Sitecore CMS 6.4 and later

Content API Cookbook

A Conceptual Overview for CMS Developers

Table of Contents

Chapter 1

Introduction

This document provides a conceptual overview of Application Programming Interfaces (APIs) that CMS developers can use to manage data in Sitecore databases, including field values, dynamic links between items, and troubleshooting information.[1]

The information in this document applies to Sitecore 6.4.

This document contains the following chapters:

[1] For more information about APIs used in presentation components, see the *Presentation Component API Cookbook* at http://sdn.sitecore.net/Reference/Sitecore%206/Presentation%20Component%20API%20Cookbook.aspx

Chapter 2

Working with Databases

This chapter provides information about APIs for accessing Sitecore databases and publishing from the Master database to one or more publishing target databases.

This chapter contains the following sections:

- How to Access Sitecore Databases
- How to Publish

2.1 How to Access Sitecore Databases

You can use the `Sitecore.Data.Database` class to access Sitecore databases.[2] Each Sitecore instance can access multiple Sitecore databases. A default Sitecore configuration includes three databases:

- Master — Contains all versions of all content supporting CMS users.
- Web — Contains published versions from Master supporting the Web site(s).
- Core — Contains data controlling the Sitecore CMS user interfaces.

You can use the techniques described in the following sections to access Sitecore databases.

It is important to remember that:

- You should *always* use Sitecore APIs to access Sitecore databases.
- You should *not* use SQL commands to access Sitecore databases.
- All code runs in the security context of the context user by default.
- Every attempt to access databases, items, fields, and other resources that do not exist may return Null or raise an exception.

  You can use a security user switcher or security disabler for specific tasks if you know the context user does not have the access rights required to complete an operation.

  For more information about using a security user switcher or security disabler, see the section *How to Resolve Item Access Rights*.
- Developers should check for Null when accessing items and fields.

  For brevity's sake, the code examples in this document do not always check for Null.
- The Sitecore APIs access Sitecore databases.

  You can also access external data stores using the appropriate .NET APIs.
- If you create, update, or remove items from a publishing target database, publishing from Master to that publishing target will overwrite those changes.

  You should avoid writing to publishing target databases. To reduce the risk of writing to a publishing target database, do not write to a Sitecore database from a presentation component.
- You can access data templates, and fields by specifying the name, partial path, full path, ID, and potentially other criteria such as language and version of various API methods.

  For best performance, use IDs whenever possible, but use constants or other features to avoid hard-coding strings in more than one class.

2.1.1 How to Access the Context Database

The context database is the primary database associated with the logical site accessed by the Web client. For presentation components running on the published Web sites, the context database is one of the publishing target databases, such as Web.[3] In the Page Editor, the context database is the

[2] For more information about Sitecore databases, see the *Content Reference* manual at http://sdn.sitecore.net/Reference/Sitecore%206/Content%20Reference.aspx.
[3] For more information about presentation components, see the *Presentation Component Reference* manual and the *Presentation Component Cookbook* at http://sdn.sitecore.net/Reference/Sitecore%206/Presentation%20Component%20Reference.aspx and http://sdn.sitecore.net/Reference/Sitecore%206/Presentation%20Component%20Cookbook.aspx.

Master database. In the Desktop, the Content Editor, and other CMS user interfaces, the context database is the Core database.

Important
Presentation components almost exclusively access the context database instead of referencing a database by name.

You can use the `Sitecore.Context.Database` property to access the context database. For example, to access the context database:

```
        Sitecore.Data.Database context = Sitecore.Context.Database;
```

Important
CMS user interface components use configuration data in the context database to manage data in the content database. For more information about the content database, see the section *How to Access the Content Database*.

2.1.2 How to Access a Database by Name

You can use the `Sitecore.Configuration.Factory.GetDatabase()` method to access a specific database. For example, to access the Master database:

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

Logic that publishes or import data typically accesses the Master database by name.

Note
If the first parameter does not match the `id` of any of the `/configuration/sitecore/databases/database` elements in `web.config`, the `Sitecore.Configuration.Factory.GetDatabase()` method throws an exception. This comparison is case-sensitive.

2.1.3 How to Access the Content Database

CMS user interface components such as the Content Editor interact with the content database. The default content database is the Master database. User interfaces such as the Sitecore Desktop allow the user to change the content database to another database.

You can use the `Sitecore.Context.ContentDatabase` property to access the content database. For example, to access the context database:

```
        Sitecore.Data.Database content = Sitecore.Context.ContentDatabase;
```

2.2 How to Publish

You can publish an item, all publishable versions of an item and its descendants, or the Master database.

2.2.1 How to Publish an Item or a Branch of Items

You can use the same APIs to publish an individual item or an item including all of its publishable descendants. To publish the `/Sitecore/Content/Home` item in all languages to the Web database:

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

Sitecore.Data.Database[] targetDatabases = { target };
Sitecore.Globalization.Language[] languages = master.Languages;
bool deep = false;
bool compareRevisions = true;

Sitecore.Publishing.PublishManager.PublishItem(home, targetDatabases, languages,
  deep, compareRevisions);
```

To publish an item and all of its publishable descendants, set the `deep` property to `true` when calling the `PublishItem()` method.

When the `compareRevisions` parameter is set to `true`, Sitecore compares the value of the `Revision` field in the versions of the item in the source and the target databases during the publishing operation. If the values of the `Revision` fields are the same, the version is not published. This logic is also known as smart publish.

To make Sitecore publish items independently of the `Revision` field values, the `compareRevisions` value should be set to `false`. This logic is also known as full publish.

Remember about security restrictions when using a publishing code. Security checking depends on the `Publishing.CheckSecurity` setting in the Web.config file.
When `Publishing.CheckSecurity` is set to `true`, Sitecore uses the `SecurityEnabler` in the publishing code and publishing will be performed under the context user. To change the user for publishing, you may use the following code sample:

```
using (new UserSwitcher(some_admin_user))
{
//perform publishing here
}
```

Note that `SecurityDisabler` doesn't work in this case.

When `Publishing.Checksecurity` is set to `false`, you may use `UserSwitcher` to make the information on item updates relevant (like Updated by: some_admin_user instead of Updated by: sitecore\anonymous).
If you are not going to use this information, you may use `SecurityDisabler`:

```
using (new SecurityDisabler())
{
//perform publishing here
}
```

Note
The value in the `Revision` field is automatically updated every time an item is saved, unless the code that modifies the item is using `ItemEditing` or `EditContext` objects where the value of the `updateStatistics` argument is set to `false`. If you set `updateStatistics` to `false`, you must either manually set the `Revision` field to a new `Guid` when updating the item or set `compareRevisions` to `false` when publishing.

2.2.2 How to Publish the Master Database

You can publish every publishable version of all the publishable items in every language in the master database. For example, to publish the Master database incrementally in every language to the Web database:

```
        Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
        Sitecore.Data.Database target = Sitecore.Configuration.Factory.GetDatabase("web");

        Sitecore.Data.Database[] targetDatabases = { target };
        Sitecore.Globalization.Language[] languages = master.Languages;

        Sitecore.Publishing.PublishManager.PublishIncremental(master, targetDatabases,
languages);
```

If you prefer to use smart publishing for all the publishable items, simply call the `PublishSmart()` method instead of the `PublishIncremental()` method.

If you prefer to use full publishing for the all publishable items, simply call the `Republish()` method instead of the `PublishIncremental()` method. A full publish of the entire database is an expensive operation that can take a long time.

Note
If you use smart publish, the note about `updateStatistics` in the previous section also applies. If you set `updateStatistics` to `false` in your code, you should perform a full publish (or manually set the `Revision` field to a new `Guid` when updating the items).

Chapter 3

Working with Items

This chapter provides information about APIs for accessing, creating, updating, moving, deleting, and performing other operations on items.

This chapter contains the following sections:

3.1 How to Access an Item

You can use the `Sitecore.Data.Items.Item` class to access any item.

Note
Sitecore provides specialized classes to represent specific types as items, such as `Sitecore.Data.Items.TemplateItem` to represent a data template and `Sitecore.Data.Items.MediaItem` to represent a media item.

You can use the `Sitecore.Data.Database.GetItem()` method to retrieve a `Sitecore.Data.Item.Item`. You can specify the ID of the item or the path to the item as the first parameter to the `Sitecore.Data.Database.GetItem()` method. For example, to access 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");
```

If the item does not exist, or the context user does not have read access rights to the item, the `Sitecore.Data.Database.GetItem()` method returns Null.

Note
Sitecore does compare case when evaluating item paths.

3.1.1 How to Access System Items

You can use members of the `Sitecore.ItemIDs` class to access system items without hard-coding paths.[4] For example, to access the `/Sitecore/Media Library` item in the Master database:

```
        Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
        Sitecore.Data.Items.Item mediaRoot =
master.GetItem(Sitecore.ItemIDs.MediaLibraryRoot);
```

3.1.2 How to Access System Data Templates

You can use the `Sitecore.TemplateIDs` class to access system data templates.[5] For example, to access the standard template in the Master database:[6]

```
        Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
        Sitecore.Data.Items.TemplateItem standard =
          master.Templates[Sitecore.TemplateIDs.StandardTemplate];
```

3.1.3 How to Access the Context Item

In presentation and other components, processing often begins with the context item.[7] You can access the context item using the `Sitecore.Context.Item` property. For example, to access the context item:

```
        Sitecore.Data.Items.Item contextItem = Sitecore.Context.Item.
```

---

[4] For more information about the members of the `Sitecore.ItemIDs` class, see the Sitecore API documentation at http://sdn.sitecore.net/Reference/Sitecore%206.aspx.
[5] For more information about the members of the `Sitecore.TemplateIDs` class, see the Sitecore API documentation at http://sdn.sitecore.net/Reference/Sitecore%206.aspx.
[6] For more information about the standard template, see the *Data Definition Reference* manual at http://sdn.sitecore.net/Reference/Sitecore%206/Data%20Definition%20Reference.aspx.
[7] For more information about the context item, see the *Presentation Component Reference* manual at http://sdn.sitecore.net/Reference/Sitecore%206/Presentation%20Component%20Reference.aspx.

3.1.4 How to Resolve Item Access Rights

Sitecore APIs operate in the context of a specific user with particular security access rights. Sitecore APIs may return Null or throw exceptions if the context user does not have a required access right. You can use a security user switcher or security disabler to work around access right limitations.[8] For examples using a security disabler, see the section *How to Place an Item in Editing Mode*.

Note
We recommend that you provide the context user with the appropriate access rights instead of using a security user switcher or a security disabler.

The code examples in this document do not use a security user switcher or security disabler. This implies that the context user has the required access rights in order for the logic to succeed.

3.1.5 How to Place an Item in Editing Mode

Sitecore APIs that update items may throw exceptions if the item is not in editing mode. You can place an item in editing mode using methods of the `Sitecore.Data.Items.Item.Editing` property, or by using the `Sitecore.Data.Items.EditContext` class.

For example, the following code places the `/Sitecore/Content/Home` item in the Master database in editing mode within a security disabler using methods in the `Sitecore.Data.Items.Item.Editing` class:

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

//TODO: eliminate SecurityDisabler if possible

using (new Sitecore.SecurityModel.SecurityDisabler())
{
  home.Editing.BeginEdit();

  try
  {
    //TODO: update home

    home.Editing.EndEdit();
  }
  catch (Exception ex)
  {
    home.Editing.CancelEdit();
  }
}
```

If you do call the `Sitecore.Data.Items.Item.Editing.CancelEdit()` method or do not call the `Sitecore.Data.Items.Item.Editing.EndEdit()` method, Sitecore does not commit the changes.

Important
Developers should use `try`/`catch` blocks as shown in this example. For brevity, code examples in this document do not always include `try`/`catch` blocks.

Alternatively, you can use the `Sitecore.Data.Items.EditContext` class with a C# `using` statement to place an item in editing mode. The closure of the `using` statement invokes the `Sitecore.Data.Items.EditContext.Dispose()` method, which commits any changes made within that segment of code.

For example, the following code places the `/Sitecore/Content/Home` item in the Master database in editing mode using a `Sitecore.Data.Items.EditContext`:

[8] For more information about security user switchers and security disablers, see the *Security API Cookbook* at http://sdn.sitecore.net/Reference/Sitecore%206/Security%20API%20Cookbook.aspx.

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

        //TODO: eliminate SecurityDisabler if possible

        using (new Sitecore.SecurityModel.SecurityDisabler())
        {
          using (new Sitecore.Data.Items.EditContext(home))
          {
            //TODO: process home
          }
        }
```

Note
If you use the `Sitecore.Data.Items.EditContext` class, you cannot explicitly rollback changes. If code within the `using` statement throws an exception, the closing of the using statement automatically commits any changes made before the exception.

3.2 How to Access Items Related to Another Item

This section provides information about APIs you can use to access items related to another item.

3.2.1 How to Access the Children of an Item

You can use the `Sitecore.Data.Items.Children` property to access the children of an item. For example, to access the children of 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");

foreach(Sitecore.Data.Items.Item child in home.Children)
{
  //TODO: process child
}
```

The system creates a new `Sitecore.Collections.Childlist` object each time you access the `Sitecore.Data.Items.Item.Children` property. The implementation of the C# `foreach` statement accesses the property only once. If you access the children of an item using a C# `for` statement rather than a C# `foreach` statement, use a variable to contain the original value of the `Sitecore.Data.Items.Item.Children` property to avoid repeatedly redefining the collection. For example, to access the children of the `/Sitecore/Content/Home` item in the Master database using a C# `for` statement:

```
Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
Sitecore.Data.Items.Item home = master.GetItem("/sitecore/content/home");
Sitecore.Collections.ChildList children = home.Children;

for(int childIndex=0; childIndex<children.Count ; childIndex++)
{
  //TODO: process children[childIndex]
}
```

3.2.2 How to Access a Branch of Items

You can access all of the items in a branch using a recursive method and the `Sitecore.Data.Items.Item.Children` property. For example, to process the `/Sitecore/Content/Home` item in the Master database, and each descendant of that item:

```
Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
Sitecore.Data.Items.Item home = master.GetItem("/sitecore/content/home");
ProcessRecursively(home);
...
private void ProcessRecursively(Sitecore.Data.Items.Item item)
{
  //TODO: process item

  foreach(Sitecore.Data.Items.Item child in item.Children)
  {
    ProcessRecursively(child);
  }
}
```

Warning
If the recursive method passes its argument to itself, then that method implements an infinite loop.

To avoid processing the root item in the branch, move the processing logic within the loop into the recursive method that iterates the children, and in that logic, process the child item instead of the item passed to the recursive method. For example, to process only the descendants of 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");
```

```
        ProcessRecursively(home);
        ...
        private void ProcessRecursively(Sitecore.Data.Items.Item item)
        {
          foreach(Sitecore.Data.Items.Item child in item.Children)
          {
            //TODO: process child

            ProcessRecursively(child);
          }
        }
```

Note
You can also use Sitecore query to access an entire branch using either the `descendant` or `descendant-or-self` axis. For more information about Sitecore query, see the section *How to Access Items Using Sitecore Query*.

3.2.3 How to Access the Parent of an Item

You can use the `Sitecore.Data.Items.Item.Parent` property to access the parent of an item. For example, to access the parent item (`/Sitecore/Content`) of 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.Item parent = home.Parent;
```

3.2.4 How to Access the Ancestors of an Item

You can use the `Sitecore.Data.Items.Item.Parent` property in a recursive method to access the ancestors of an item. For example, to access the ancestors of 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");
        ProcessRecursively(home.Parent);
        ...
        private void ProcessRecursively(Sitecore.Data.Items.Item item)
        {
          //TODO: process item

          if (item.Parent != null )
          {
            ProcessRecursively(item.Parent);
          }
        }
```

Warning
If the recursive method passes its argument to itself, then that method implements an infinite loop.

Alternatively, you can use the `Sitecore.Data.Items.Item.Axes.GetAncestors()` method to access the ancestors of an item. For example, to axes the ancestors of the context item:

```
        Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
        Sitecore.Data.Items.Item home = master.GetItem("/sitecore/content/home");
        Sitecore.Data.Items.Item[] ancestors = home.Axes.GetAncestors();

        foreach(Sitecore.Data.Items.Item ancestor in ancestors)
        {
          //TODO: process ancestor
        }
```

An important difference between the techniques is that the `Sitecore.Data.Items.Item.Axes.GetAncestors()` method accesses items in document order from the root down, while the recursive method using `Sitecore.Data.Items.Item.Parent` accesses items in the reverse document order from the branch or leaf item up.

Note
You can also use Sitecore query to access the ancestors of an item `ancestor` or `ancestor-or-self` axis. For more information about Sitecore query, see the section *How to Access Items Using Sitecore Query*.

3.3 How to Access Items Using Sitecore Query

You can use the `Sitecore.Data.Database.SelectItems()` method to retrieve items in a database that match a Sitecore query.

It is important to remember that:

- Sitecore query is not always the most efficient way to locate items in repository with a large volume of data. Consider using a search index or another solution where the system must frequently match items in a large branch.
- Sitecore query syntax is not the same as XPath syntax.
- Do not assume that Sitecore query returns items in document order or reverse document order.
- Always check for Null before accessing Sitecore query results.

Note
The `value` attribute of the `/configuration/sitecore/settings/setting` element in `web.config` with `name` `Query.MaxItems` specifies the maximum number of items returned by Sitecore query.

For example, to access all items based on the `Common/Folder` data template in the Master database:

```
        Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
        string query = String.Format("//*[@@templateid='{0}']",
          Sitecore.TemplateIDs.Folder);
        Sitecore.Data.Items.Item[] queried = master.SelectItems(query);

        if (queried!= null)
        {
          foreach(Sitecore.Data.Items.Item item in queried)
          {
            //TODO: process item
          }
        }
```

You can use the `Sitecore.Data.Items.Item.Axes.SelectItems()` method to access items matching a Sitecore query relative to another item.[9] For example, to access the items on the `ancestor-or-self` axis of 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.Item[] queried = home.Axes.SelectItems("ancestor-or-self::*");

        if (queried!=null)
        {
          foreach(Sitecore.Data.Items.Item item in queried)
          {
            //TODO: process item
          }
        }
```

[9] For more information about Sitecore query, see http://sdn.sitecore.net/Reference/Using%20Sitecore%20Query.aspx.

3.4 How to Access Media Items

You can use the Sitecore.Data.Items.MediaItem class to access media items. Pass a Sitecore.Data.Items.Item representing the media item as the first parameter to the constructor for Sitecore.Data.Items.MediaItem. For example, to access the /Sitecore/Media Library/Files/Sample media item in the Master database:

```
        Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
        Sitecore.Data.Items.Item item = master.GetItem("/sitecore/media
library/files/sample");

        if (item==null)
        {
          //TODO: handle case that item does not exist
        }
        else
        {
          Sitecore.Data.Items.MediaItem media = new Sitecore.Data.Items.MediaItem(item);

          //TODO: process media
        }
```

3.4.1 How to Import Media Files in a .ZIP File

To import a .zip file containing images, use the Sitecore.Resources.Media.MediaUploader class. To unpack the file that you want to upload, set the Unpack property to the *true* value. For an example of its use please see the Sitecore.Shell.Controls.RichTextEditor.DBContentProvider.StoreFile method.

3.5 How to Access Alternate Languages of an Item

Each item can contain multiple languages. You can use the `Sitecore.Globalization.Language` class to specify a language when accessing an item using the `Sitecore.Data.Database.GetItem()` method. For example, to access the current version of the `/Sitecore/Content/Home` item in the default `en` language:

```
        Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
        Sitecore.Globalization.Language language = 
Sitecore.Globalization.Language.Parse("en");
        Sitecore.Data.Items.Item home = master.GetItem("/sitecore/content/home",language);
```

Note
If you do not specify a language, Sitecore uses the context language by default. Sitecore user interface components specify the content language.

You can use the `Sitecore.Data.Items.Item.Versions.Count` property to determine if any versions exist for a language. For example, to access the current version in each language of 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");

        foreach (Sitecore.Globalization.Language language in home.Languages)
        {
          Sitecore.Data.Items.Item langItem = home.Database.GetItem(home.ID, language);

          if (langItem.Versions.Count > 0)
          {
            //TODO: process langItem
          }
          else
          {
            //TODO: handle case that version data exists for language
          }
        }
```

3.6 How to Access Alternate Versions of an Item

Each item can contain multiple languages. Each language can contain multiple versions. You can use the `Sitecore.Data.Version` class to specify a version when accessing an item using the `Sitecore.Data.Database.GetItem()` method. For example, to access the first version of the `/Sitecore/Content/Home` item in the Master database in the default `en` language:

```
        Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
        Sitecore.Globalization.Language language =
Sitecore.Globalization.Language.Parse("en");
        Sitecore.Data.Items.Item home = master.GetItem("/sitecore/content/home",
          language, Sitecore.Data.Version.Latest);
```

To access the second version of the `/Sitecore/Content/Home` item in the Master database in the default `en` language:

```
        Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
        Sitecore.Globalization.Language language =
Sitecore.Globalization.Language.Parse("en");
        Sitecore.Data.Version second = new Sitecore.Data.Version(2);
        Sitecore.Data.Items.Item home = master.GetItem("/sitecore/content/home",
          language,second);
```

Note
If you do not specify a version, Sitecore uses the current version by default.

You can access the identifier of a version using the `Sitecore.Data.Items.Version.Number` property. For example, to access the version number of 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");
        int verNumber = home.Version.Number;
```

You can use the `Sitecore.Data.Items.Item.Versions.Count` property to determine if any versions of an item exist for a language. You can use the `Sitecore.Data.Items.Item.Versions.GetVersions()` method to access all versions of an item in a language. For example, to access all versions in each language of 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");

        foreach (Sitecore.Globalization.Language language in home.Languages)
        {
          Sitecore.Data.Items.Item langItem = master.GetItem(home.ID, language);

          if (langItem.Versions.Count > 0)
          {
            foreach (Sitecore.Data.Items.Item verItem in langItem.Versions.GetVersions())
            {
              //TODO: process verItem
            }
          }
          else
          {
              //TODO: handle case that no versions exist in language
          }
        }
```

3.7 How to Create an Item

You can create use the `Sitecore.Data.Items.Item.Add()` method to create an item. The parent item and the data template for the new item must exist before you create the item. For example, to create the `/Sitecore/Content/Home/MyItem` item using the `Sample/Sample Item` data template 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 sample = master.Templates["sample/sample item"];
        Sitecore.Data.Items.Item myItem = home.Add("MyItem", sample);
```

3.7.1 How to Create a Version of an Item in a Language

You can create a version of an item in a language using the `Sitecore.Data.Items.Versions.Add()` method. For example, to add the first version to the `/Sitecore/Content/Home` item in the Master database for each language for which version data does not already exist:

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

        foreach (Sitecore.Globalization.Language language in home.Languages)
        {
          Sitecore.Data.Items.Item langItem = home.Database.GetItem(home.ID, language);

          if (langItem.Versions.Count < 1)
          {
            langItem = langItem.Versions.AddVersion();
          }
        }
```

3.7.2 How to Create a Media Library Item

You can create media library items by creating files to a file system monitored by Sitecore, or by invoking APIs.

How to Create Media Items Using the File System

You can create media library items by copying or moving files into a Sitecore file system, typically the `/upload` directory. If you create directories and files in the directory specified by the value attribute of the `/configuration/sitecore/sc.variable` element in `web.config` with `name mediaFolder`, Sitecore will create corresponding media folders and media items under `/Sitecore/Media Library` in the Master database.

Important
Sitecore only creates media items if ASP.NET raises file system events. Before creating the media files in the Sitecore file system, ensure the ASP.NET process is active by requesting a resource processed by ASP.NET, such as the home page.

Warning
Excessive file system activity can overwhelm the ASP.NET worker process. You can monitor the media import process in the Sitecore log file to determine if it completes. If importing media fails, try importing a smaller batch.

Tip
You can delete the files after Sitecore creates media library items from them.

How to Create Media Items Using APIs

You can use the `Sitecore.Resources.Media.MediaCreator` and `Sitecore.Resources.Media.MediaCreatorOptions` classes to create media items from files. For example, to create the media item `/Sitecore/Media Library/Images/Sample` in the Master database from the file `C:\temp\sample.jpg`:

```
        Sitecore.Resources.Media.MediaCreatorOptions options =
          new Sitecore.Resources.Media.MediaCreatorOptions();
        options.Database = Sitecore.Configuration.Factory.GetDatabase("master");
        options.Language = Sitecore.Globalization.Language.Parse(
          Sitecore.Configuration.Settings.DefaultLanguage);
        options.Versioned =
Sitecore.Configuration.Settings.Media.UploadAsVersionableByDefault;
        options.Destination = "/sitecore/media library/images/Sample";
        options.FileBased = Sitecore.Configuration.Settings.Media.UploadAsFiles;
        Sitecore.Resources.Media.MediaCreator creator =
          new Sitecore.Resources.Media.MediaCreator();
        Sitecore.Data.Items.MediaItem sample =
          creator.CreateFromFile(@"C:\temp\sample.jpg",options);
```

3.8 How to Access the Icon for an Item

You can use the `Sitecore.Data.Items.Item.Appearance.Icon` property to access the icon for an item. If the icon contains a themed image, you can use the `Sitecore.Resources.Images.GetThemedImageSource()` method to convert the relative path to a full path. For example, to determine the full path of the icon for 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");
        string icon = home.Appearance.Icon;

        if (icon.StartsWith("~"))
        {
          icon = Sitecore.StringUtil.EnsurePrefix('/', icon);
        }
        else if (!(icon.StartsWith("/" ) && icon.Contains(":")))
        {
          icon = Sitecore.Resources.Images.GetThemedImageSource(icon);
        }
```

You can use the `Sitecore.Data.Items.Item.Appearance.Icon` property to set the icon for an item. For example, to set the icon for the `/Sitecore/Content/Home` item in the Master database to the themed image `network/16x16/home.png`:

```
        Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
        Sitecore.Data.Items.Item home = master.GetItem("/sitecore/content/home");
        home.Editing.BeginEdit();
        home.Appearance.Icon = "network/16x16/home.png";
        home.Editing.EndEdit();
```

3.9 Item Operations: Rename, Move, Copy, and Delete

This section provides information about APIS to invoke operations to rename, move, copy, and delete items.

3.9.1 How to Rename an Item

You can use the `Sitecore.Data.Items.Item.Name` property to rename an item. For example, to rename the `/Sitecore/Content/Home/Sample` item in the Master database to `/Sitecore/Content/Home/Changed`:

```
Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
Sitecore.Data.Items.Item sample = master.GetItem("/sitecore/content/home/sample");
sample.Editing.BeginEdit();
sample.Name = "Changed";
sample.Editing.EndEdit();
```

3.9.2 How to Move an Item

You can use the `Sitecore.Data.Items.Item.MoveTo()` method to move an item or branch. For example, to move the `/Sitecore/Content/Home/Sample` item in the Master database to `/Sitecore/Content`:

```
Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
Sitecore.Data.Items.Item sample = master.GetItem("/sitecore/content/home/sample");
Sitecore.Data.Items.Item content = master.GetItem(Sitecore.ItemIDs.ContentRoot);
sample.MoveTo(content);
```

3.9.3 How to Copy an Item and Its Descendants

You can use the `Sitecore.Data.Items.Item.CopyTo()` method to copy an item and its descendants. For example, to copy the `/Sitecore/Content/Home` item and any descendants in the Master database to create `/Sitecore/Content/Sibling`:

```
Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
Sitecore.Data.Items.Item home = master.GetItem("/sitecore/content/home");
Sitecore.Data.Items.Item content = master.GetItem(Sitecore.ItemIDs.ContentRoot);
home.CopyTo(content,"Sibling");
```

3.9.4 How to Delete an Item and Its Descendants

You can use the `Sitecore.Data.Items.Item.Delete()` method to delete an item and its descendants. For example, to delete the `/Sitecore/Content/Home/Sample` item in the Master database:

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

Note
The `Sitecore.Data.Items.Item.Delete()` method deletes the item and all of its descendants.

How to Delete the Descendants of an Item

You can use the `Sitecore.Data.Items.Item.DeleteChildren()` method to delete the descendants of an item. For example, to delete the descendants of the `/Sitecore/Content/Home` item in the Master database:

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

```
        Sitecore.Data.Items.Item sample = master.GetItem("/sitecore/content/home/sample");
        sample.DeleteChildren();
```

3.10 How to Create a Proxy Item

You can create a proxy definition item using the `System/Alias` data template.[10] For example, to create the proxy definition item `/Sitecore/System/Proxies/MyProxy` to proxy `/Sitecore/Content/Home/Sample` and its descendants as `/Sitecore/Layout/Sample` in the Master database:

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

        if (master.ProxiesEnabled)
        {
          Sitecore.Data.Items.Item proxies = master.GetItem("/sitecore/system/proxies");
          Sitecore.Data.Items.TemplateItem proxy =
master.Templates[Sitecore.TemplateIDs.Proxy];
          Sitecore.Data.Items.Item proxyDef = proxies.Add("MyProxy", proxy);
          Sitecore.Data.Items.Item sample = master.GetItem("/sitecore/content/home/sample");
          proxyDef.Editing.BeginEdit();
          Sitecore.Data.Items.Item parent = master.GetItem(Sitecore.ItemIDs.LayoutRoot);
          proxyDef.Fields[Sitecore.FieldIDs.ProxyTargetItem].Value = parent.ID.ToString();
          proxyDef.Fields[Sitecore.FieldIDs.ProxySourceItem].Value = sample.ID.ToString();
          proxyDef.Fields[Sitecore.FieldIDs.ProxyInsertionType].Value = "Entire sub-tree";
          proxyDef.Editing.EndEdit();
        }
        else
        {
          //TODO: handle case that proxies are disabled in the database
        }
```

To proxy an individual item, set the value of the `Sitecore.FieldIDs.ProxyInsertionType` field in the proxy definition item to `Root item only`. To set proxy items from a different database, set the value of the `Sitecore.FieldIDs.ProxySourceDatabase` field in the proxy definition item to the name of the source database.

Note
To enable proxies in a database, set the value of the `<proxiesEnabled>` element to `true` in the `/configuration/sitecore/databases/database` element in `web.config` with the appropriate `id`.

[10] For more information about proxies, see the *Content Reference* manual at http://sdn.sitecore.net/Reference/Sitecore%206/Content%20Reference.aspx and the *Data Definition Cookbook* at http://sdn.sitecore.net/Reference/Sitecore%206/Data%20Definition%20Cookbook.aspx.

3.11 How to Create an Alias

You can create an alias definition item using the `System/Alias` data template.[11] For example, to create the alias definition item `/Sitecore/System/Aliases/MyAlias` to cause the URL /MyAlias.aspx to activate the `/Sitecore/Content/Home/Sample` item:

```
        Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
        Sitecore.Data.Items.TemplateItem alias = master.Templates[Sitecore.TemplateIDs.Alias];
        Sitecore.Data.Items.Item aliases = master.GetItem("/sitecore/system/aliases");
        Sitecore.Data.Items.Item myAlias = aliases.Add("MyAlias", alias);
        Sitecore.Data.Items.Item sample = master.GetItem("/sitecore/content/home/sample");
        myAlias.Editing.BeginEdit();
        Sitecore.Data.Fields.LinkField linkField = myAlias.Fields["linked item"];
        linkField.LinkType = "internal";
        Sitecore.Links.UrlOptions urlOptions =
Sitecore.Links.LinkManager.GetDefaultUrlOptions();
        urlOptions.AlwaysIncludeServerUrl = false;
        linkField.Url = Sitecore.Links.LinkManager.GetItemUrl(sample,urlOptions);
        linkField.TargetID = sample.ID;
        myAlias.Appearance.Icon = sample.Appearance.Icon;
        myAlias.Editing.EndEdit();
```

[11] For more information about aliases, see the *Content Reference* manual at http://sdn.sitecore.net/Reference/Sitecore%206/Content%20Reference.aspx and the *Data Definition Cookbook* at http://sdn.sitecore.net/Reference/Sitecore%206/Data%20Definition%20Cookbook.aspx.

Chapter 4

Working with Fields

This chapter provides information about APIs to access and update various types of data template field values.

This chapter contains the following sections:

- How to Access Fields
- How to Access the Standard Value of a Field
- How to Determine if a Field Contains Its Standard Value
- How to Reset a Field to Its Standard Value

4.1 How to Access Fields

You can use the APIs described in this section to read and write field values. Excluding the Attachment system field type used to store binary data for media items, Sitecore stores all field values as text. You can access any field as a simple text value, or use specialized classes in the `Sitecore.Data.Fields` namespace.

Some field types consist of a simple text value. Checkbox field stores the one character (“`1`”) when selected. Rich Text Editor (RTE) fields contain XHTML. Numerous field types contain the ID of a single item or multiple IDs separated by pipe characters (“`|`”). Other field types contain XML or data in proprietary formats.

Tip
To determine the string format of any field type, open in the Content Editor, select an item containing the field, and then view raw field values.[12]

You can access the string value of any field using the collection exposed by the `Sitecore.Data.Items.Item` class. For example, to access the value of the `Title` field in 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");
string title = home["title"];
```

Important
If the specified field does not exist, the collection exposed by `Sitecore.Data.Items.Item` returns an empty string, never Null.

You can update the value of any field using the collection exposed by the `Sitecore.Data.Items.Item` class. For example, to update the value of the `Title` field in 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");
home.Editing.BeginEdit();
home["title"] = "//TODO: replace with appropriate value";
home.Editing.EndEdit();
```

You can clear any field by setting its value to an empty string using the collection exposed by the `Sitecore.Data.Items.Item` class. For example, to clear the `Title` field in 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");
home.Editing.BeginEdit();
home["title"] = String.Empty;
home.Editing.EndEdit()
```

Important
Clearing the value of a field does not reset that field to its standard value. For instructions to reset a field to its standard value, see the section How to Reset a Field to Its Standard Value.

You cannot set a field value to Null.

[12] For instructions to view raw field values, see the *Client Configuration Cookbook* at http://sdn.sitecore.net/Reference/Sitecore%206.aspx.

Alternatively, you can access any field as an instance of the `Sitecore.Data.Fields.Field` class using the `Sitecore.Data.Items.Item.Fields` property. For example, to access the `Title` field in 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.Fields.Field titleField = home.Fields["title"];

if(titleField!=null)
{
  string title = titleField.Value;
}
```

Important
If you access to the field by name (`item.Fields[System.String]`) and it does not exist in the item, then the `Sitecore.Data.Items.Item.Fields` collection returns `Null`.

If you access to the field by ID (`item.Fields[Sitecore.Data.ID]`) and it does not exist in the item, then the `Sitecore.Data.Items.Item.Fields` collection returns a “new” `Sitecore.Data.Fields.Field` object which does not belong to the item template.

You can use the `Sitecore.Data.Fields.Field.Value` property to update the value of a field. For example, to update the value of the `Title` field in 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.Fields.Field titleField = home.Fields["title"];

if(titleField!=null)
{
  home.Editing.BeginEdit();
  titleField.Value = "//TODO: replace with appropriate value";
  home.Editing.EndEdit();
}
```

Important
Sitecore retrieves data only when necessary. To ensure that Sitecore retrieves all field values for an item, invoke the `Sitecore.Data.Items.Item.ReadAll()` method. For example, to iterate the fields of the context item:

```
Sitecore.Data.Items.Item item = Sitecore.Context.Item;
item.Fields.ReadAll();

foreach (Sitecore.Data.Fields.Field field in item.Fields)
{
  // TODO: process field
}
```

4.1.1 How to Access System Fields

You can use the `Sitecore.FieldIDs` class to access system fields.[13] For example, to access the archive date field of the `/Sitecore/Content/Home/Sample` item in the Master database:

```
Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
Sitecore.Data.Items.Item sample = master.GetItem("/sitecore/content/home/sample");
Sitecore.Data.Fields.DateField archiveField =
  sample.Fields[Sitecore.FieldIDs.ArchiveDate];
DateTime archiveWhen = archiveField.DateTime;
```

[13] For more information about the members of the `Sitecore.FieldIDs` class, see the Sitecore API documentation at http://sdn.sitecore.net/Reference/Sitecore%206.aspx.

4.1.2 How to Access a Field Using the FieldRender Web Control

In presentation components, whenever possible, use the FieldRenderer Web control to output field values.[14] For example, to output the `Title` field in the context item from a Web control:

```
namespace Namespace.Web.UI.WebControls
{
  public class WebControlName : Sitecore.Web.UI.WebControl
  {
    protected override void DoRender(System.Web.UI.HtmlTextWriter output)
    {
      string html = Sitecore.Web.UI.WebControls.FieldRenderer.Render(
        Sitecore.Context.Item, "title");
      output.Write(html);
    }
  }
}
```

Alternatively, you can add the FieldRenderer Web control to a layout or sublayout. For example:

```
<sc:FieldRenderer runat="server" ID="fieldControl" />
```

In the code behind for the layout or sublayout, set properties of the control. For example, to render the `Title` field in the context item:

```
namespace Namespace.Web.UI
{
  public partial class SublayoutName : System.Web.UI.UserControl
  {
    protected void Page_Load(object sender, EventArgs e)
    {
      fieldControl.DataSource = Sitecore.Context.Item.Paths.FullPath;
      fieldControl.FieldName = "title";
    }
  }
}
```

Important
The FieldRenderer Web control does not support all field types. You can use the FieldRenderer Web control with Date, Datetime, Image, Integer, Multi-Line Text, Number, Rich Text, and Single-Line Text fields.

4.1.3 How to Access Checkbox Fields

You can use the `Sitecore.Data.Fields.CheckboxField` class to access data template fields of type Checkbox. To determine if the user has selected a Checkbox, access the `Sitecore.Data.Fields.CheckboxField.Checked` property. For example, to determine if the Checkbox field named `CheckboxField` is selected in 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.Fields.CheckboxField checkboxField = home.Fields["checkboxfield"];

if (checkboxField != null && checkboxField.Checked)
{
  //TODO: handle case that checkbox exists and is selected
}
```

To select a Checkbox field, set the `Sitecore.Data.Fields.CheckboxField.Checked` property. For example, to ensure the Checkbox field named `CheckboxField` is selected in the `/Sitecore/Content/Home` item in the Master database:

[14] For more information about the FieldRenderer Web control, see the *Presentation Components Reference* manual and the *Presentation Components Cookbook* at http://sdn.sitecore.net/Reference/Sitecore%206.aspx.

```
        Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
        Sitecore.Data.Items.Item home = master.GetItem("/sitecore/content/home");
        Sitecore.Data.Fields.CheckboxField checkboxField = home.Fields["checkboxfield"];

        if (checkboxField != null && ! checkboxField.Checked )
        {
          home.Editing.BeginEdit();
          checkboxField.Checked = true;
          home.Editing.EndEdit();
        }
```

4.1.4 How to Access Date and Datetime Fields

You can use the `Sitecore.Data.Fields.DateField` class to access data template fields of type Date and Datetime. The `Sitecore.Data.Fields.DateField.Value` property contains the date and time as a string in the ISO format used by Sitecore (`yyyyMMddTHHmmss`).[15] You can convert a value in the ISO format to a `System.DateTime` structure using the `Sitecore.DateUtil.IsoDateToDateTime()` method. For example, to access the Date or Datetime field named `DateTimeField` in 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.Fields.DateField dateTimeField = home.Fields["datetimefield"];
        string dateTimeString = dateTimeField.Value;
        DateTime dateTimeStruct = Sitecore.DateUtil.IsoDateToDateTime(dateTimeString);
```

Alternatively, you can access the `Sitecore.Data.Fields.DateField.DateTime` property, which contains a `System.DateTime` structure representing the same value. For example, to access the Date or Datetime field named `DateTimeField` in the `/Sitecore/Content/Home` item in the Master database as a `System.DateTime` structure:

```
        Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
        Sitecore.Data.Items.Item home = master.GetItem("/sitecore/content/home");
        Sitecore.Data.Fields.DateField dateTimeField = home.Fields["datefield"];
        DateTime dateTimeStruct = dateTimeField.DateTime;
```

You can update the value of a Date or Datetime field by updating the `Sitecore.Data.Fields.DateField.Value` property to a string in the ISO format. You can use the `Sitecore.DateUtil.ToIsoDate()` method to convert a `System.DateTime` structure to the ISO format. For example, to update the value of the Datetime field named `DateTimeField` in the `/Sitecore/Content/Home` item in the Master database to the current system date:

```
        Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
        Sitecore.Data.Items.Item home = master.GetItem("/sitecore/content/home");
        Sitecore.Data.Fields.DateField dateTimeField = home.Fields["datetimefield"];

        if (dateTimeField != null)
        {
          home.Editing.BeginEdit();
          dateTimeField.Value = Sitecore.DateUtil.ToIsoDate(DateTime.Now);
          home.Editing.EndEdit();
        }
```

Note
If the user has not specified a value for a field of type Date or Datetime, then the `Sitecore.Data.Fields.DateField.Value` property contains an empty string, and the `Sitecore.Data.Fields.DateField.DateTime` property contains `System.DateTime.MinValue`.

[15] For more information about .NET date format patterns, see http://msdn.microsoft.com/en-us/library/73ctwf33.aspx.

Note
By default, a field of type Date contains a time value of midnight.

4.1.5 How to Access File Fields

You can use the `Sitecore.Data.Fields.FileField` class to access data template fields of type File. You can use the `Sitecore.Data.Fields.FileField.MediaItem` property to access the media item selected in the field. For example, to access the media item referenced by the File field named `FileField` in 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.Fields.FileField fileField = home.Fields["filefield"];
        Sitecore.Data.Items.Item file = fileField.MediaItem;

        if (fileField==null)
        {
          //TODO: handle case that field does not exist
        }
        else if (file==null)
        {
          //TODO: handle case that user has not selected a file
        }
        else
        {
          Sitecore.Data.Items.MediaItem media = new Sitecore.Data.Items.MediaItem(file);
          //TODO: process media
        }
```

You can update the value of a File field by updating the `Sitecore.Data.Fields.FileField.MediaID` and `Sitecore.Data.Fields.FileField.Src` attributes. For example, to ensure the File field named `FileField` in the `/Sitecore/Content/Home` item in the Master database specifies the `/Sitecore/Media Library/Files/Sample` media item:

```
        Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
        Sitecore.Data.Items.Item home = master.GetItem("/sitecore/content/home");
        Sitecore.Data.Fields.FileField fileField = home.Fields["filefield"];
        Sitecore.Data.Items.Item sample = master.GetItem("/sitecore/media
library/files/sample");

        if (fileField == null )
        {
          //TODO: handle case that field does not exist
        }
        else if (sample == null)
        {
          //TODO: handle case that sample does not exist
        }
        else if (fileField.MediaID != sample.ID)
        {
          home.Editing.BeginEdit();
          fileField.MediaID = sample.ID;
          fileField.Src = Sitecore.Resources.Media.MediaManager.GetMediaUrl(sample);
          home.Editing.EndEdit();
        }
```

4.1.6 How to Access General Link Fields

You can use the `Sitecore.Data.Fields.LinkField` class to access data template fields of type General Link. Depending on the type of link in the field, you can use the following properties of the `Sitecore.Data.Fields.LinkField` class:

Property	Value
`Anchor`	The `name` attribute of the HTML `<a>` element, without the leading hash character (“`#`”).

Property	Value
`Class`	The `class` attribute of the HTML `<a>` element.
`IsInternal`	True for an internal link, False for media and other types of links.
`IsMediaLink`	True for media links.
`LinkType`	A token identifying the type of link (`internal`, `media`, `external`, `mailto`, `anchor`, or `javascript`).
`QueryString`	Query string parameters to add to the URL.
`Target`	The `target` attribute of the HTML `<a>` element.
`TargetID`	The ID of item specified d by internal or media link.
`TargetItem`	The `Sitecore.Data.Items.Item` specified by an internal or media link.
`Text`	The text content of the HTML `<a>` element.
`Title`	The `title` attribute of the HTML `<a>` element.
`Url`	The URL of the link, except for media items, for which the `Url` property contains the path to the media item relative to `/Sitecore/Media Library`.

To determine the URL in the General Link field named `GeneralLinkField` in 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.Fields.LinkField linkField = home.Fields["generallinkfield"];
string url = String.Empty;

switch(linkField.LinkType)
{
  case "internal":
  case "external":
  case "mailto":
  case "anchor":
  case "javascript":
    url = linkField.Url;
    break;
  case "media":
    Sitecore.Data.Items.MediaItem media =
      new Sitecore.Data.Items.MediaItem(linkField.TargetItem);
    url = Sitecore.StringUtil.EnsurePrefix('/',
      Sitecore.Resources.Media.MediaManager.GetMediaUrl(media));
    break;
  case "":
    break;
  default:
    string message = String.Format("{0} : Unknown link type {1} in {2}",
      this.GetType(), linkField.LinkType, home.Paths.FullPath);
    Sitecore.Diagnostics.Log.Error(message,this);
    break;
}
```

You can use the `Sitecore.Data.Fields.LinkField.Clear()` method to remove the data from a field of type General Link. For example, to remove any link from the General Link field named `GeneralLinkField` in 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.Fields.LinkField linkField = home.Fields["generallinkfield"];
home.Editing.BeginEdit();
linkField.Clear();
home.Editing.EndEdit();
```

To update the General Link field named `GeneralLinkField` in the `/Sitecore/Content/Home` item in the Master database to the `/Sitecore/Content/Home/Sample` item:

```
        Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
        Sitecore.Data.Items.Item home = master.GetItem("/sitecore/content/home");
        Sitecore.Data.Fields.LinkField linkField = home.Fields["generallinkfield"];
        Sitecore.Data.Items.Item sample = master.GetItem("/sitecore/content/home/sample");
        home.Editing.BeginEdit();
        linkField.Clear();
        linkField.LinkType = "internal";
        Sitecore.Links.UrlOptions urlOptions =
Sitecore.Links.LinkManager.GetDefaultUrlOptions();
        urlOptions.AlwaysIncludeServerUrl = false;
        linkField.Url = Sitecore.Links.LinkManager.GetItemUrl(sample,urlOptions);
        linkField.TargetID = sample.ID;
        home.Editing.EndEdit();
```

To update the General Link field named `GeneralLinkField` in the `/Sitecore/Content/Home` item in the Master database to the `/Sitecore/Media Library/Files/Sample` media item:

```
        Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
        Sitecore.Data.Items.Item home = master.GetItem("/sitecore/content/home");
        Sitecore.Data.Fields.LinkField linkField = home.Fields["generallinkfield"];
        Sitecore.Data.Items.Item sample = master.GetItem("/sitecore/media
library/files/sample");
        home.Editing.BeginEdit();
        linkField.Clear();
        linkField.LinkType = "media";
        linkField.Url = sample.Paths.MediaPath;
        linkField.TargetID = sample.ID;
        home.Editing.EndEdit();
```

To update the General Link field named `GeneralLinkField` in the `/Sitecore/Content/Home` item in the Master database to the external URL `http://sitecore.net`:

```
        Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
        Sitecore.Data.Items.Item home = master.GetItem("/sitecore/content/home");
        Sitecore.Data.Fields.LinkField linkField = home.Fields["generallinkfield"];
        home.Editing.BeginEdit();
        linkField.Clear();
        linkField.LinkType = "external";
        linkField.Url = "http://sitecore.net";
        home.Editing.EndEdit();
```

To update the General Link field named `GeneralLinkField` in the `/Sitecore/Content/Home` item in the Master database to the anchor `namedAnchor`:

```
        Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
        Sitecore.Data.Items.Item home = master.GetItem("/sitecore/content/home");
        Sitecore.Data.Fields.LinkField linkField = home.Fields["generallinkfield"];
        home.Editing.BeginEdit();
        linkField.Clear();
        linkField.LinkType = "anchor";
        linkField.Url = "namedAnchor";
        home.Editing.EndEdit();
```

To update the General Link field named `GeneralLinkField` in the `/Sitecore/Content/Home` item in the Master database to the email address `email@domain.tld`:

```
        Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
        Sitecore.Data.Items.Item home = master.GetItem("/sitecore/content/home");
        Sitecore.Data.Fields.LinkField linkField = home.Fields["generallinkfield"];
        home.Editing.BeginEdit();
        linkField.Clear();
        linkField.LinkType = "mailto";
        linkField.Url = "mailto:email@domain.tld";
        home.Editing.EndEdit();
```

To update the General Link field named `GeneralLinkField` in the `/Sitecore/Content/Home` item in the Master database to a JavaScript function:

```
        Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
        Sitecore.Data.Items.Item home = master.GetItem("/sitecore/content/home");
        Sitecore.Data.Fields.LinkField linkField = home.Fields["generallinkfield"];
        home.Editing.BeginEdit();
        linkField.Clear();
        linkField.Text = "//TODO: replace with appropriate value";
        linkField.LinkType = "javascript";
        linkField.Url = @"javascript:alert('javascript')";
        home.Editing.EndEdit();
```

4.1.7 How to Access Image Fields

You can use the `Sitecore.Data.Fields.ImageField` class to access data template fields of type Image. You can use the `Sitecore.Data.Fields.ImageField.MediaItem` property to access the media item selected in the field as a `Sitecore.Data.Items.Item`. If the field does not specify an image, then the `Sitecore.Data.Fields.ImageField.MediaItem` property is Null. You can use the `Sitecore.Resources.ImageBuilder` class to construct an HTML `<img>` element. You can use the `Sitecore.Resources.Media.MediaManager.GetMediaUrl()` method to determine the URL of a media item. For example, to construct an HTML `<img>` element based on the value of the Image field named `ImageField` in 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.Fields.ImageField imageField = home.Fields["imagefield"];

        if (imageField!=null && imageField.MediaItem!=null)
        {
          Sitecore.Data.Items.MediaItem image =
            new Sitecore.Data.Items.MediaItem(imageField.MediaItem);
          string src = Sitecore.StringUtil.EnsurePrefix('/',
            Sitecore.Resources.Media.MediaManager.GetMediaUrl(image));
          string imgTag = String.Format(@"<img src=""{0}"" alt=""{1}"" />", src, image.Alt);
        }
```

Important
Use the Sitecore media library for images and other media contributed by business users. Use the file system and a source code management system for images managed by developers.

You can use the `Sitecore.Data.Fields.ImageField.Clear()` method to clear the content of an Image field. For example, to clear the Image field named `ImageField` field in 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.Fields.ImageField imageField = home.Fields["imagefield"];
        home.Editing.BeginEdit();
        imageField.Clear();
        home.Editing.EndEdit();
```

You can use the `Sitecore.Data.Fields.ImageField` class to update an Image field. For example, to update the Image field named `ImageField` in the `/Sitecore/Content/Home` item in the Master database to the `/Sitecore/Media Library/Images/Sample` image:

```
        Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
        Sitecore.Data.Items.Item home = master.GetItem("/sitecore/content/home");
        Sitecore.Data.Items.Item sampleItem =
          master.GetItem("/sitecore/media library/images/sample");
        Sitecore.Data.Items.MediaItem sampleMedia =
          new Sitecore.Data.Items.MediaItem(sampleItem);
        Sitecore.Data.Fields.ImageField imageField = home.Fields["imagefield"];

        if (imageField.MediaID != sampleMedia.ID )
        {
          home.Editing.BeginEdit();
          imageField.Clear();
          imageField.MediaID = sampleMedia.ID;

          if (!String.IsNullOrEmpty(sampleMedia.Alt))
          {
            imageField.Alt = sampleMedia.Alt;
          }
          else
          {
            imageField.Alt = sampleMedia.DisplayName;
          }

          home.Editing.EndEdit();
        }
```

4.1.8 How to Access Droplink, Droptree, and Grouped Droplink Fields

You can access fields types allowing the user to select a single item, including Droplink, Droptree, and Grouped Droplink fields, using the `Sitecore.Data.Fields.ReferenceField` class. The `Sitecore.Data.Fields.ReferenceField.TargetItem` property contains the `Sitecore.Data.Items.Item` specified by the field, or Null. For example, to access the item specified by the Droptree field named `ReferenceField` in 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.Fields.ReferenceField referenceField = item.Fields["referencefield"];

        if (referenceField==null)
        {
          //TODO: handle case that field does not exist
        }
        else if (referenceField.TargetItem==null)
        {
          //TODO: handle case that user has not selected an item
        }
        else
        {
          Sitecore.Data.Items.Item referencedItem = referenceField.TargetItem;
          //TODO: process referencedItem
        }
```

You can set the `Sitecore.Data.Fields.ReferenceField.Value` property to the ID of an item to update a field of one of the supported types. For example, to ensure the Droptree field named `ReferenceField` in the `/Sitecore/Content/Home` item in the Master database specifies the `/Sitecore/Content/Home/Sample` item:

```
        Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
        Sitecore.Data.Items.Item home = master.GetItem("/sitecore/content/home");
        Sitecore.Data.Fields.ReferenceField referenceField = home.Fields["referencefield"];

        if (referenceField==null)
        {
          //TODO: handle case that field does not exist
        }
        else
        {
          Sitecore.Data.Items.Item sample = master.GetItem("/sitecore/content/home/sample");

          if (sample==null)
          {
            //TODO: handle case that sample does not exist
          }
          else if (sample.ID.ToString()!=referenceField.Value)
          {
            home.Editing.BeginEdit();
            referenceField.Value = sample.ID.ToString();
            home.Editing.EndEdit();
          }
        }
```

4.1.9 How to Access Checklist, Multilist, Treelist, and TreelistEx Fields

You can access field types allowing the user to select multiple items, including Checklist, Multilist, Treelist, and TreelistEx, using the `Sitecore.Data.Fields.MultilistField` class. You can use the `Sitecore.Data.Fields.MultilistField.GetItems()` method to access a list of `Sitecore.Data.Item.Item` objects representing the items specified by the field. For example, to iterate over the items specified in the Multilist field named `Multiselect` in 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.Fields.MultilistField multiselectField = home.Fields["multiselect"];

if (multiselectField==null)
{
  //TODO: handle case that field does not exist
}
else
{
  Sitecore.Data.Items.Item[] items= multiselectField.GetItems();

  if (items!=null && items.Length>0)
  {
    for (int i=0; i<items.Length ; i++)
    {
      //process items[i]
    }
  }
}
```

The individual members of the list returned by `Sitecore.Data.Fields.MultilistField.GetItems()` method are never Null. If a user has deleted an item without updating the references to that item, the `Sitecore.Data.Fields.MultilistField.GetItems()` method excludes that item from its results.

Note
You can also use the `Sitecore.Data.Fields.MultilistField` class to access fields of type Droplink, Droptree, and Grouped Droplink. This approach provides you with a single programming model for all field types that store the IDs of one or more Sitecore items, and could reduce the need to update code if you change the type of the field. Because Droplink, Droptree, and Grouped Droplink do not support selection of multiple items, you should not use the `Sitecore.Data.Fields.Multilist` class to update these types of fields.

You can add items to a supported field type using the `Sitecore.Data.Fields.MulitlistField.Add()` method, and remove items using the `Sitecore.Data.Fields.MulitlistField.Remove()` method. For example, to ensure that the TreelistEx `Multiselect` field in the `/Sitecore/Content/Home` item in the Master database specifies the `/Sitecore/Content/Home/Sample1` item, but does not specify not the `/Sitecore/Content/Home/Sample2` item:

```
Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
Sitecore.Data.Items.Item home = master.GetItem("/sitecore/content/home");
Sitecore.Data.Items.Item sample1 = master.GetItem("/sitecore/content/home/sample1");
Sitecore.Data.Items.Item sample2 = master.GetItem("/sitecore/content/home/sample2");
Sitecore.Data.Fields.MultilistField multiselectField = home.Fields["multiselect"];

if(multiselectField.Contains(sample2.ID.ToString())
  || !multiselectField.Contains(sample1.ID.ToString()))
{
  home.Editing.BeginEdit();

  if(!multiselectField.Contains(sample1.ID.ToString()))
  {
    multiselectField.Add(sample1.ID.ToString());
```

```
          }

          if(multiselectField.Contains(sample2.ID.ToString()))
          {
            multiselectField.Remove(sample2.ID.ToString());
          }

          home.Editing.EndEdit();
        }
```

4.1.10 How to Access File Drop Area Fields

You can use the `Sitecore.Data.Fields.FileDropAreaField` class to access the value in a field of type File Drop Area (FDA). The `Sitecore.Data.Fields.FileDropAreaField.GetMediaItems()` method returns the media items associated with the FDA field.

You can implement a Web control based on the following example that generates an unordered list of links to the media items associated with an FDA field.

```
        namespace Sitecore.Sharedsource.Web.UI.WebControls
        {
          using System;

          public class FDALinks : Sitecore.Web.UI.WebControl
          {
            public string FieldName
            {
              get;
              set;
            }

            protected override void DoRender(System.Web.UI.HtmlTextWriter output)
            {
              if (this.FieldName == null
                || Sitecore.Context.Item == null
                || output == null)
              {
                return;
              }

              Sitecore.Data.Fields.FileDropAreaField fdaField =
                Sitecore.Context.Item.Fields[this.FieldName];

              if (fdaField == null)
              {
                return;
              }

              Sitecore.Collections.ItemList mediaItems = fdaField.GetMediaItems();

              if (mediaItems.Count < 1)
              {
                return;
              }

              output.Write("<ul>");

              foreach (Sitecore.Data.Items.Item mediaItem in mediaItems)
              {
                string mediaUrl = Sitecore.StringUtil.EnsurePrefix(
                  '/',
                  Sitecore.Resources.Media.MediaManager.GetMediaUrl(mediaItem));
                string markup = String.Format(
                  @"<li><a href=""{0}"">{1}</a>",
                  mediaUrl,
                  mediaItem.Name);
                output.Write(markup);
              }

              output.Write("</ul>");
            }
```

```
    }
  }
```

4.1.11 How to Access Word Document Fields

You can use the `Sitecore.Data.Fields.WordDocumentField` class to access the value of a Word Document field. You can use the `Sitecore.Data.Fields.WordDocumentField.Html` property to access the HTML representation of the field value. You can use the `Sitecore.Data.Fields.WordDocumentField.PlainText` property to access the plain text representation of the field value.

You can use the `Sitecore.Data.Fields.WordDocumentField.Styles` property to access the Cascading Style Sheet code associated with that HTML. For example, you can implement a Web control based on the following example that outputs the styles and HTML of a Word Document field:

```
namespace Sitecore.Sharedsource.Web.UI.WebControls
{
  using System;
  using System.Text.RegularExpressions;

  public class RenderWordDocumentFieldRenderer : Sitecore.Web.UI.WebControl
  {
    public string FieldName
    {
      get;
      set;
    }

    protected override void DoRender(System.Web.UI.HtmlTextWriter output)
    {
      Sitecore.Data.Items.Item item = this.GetItem();

      if (this.FieldName == null || item == null)
      {
        return;
      }

      Sitecore.Data.Fields.WordDocumentField wordField = item.Fields["WordField"];

      if (wordField == null
        || wordField.BlobId == Sitecore.Data.ID.Null
        || Regex.IsMatch(wordField.PlainText, "^\\s*$"))
      {
        return;
      }

      output.WriteLine(String.Format(
        "<style type=\"text/css\">{0}</style>",
        wordField.Styles));
      output.WriteLine(Sitecore.Web.UI.WebControls.FieldRenderer.Render(
        item,
        wordField.InnerField.Name));
    }
  }
}
```

You can use the `Sitecore.WordOCX.WordOCXUrlManager.GetDownloadLink()` method to retrieve a URL that provides access to the value of a Word Document field as a Word Document (`.docx` file).

For example, to retrieve a URL to acess the Word Document field named WordDocument in the context item as a Word Document file:

```
Sitecore.Data.Items.Item item = Sitecore.Context.Item;
Sitecore.Data.Fields.WordDocumentField wordField = item.Fields["WordDocument"];

if (wordField != null
  && wordField.BlobId != Sitecore.Data.ID.Null
  && !Regex.IsMatch(wordField.PlainText, "^\\s*$"))
{
  Dictionary<string, string> parameters = new Dictionary<string, string>();
```

```
          parameters["db"] = item.Database.Name;
          parameters["blobId"] = wordField.BlobId.ToString();
          Sitecore.WordOCX.WordOCXUrlManager wordManager =
            new Sitecore.WordOCX.WordOCXUrlManager(parameters);
          string url = wordManager.GetDownloadLink();
        }
```

4.2 How to Access the Standard Value of a Field

You can use the `Sitecore.Data.Fields.Field.GetStandardValue()` method to access the standard value of a field.[16] For example, to access the standard value of the Title field in 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.Fields.Field titleField = home.Fields["title"];
        string standardTitle = titleField.GetStandardValue();
```

[16] For more information about field standard values, see the *Data Definition Reference* manual and the *Data Definition Cookbook* at http://sdn.sitecore.net/Reference/Sitecore%206.aspx.

4.3 How to Determine if a Field Contains Its Standard Value

You can determine whether a field contains its standard value using the `Sitecore.Data.Fields.Field.ContainsStandardValue` property. For example, to determine if the `Title` field in the `/Sitecore/Content/Home` item in the Master database contains its standard value:

```
Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
Sitecore.Data.Items.Item home = master.GetItem("/sitecore/content/home");
Sitecore.Data.Fields.Field titleField = home.Fields["title"];

if (titleField.ContainsStandardValue)
{
  //TODO: handle case that field contains standard value
}
```

The `Sitecore.Data.Fields.Field.ContainsStandardValue` property is False if there is no standard value for a field.

Note
A field can contain the same value as its standard value without actually containing that standard value. For example, an item contains its standard value. The user updates that field; the field no longer contains its standard value. The user updates the field again, setting the value to the same value as the standard value for the field, but without resetting the field to its standard value. The field now contains the same value as its standard value, but does not contain its standard value; the `Sitecore.Data.Fields.Field.ContainsStandardValue` property is False despite the fact that the value of the field is equal to its standard value.

4.4 How to Reset a Field to Its Standard Value

You can use the `Sitecore.Data.Fields.Field.Reset()` method to reset a field to its standard value. For example, to reset the value of the `Title` field in the `/Sitecore/Content/Home` item in the Master database to its standard value:

```
Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
Sitecore.Data.Items.Item home = master.GetItem("/sitecore/content/home");
Sitecore.Data.Fields.Field titleField = home.Fields["title"];
home.Editing.BeginEdit();
titleField.Reset();
home.Editing.EndEdit();
```

Updating the value of a `Sitecore.Data.Fields.Field` to an empty string or the standard value of the field does not cause the value of the field to revert to the value defined in the standard values item associated with its data template. Use the `Sitecore.Data.Fields.Field.Reset()` method to reset a field to its standard value.

When you reset a field to its standard value, Sitecore does not expand tokens such as `$name` in the standard value. You can use a master variable replacer to replace tokens.

```
Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
Sitecore.Data.Items.Item home = master.GetItem("/sitecore/content/home");
Sitecore.Data.Fields.Field titleField = home.Fields["title"];
home.Editing.BeginEdit();
titleField.Reset();
home.Editing.EndEdit();
home.Editing.BeginEdit();
Sitecore.Data.MasterVariablesReplacer replacer =
Sitecore.Configuration.Factory.GetMasterVariablesReplacer();
replacer.ReplaceField(home,titleField);
home.Editing.EndEdit();
```

Note
You must commit the reset operation before using the master variable replacer on the field value.

Chapter 5

Working with Dynamic Links

This chapter contains information about configuring and using Sitecore dynamic link management APIs.[17]

This chapter contains the following sections:

[17] For more information about Sitecore dynamic links, see the guide to working with Dynamic Links at http://sdn.sitecore.net/Reference/Sitecore%206/Dynamic%20Links.aspx.

5.1 How to Access the Friendly URL of a Content Item

You can use the `Sitecore.Links.LinkManager.GetItemUrl()` method to access the friendly URL of a content item.[18] For example, to access the friendly URL of 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");
        string url = Sitecore.Links.LinkManager.GetItemUrl(home);
```

[18] For more information about dynamic URLs, see the guide to working with Dynamic Links at http://sdn.sitecore.net/Reference/Sitecore%206/Dynamic%20Links.aspx.

5.2 How to Access the URL of an RSS Feed

The URL of an RSS feed is the default URL of the feed definition item. [19] You can use the same APIs to access the URL of an RSS feed that you use to access the URL of any item. For more information about the APIs that you can use to access the URL of an item, see the previous section*, How to Access the Friendly URL of a Content Item*.

You can use the `Sitecore.Syndication.FeedManager.GetFeedUrl()` method to retrieve an RSS URL. The first parameter to the `Sitecore.Syndication.FeedManager.GetFeedUrl()` is a feed definition item. The second parameter indicates whether to include authentication information in the URL.

Note
Use RSS URL authentication features only for Sitecore client RSS feeds.[20]

For more information about syndication APIs, see *Chapter 6, Syndication APIs*.

[19] For more information about Sitecore RSS features, see the *Presentation Component Cookbook* at http://sdn.sitecore.net/Reference/Sitecore%206/Presentation%20Component%20Cookbook.aspx.
[20] For more information about Sitecore client RSS feeds, see the *Client Configuration Cookbook* at http://sdn.sitecore.net/Reference/Sitecore%206/Client%20Configuration%20Cookbook.aspx.

5.3 How to Access the Friendly URL of a Media Item

You can use the `Sitecore.Resources.Media.MediaManager.GetMediaUrl()` method to access the friendly URL of a media item. For example, to access the friendly URL of the media item `/Sitecore/Media Library/Images/Sample` in the Master database:

```
Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
Sitecore.Data.Items.Item sampleItem = master.GetItem(
  "/sitecore/media library/images/sample");
Sitecore.Data.Items.Item sampleMedia = new Sitecore.Data.Items.MediaItem(sampleItem);
string url = Sitecore.StringUtil.EnsurePrefix('/',
  Sitecore.Resources.Media.MediaManager.GetMediaUrl(sampleMedia));
```

Warning
Sitecore does not automatically include the leading slash character (“`/`”) in media URLs. This results in relative URLs for media items, which IIS resolves to the document root due to the tilde character (“`~`”). In solutions with very deep information architectures, relative media URLs can exceed limits imposed by the Web client or the Web server. Use the `Sitecore.StringUtil.EnsurePrefix()` method as shown in the previous example to ensure media URLs include the leading slash character.

You can use the `Sitecore.Resources.Media.MediaUrlOptions` class to specify media options. For example, to retrieve the URL of the thumbnail of the `/Sitecore/Media Library/Images/Sample` media item in the Master database:

```
Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
Sitecore.Data.Items.Item sampleItem = master.GetItem(
  "/sitecore/media library/images/sample");
Sitecore.Data.Items.MediaItem sampleMedia =
  new Sitecore.Data.Items.MediaItem(sampleItem);
Sitecore.Resources.Media.MediaUrlOptions mediaOptions =
  new Sitecore.Resources.Media.MediaUrlOptions();
mediaOptions.Thumbnail = true;
string url = Sitecore.StringUtil.EnsurePrefix('/',
  Sitecore.Resources.Media.MediaManager.GetMediaUrl(sampleMedia, mediaOptions));
```

5.4 How to Transform Dynamic Links in HTML to Friendly URLs

You can use the FieldRenderer Web control, the `renderField` pipeline, or the `Sitecore.Links.LinkManager.ExpandDynamicLinks()` method to transform dynamic links into friendly URLs.[21]

You can use the `Sitecore.Links.LinkManager.ExpandDynamicLinks()` method to transform dynamic links in Rich Text Editor (RTE) fields, including both content and media links, to friendly URLs. For example, to transform dynamic links in the `Text` field in 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");
        string containsDynamicLinks = home.Fields["text"].Value;
        string containsFriendlyLinks = Sitecore.Links.LinkManager.ExpandDynamicLinks(
          containsDynamicLinks, Sitecore.Configuration.Settings.Rendering.SiteResolving);
        string finalMarkup = 
System.Text.RegularExpressions.Regex.Replace(containsFriendlyLinks,
          "([^/])~/media", "$1/~/media");
```

[21] For more information about the FieldRenderer Web control, see the *Presentation Component Reference* manual at http://sdn.sitecore.net/Reference/Sitecore%206/Presentation%20Component%20Reference.aspx .

Chapter 6

Syndication APIs

This chapter describes APIs related to Sitecore Really Simple Syndication (RSS) features.[22]

This chapter first describes Sitecore syndication APIs, and then describes ASP.NET system syndication APIs.

This chapter contains the following section:

- Sitecore.Syndication Classes
- ASP.NET Syndication Classes

[22] For more information about Sitecore RSS features, see the *Presentation Component Cookbook* at http://sdn.sitecore.net/Reference/Sitecore%206/Presentation%20Component%20Cookbook.aspx.

6.1 Sitecore.Syndication Classes

This section describes the classes in the `Sitecore.Syndication` namespace that implement Sitecore RSS features.

6.1.1 Sitecore.Syndication.FeedManager

You can use the `Sitecore.Syndication.FeedManager.GetFeedUrl()` method to access the URL of an RSS feed. For more information about the `Sitecore.Syndication.FeedManager.GetFeedUrl()` method, see the section *How to Access the URL of an RSS Feed*.

6.1.2 Sitecore.Syndication.FeedUtil

The `Sitecore.Syndication.FeedUtil` class contains static utility methods related to syndication.

The `Sitecore.Syndication.FeedUtil.IsConfiguredForFeed()` method returns True for any item that you can include in an RSS feed. For an example that uses the `Sitecore.Syndication.FeedUtil.IsConfiguredForFeed()` method, see the section *Example: Syndicate Children of Multiple Items*.

The `Sitecore.Syndication.FeedUtil.IsFeed()` method returns True if the parameter is a feed definition item.

6.1.3 Sitecore.Syndication.PublicFeed

The `Sitecore.Syndication.PublicFeed` class uses feed definition items to construct RSS feeds. The `Sitecore.Syndication.PublicFeed` class syndicates the items identified by a Sitecore query or by the children of a data source item specified in the feed definition item.[23]

You can use the `Sitecore.Syndication.PublicFeed` class as a base class to develop custom feeds. Specify your class in the **Type** field in the **Extensibility** section of the feed definition item. If you do not specify a value for the **Type** field in the **Extensibility** section of the feed definition item, then Sitecore uses the `Sitecore.Syndication.PublicFeed` class to format the feed.

The `Sitecore.Syndication.PublicFeed.FeedItem` property exposes the feed definition item associated with the feed.

You can implement a feed that determines the items to syndicate using custom logic by overriding the `Sitecore.Syndication.PublicFeed.GetSourceItems()` method. For an example that overrides the `Sitecore.Syndication.PublicFeed.GetSourceItems()` method, see the section Example: Syndicate Children of Multiple Items.

Example: Syndicate Children of Multiple Items

You can implement a custom RSS feed based on the following example that syndicates the children of all of the items selected in the field named **Sources** in the feed definition item.

1. In the Visual Studio Web application project, compile a class based on the following example:

```
namespace Sitecore.Sharedsource.Syndication
{
  public class SelectionFeed : Sitecore.Syndication.PublicFeed
  {
    public override System.Collections.Generic.IEnumerable<Sitecore.Data.Items.Item>
```

[23] For more information about Sitecore query, see the *Data Definition Reference* manual at http://sdn.sitecore.net/Reference/Sitecore%206/Data%20Definition%20Reference.aspx.

```
              GetSourceItems()
            {
              Sitecore.Data.Fields.MultilistField entries = this.FeedItem.Fields["Sources"];

              if (entries == null || entries.Count < 1)
              {
                return new Sitecore.Data.Items.Item[0];
              }

              Sitecore.Collections.ItemList list = new Sitecore.Collections.ItemList();

              foreach (Sitecore.Data.Items.Item parent in entries.GetItems())
              {
                foreach (Sitecore.Data.Items.Item child in parent.Children)
                {
                  if (Sitecore.Syndication.FeedUtil.IsConfiguredForFeed(child))
                  {
                    list.Add(child);
                  }
                }
              }

              if (list.Count < 1)
              {
                return new Sitecore.Data.Items.Item[0];
              }

              list.Sort(new Sitecore.Data.Comparers.UpdatedComparer());
              return list.ToArray();
            }
          }
        }
```

1. In the **Template Manager** or the **Content Editor**, in the `System/Feeds/RSS Feed` data template, in the **Extensibility** section, add a **Treelist** field named **Sources**, and set the **Source** property of this field to the `/sitecore/content` item.
2. In the **Content Editor**, in the feed definition item, in the **Extensibility** section, in the **Type** field, enter the signature of the class:

```
Sitecore.Sharedsource.Syndication.MultiParentFeed, Assembly
```

3. In the **Content Editor**, in the feed definition item, in the **Extensibility** section, in the **Sources** field, select items with children to syndicate.

6.2 ASP.NET Syndication Classes

You can use the classes described in the following sections to manipulate RSS feed entries.

Note
To use classes in the `System.ServiceMode.Syndication` namespace, in the Visual Studio Web application project, add a reference to the `System.ServiceModel.Web` assembly. Set the **Copy Local** property of the `System.ServiceModel.Web` reference to **False**.

6.2.1 System.ServiceModel.Syndication.SyndicationItem

The `System.ServiceModel.Syndication.SyndicationItem` class represents an item in syndication feed. You can use the `System.ServiceModel.Syndication.SyndicationItem class` to manipulate properties of RSS feed entries.

Example: Syndication Entry Title Length Limit

You can implement a custom feed based on the following example that limits the length of RSS entry titles.

1. In the Visual Studio Web application project, add a class based on the following example:

```
namespace Sitecore.Sharedsource.Syndication
{
  using System;
  using System.ServiceModel.Syndication;

  public class LimitedTitleFeed : Sitecore.Syndication.PublicFeed
  {
    protected override SyndicationItem RenderItem(Sitecore.Data.Items.Item item)
    {
      SyndicationItem entry = base.RenderItem(item);
      int titleLengthLimit;

      if (this.FeedItem != null
        && Int32.TryParse(this.FeedItem["TitleLengthLimit"], out titleLengthLimit))
      {
        string title = entry.Title.Text.Substring(0, titleLengthLimit - 3);
        title = title.TrimEnd(new[] { ' ', ',', ';' }) + "...";
        entry.Title = new TextSyndicationContent(title);
      }

      return entry;
    }
  }
}
```

2. In the **Template Manager** or the **Content Editor**, create a custom feed data template that inherits from the `System/Feeds/RSS Feed` data template, and add an integer field named **TitleLengthLimit**.
3. In the **Content Editor**, create a feed based on the custom feed data template.
4. In the **Content Editor**, in the feed definition item, in the **TitleLengthLimit** field, enter the maximum number of characters to allow in the title of a syndication entry.
5. In the **Content Editor**, in the feed definition item, in the **Extensibility** section, in the **Type** field, enter the signature of the .NET type, such as the following:

```
Sitecore.Sharedsource.Syndication.SelectionFeed, Assembly
```

Example: Syndication Entry Categories

You can use the RSS format to associate multiple topic categories with each syndication entry. You can manage a list of categories as a folder containing category definition items. You can add a selection field to each of the data templates for syndicated items to allow the CMS user to select any number of category definition items. You can include the categories in each syndication entry by populating the `System.ServiceModel.Syndication.SyndicationItem.Categories` list with the names of the category definition items.

You can implement a custom feed based on the following example that includes categories for syndication entries.

1. In the Visual Studio Web application project, add a class based on the following example:

```
namespace Sitecore.Sharedsource.Syndication
{
  using System.ServiceModel.Syndication;

  public class CategorizedFeed : Sitecore.Syndication.PublicFeed
  {
    protected override SyndicationItem RenderItem(Sitecore.Data.Items.Item item)
    {
      Sitecore.Diagnostics.Assert.ArgumentNotNull(item, "item");
      SyndicationItem entry = base.RenderItem(item);
      Sitecore.Data.Fields.MultilistField entries = item.Fields["Categories"];

      if (entries != null)
      {
        foreach (Sitecore.Data.Items.Item category in entries.GetItems())
        {
          if (category != null)
          {
            entry.Categories.Add(new SyndicationCategory(category.Name));
          }
        }
      }

      return entry;
    }
  }
}
```

2. In the **Content Editor**, in the feed definition item, in the **Extensibility** section, in the **Type** field, enter the signature of the .NET type, such as the following:

```
Sitecore.Sharedsource.Syndication.CategorizedFeed, Assembly
```

3. In the **Content Editor**, create a folder containing category definition items.
4. In the **Template Manager** or the **Content Editor**, in the data template(s) used for syndication entries, add a selection field named **Categories**, and then set the **Source** property of the **Categories** field to the folder containing category definition items that you created in the previous step.
5. In the **Content Editor**, in the syndication entries, in the **Categories** field, select relevant categories.

Tip
Create a data template containing the **Categories** field, and add that data template as a base template for templates that support syndication.

Chapter 7

Miscellaneous Content API

This chapter contains information about unclassified content APIs that are not covered in the previous chapters of this manual.

This chapter contains the following sections:

- Configuring Reminder

7.1 Configuring Reminder

In Sitecore CMS, you can modify the subject or text of the Reminder functionality.

To change the subject of the reminder e-mail:

1. In the `web.config` file, find the `Tasks.EmailReminderSubject` property which defines the subject of the e-mail reminder. The default value of the property is:

   Reminder from Sitecore.

2. Change the default value to the one you want.

To change the e-mail body text of the reminder e-mail:

1. In the `web.config` file, find the `Tasks.EmailReminderText` property which defines the e-mail body of the email reminder. This text introduces the e-mail reminder. A custom text can be appended to this text. The default value of the property is:

   This is a reminder from Sitecore regarding the item: '{item}' in the database '{database}'

   The `{item}` and `{database}` tokens are replaced with the appropriate Item path and `Database` name before the message is sent.

2. Change the default value to the one you want.

Chapter 8

Troubleshooting Content APIs

This chapter contains troubleshooting information for common issues Sitecore developers experience when using content APIs.

This chapter contains the following sections:

8.1 Could Not Find Configuration Node

You may see messages such as the following in the browser if you attempt to use the `Sitecore.Configuration.Factgory.GetDatabase()` method to access a database that does not exist, or do not enter the database name in the same character case as its configuration in `web.config`:

System.InvalidOperationException: Could not find configuration node

Ensure that the value passed as the first parameter to the `Sitecore.Configuration.Factgory.GetDatabase()` method matches the `id` attribute of the appropriate `/configuration/sitecore/databases/database` element in `web.config`.

8.2 Object Reference Not Set to an Instance of an Object

You may see messages such as the following in the browser if you attempt to access an item that does not exist, has not been published, or to which the context user does not have the `item:read` security access right:

System.NullReferenceException: Object reference not set to an instance of an object

Ensure that the ID or path specified for the item is correct, that the code accesses the correct database, that the item exists in that database, and that the context user has the `item:read` access right to the item. You may need to publish the item or its data template, or use a security user switcher or security disabler. For more information about using a security user switcher or security disabler, see the section *How to Resolve Item Access Rights*.

8.3 Item Is Not in Editing Mode

You may see messages such as the following in the browser if you attempt to update an item without placing it in editing mode:

Sitecore.Exceptions.EditingNotAllowedException: Item is not in editing mode

For information about placing an item in editing mode, see the section Note
We recommend that you provide the context user with the appropriate access rights instead of using a security user switcher or a security disabler.

The code examples in this document do not use a security user switcher or security disabler. This implies that the context user has the required access rights in order for the logic to succeed.

How to Place an Item in Editing Mode.

8.4 The Current User Does Not Have Write Access to This Item

You may see messages such as the following in the browser if you attempt to update an item to which the context user does not have the `item:write` security access right:

System.UnauthorizedAccessException: The current user does not have write access to this item

Ensure that the context user has the `item:write` access right to the item. You may need to use a security user switcher or security disabler. For more information about using a security user switcher or security disabler, see the section *How to Resolve Item Access Rights*.

8.5 Add Access Required

You may see messages such as the following in the browser if you attempt to add an item under an item to which the context user does not have the `item:create` security access right:

Sitecore.Exceptions.AccessDeniedException: AddFromTemplate - Add access required

Ensure that the context user has the item:create access right to the parent item. You may need to use a security user switcher or security disabler. For more information about using a security user switcher or security disabler, see the section *How to Resolve Item Access Rights*.

Note
When you create an item, Sitecore automatically places the first version in the context language in the initial state of the default workflow specified in the standard values of the item's data template.

To update the workflow state of an item, you can refer to the methods:

- Sitecore.Workflows.Simple.Workflow.Start
- Sitecore.Workflows.Simple.Workflow.Execute