can include tokens such as `$name` and `$now` in the standard value of a field. For example, to set the standard value for the Title field in the `User Defined/Example Data Template` data template in the Master database to `$name`:

```
Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
Sitecore.Data.Items.TemplateItem example =
  master.Templates["user defined/example data template"];
Sitecore.Data.Items.Item standardValues = example.StandardValues;
standardValues.Editing.BeginEdit();
standardValues.Fields["title"].Value = "$name";
standardValues.Editing.EndEdit();
```

For more information about setting field values and the tokens you can use in standard values, see the *Content API Cookbook* and the *Data Definition Cookbook*, respectively.

How to Specify a Default Workflow for Items Based on a Data Template

You can set the value of the `Sitecore.FieldIDs.DefaultWorkflow` field in the standard values for a data template to specify a default workflow for all items based on that data template. For example, to set the default workflow for the `User Defined/Example Data Template` data template in the Master database to `/Sitecore/System/Workflows/Sample Workflow`:

```
Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
Sitecore.Data.Items.TemplateItem example =
  master.Templates["user defined/example data template"];
Sitecore.Data.Items.Item standardValues = example.StandardValues;
Sitecore.Data.Items.Item sampleWorkflow =
  master.GetItem("/sitecore/system/workflows/sample workflow");
standardValues.Editing.BeginEdit();
standardValues.Fields[Sitecore.FieldIDs.DefaultWorkflow].Value =
  sampleWorkflow.ID.ToString();
standardValues.Editing.EndEdit();
```

How to Specify Default Layout Details for Items Based on a Data Template

You can set the value of the `Sitecore.FieldIDs.LayoutField` field in the standard values of a data template to specify default layout details for all items based on that data template. For example, to copy layout details from the `/Sitecore/Content/Home` item to the standard values of the `User Defined/Example Data Template` data template in the Master database:

```
Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
Sitecore.Data.Items.Item home = master.GetItem("/sitecore/content/home");
Sitecore.Data.Items.TemplateItem example =
  master.Templates["user defined/example data template"];
Sitecore.Data.Items.Item standardValues = example.StandardValues;
standardValues.Editing.BeginEdit();
standardValues.Fields[Sitecore.FieldIDs.LayoutField].Value =
  home.Fields[Sitecore.FieldIDs.LayoutField].Value;
standardValues.Editing.EndEdit();
```

How to Specify Default Insert Options for Items Based on a Data Template

You can set the values of the `Sitecore.FieldIDs.Branches` and `__Insert Rules` fields in the standard values of a data template to specify default insert options for all items based on that data template. For example, to ensure insert options for the `User Defined/Example Data Template` data template in the Master database include the `Sample/Sample` Item and `Common/Folder` data templates, and that insert rules include those defined in the `/Sitecore/Content/Home` item:

```
Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
Sitecore.Data.Items.TemplateItem example =
  master.Templates["user defined/example data template"];
Sitecore.Data.Items.Item standardValues = example.StandardValues;
standardValues.Editing.BeginEdit();
Sitecore.Data.Fields.MultilistField standardInsertOptions =
  standardValues.Fields[Sitecore.FieldIDs.Branches];

foreach (string templateKey in new string[] { "sample/sample item",
  Sitecore.TemplateIDs.Folder.ToString() })
{
  Sitecore.Data.Items.TemplateItem template = master.Templates[templateKey];

  if (!standardInsertOptions.Contains(template.ID.ToString()))
  {
    standardInsertOptions.Add(template.ID.ToString());
  }
}

Sitecore.Data.Fields.MultilistField standardInsertRules =
  standardValues.Fields["__Insert Rules"];
Sitecore.Data.Items.Item home = master.GetItem("/sitecore/content/home");
Sitecore.Data.Fields.MultilistField homeInsertRules = home.Fields["__Insert Rules"];

foreach(string homeInsertRuleId in homeInsertRules.Items)
{
  if (!standardInsertRules.Contains(homeInsertRuleId))
  {
    standardInsertRules.Add(homeInsertRuleId);
  }
}

standardValues.Editing.EndEdit();
```

How to Specify a Default Subitems Sorting Rule for Items Based on a Data Template

You can set the value of the `Sitecore.FieldIDs.SubmitemsSorting` field in the standard values for a data template to specify the default subitem sorting rule for all items based on the data template. For

example, to set the default subitem sorting rule for the `User Defined/Example Data Template` data template in the Master database to the `/Sitecore/System/Settings/Subitems Sorting/Updated` subitems sorting rule:

```
Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
Sitecore.Data.Database core = Sitecore.Configuration.Factory.GetDatabase("core");
Sitecore.Data.Items.Item rule =
  core.GetItem("/sitecore/system/settings/subitems sorting/updated");
Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
Sitecore.Data.Items.TemplateItem example =
  master.Templates["user defined/example data template"];
Sitecore.Data.Items.Item standardValues = example.StandardValues;
standardValues.Editing.BeginEdit();
standardValues.Fields[Sitecore.FieldIDs.SubitemsSorting].Value = rule.ID.ToString();
standardValues.Editing.EndEdit();
standardValues.Editing.EndEdit();
```

How to Prepare your Data Template to Be Hydrated from the Index

In the Object Relational Mapping (ORM) context, hydration allows you to convert data between relational databases, XML, and indexes. In Sitecore, it is the process of mapping fields in data templates to properties in corresponding classes.

When you create data templates in Sitecore, they are stored in the database and added to the content tree. This section describes how to map the data templates to classes in which the properties *hydrate* their values from the index.

The following table lists the fields in a simple product data template in the content tree:

Field	Type
Name	Single Line Text
Price	Number
Specification	Rich Text
Reviews Collection	Multilist
Rating	Number
Introduction Blurb	Multi Line Text
Release Date	Date
Stock Count	Number
In Stock	Checkbox

This data template maps to a class in your code as follows:

Property	Type
Name	String
Price	Float
Specification	String
Reviews Collection	IEnumerable<Review>
Rating	Int
Introduction Blurb	String

Property	Type
Release Date	DateTime
Stock Count	Int
In Stock	bool

Now, you should specify which data template property value is stored in the index, database, or other sources.

The following table lists some property values that are distributed across different sources:

Property	Location
Name	Index
Price	Web Service
Specification	Index
Reviews Collection	Database
Rating	Index
Introduction Blurb	Index
Release Date	Index
Stock Count	Web Service
In Stock	Index

There is no ideal way to distribute the properties across different sources. It varies from case to case. For the best performance, all the properties should be stored in the index but this is not always possible. Some properties like `Stock Count` are expensive to store in the index because they frequently change and have dynamic logic. To limit the size of the index on the disk, there may be some properties such as `reviews` that should be stored in a separate database.

The following class represents the *Product* data template:

```
public class Product
{
  [IndexField("productname")]
  public string Name
  {
    get;
    set;
  }
  public float Price
  {
    get;
    internal set;
  }
  public string Specification
  {
    get;
    set;
  }
  public IEnumerable<Review> ReviewsCollection
  {
    get;
    set;
  }
  [IndexField("productrating")]
  public int Rating
```

```
        {
          get;
          set;
        }
        [IndexField("productblurb")]
        public string IntroductionBlurb
        {
          get;
          set;
        }
        public DateTime ReleaseDate
        {
           get;
           set;
        }
        public int StockCount
        {
          get;
          set;
        }
        public bool InStock
        {
          get;
          set;
        }
      }
```

The following table describes how the properties in the `Product` class are mapped to the different sources:

Property	Description
Name	This property is mapped to an index field that has another name because there is no field called *Name* in the index. The hydration process maps the field value of the `productName` field value in the index to the `Name` property in the class.
Price	This property has a `get`, a `set` and its value is not retrieved from the index. However, if there is a *Price* field in the index, Sitecore will automatically hydrate this value. This can also be overridden to get this value from a web service.
Specification	This property's name matches the field in the index. The hydration process maps it automatically without the `[IndexField]` attribute.
Review Collection	This property's value is stored in the database because you can use the ID to look it up.
Rating	This property is mapped to the `productrating` field in the index.
Introduction Blurb	This property is mapped to the `productblurb` field in the index.
Release Date	This property's name matches the field in the index. The hydration process maps it automatically without the `[IndexField]` attribute.
Stock Count	This property has a `get`, a `set` and its value is not retrieved from the index. However, if there is a stock count field in the index, Sitecore will automatically hydrate this value. This can also be overridden to get this value from a web service.

Property	Description
In Stock	This property is stored in the index because the logic is simple. Sitecore automatically maps its name because it matches a field in the index called `instock`. **Note** By default, all the field names in the index are in lower case. Even if you use Camel or any other casing system, Sitecore adapts to it and uses lower case to hydrate the property.

Because we are using Lucene, you can see the mappings in the `Sitecore.ContentSearch.Lucene.DefaultConfiguration.config` file.

Now, you can run queries against the products in Sitecore.

Note
To save disk space, Sitecore does not store the values of the fields in the index, by default. This is because you may not use the search index to hydrate items.

When configuring the mappings, you have the following choices:

- You can globally make mappings for a field type such as Single-Line Text
- You can specifically do it at the field name level.

To keep the index light-weight, you should map by field name. To avoid duplicate field names in templates, you have the following solutions:

- Abstract the fields into a base template
- Add a prefix to the fields such as *ProductName* instead of just *Name*.

Currently, we do not support the mapping of fields to types for field ID.

The following snippet is the default configuration for the field mapping in the `Sitecore.ContentSearch.Lucene.DefaultConfiguration.config` file:

```
        <fieldNames hint="raw:AddFieldByFieldName">
          <fieldType fieldName="parsedlanguage" storageType="YES" indexType="TOKENIZED"
          VectorType="NO" boost="1f" type="System.String"
          settingType="Sitecore.ContentSearch.LuceneProvider.LuceneSearchFieldConfiguration,
          Sitecore.ContentSearch.LuceneProvider">
          <Analyzer
          type="Sitecore.ContentSearch.LuceneProvider.Analyzers.LowerCaseKeywordAnalyzer,
          Sitecore.ContentSearch.LuceneProvider" />
          </fieldType>
          <fieldType fieldName="_templatename" storageType="YES" indexType="TOKENIZED"
          vectorType="NO" boost="1f" type="System.String"
          settingType="Sitecore.ContentSearch.LuceneProvider.LuceneSearchFieldConfiguration,
          Sitecore.ContentSearch.LuceneProvider">
          <Analyzer
          type="Sitecore.ContentSearch.LuceneProvider.Analyzers.LowerCaseKeywordAnalyzer,
          Sitecore.ContentSearch.LuceneProvider" />
          </fieldType>
          <fieldType fieldName="_securitytoken" storageType="YES" indexType="TOKENIZED"
          vectorType="NO" boost="1f" type="System.String"
          settingType="Sitecore.ContentSearch.LuceneProvider.LuceneSearchFieldConfiguration,
          Sitecore.ContentSearch.LuceneProvider">
          <Analyzer
          type="Sitecore.ContentSearch.LuceneProvider.Analyzers.LowerCaseKeywordAnalyzer,
```

```
        Sitecore.ContentSearch.LuceneProvider" />
        </fieldType>
        <fieldType fieldName="calculateddimension" storageType="YES" indexType="TOKENIZED"
        vectorType="NO" boost="1f" type="System.String"
        settingType="Sitecore.ContentSearch.LuceneProvider.LuceneSearchFieldConfiguration,
        Sitecore.ContentSearch.LuceneProvider">
        <Analyzer
        type="Sitecore.ContentSearch.LuceneProvider.Analyzers.LowerCaseKeywordAnalyzer,
        Sitecore.ContentSearch.LuceneProvider" />
        </fieldType>
        <fieldType fieldName="sizerange" storageType="YES" indexType="TOKENIZED"
        vectorType="NO" boost="1f" type="System.String"
        settingType="Sitecore.ContentSearch.LuceneProvider.LuceneSearchFieldConfiguration,
        Sitecore.ContentSearch.LuceneProvider">
        <Analyzer
        type="Sitecore.ContentSearch.LuceneProvider.Analyzers.LowerCaseKeywordAnalyzer,
        Sitecore.ContentSearch.LuceneProvider" />
        </fieldType>
        <fieldType fieldName="title" storageType="NO"  indexType="TOKENIZED"
        vectorType="NO" boost="1f" type="System.String"
        settingType="Sitecore.ContentSearch.LuceneProvider.LuceneSearchFieldConfiguration,
         Sitecore.ContentSearch.LuceneProvider" />
        <fieldType fieldName="text" storageType="NO"  indexType="TOKENIZED"
        vectorType="NO" boost="1f" type="System.String"
        settingType="Sitecore.ContentSearch.LuceneProvider.LuceneSearchFieldConfiguration,
        Sitecore.ContentSearch.LuceneProvider" />
        <fieldType fieldName="version" storageType="NO"  indexType="TOKENIZED"
        vectorType="NO" boost="1f" type="System.Int32"
        settingType="Sitecore.ContentSearch.LuceneProvider.LuceneSearchFieldConfiguration,
        Sitecore.ContentSearch.LuceneProvider" />
      </fieldNames>
```

Now, you should specify whether or not the properties store their data and how to store it.

To be precise, you can add the following field mapping to the configuration of the *Web* and *Master* indexes because they are the most needed. However, you should add it in the default field mapping index because later you can introduce other indexes that may also need this mapping:

```
      <fieldType fieldName="productname" storageType="YES"  indexType="TOKENIZED"
      vectorType="NO" boost="4f" type="System.String"
      settingType="Sitecore.ContentSearch.LuceneProvider.LuceneSearchFieldConfiguration,
      Sitecore.ContentSearch.LuceneProvider" />
      <fieldType fieldName="specification" storageType="YES"  indexType="TOKENIZED"
      vectorType="NO" boost="1f" type="System.String"
      settingType="Sitecore.ContentSearch.LuceneProvider.LuceneSearchFieldConfiguration,
      Sitecore.ContentSearch.LuceneProvider" />
      <fieldType fieldName="productrating" storageType="YES"  indexType="TOKENIZED"
      vectorType="NO" boost="1f" type="System.Int32"
      settingType="Sitecore.ContentSearch.LuceneProvider.LuceneSearchFieldConfiguration,
      Sitecore.ContentSearch.LuceneProvider" />
      <fieldType fieldName="
      productblurb" storageType="YES"  indexType="TOKENIZED"    vectorType="NO" boost="2f"
      type="System.String"
      settingType="Sitecore.ContentSearch.LuceneProvider.LuceneSearchFieldConfiguration,
      Sitecore.ContentSearch.LuceneProvider" />
      <fieldType fieldName=" releasedate"
      storageType="YES"  indexType="TOKENIZED" vectorType="NO" boost="1f"
      type="System.DateTime"
      settingType="Sitecore.ContentSearch.LuceneProvider.LuceneSearchFieldConfiguration,
      Sitecore.ContentSearch.LuceneProvider" />
      <fieldType fieldName="
      productrating" storageType="YES"  indexType="TOKENIZED"    vectorType="NO" boost="1f"
      type="System.Boolean"
      settingType="Sitecore.ContentSearch.LuceneProvider.LuceneSearchFieldConfiguration,
      Sitecore.ContentSearch.LuceneProvider" />
```

In the previous index configuration, you have specified that all the fields are stored and added a globally static boost of `4f` to `productname` and `2f` to `instock`. This means that it is more important to find a

match on the `productname` than finding a match on the `blurb` or `specification`. You have also added a boost to `blurb` of `2f` because it is more important than `specification`. You can consider boosting `InStock`, but this is not the place to boost it. We can achieve this in a more flexible way by using the boosting rules in the *Sitecore Search and Indexing Guide*.

To search for products in a bucket, use the following snippet:

```
namespace Sample.SearchInBucket
{
    using System;
    using System.Collections.Generic;
    using System.Diagnostics;
    using System.Linq;
    using Sitecore.Buckets.Util;
    using Sitecore.ContentSearch;
    using Sitecore.ContentSearch.Linq;
    using Sitecore.ContentSearch.Utilities;
    using Sitecore.Data.Items;
    using Sitecore.Web.UI.WebControls;

    public partial class SampleControl : System.Web.UI.UserControl
    {
        /// <summary>
        /// The page_load.
        /// </summary>
        /// <param name="sender">
        /// The sender.
        /// </param>
        /// <param name="e">
        /// The e.
        /// </param>
        protected void Page_Load(object sender, EventArgs e)
        {
          Var productsBucket =
          Sitecore.Context.Database.GetItem("/sitecore/content/home/products");
          using (var context =
          Sitecore.ContentSearch.ContentSearchManager.GetIndex(productsBucket).
          CreateSearchContext())
          {
            var results = context.GetQueryable<Product>().Where(i => i.InStock);
          }
        }
    }
}
```

Now, the `results` variable has retrieved all the products that are in stock and placed them in the content tree under `productBucket`.

If you inspect the values in this object, you should have:

```
Name: Product 1
Price: null
Specifications: This is an awesome product
ReviewsCollection: null
Rating: 3
IntroductionBlurb: This is good.
ReleaseDate: 12/12/2012 12:12:12
StockCount: null
InStock: true
```

The previous object has been through the hydration process and has extracted all it could from the index:

- Some hydration processes were done because of the attribute mappings, for example, `[IndexField("productname")]`.
- Some hydration processes were not done because there was a one to one mapping between the names and field names of the property in the search index.

- Some hydration processes resulted in null because the index was not able to resolve them in either of the previous ways. However, this is an intended behavior because you may want to use a web service to search for the other information in the database.

  To use a web service to get these values:

```
foreach (var product in results)
{
  product.Price = WebServiceCallForPrice();
  //In reality you would store reviewscollection in the index, but this is simply to show
  that you can hydrate the item in many ways.
  product.ReviewsCollection =
  Sitecore.Context.Database.GetItem("/sitecore/content/home/reviews/" + product.Name);
  product.StockCount = WebServiceCallForStockCount();
}
```

Chapter 3

Branch Templates

This chapter contains information about and examples of APIs that you can use to access and manipulate data templates and branch templates.

This chapter contains the following sections:

- How to Access an Existing Branch Template
- How to Create a Branch Template
- How to Add an Item to a Branch Template
- How to Access Field Values in Items in Branch Templates
- How to Control Access to a Branch Template

3.1 How to Access an Existing Branch Template

You can use the `Sitecore.Data.Items.BranchItem` class to access branch templates. You can use the `Sitecore.Data.Database.Branches` collection property to access an existing branch template. For the key to the collection, use the path to the branch template definition item relative to the `/Sitecore/Templates` item, without the leading slash character ("`/`"). For example, to access the `Branches/User Defined/Example Branch Template` branch template (`/Sitecore/Templates/Sitecore/Branches/User Defined/Example Branch Template`) in the Master database:

```
Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
Sitecore.Data.Items.BranchItem exampleBranch =
  master.Branches["branches/user defined/example branch template"];

if (exampleBranch==null)
{
  //TODO: handle case that branch template does not exist
}
```

Note
You should not create a branch template inside a bucket unless it is necessary. When the branch is created inside a bucket, you must run `Sync`.

3.2 How to Create a Branch Template

You can use the `Sitecore.Data.Items.Item.Add()` method to create a branch template. For example, to ensure the `Branches/User Defined/Example Branch Template` branch template exists in the Master database:

```
Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
Sitecore.Data.Items.BranchItem exampleBranch =
  master.Branches["branches/user defined/example branch template"];

if (exampleBranch == null)
{
  Sitecore.Data.Items.Item parent =
    master.GetItem("/sitecore/templates/branches/user defined");
  Sitecore.Data.Items.TemplateItem example =
    master.Templates[Sitecore.TemplateIDs.BranchTemplate];
  exampleBranch = new Sitecore.Data.Items.BranchItem(
    parent.Add("Example Branch Template", example));
}
```

3.2.1 How to Create a Branch Template Containing a Single Item

You can use the `Sitecore.Data.Masters.CreateMaster()` method to create a branch template containing a single item named `$name`. You may want to copy the icon from the data template associated with the item to the branch template definition item. For example, to create the `Branches/User Defined/Sample Item` branch template containing a single item named `$name` based on the `Sample/Sample Item` data template in the Master database, copying the icon from the data template to the branch template definition item:

```
Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
Sitecore.Data.Items.Item parent =
  master.GetItem("/sitecore/templates/branches/user defined");
Sitecore.Data.Items.TemplateItem exampleTemplate =
  master.Templates["sample/sample item"];
Sitecore.Data.Items.Item exampleItem =
  Sitecore.Data.Masters.Masters.CreateMaster(parent,exampleTemplate.ID);
Sitecore.Data.Items.BranchItem exampleBranch =
  new Sitecore.Data.Items.BranchItem(exampleItem);
exampleBranch.InnerItem.Editing.BeginEdit();
exampleBranch.InnerItem.Appearance.Icon = exampleTemplate.Icon;
exampleBranch.InnerItem.Editing.EndEdit();
```

3.3 How to Add an Item to a Branch Template

You can use the `Sitecore.Data.Items.Item.Add()` method to add an item to a branch template. For example, to add an item named `$name` based on the `Sample/Sample Item` data template to the `Branches/User Defined/Example Branch Template` branch template in the Master database if that item does not already exist:

```
Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
Sitecore.Data.Items.Item exampleBranch =
  master.Branches["branches/user defined/example branch template"];
Sitecore.Data.Items.Item child = exampleBranch.Children["$name"];

if (child == null)
{
  Sitecore.Data.Items.TemplateItem sampleTemplate =
    master.Templates["sample/sample item"];
  child = exampleBranch.Add("$name",sampleTemplate);
}
```

3.3.1 How to Copy Items to a Branch Template

You can use the `Sitecore.Data.Items.Item.CopyTo()` method to copy an existing item or hierarchy of items to a branch template. For example, to copy the `/Sitecore/Content/Home` item to the `Branches/User Defined/Sample Item` branch template in the Master database:

```
Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
Sitecore.Data.Items.BranchItem exampleBranch =
  master.Branches["branches/user defined/sample item"];
Sitecore.Data.Items.Item sampleItem = exampleBranch.InnerItem.Children["$name"];
Sitecore.Data.Items.Item home = master.GetItem("/sitecore/content/home");
home.CopyTo(exampleBranch.InnerItem, home.Name);
```

3.4 How to Access Field Values in Items in Branch Templates

You can use the `Sitecore.Data.Items.Item.Fields` collection property access field values in the items of a branch template.

For example, to set the value of the Title field in the child named `$name` beneath under the `Branches/User Defined/Example Branch Template` branch template in the Master database to `$name`:

```
Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
Sitecore.Data.Items.BranchItem sampleBranch =
  master.Branches["branches/user defined/example branch template"];
Sitecore.Data.Items.Item name = sampleBranch.InnerItem.Children["$name"];
name.Editing.BeginEdit();
name.Fields["title"].Value = " $name";
name.Editing.EndEdit();
```

For more information about setting field values, see the *Content API Cookbook*.

3.5 How to Control Access to a Branch Template

You can manipulate the security access rights of a branch template definition item to control who can insert items using that branch template. You can manipulate the security access rights for the descendants of a branch template definition item to control who can access items inserted using that branch template. For more information about security APIs, see the *Security API Cookbook*.

3.5.1 How to Control Who Can Use a Branch Template

You can control which users can use a branch template to insert items by configuring read access rights in the branch template definition item. For example, to specify that only administrators and the `sitecore\user` user can create items using the `Branches/User Defined/Example Branch Template` branch template in the master database, disable security inheritance for the branch template definition item, and then grant the read access right to the context user:

```
Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
Sitecore.Data.Items.BranchItem exampleBranch =
  master.Branches["branches/user defined/example branch template"];
Sitecore.Security.AccessControl.AccessRuleCollection accessRules =
  new Sitecore.Security.AccessControl.AccessRuleCollection();
Sitecore.Security.Accounts.Role everyone =
  Sitecore.Security.Accounts.Role.FromName("  everyone");
accessRules.Helper.AddinheritancePermission(everyone, AccessRight.Any,
  PropagationType.Entity, InheritancePermission.Deny);
accessRules.Helper.AddinheritancePermission(everyone, AccessRight.Any,
  PropagationType.Descendants, InheritancePermission.Deny);
Sitecore.Security.Accounts.User user =
  Sitecore.Security.Accounts.User.FromName(@"sitecore\user",false);

foreach (Sitecore.Security.AccessControl.AccessRight accessRight in
  new Sitecore.Security.AccessControl.AccessRight[] {
Sitecore.Security.AccessControl.AccessRight.ItemRead,
Sitecore.Security.AccessControl.AccessRight.ItemWrite })
{
  accessRules.Helper.AddAccessPermission(user, accessRight,
      Sitecore.Security.AccessControl.PropagationType.Entity, AccessPermission.Allow);
  accessRules.Helper.AddAccessPermission(user, accessRight,
    Sitecore.Security.AccessControl.PropagationType.Descendants,
    AccessPermission.Allow);
}

exampleBranch.InnerItem.Security.SetAccessRules(accessRules);
```

Descendant items by default inherit access rights from the branch template definition item. Access rights for a branch template definition item control which accounts can use the branch template to insert items, as well as which accounts can manipulate the items within the branch template.

3.5.2 How to Control Who Can Access Items Created from a Branch Template

You can control which users can perform various operations on items created from a branch template by applying access rights to the items under the branch template definition item. When a user inserts an item using a branch template, Sitecore copies the access rights defined in the items within the branch template to the created items. For example, to specify that everyone has read access to all items created from a the `Branches/User Defined/Example Branch Template` branch template in the Master database:

```
Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
Sitecore.Data.Items.BranchItem exampleBranch =
```

```
        master.Branches["branches/user defined/example branch template"];
      Sitecore.Security.AccessControl.AccessRuleCollection emptyAccessRules =
        new Sitecore.Security.AccessControl.AccessRuleCollection();
      Sitecore.Security.AccessControl.AccessRuleCollection accessRules =
          new Sitecore.Security.AccessControl.AccessRuleCollection();
      Sitecore.Security.Accounts.Role everyone =
        Sitecore.Security.Accounts.Role.FromName("__everyone");
      accessRules.Helper.AddinheritancePermission(everyone, AccessRight.Any,
        PropagationType.Entity, InheritancePermission.Allow);
      accessRules.Helper.AddinheritancePermission(everyone, AccessRight.Any,
        PropagationType.Descendants, InheritancePermission.Allow);
      accessRules.Helper.AddAccessPermission(everyone,
        Sitecore.Security.AccessControl.AccessRight.ItemRead,
        Sitecore.Security.AccessControl.PropagationType.Entity, AccessPermission.Allow);

      foreach( Sitecore.Data.Items.Item child in exampleBranch.InnerItem.Children )
      {
        child.Security.SetAccessRules(accessRules);

        foreach( Sitecore.Data.Items.Item descendant in child.Axes.GetDescendants())
        {
          descendant.Security.SetAccessRules(emptyAccessRules);
        }
      }
```

Note
When using a branch template to insert one or more items, Sitecore does not copy the access rights defined in the branch template definition item to the inserted items. The access rights defined for a branch template definition item affect who can use and maintain the branch template, but do not affect the access rights of items created using that branch template. If you do not set access rights on the items within the branch template, then the new items inherit access rights from their new parent.