Sitecore CMS 6 and later

Sitecore Project Review Recommended Practices

A developer's guide to optimizing usability and performance through recommended practices.

Table of Contents

Chapter 1

Introduction

This document describes the Sitecore recommended practices that Sitecore Professional Services look for when they perform a web site project review. The focus is on usability and performance.

These are the best practices in typical circumstances. As with any guidelines, you should take the specifics of the implementation requirements into account before making use of these recommendations.

This document is designed to help Sitecore developers and administrators create optimized websites.

The information in this document is valid for Sitecore 6 and later.

This document contains the following chapters:

- **Introduction**
  This introduction that describes the aim and content of the document.
- **Sitecore Recommended Practices**
  This chapter contains a list of the recommended best practices grouped according to the areas they deal with.

Chapter 2

Sitecore Recommended Practices

These recommended practices have been gathered from performing Sitecore solution reviews in the field, Sitecore documentation, as well as information gathered from various blogs regarding the subject.

The sections reflect how a Sitecore Professional Services individual would review a site, look at how recommended practices have been applied, and make recommendations along the way.

This document is not a complete representation of all the recommended practices, and will remain dynamic so that it may be improved in the future.

This chapter contains the following sections:

- Templates and _Standard Values
- Information Architecture (Content Structure)
- Security (Users and Roles)
- Workflow
- Media
- Caching
- Presentation
- Page Editor
- Solution Code
- System Architecture (Hardware and Software)

2.1 Templates and _Standard Values

This section contains Sitecore recommended practices for templates and _Standard Values.

- **Naming conventions** —Use simple, relevant, and easy to understand names for templates, fields, and sections. By default, Sitecore displays the names you provided to both technical and non-technical users. Choose names that business users, such as content authors, will easily recognize and understand.
- **Folder structure** — Provide a folder structure to classify by web site, site section, section content, function, and so on. This makes it easier to navigate and find a given template.
- **Make good use of inheritance** — Place commonly used sections and fields in their own template, so that more specific templates can inherit them. For example, the **Title** and **Text** fields in the **Page Title and Text** section are used in multiple different content templates. Rather than duplicate theses fields in each content template, simply inherit the **Page Title and Text** template.
- **Avoid duplicate field names** — Even though sections separate fields, do not reuse a field name within the same template. Be careful not to have two fields with the same name in the inheritance chain either. If you really need two fields with the same apparent name in different sections in the Content Editor, use the **Title** field of the field definition item. Whenever this field is set, it is used by the Content Editor instead of the name.
- **Assign icons to templates** — This provides a visual clue to the type of item that will be created.
- **_Standard Values** — Define layout details, initial workflow, and insert options to a template. This reduces administration and centrally manages system settings, rather than setting them on individual items.
- **_Standard Values** — Use tokens that are meaningful and provide clues to business users, such as content authors, as to what information is expected in a given field to provide default values.
- **_Standard Values** — Use tokens, such as `$name`, to reduce the amount of text that a content author is required to enter when creating a new item.
- **Image Fields** — Define the source field to show the point in the media library that is relevant to the item being created.
- **Rich Text Editor Fields** — Set the source property to a profile (found in the core database under `/sitecore/system/Settings/Html Editor Profiles`) that only shows the controls you want to make available to your business users.
- Do not override, or delete any templates located under `/Sitecore/Templates/System`.
- Use the *Help* option in the individual field definition items to provide extra information to users about fields. Also consider using the **Title** field of the definition item to present a different name for the field to the user.
- Remember that the name of your templates appear as insert options. Try to give them meaningful names. Use the **Display Name** field to provide a different text for the user than the name of the template.

2.2 Information Architecture (Content Structure)

This section contains Sitecore recommended practices for information architecture.

- Provide a content structure that closely mimics that of the web site. This allows you to easily build navigational controls, allows you to use security rules to easily deny or grant access to various parts of the content tree, as well as, simplifying the task of finding any given item within the content structure.
- Limit the number of items under any given node that share the same parent, to 100 items or less for performance and usability.
- Limit the number of versions of any item to the fewest possible. Sitecore recommends keeping 10 or fewer versions on any item, but policy may dictate this to be a higher number. There is a shared source module, Version Manager, available at: http://trac.sitecore.net/VersionManager to help manage versions.
- For business users, such as content authors, turn off *Standard Fields* to improve performance.
- Avoid using the Rich Text Editor to define structure — this is the job of the presentation layer.
- Use the special syntax to restrict the results on Treelists, DropTrees, and TreelistEx to make sure users can only select the appropriate items, or Sitecore query in the other selection fields.
- Use TreelistEx instead of Treelist when showing very big trees — like the *Home* node and its descendants — or have lots of Treelist fields in one single item. TreelistEx only computes the tree when you click *Edit* whereas a Treelist will compute it every time it is rendered.
- Remember the structure of the tree has an impact on URLs by default. If you put items inside a folder, that folder name becomes part of the URL.
- Do not put items that should be accessed by a URL (and have presentation details set) outside of the Home item of the site, as this adds `/Sitecore/content` to the URL.
- Remember to set Insert Options appropriately to enforce the topology of the tree by restricting the types of items users can create in the different parts of the tree. This can be further restricted through security.
- Insert options can be tailored further through Rules (`/sitecore/system/Settings/Rules/Insert Options`) This allows you to add/remove insert options according to dynamic conditions (number of existing children), or type of content. This feature is particularly useful when you want to avoid having to create new “Folder templates” to assign the insert options.

2.3 Security (Users and Roles)

This section contains Sitecore recommended practices for security (users and roles).

- **Break inheritance rather than explicitly deny access rights** — It is a recommended practice to break inheritance in cases where the access right should be denied instead of explicitly denying it for a security account. If you deny an access right explicitly to a security account, the only way to override this denial of access is to do it directly on a user account. This creates an overhead in security management when you would like a user to inherit this role and some other role that should allow the same right access.

  Example:

  Role A denies write access to a `/home/contact us` item. Role B allows write access to the same item. User AB has been assigned role A and role B. But due to the explicit denial of write access to the item, user AB cannot get write access to it.

  When you break security inheritance it sets permissions to the default deny state, which can be overridden, by assigning explicit allow access to one of the roles. Explicit deny access can never be overridden by explicit allow access.

  For more information about security roles and inheritance, see http://sdn.sitecore.net/upload/sitecore6/security_administrators_cookbook-usletter.pdf, section 5.3.

- **Apply security to roles rather than users** — We recommend that you configure security rights for a role security account rather than a user account. Even if there is only one user in the system with this unique access level, it is still better to have permissions configured for a role account. It is much easier to maintain a security system in which all the security configuration is set on role security accounts.
- Limit access to the parts of the content tree that are relevant to the user that is logged into the system. For example, a person who is responsible for news items may not require access to the *About Us* page. This not only allows the user to focus only on what is relevant, but also provides performance benefits by not having to render the entire content tree.
- Limit access to the ribbon items by disabling features that are not relevant to individual users.
- No users should have empty or obvious passwords.
- Use the profile setting of the user properties to specify that a user should always use a certain interface no matter what interface they select in the login screen.
- Make sure that users belong to only the required Sitecore Client roles.
- Administrator user accounts should only be used to perform administrator tasks (mainly unlocking other user's items).

2.4 Workflow

This section contains Sitecore recommended practices for workflow.

- We recommend that you enable workflow. Workflows ensure that items move through a predefined set of states before they become publishable. This ensures that content always receives the appropriate reviews and approval before it is published on the live web site.
- Workflow can be very useful even when approval is not required. A simple two state workflow (Edit, Publish) can be useful to ensure that items that are currently being edited are not published when you run the *Publish* wizard. In addition, this provides auto-versioning. By using an auto-publish action in the *Publish* state, users who are not members of the *Publishing* role can also publish content.
- Apply appropriate workflow security.
- Limit workflow publishing to specific roles and users. In the workflow, provide a default final step with the publish action.
- Avoid email notification of every workflow state. Only provide email notification where it makes sense for a critical workflow event. Alternatively, you can use of the RSS feeds instead of email.
- Try to minimize the number of workflows, states, and the number of actors involved.

2.5 Media

This section contains Sitecore recommended practices for media.

- We recommended that you store media in the database. Blob bases storage approach is recommended as it simplifies publishing and deployment of media assets.

  For example, you can easily publish media that is stored in the database to a production target, whereas file system based media requires synchronization of directories and files between servers.

- The folder structure of the media library should be organized so that it is easy to navigate for non-technical users.
- Give media items names that can easily be recognized and understood by business users, such as content authors.
- Media items should be optimized, using a graphics program such as Photoshop to decrease the size of a rendered page thereby increasing performance.
- **Image Fields** — Define the source field to show the point in the media library that is relevant to the item being created.
- Include the path to static media items (that is, header images, CSS files, JavaScript files, and so on.) in the *IgnoreUrlPrefixes* setting located in the web.config file. This will prevent those media items from being processed through the Sitecore pipelines, improving performance.

2.6 Caching

This section contains Sitecore recommended practices for caching.

- We recommend that you configure and tune the Sitecore database caches — prefetch, data, and item. For more information, see: http://sdn.sitecore.net/upload/sitecore6/sc62keywords/cache_configuration_reference_us.pdf section 4.2, Tuning (database caches)
- Cache rendered output, especially expensive logic and components that are retrieved frequently. For more information about setting up caching for renderings, see: http://sdn.sitecore.net/upload/sitecore6/64/presentation_component_reference-usletter.pdf

2.7 Presentation

This section contains Sitecore recommended practices for presentation.

- Limit the number of layouts in favor of sublayouts. The aim should be to have one layout per site per device.
- Place components that appear in every single page statically (in the markup) of the layout. Use placeholders to dynamically bind components that only appear in some pages. Group components that are always used together in sublayouts.
- Set presentation details on the _standard values whenever possible.
- Use the FieldRenderer or the other Sitecore presentation controls to enable the PageEditor functionality. You can still use those components for rendering fields that should not be editable setting the disable-web-editing attribute.
- Make sure to use the appropriate image manipulation parameters to ensure that images are re-sized on the server and unnecessarily large files are not sent to the browser.
- Make sure to use the DataSource — by using `$sc_item` instead of `$sc_currentitem` in xslts, the DataSource attribute on WebServer controls or with the appropriate code in Subalyouts. This facilitates creating multi-variant tests (MVT) and personalization rules in the OMS.
- Leverage the presentation component parameters. You could use the parameters to allow users to configure/modify the behavior of the component, for example, the number of items shown in a list, the CSS classes to use, the URL for a feed it is showing, and so on.
- Make sure to create a template for any parameters in the presentation component, and set the Parameters Template Property.
- Test components appropriately. For example, a menu rendering might work fine with a few items, but struggle when you have large amounts of data.

2.8 Page Editor

This section contains Sitecore recommendations for the Page Editor.

- Sitecore recommends that you enable and configure the Page Editor for non-technical users.
- Configure the placeholder settings so that users are able to add the right components to the page. In addition, use the **Editable** check-box to disable the ability to add components in the Page Editor to a particular placeholder. Also use security (disable Write access) to disable the placeholder for certain users only.
- Consider using the Page Editor buttons, and the Datasource Location, and Datasource Template fields in Sitecore 6.4 and later.
- Make sure you test your CSS. It might have an impact the way the PageEditor displays.

2.9 Solution Code

This section contains overview of Sitecore recommended practices for solution code.

- Sitecore highly recommends using C#. All of the Sitecore programming examples are in C# and C# is the language spoken by Sitecore Support.
- Follow best practices for Web, Windows, IIS, ASP.NET, XML, XSL, SQL, .NET, CSS, JavaScript, and other technologies. For example, comment your code to describe the intent, not the implementation.
- Use GUIDs where possible instead of path / name. This will improve performance, as well as prevent any breakage if you move content to a new location in the content tree.
- Use the Sitecore Search API instead of query or fast query. This allows querying based on indexes, rather than the content structure. For more information about search, see Alex Shyba's blog post http://sitecoreblog.alexshyba.com/2011/02/8-reasons-to-use-new-search-in-sitecore.html and http://sdn.sitecore.net/Reference/Sitecore%206/Sitecore%20Search%20and%20Indexing.aspx
- Avoid //, descendant and descendant-or-self axis in XSL renderings.
- Use `App_Config/Include` files instead of directly modifying the `web.config` file.
- Don't hardcode information in the code, use `<settings>` in a `*.config` file.
- Use parameters and properties when creating pipelines/events.
- Don't hardcode labels and other static text information, consider modeling it with items, or create a dictionary in Sitecore.

2.10 System Architecture (Hardware and Software)

This section contains Sitecore recommended practices for System Architecture (hardware and software).

Sitecore recommends that:

- All servers — Web and Database, run on Windows Server 2008 R2 64 bit.
- The Web server(s) run IIS 7.x with the latest security updates.
- You use IIS 7.x Integrated mode.
- The Database server(s) run Microsoft SQL Server 2008 R2 64 bit with the latest security updates.
- The Web and Database servers are on separate computers.
- You implement maintenance plans to defragment indexes on the core, master, and web databases on a scheduled basis on SQL Server.
- On IIS — you enable static content compression.
- On IIS — you enable dynamic content compression on the CMS server.
- On IIS — you make sure that the content expires header is set in the common headers, particularly for the /sitecore folder.
- You read the Optimizing Performance in Sitecore article on the SDN for additional information about improving performance.