Sitecore Media Facilities

Author: Sitecore Corporation

Date: Friday, 10 August 2007

Release: Rev. 1.0

Language: English

Table of Contents

Chapter 1

Configuration Options

The present article describes the media storage architecture of Sitecore CMS and applied guides on Sitecore CMS configuration.

Users who wish to configure the media storage quickly for testing purposes should navigate to the following sections of the article:

- Configuration Options Comparison - the overview of the two principal media storage methods.
- Basic Media Storage Configuration Instructions.

1.1 Architectural Notes

All data in Sitecore is stored as text in an XML repository which abstracts an underlying relational database. Every item in the XML repository is based on a template, which defines the data structure – the fields which make up the data item. Items are therefore structured data, though unstructured data can be embedded in an item for instance using the Rich Text field type.

Some data cannot efficiently be structured into an XML hierarchy, for instance the images, movies, PDF and MP3 files. Such resources are stored in Sitecore's Media Library, which provides a number of media templates used to manage metadata about these media, for instance alternate text for images. The actual content of each media file can be either stored on the file system (in which case one of the fields in the media template references the location of the file on disk) or encoded and stored as a blob in the database (in which case one of the fields in the media template references the database record). This document uses the term "database media" for media items where the binary data from the file is stored in the database, "filesystem media" for media items where the binary is stored on the file system.

Just like file-based media, media items in Sitecore should be organized into a hierarchy similar to a folder structure. In Sitecore, a folder is just an item based on a template that has no fields – a folder is an item that is just a container for other items. When files and folders are copied into Sitecore's media library file systems as described below, Sitecore automatically creates the appropriate folder path in the media library and creates media items for the new files in the media library location corresponding to the original file system location. Certain Sitecore item naming restrictions are applied so the actual path to a media item may differ from its original file system path. Moving, renaming and deleting media on the media file systems is not supported, as these may not update the media library items appropriately.

Every field in a Sitecore template is versionable by default, meaning that Sitecore can keep track of values in the field in previous versions of the item. Sitecore provides two templates for each type of media item, a versionable template and an unversionable template. In general, versioning of media should only be implemented when versioning is expressly required and implications such as increased database storage space over time are understood.

Because these settings affect the way data is managed by Sitecore, and as there is no default feature for converting a database media asset to a file media asset, these settings should be considered and defined during installation rather than afterwards.

1.2 Configuration Options Comparison

Consider the following factors for any new Sitecore installation: each setting has advantages and disadvantages for working with various types of media.

Database Media Storage

Advantages

- Media can be published like any other resource
- CMS services such as locking, security, versioning and workflow are supported for media
- Images can be dynamically manipulated, for instance shrinking or stretching
- Changes to the media library do not need to be synchronized with the file system

Disadvantages

- Very large media can result in performance issues
- URLs for media items are altered, for instance ending in the .ashx extension which invokes ASP.NET processing rather than original file extensions
- Can result in increased database and file system requirements as media are both stored in two databases (master and web) as well as being cached on disk

Filesystem Media Storage

Advantages

- Media can be manipulated on the file system, for instance copying the media folder to another system

Disadvantages

- For multi-server environments, media must be propagated from the CMS server to the content delivery server(s) using the Sitecore Staging module (see http://sdn5.sitecore.net/Products/Staging.aspx)
- CMS services such as locking, security, versioning and workflow can be applied to the metadata only. Every change of the file itself will require a new copy of the file in the file system.

Every Sitecore media item involves two components: binary data generally originating from a media file and textual metadata stored in the fields making up the media item. When an item is

added to Sitecore's media library, the data can either be managed on the file system or encoded and stored in the Sitecore content repository. Each Sitecore server is configured to store data on the file system or in the database by default, but an advanced upload UI is available to some users allowing them to control whether uploads are stored on the file system or in the database. Using logic described under Media Path Naming Algorithms, Sitecore stores a URL in the Path field of the new media item. For media stored on the file system, the File Path field is populated with the path to the file on disk relative to the document root. By default, the Sitecore URL for a media item contains the .ashx extension mapping the incoming request to ASP.NET for processing. While this can be avoided by mapping the various media extensions to the ASP.NET DLL, this is generally not recommended – if the URL of a media item must contain the original file extension, media should be managed on the file system. File system media may be referenced by the URL in their File field when image manipulation or other services are required or by the value of their File Path field when the original extension is needed, though this can generally be avoided using forceDownload as described under Media Field Types and Accessing Media Field Values. Accessing media by the Path URL will always be slower than accessing media by File Path, though for the larger media the difference should not be considerable.

1.3 Configuration

1.3.1 Filesystem Media

These settings should be used in case if media files will be stored on disk.

1. Set *Media.UploadAsFiles* to **true** in web.config:

```
<setting name="Media.UploadAsFiles" value="true">
```

2. Specify the MediaFolder path in web.config (files dropped into this folder will be copied into the *Media.FileFolder* and appropriate items will be created in the media library):

```
<sc.variable name="mediaFolder" value="/upload">
```

3. Specify the Media.FileFolder path. Uploaded files will be stored in this folder:

```
<setting name="Media.FileFolder" value="/App_Data/MediaFiles">
```

Note: For multi-server environments, media must be propagated from the CMS server to the content delivery server(s) using the Sitecore Staging module (see http://sdn5.sitecore.net/Products/Staging.aspx)

1.3.2 Database Media

These settings should be used in case if media files will be stored in the database.

1. Set *Media.UploadAsFiles* to **false** in web.config:

```
<setting name="Media.UploadAsFiles" value="true">
```

2. Specify the MediaFolder path in web.config (if a file system is copied into this folder Sitecore will automatically create media items for each and import the data to the database):

```
<sc.variable name="mediaFolder" value="/upload">
```

3. Specify the Media.FileFolder path for cases when users will need to store a media file on disk:

```
<setting name="Media.FileFolder" value="/App_Data/MediaFiles">
```

1.3.3 Architectural Notes

Organizations must choose a default (database or file-system storage) and should use the other technology only when appropriate. For instance if database storage is the default then file-based storage might only be used for very large PDFs when versioning and other CMS services are not needed, but if file-based storage is the default then database storage could be used only for media which require security. The default storage option is defined by the Media.UploadAsFiles setting in web.config which defaults to false, causing media to be stored in the database by default (setting Media.UploadAsFiles to true will cause the default to be file-based storage).

Sitecore provides two base file system directories for working with media, which should be considered drop-off points (files dropped into these folders will be imported into the media library). Whether using file storage or database storage, once files exist under a Sitecore media directory, they should not be removed, renamed or deleted, as this can break references in Sitecore. Additionally, directories should not be created manually in these media directories - Windows will first create a directory named something like "New Folder". Instead, create the media directory structure elsewhere before dragging it into the media root.

Media stored in the database are associated with the directory specified by the MediaFolder setting in web.config, which defaults to /upload (as defined by the mediaFolder variable also defined in web.config). When media stored in the database are uploaded to Sitecore, they are not written to the file system – this folder exists to simplify import (if a file system is copied into this folder Sitecore will automatically create media items for each and import the data to the database).

When Media.UploadAsFiles is false, by default media are stored in the database and never created on the file system. Database media items are created from files copied into directory specified by MediaFolder but the files are not copied to Media.FileFolder and are not used after the media item has been created. If the user chooses “Upload as file” in the advanced upload UI, the file is written to Media.FileFolder.

When Media.UploadAsFiles is true, media are stored on the file system under the directory specified by Media.FileFolder. If the user disables “Upload as file” in the advanced upload UI, the file is not written to disk but stored in the database instead and no file is written to disk.

Sitecore monitors activity in the directory specified by MediaFolder. When a new file appears, if Media.UploadAsFiles is false, a media item is created and the binary data is stored in the database. If Media.UploadAsFiles is true, a media item is created, any corresponding path is created under Media.FileFolder, the file is copied to this path which is referenced by the new media item; the file under MediaFolder remains but is not used. The ASP.NET worker process must be active for monitoring of the MediaFolder to be in effect – ensure the Sitecore user interface is responding before placing files for import into MediaFolder and monitor the logs while importing media to ensure ASP.NET is not overwhelmed by file system activity. The folder specified by Media.FileFolder is not monitored by Sitecore – regardless of how Media.UploadAsFiles is set, no media items are created from file deposited in Media.FileFolder. MediaFolder and Media.FileFolder should never be set to the same path.

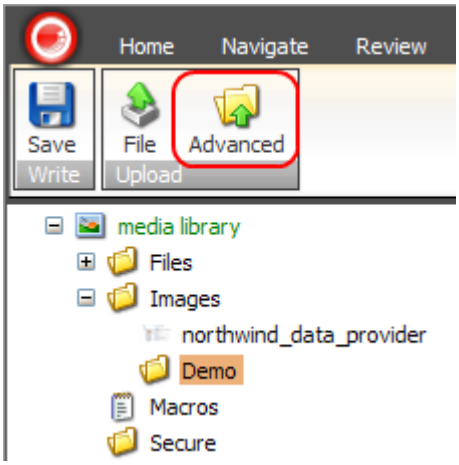
Under various conditions when a new file or directory is created in an NTFS file system, a temporary name such as “New Directory” is assigned before the new file system resource is named. It is therefore important not to create and rename files and directories under MediaFolder,

but rather to copy existing files and directory structures into this directory which avoids system renaming.

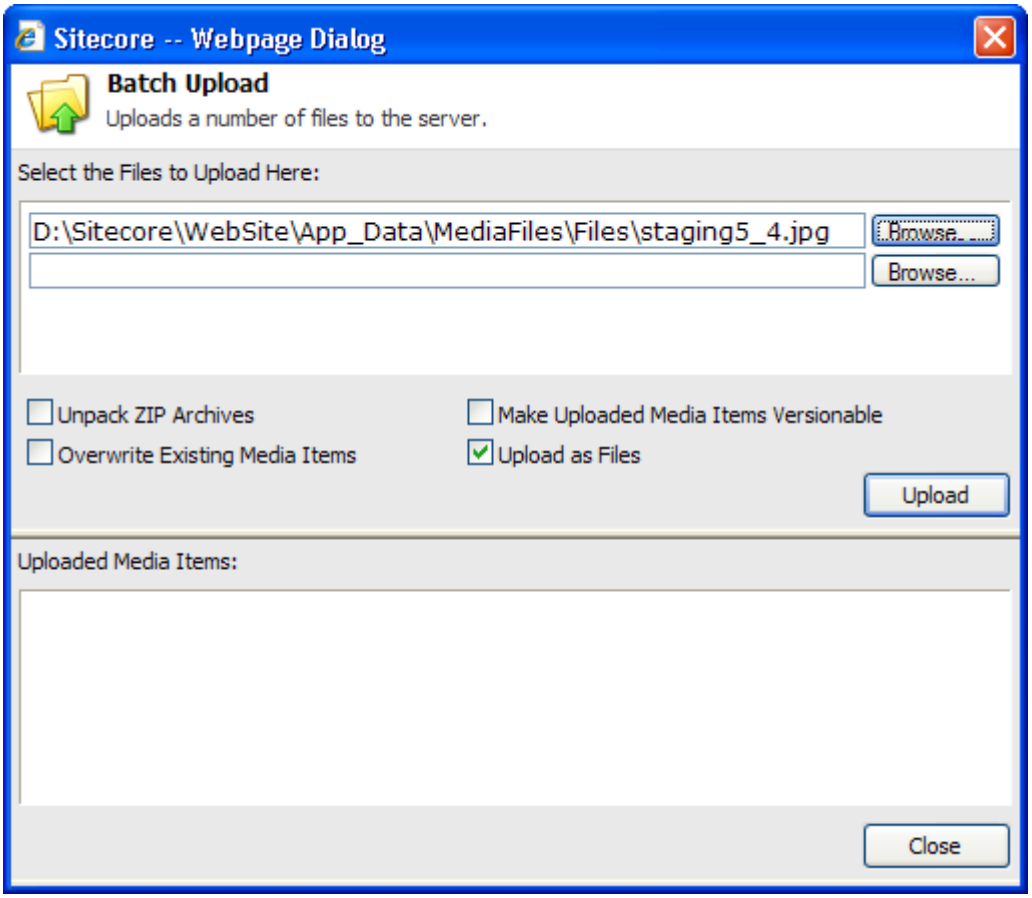
1.3.4 Advanced Upload UI

Sitecore provides an advanced upload dialog with various options which allow to specify where an item will be stored.

To access the advanced upload dialog, navigate to the folder in the Media Library where the files are supposed to be uploaded and select the **Advanced** button in the Upload tab.



The following dialog will start:



The dialog has the following options:

- **Unpack ZIP Archives**
  Check to instruct Sitecore CMS to unpack zip archives after uploading. The archives will not be deleted after unpacking.

- **Make Uploaded Media Items Versionable**
  Check to make the uploaded media items versionable.

- **Overwrite Existing Media Items**
  Check to instruct Sitecore CMS to overwrite the existing files with the new ones if their names are the same. If unchecked, the postfix including the number will be added to the new filename.

- **Upload as Files**
  Check to upload the files as the filesystem media, uncheck to upload the files as the database media.

After the files are uploaded, their thumbnails will be displayed in the Uploaded Media Items field.

Chapter 2

Development Options

This section contains in-depth information for developers.

2.1 Media Library web.config Settings

Additional information is provided in inline comments within the default web.config file.

Setting:	Default:	Description:
MediaFolder	/upload	Sitecore will automatically monitor this directory for new files and create corresponding media items. If Media.UploadAsFiles is false, these files are left in the MediaFolder directory but the files are not used – their database representations are used instead. If Media.UploadAsFiles is true, these files are copied to Media.FileFolder and the files in this directory are used (MediaFolder is used only for automated media imports).
Media.AutoSetAlt	false	If set to true, the Alt field of media items will be automatically set to the item name as determined by Sitecore media naming algorithms (see Media Path Naming Algorithms) when an image media item is created through uploading or directly on the file system, otherwise the field will be left blank.
Media.FileFolder	/App_Data/ MediaFiles	Base directory for media stored on the file system. This directory does not exist by default but is created automatically if needed. Sitecore does not monitor this directory and create media items when files appear.

Media.CacheFolder	/App_Data/ MediaCache	The folder under which media files are cached
Media.CachingEnabled	true	Enables or disables disk-based caching of media files (irrelevant when files are retrieved by File Path).
Media.DefaultImageFormat	Jpeg	Sitecore will assume an image is in this format if the actual format cannot be determined from a file's extension. It must be possible to parse the value of this setting to match a valid System.Drawing.Imaging.ImageFormat value (see http://msdn2.microsoft.com/en-us/library/k5290c89(vs.71).aspx).
Media.IncludeExtensionsInItemNames	false	Indicating whether to include a file extension when generating an item name from a file name. File.jpg in the file system becomes file. Some organizations may prefer these friendlier names in the CMS user interfaces while others may prefer to differentiate various types of documents by including the extension.
Media.InterpolationMode	High	Interpolation mode for use in resizing images. It must be possible to parse the value of this setting to match a valid System.Drawing.Drawing2D.InterpolationMode (see http://msdn2.microsoft.com/en-us/library/system.drawing.drawing2d.interpolationmode.aspx)
Media.MaxSizeInDatabase	20MB	Indicates the maximum allowed size of media intended to be stored in a database. The data provider will thrown an exception which the upload UI will use to inform the user the resource is too large. Note that the maxRequestLength attribute of /configuration/system.web/httpRuntime limits the amount of data that can be uploaded; this value should always be higher than Media.MaxSizeInDatabase.

Media.MaxSizeInMemory	40MB	Indicates the maximum allowed size of a media object in memory during processing, for instance generating thumbnails or otherwise manipulating including use of the image editor.
Media.UploadAsFiles	false	When false, media will be stored in the database by default; when true media will be stored on the filesystem by default.
Media.RequestExtension	ashx	Extension used to map media URLs to ASP.NET for processing by Sitecore.
Media.SecureFolder	/secure	This setting is deprecated starting from version 5.3.1 rev. 070504. This setting is used to optimize FastMediaCache. When true the security checks in media cache are performed only on requests for media stored in the specidied Sitecore directory.
Media.UploadAsVersionableByDefault	false	When true, templates supporting versioning of media are used; when false, templates not supporting versioning of media are used.

Media.UseItemPaths	false	When false, the value of the Path field for new media items includes a prefix, the compressed GUID of the media item and the .ashx extension (depending on Media.RequestExtension): ~/media/F11C99D0112F4363A1A79AAA89ACC96A.ashx When true, the value of the Path field and hence the default URL for new media items will be based on the path to the item rather than its compressed GUID. Under default conditions: ~/media/dir1/dir2/sample.ashx Neither approach is known to be faster than the other – some organizations may prefer the unique GUID or the hiding of media library structure, and if the media library is too deep using the path may exceed URL limits imposed by IIS due to its reliance on NTFS. Other organizations may prefer friendlier URLs for media.
Media.WhitespaceReplacement	" " (a single space)	The character to use when replacing whitespace (in general case - the character matching the .NET regular expression \W) in the names of uploaded media.

2.2 /App_Config/MimeTypes.config and mediaType Entries in web.config

Sitecore 5.3.1 introduces /App_Config/MimeTypes.config mapping file extensions to the appropriate mime types. Each /configuration/mediaType entry in MimeTypes.config maps a comma-separated list of extensions to the appropriate mime type for those extensions. For instance the following line maps the two extensions .au and .snd to the audio/basic mime type:

```
<mediaType extensions="au,snd"><mimeType>audio/basic</mimeType></mediaType>
```

For convenience of administration, the most common mime types are configured directly in web.config and MimeTypes.config only comes into play for files with extensions which have no corresponding mediaType entry in web.config. These entries have the following format:

```
<mediaType name="JPEG image" extensions="jpg, jpeg">
```

This entry applies a name to the media type and maps file extensions to this type.

```
  <mimeType>image/jpeg</mimeType>
```

This entry defines the mime type to use for media assets created from files with these extensions.

```
<forceDownload>false</forceDownload>
```

A true value for the forceDownload element causes Sitecore to apply an HTTP "Content-Disposition = attachment; filename=" header when linking to the .ashx URL of the media item, causing the browser to prompt the user to open/save as rather than opening the resource in the browser.

```
<sharedTemplate>system/media/unversioned/jpeg</sharedTemplate>
```

This Sitecore template will be used for unversioned media items matching the extensions associated with this media type.

```
<versionedTemplate>system/media/versioned/jpeg</versionedTemplate>
```

This Sitecore template will be used for versioned media items matching the extensions associated with this media type.

```
<mediaValidator type="Sitecore.Resources.Media.ImageValidator" />
```

A method in this class is executed to generate media warnings in the Sitecore UI such as if an image does not have the Alt field populated.

```
<thumbnails>
  <generator type="Sitecore.Resources.Media.ImageThumbnailGenerator,
Sitecore.Kernel">
    <extension>png</extension>
  </generator>
  <width>150</width>
  <height>150</height>
  <backgroundColor>#FFFFFF</backgroundColor>
</thumbnails>
```

This section defines how the default thumbnail for a media asset will be generated. Thumbnails are generated on disk and cached just like any other request for a media item using the Path property.

```
<prototypes>
  <media type="Sitecore.Resources.Media.JpegMedia, Sitecore.Kernel" />
</prototypes>
```

A method in the specified class will be invoked to populate metadata when new items of this type are created.

2.3 Media Path Naming Algorithms

Often, media directory and file names must be normalized to match Sitecore item naming requirements. The rules for naming media items and paths are the same for both database and file media.

When files appear in directories under MediaFolder, Sitecore constructs the corresponding path under /sitecore/media library if needed and creates the media item under that path, but individual directories in the path may be renamed to meet Sitecore item naming requirements.

Whether a directory or media item appears on the file system or is uploaded through the user interface, the name of the corresponding item in Sitecore's media library will be based on the original file system entry name. When directories appear under the MediaFolder file system, corresponding folders in the media library are created as follows:

1. The value of MediaFolder is replaced with "/sitecore/media library" (/upload/dir1/dir2 becomes /sitecore/media library/dir1/dir2)

2. Any character matching the .NET regular expression \W (any nonword character) is replaced with Settings.Media.WhitespaceReplacement

When files appear under the MediaFolder file system, corresponding items are created in the media library as follows:

1. The corresponding media folder is determined and created using the algorithm above

2. If Settings.Media.IncludeExtensionsInItemNames is false, the dot (".") and extension are removed; otherwise, Settings.Media.WhitespaceReplacement replaces the original dot before the extension

3. Any character matching the .NET regular expression \W (dot (".") by default) is replaced with Settings.Media.WhitespaceReplacement

The ItemNameValidation and InvalidItemNameChars settings in web.config are applied to all items including media.

2.4 Sitecore security

Security can be applied to media items just as to any other Sitecore item.

Media stored on disk is also protected by Sitecore security. Extranet users should not have direct access to the folder defined by Media.FileFolder setting. Extranet users will retrieve media files via URLs generated by Sitecore which are validated against security.

Administrators can prevent extranet users from uploading media files by restricting access to the media library. Don't forget to unprotect the media library item (Configure tab in the Content Editor) if you want to change the rights for the item.

2.5 Development Options

Fields which will always reference image media assets should use the field type Image. Fields which could reference any other type of media asset should use the field type File, which does not support overriding attributes such as height and width. Media assets have a field named "Path" which contains the URL of the media item ending with .ashx; a File Path field is also populated with the disk path for file media assets. Logic such as image manipulation can be applied to all media assets when accessed by Path, but when File Path is used the original media file will always be served. File media items can be referenced by their File Path on disk or by their Path in Sitecore's media library. Because all media items have a Path, all Sitecore components such as the HTML editor and XSL extension functions use the Path.

To generate an HTML image element from a field of type Image in XSL:

```
<sc:image field="image_field" select="." />
```

To create an HTML link to the media item referenced by a field of type Image or File in XSL:

```
<a href="{sc:fld( 'image_or_file_field', ., 'src' )}">Link to an image or
file</a>
```

To create an HTML link to a file media referenced in a field of type File or Image by File Path rather than Path in XSL:

```
<xsl:variable name="mediaitem" select="sc:item( sc:fld( 'image_or_file_field',
., 'mediaid' ), .)" />
<a href="{sc:fld( 'path', $mediaitem )}" >Link to file media by Path</a>
```

See http://sdn5.sitecore.net/FAQ/XSL/Rendering%20Not%20Executing%20Certain%20Statements.aspx for an explanation of the {} syntax.

Use the following code to get an Image or File field in .NET:

```
Sitecore.Data.Fields.ImageField image = Sitecore.Context.Item.Fields["image"];
if (image != null )
{
  // field exists
  if (image.Src != "")
  {
    // User has specified a resource
    // URL is image.Src
    // Path is image.MediaItem.Fields["path"]
    // File Path is image.MediaItem.Fields["file path"]
```

It is the developer's responsibility to ensure the field exists, the user has specified a media item, and that the media item exists whether the media item has a file path or other conditions depending on requirements.

Use the MediaManager class to get URLs of the images:

```
ImageField imageField = Sitecore.Context.Item.Fields["image"];
 string url = string.Empty;
if (imageField != null)
{
    MediaItem mediaItem = imageField.MediaItem;
    if (mediaItem != null)
    {
    MediaUrlOptions options = new MediaUrlOptions();
    options.Height = 300;
    options.Width = 200;
    options.UseItemPath = true;
    url = MediaManager.GetMediaUrl(mediaItem, options);
    }
 }
```

The main properties of the MediaUrlOptions class are described below:

Name	Legal Values	Default Value	Description
AllowStretch	true | false	false	Allow stretching the image beyond its original size?
BackgroundColor	Color names (such as black or red) and HTML hex color codes (such as CE55E2)	black	Background color for the border added when an image is stretched beyond its original size (and allowStretch=false).
Database	Any Sitecore database defined on the site.	content database of the current site	The name of the Sitecore to pull the image from.

DisableMediaCache	true | false	false	Disable the media cache for this request? If true, the image will always be retrieved from the database, bypassing the media cache.
Height	Any positive integer		The height of the image. Be sure to include as=true if the height will be larger than its original size.
language	Any valid language name		Retrieve the image from a specific language version of the item.
maxHeight	Any positive integer		Maximum height of the image to display. Scale the image down to this size if necessary.
maxWidth	Any positive integer		Maximum width of the image to display. Scale the image down to this size if necessary.
scale	Any positive floating point number using a dot as a decimal point (such as 1.5, which corresponds to 150%)		Scale factor for the image to display. Be sure to include as=true if the image will be scaled to larger than its original size.
thumbnail	true | false	false	Display a thumbnail of the requested file, useful for images as well as other media types, such as PDF, flash, and so on.
version	Any positive integer		Retrieve the image from a specific version of the item.
width	Any positive integer		The width of the image. Be sure to include as=true if the width will be larger than its original size.

Additional information on using the <sc:image> element in XSL can be found by following the link below:

http://sdn5.sitecore.net/Articles/XSL/5%203%20Enhancements/Image%20Enhancements.aspx

2.6 Supporting Original Media Extensions

By default, the Sitecore URL for a media item contains the .ashx extension mapping the incoming request to ASP.NET for processing. While this can be avoided by mapping the various media extensions to the ASP.NET DLL, this is generally not recommended – if the URL of a media item must contain the original file extension, media should be managed on the file system.

But in case an organization needs the original image file extensions in the URLs the instructions below should be used.

2.6.1 Using the Image Enhancements

The XSL parameters called image enhancements are described on the following page: http://sdn5.sitecore.net/articles/xsl/5%203%20enhancements/image%20enhancements.aspx

The column Shorthand contains the actual parameters that should be used in the URL, for example:

This is an URL for the jpg image from the media library:

http://server_url/~/media/Files/picture1 JPG.ashx

The next URL returns an image with different dimensions:

http://server_url/~/media/Files/4453 JPG.ashx?w=400&h=400

The next URL returns a thumbnail for the media item:

http://localhost/~/media/Files/4453%20JPG.ashx?thn=1&w=32&h=32

An example from XSLT:

```
<xsl:variable name="path" select="concat(sc:fld('image', ., 'src'),
'?w=400&amp;h=400')" />
<img src="{$path}" alt="" />
```

These enhancements can be used for both the database media and the filesystem media, provided that filesystem media is retrieved via an http handler (URL ending with .ashx by default).

2.6.2 Changing Extensions from ASHX to Image Extensions

It is possible to change the extensions of media items from ashx to the original image file extensions by mapping the various media extensions to the ASP.NET DLL.

For example we want to have .jpg extensions for the jpeg images.

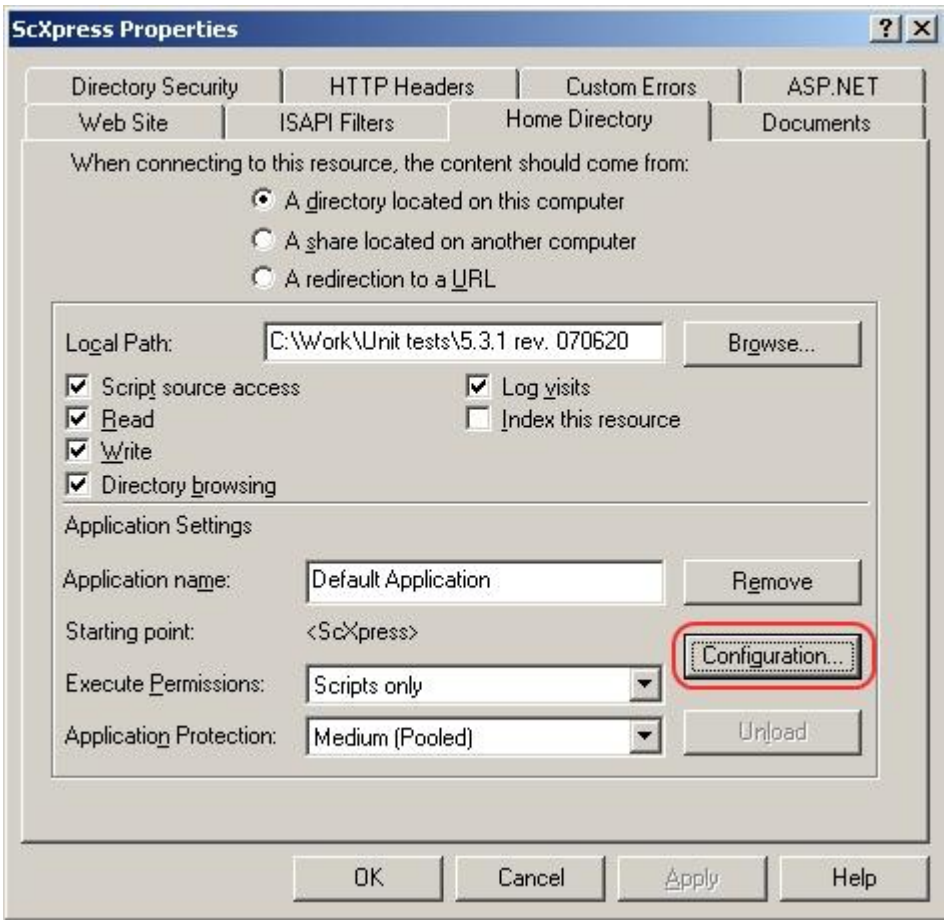
Navigate to a media item in the Media Library and change the extension from ashx to jpg in the *Path* field.

For instance:

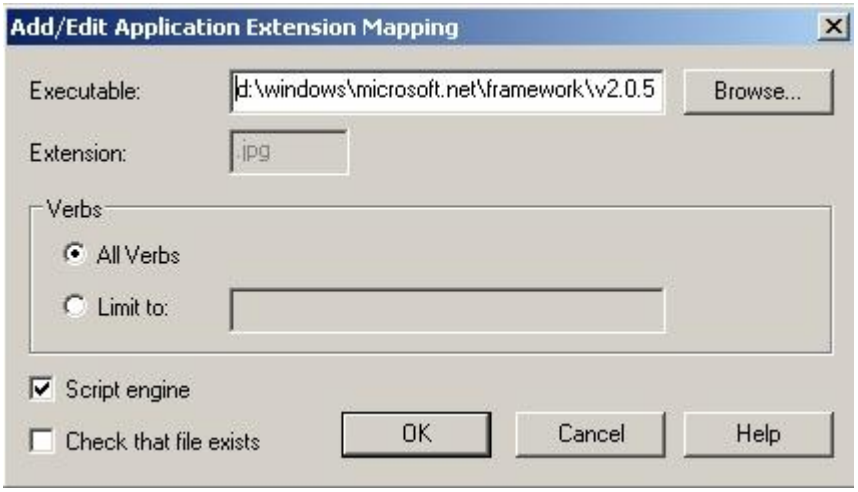
The media item has such value in path field - **~/media/Files/picture1.ashx**

we change the extension to jpg - **~/media/Files/picture1.jpg**

Go to IIS manager, select the site, select options in popup menu, click *Home Directory* tab, and click the configuration button:



Set ASP.NET handler for files with jpg extension:



Don't forget to uncheck the *Check that file exists* checkbox.

After that the next request http://server_url/~/media/Files/picture1.jpg will return an image from the media library. All Sitecore controls will work with such media item correctly. Image enhancements can also be used with such URLs.

The disadvantage of this method is that you have to change the extensions in *Path* field for all items and you should add an extension mapping for every extension which you intend to use.

2.7 Environment Considerations

Sitecore generates absolute (from the document root) but not fully qualified (no protocol or hostname) URLs by default, so the same references can be used from any URL.

When both content editorial (CMS) and content delivery (published site) environments are hosted on a single Sitecore instance, as soon as file media are created, the files are available both in the content editorial and content delivery environments, though in most cases it is necessary to publish media metadata as well. When the content delivery environment is hosted on one or more servers separate from the content editorial server, a mechanism is required to transfer file media from the content editorial server to the content delivery server(s). This is most commonly accomplished with the Sitecore Staging module using SOAP or FTP (see http://sdn5.sitecore.net/Products/Staging.aspx).

Whether the content editorial and content delivery environments are on the machine or content delivery occurs on separate "runtime" servers, media publishing transfers both the metadata and the binary data from the master to the web database(s) for database media so there is no need to configure Staging to transfer separate media files.