**Version**

**Document Control**

|  |  |  |  |
| --- | --- | --- | --- |
| Draft | Author | Date | Comment |
| 0.9 | J. Ritmeijer | 11/12/2008 | Initial release |
| 0.91 | J. Ritmeijer | 26/01/2009 | Added info about how to refresh translations and breadcrumbs on all WFEs. See 4.1. |
| 1.0 | J. Ritmeijer | 18/05/2009 | Added information on versioning and made ready for public consumption. |

**Purpose and audience of document**

This document contains guidelines and assistance for developing commercial SharePoint solutions.

The intended audience is anyone involved in developing SharePoint applications.

.

**Disclaimer**

© Muhimbi. All rights reserved. No part of this document may be altered, reproduced or distributed in any form without the expressed written permission of Muhimbi.

This document was created strictly for information purposes. No guarantee, contractual specification or condition shall be derived from this document unless agreed to in writing. Muhimbi reserves the right to make changes in the products and services described in this document at any time without notice and this document does not represent a commitment on the part of Muhimbi in the future.

While Muhimbi uses reasonable efforts to ensure that the information and materials contained in this document are current and accurate, Muhimbi makes no representations or warranties as to the accuracy, reliability or completeness of the information, text, graphics, or other items contained in the document. Muhimbi expressly disclaims liability for any errors or omissions in the materials contained in the document and would welcome feedback as to any possible errors or inaccuracies contained herein.

Muhimbi shall not be liable for any special, indirect, incidental, or consequential damages, including without limitation, lost revenues or lost profits, which may result from the use of these materials. All offers are non-binding and without obligation unless agreed to in writing.

**Contents**

[1 Introduction 6](#_Toc230492850)

[1.1 Customising this document 6](#_Toc230492851)

[2 Development Tools 7](#_Toc230492852)

[2.1.1 Visual Studio 2008 7](#_Toc230492853)

[2.1.2 WSPBuilder 7](#_Toc230492854)

[2.1.3 Source Code Control 8](#_Toc230492855)

[2.1.4 Bug & Issue tracking 8](#_Toc230492856)

[3 User Interface 10](#_Toc230492857)

[3.1 Application and Central Administration pages 10](#_Toc230492858)

[3.1.1 SharePoint Web and User Controls 10](#_Toc230492859)

[3.1.2 Breadcrumbs 10](#_Toc230492860)

[3.2 Company specific branding 11](#_Toc230492861)

[3.2.1 aspx page 12](#_Toc230492862)

[3.2.2 aspx.cs (Code behind) page 12](#_Toc230492863)

[3.2.3 Resource file 12](#_Toc230492864)

[3.2.4 Feature definition 12](#_Toc230492865)

[3.3 ‘Code Behind’ 12](#_Toc230492866)

[3.4 Style Sheets 13](#_Toc230492867)

[3.5 Browser compatibility 13](#_Toc230492868)

[3.6 Screen resolution 13](#_Toc230492869)

[4 Internationalisation / Localisation 14](#_Toc230492870)

[4.1 Multi lingual support 14](#_Toc230492871)

[4.2 Dates, numbers and currencies 15](#_Toc230492872)

[5 Coding guidelines 16](#_Toc230492873)

[5.1 Development Language 16](#_Toc230492874)

[5.2 File headers 16](#_Toc230492875)

[5.3 Commenting 17](#_Toc230492876)

[5.4 Code Formatting 17](#_Toc230492877)

[5.5 Naming conventions 18](#_Toc230492878)

[5.5.1 C# 18](#_Toc230492879)

[5.5.2 Namespaces 19](#_Toc230492880)

[5.5.3 Feature names 19](#_Toc230492881)

[6 Exception Management & Logging 20](#_Toc230492882)

[6.1 Writing to the SharePoint Trace log 21](#_Toc230492883)

[6.2 Writing to the Event log 22](#_Toc230492884)

[7 SharePoint Development & Technologies 23](#_Toc230492885)

[7.1 Application Pages 23](#_Toc230492886)

[7.2 Administration Pages 23](#_Toc230492887)

[7.3 Event Receivers 24](#_Toc230492888)

[7.4 Web Parts 24](#_Toc230492889)

[7.5 Features & Custom Actions 24](#_Toc230492890)

[7.6 Maximum file and folder lengths in SharePoint 24](#_Toc230492891)

[7.7 SPGridview 25](#_Toc230492892)

[7.8 Storing settings 25](#_Toc230492893)

[7.9 Modifying the web.config 26](#_Toc230492894)

[7.10 Elevating privileges 26](#_Toc230492895)

[7.11 Preventing memory leaks 27](#_Toc230492896)

[8 Dealing with version numbers 28](#_Toc230492897)

[8.1 Determining the ‘strong named’ version number 28](#_Toc230492898)

[8.2 Determining the real version number 29](#_Toc230492899)

[9 Sharing Libraries and DLLs between SharePoint solutions 30](#_Toc230492900)

[9.1 The problem 30](#_Toc230492901)

[9.2 The Solution for First party libraries 31](#_Toc230492902)

[9.3 The Solution for Third party libraries 32](#_Toc230492903)

[10 Scalability 33](#_Toc230492904)

[11 Security 34](#_Toc230492905)

[11.1 SSL 34](#_Toc230492906)

[11.2 Anonymous Access 34](#_Toc230492907)

[11.3 Forms Authentication 34](#_Toc230492908)

[11.4 Audit log 34](#_Toc230492909)

[11.5 Validate user data / Prevent cross-site scripting attacks 35](#_Toc230492910)

[11.6 Non privileged users vs Administrators 35](#_Toc230492911)

[11.7 Trust levels 35](#_Toc230492912)

[12 Compatibility 36](#_Toc230492913)

[12.1 32/64 bit 36](#_Toc230492914)

[12.2 Windows Server 2003 / 2008 36](#_Toc230492915)

[12.3 Different patch levels 36](#_Toc230492916)

[12.4 WSS3 vs MOSS 36](#_Toc230492917)

[12.5 .net Framework version 37](#_Toc230492918)

[12.6 International versions 37](#_Toc230492919)

[13 Creating a new project 38](#_Toc230492920)

[13.1 In TFS 38](#_Toc230492921)

[13.2 On the developer workstation 38](#_Toc230492922)

[13.3 Setting up a SharePoint project 39](#_Toc230492923)

[14 Source code control 40](#_Toc230492924)

[14.1 TFS Project Folder Structure 40](#_Toc230492925)

[14.1.1 Main 40](#_Toc230492926)

[14.1.2 Development 41](#_Toc230492927)

[14.1.3 Releases 41](#_Toc230492928)

[14.2 Client Folder structure 41](#_Toc230492929)

[14.3 Branching guidelines 42](#_Toc230492930)

[14.3.1 Typical branching scenarios 42](#_Toc230492931)

[14.3.2 How to branch 42](#_Toc230492932)

[14.4 Working with shared libraries and files 43](#_Toc230492933)

[15 Appendix - Sample files 44](#_Toc230492934)

[15.1 Feature.XML 45](#_Toc230492935)

[15.2 Elements.xml 46](#_Toc230492936)

# Introduction

Muhimbi develops commercial products for the SharePoint market. As the market demands a high level of quality this document has been developed to provide guidance for day-to-day development.

Please note that a certain level of SharePoint development knowledge is expected from all readers. This document is by no means a training guide and is under constant revision.

Some topics such as Content Types and Work Flow are not touched upon at all. These topics may be added in the future.

It is worth checking the SharePoint Guidance section of the MSDN Patterns & Practices web site at http://msdn.microsoft.com/en-us/library/dd203468.aspx as well as *Best Practices: Common Coding Issues When Using the SharePoint Object Model* at http://msdn.microsoft.com/en-us/library/bb687949.aspx.

## Customising this document

*Note that this document is mainly targeted at SharePoint Developers who work for or with Muhimbi. However, this document may be of use to external developers as well. Naturally some sections may not be relevant to those developers or may need to be revised depending on the needs and circumstances.*

*Feel free to rebrand this document for use in other organisations. We would appreciate it if the customised version of this document would accredit Muhimbi and contain a link to our website.*

*Please provide constructive feedback to guidelines@muhimbi.com.*

# Development Tools

The key tools used for SharePoint development are listed in this chapter. Feel free to use additional tools if that increases productivity, but if any of those new tools replace any of the existing ones then a discussion is required.

### Visual Studio 2008

This version of VS is installed by default on all development workstations.

### WSPBuilder

WSPBuilder is installed on each development server. Its main purpose is to create WSP files without the need to manually maintain DDF files.

Generally we only use it at the beginning of the project to create and deploy the initial feature and in the end to package it up, as creating and deploying a WSP every time you build the project slows things down. A standard XCOPY works faster.

A typical post build event looks as follows:

|  |
| --- |
| set gacutil="$(SolutionDir)..\..\SharedBinaries\GACUtil\gacutil.exe"  set wspbuilder="$(SolutionDir)..\..\SharedBinaries\WSPBuilder\wspbuilder.exe"  set useWSPBuilder=false  @REM Do we want to build using XCopy or WSPBuilder?  if %useWSPBuilder%==false goto useXCopy  @echo \*\* Build using WSP Builder  @echo \*\* Remove files from the WSPBuilder GAC Directory  del /F /Q "$(ProjectDir)GAC\\*.\*"  @echo \*\* Move dependent DLLs to GAC directory to allow WSPBuilder to package them  move /y "$(TargetDir)Muhimbi.SharePoint.Common.dll" "$(ProjectDir)GAC"  move /y "$(TargetDir)Muhimbi.SharePoint.Diagnostics.dll" "$(ProjectDir)GAC"  cd "$(ProjectDir)"  %wspbuilder% -BuildDDF True -ResetWebServer False -Outputpath   "$(ProjectDir)Solution" -Cleanup False -WSPName Muhimbi.Licensing.wsp   -BuildSafeControls False    goto end  :useXCopy  @echo \*\* Building using XCOPY  xcopy /E /Y "$(ProjectDir)12" "C:\Program Files\Common Files\Microsoft Shared\web server extensions\12"  @echo \*\* Installing GAC assemblies, revise as needed  %gacutil% /if "$(TargetDir)Muhimbi.SharePoint.Common.dll"  %gacutil% /if "$(TargetDir)Muhimbi.SharePoint.Diagnostics.dll"  %gacutil% /if "$(TargetDir)Muhimbi.Licensing.Base.dll"  %gacutil% /if "$(TargetDir)Muhimbi.Licensing.Validator.dll"  @echo \*\* Installing $(OutDir)$(TargetFileName) into the GAC...  %gacutil% /if "$(TargetPath)"  echo \*\* Recycling App Pools  "$(SolutionDir)\RecycleAppPools.vbs"  :end  @echo \*\* rebuilding sitemaps and translations only uncomment when making changes to either as it is a bit slow  REM %stsadm% -o copyappbincontent |

Note that in this example *useWSPBuilder* has been set to *false* to copy the files to the 12 hive directly.

For more information see http://www.codeplex.com/wspbuilder

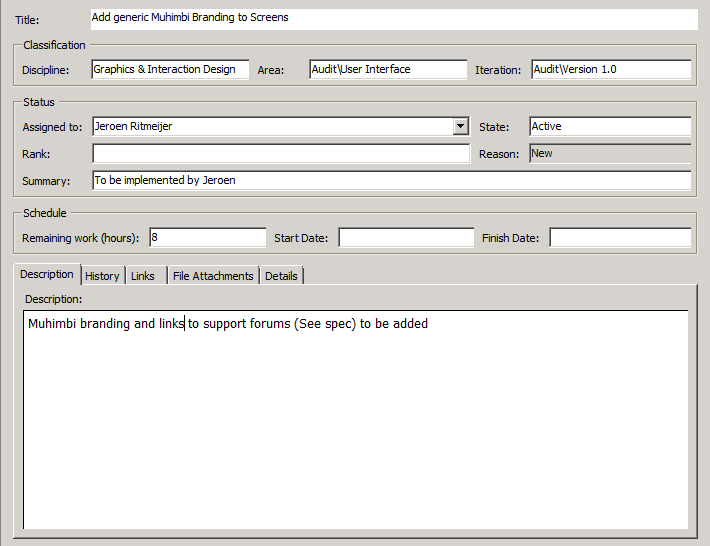
### Source Code Control

Team Foundation Server 2008 is used for Source Code control. For details see chapter 14.

### Bug & Issue tracking

Team Foundation Server 2008 is used for Bug, Issue and work item tracking. The process is fairly straight forward. Guidelines are provided below:

1. Be as specific as possible and enter as much information as possible. Paste in email trails if available.
2. During every change, fill out a description under the History Tab. Copy emails if available.
3. Specify any linked issues (Duplicated items, related issues, requirements etc) under the Links tab.
4. Screenshots and other supporting information can be added to the ‘File Attachments’ tab.
5. Enter a summary of the current status, e.g. ‘waiting for meeting with customer’ in the Summary field.
6. Setup TFS alerts to be automatically notified when new work items are assigned.



*Typical work item*

At the beginning of the project, make sure that all requirements in the Requirements Documentation are copied into the system as Work Items or Quality of Service Scenarios. This allows progress to be tracked of all requirements all the way through to the final implementation.

# User Interface

The applications we develop integrate seamlessly with SharePoint and sport the same look and feel as the facilities that ship with the platform.

This section contains some helpful guidance with regards to developing SharePoint user interfaces.

## Application and Central Administration pages

### SharePoint Web and User Controls

When developing Application Pages, developers should be careful not to ‘reinvent the wheel’. SharePoint ships with a large number of controls for capturing input, selecting web applications or picking users. All these controls share a common look and feel.

For a simple example of using the standard controls see the following blog posting: http://blog.rafelo.com/2008/10/using-inputformsection-and.html

Some tips:

1. Don’t build your own HTML unless absolutely necessary.
2. Look at existing Central Administration and Application pages stored in the 12\TEMPLATE\Admin and 12\TEMPLATE\LAYOUTS directories.
3. Only use WSS 3.0 controls. Don’t use any MOSS specific ones (see 12.4).

### Breadcrumbs

To add breadcrumbs to application pages read the following articles. Make sure to read the comments as well:

1. Regular pages:   
   http://weblogs.asp.net/jan/archive/2008/04/16/adding-breadcrumb-navigation-to-sharepoint-application-pages-the-easy-way.aspx
2. Central Administration pages: http://weblogs.asp.net/jan/archive/2008/10/10/adding-breadcrumb-navigation-to-application-pages-in-sharepoint-central-administration.aspx

Some tips:

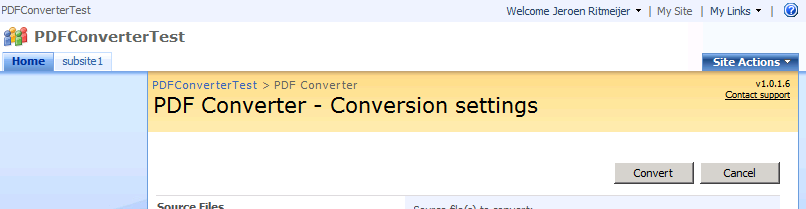
1. If your breadcrumb is not displayed in Central Administration screens then make sure the page uses the ~/\_admin/admin.master master page file.
2. Locations for use in the Breadcrumbs are deployed using .sitemap files. Make sure these files are processed by SharePoint and copied to the correct folders using an event receiver (***see #1***).
3. Sitemaps for Central Administration are stored in 12\Template\Admin. Make sure the file is unique by including the feature name in the file name.
4. Sitemaps for regular Application Pages are stored in 12\Template\Layouts. Make sure the file is unique by including the feature name in the file name.
5. Do not add elements to the sitemap programmatically unless absolutely necessary.

## Company specific branding

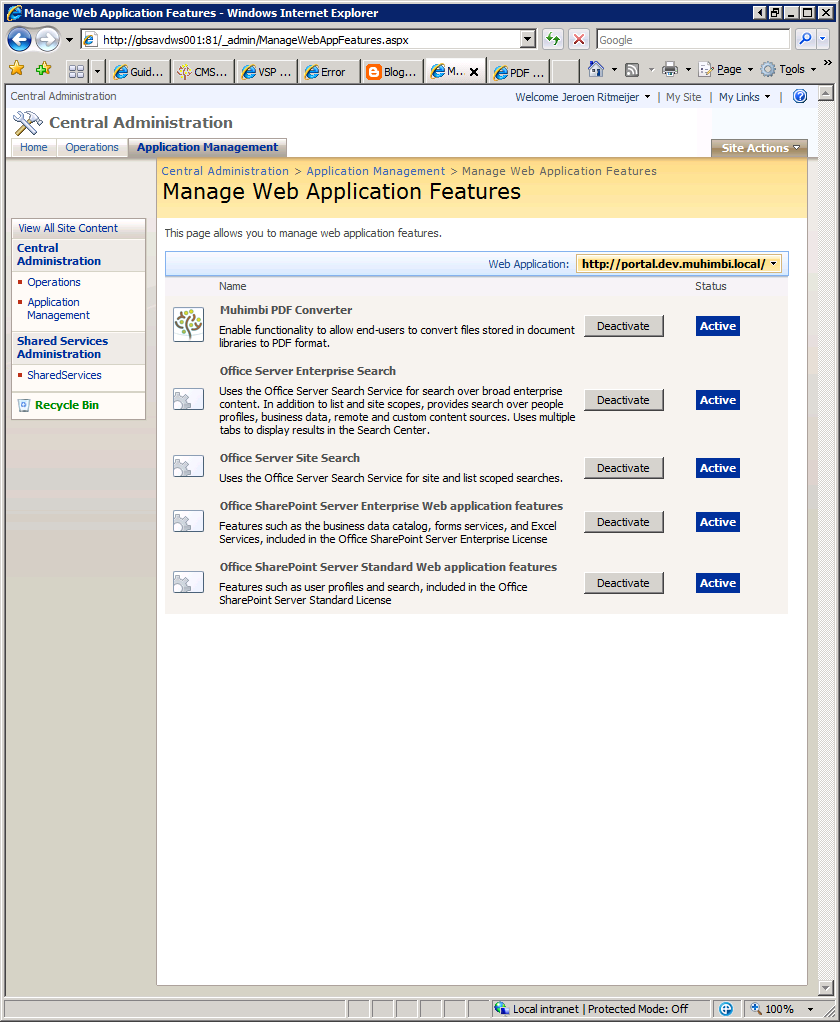
Muhimbi applies very limited branding to their applications. After all, most of our applications will run as part of someone else’s system. Having said that, the following visual elements will need to be added to all our applications.

1. Version number
2. Link to the support forum on the Muhimbi site
3. Muhimbi branded feature image (ImageURL)

This should be positioned in a very unobtrusive, but still visible, place. For example:



*Version number and support link added to an Application Page*



*Muhimbi branded feature icon*

This can be added as outlined below. Note that this is just sample code, feel free to refactor in such a way that it fits in with other libraries used by the solution.

### aspx page

Place the following code as the first line in the ‘PlaceHolderPage  
TitleInTitleArea’ content placeholder:

<asp:Label ID="lblMuhimbiVersion" runat="server" style="float:right; width:100px; position:relative; top:-14px; text-align: right; font-family:Tahoma; font-size:7pt; margin-right:8px"/>

### aspx.cs (Code behind) page

Place the following code in the code behind page and call it from the OnLoad method. Make sure to add the name of the resource file (without .resx)

/// <summary>

/// Render the version number and a link to the Muhimbi support site

/// </summary>

private void RenderVersionNumber()

{

AssemblyFileVersionAttribute fileVersion = (AssemblyFileVersionAttribute)  
 Attribute.GetCustomAttribute(Assembly.GetCallingAssembly(),  
 typeof(AssemblyFileVersionAttribute));

string version = fileVersion.Version;

string versionTemplate = (string)HttpContext  
 .GetGlobalResourceObject(*resourceFile*, "muhimbi\_message");

lblMuhimbiVersion.Text = string.Format(versionTemplate, version);

}

Make sure lblMuhimbiVersion is properly declared:

protected Label lblMuhimbiVersion;

### Resource file

Place the following in the resource files for the various languages:

<data name="muhimbi\_message" xml:space="preserve">

<value><![CDATA[v{0}<br /><a href="http://www.muhimbi.com/support.aspx" style="text-decoration:underline">Contact support</a>]]></value>

</data>

### Feature definition

Place ‘muhimbi\_feature.gif’ in the ‘12/template/images/*feature name*’ folder and add the following attribute to the feature defintion:

ImageUrl="feature name/muhimbi\_feature.gif"

## ‘Code Behind’

Do not place any server side source code in-line. Presentation based logic should go in a ‘code behind’ class and business logic should be placed in separate classes.

This is similar to traditional ASP. NET development, the difference is that you need to create the ‘code behind class’ manually.

## Style Sheets

SharePoint ships with a large number of CSS styles out of the box. To ensure that our applications automatically adopt whatever custom style sheet is in use and adopt the native look and feel, we need to make sure the relevant styles are applied to all paragraphs, table cells etc.

For a visual reference of the SharePoint styles, see

<http://www.heathersolomon.com/content/sp07cssreference.htm>

## Browser compatibility

Although Internet Explorer is the most commonly used browser when we are talking about SharePoint, we need to make sure that all our applications work as well, or as badly, in other browsers as the Out Of The Box SharePoint facilities do. Therefore all applications need to be tested in:

* IE6+7+8
* Firefox 2+3,
* Safari,
* Google Chrome

## Screen resolution

Our applications are optimised for 1024x768 screens, but must be usable on 800x600 as well as very high resolutions (1600x1050).

# Internationalisation / Localisation

Our applications target a worldwide audience. Therefore it is essential that we take language and regional differences into account.

## Multi lingual support

SharePoint provides excellent support for multiple languages. SharePoint allows anything including Application Pages, Breadcrumbs and Feature descriptions to be translated.

For Details see:

1. http://tomblog.insomniacminds.com/2008/02/25/sharepoint-internals-resources/
2. http://www.mikhaildikov.com/2007/03/sharepoint-resources-types-use-and\_2163.html

Some tips:

1. Similarly to sitemap files, translation files need to be copied to the correct directories as part of a feature installation. Unfortunately SharePoint does not make it easy to activate this information to multiple Web Front End Servers. Therefore a generic jobs has been developed by Muhimbi that ‘refreshes’ this information on all servers in a webfarm. It is used as follows:
   1. Reference Muhimbi.SharePoint.Common
   2. For each feature that contains breadcrumbs and / or translations that go into the inetpub......./App\_GlobalResources directory call the following method in the FeatureInstalled event:

*FeatureInstallationJob.ScheduleJob(FeatureInstallationJob.ApplicationContentDestinations.WebApplications, properties.Definition.DisplayName);*

* 1. Note that *ApplicationContentDestinations* is an enumeration. Please pick the appropriate value depending on where the feature / translations are being used.

1. Make sure that during day to day development ‘locale independent’ Resource files are used and maintained. Toward the end of the development cycle these can then be translated.
2. Resource files for Feature Descriptions and Custom Actions are located in:

\12\TEMPLATE\FEATURES\<feature>\Resources\Resources.<culture>.resx

1. Make sure that ‘RequireResources’ is set to ‘false’ in feature.xml. Otherwise locales for which no translation is available will not fall back to the English translations.
2. Set the ‘Build Action’ for each resource file to ‘None’. Otherwise the GAC will contain a separate entry for each supported locale.
3. Resource files for Application pages are stored in:

\12\CONFIG\Resources\

1. Resource files for Central Administration Application pages are stored in:

\12\CONFIG\AdminResources\

1. Content in resource files can be referenced as follows:
   1. C#: HttpContext.GetGlobalResourceObject("MyResource", "MyName").ToString();
   2. ASPX Properties: <%$Resources:MyResource, MyName%>
   3. ASPX Text: <asp:literal runat="server" Text="<%$Resources:MyResource, MyName%>" />
   4. XML (features, sitemaps): $Resources:MyResource, MyName

## Dates, numbers and currencies

Different locales use different formats for dates, numbers and currencies. Don’t assume everyone uses the same formats as you do. Always make sure the locale specified for the current SPWeb / user is correctly applied to the application under development.

The user’s culture can be retrieved from the CurrentThread as per the example below.

|  |
| --- |
| colName = new BoundField();  colName.DataField = "LicenseExpiry";  colName.HeaderText = "some header"; //TODO: load from resource file colName.SortExpression = "LicenseExpiry";  colName.DataFormatString = "{0:" +  System.Threading.Thread.CurrentThread.CurrentCulture.  DateTimeFormat.ShortDatePattern + "}";  colName.HeaderStyle.Width = new Unit("5%");  colName.ItemStyle.Width = new Unit("5%");  licenseGrid.Columns.Add(colName); |

Don’t assume that SharePoint will take care of everything automatically. It does in many cases, but there are some exceptions so pay special attention during testing.

See <http://blog.bittercoder.com/PermaLink,guid,51d881b6-8480-4c7c-adb4-df5e1e6db1ce.aspx>

Note that it is common practice to store timestamps in UTC format, e.g. the Audit Log stores information this way. Make sure to convert any timestamps saved in this time zone to the user’s time zone using the DateTime.ToLocalTime() method.

# Coding guidelines

## Development Language

All server side code is developed in C#. Any language construct that works with the .net Framework 3.0 can be used, so Linq is out of the question as it was not introduced until version 3.5.

Any client side code is written in JavaScript. For details on which browser versions need to be supported see 3.5 *Browser compatibility*.

## File headers

Some of our licenses provide access to the full source code. Therefore it is essential that a header is added to each file including XML, ASPX, CS and CONFIG files.

The header is as follows:

|  |
| --- |
| '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  ' <Solution Name>  '  ' Copyright 2009, Muhimbi Ltd  ' www.muhimbi.com  '  ' All rights reserved  '  ' Document author: <original author name>  '  ' FILENAME : <filename>  ' PURPOSE : <small description>  ' TO-DO : <any major to-do topics>  ' KNOWN ISSUES : <any major known issues>  ' HISTORY : <date> - <initials> - <comments>  ' <date> - <initials> - <comments>  '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |

Please comment out the header using whatever facility is applicable to the file format.

Note that the best way to comment out in ASPX pages is to use the <%-- --%> syntax as anything else may cause problems when using content place holders.

Unfortunately the header needs to be maintained by hand as our Source Code Control System (TFS2008) does not support dynamic keyword expansion like Visual SourceSafe does.

## Commenting

Over time our solutions are maintained by different developers. In addition to our own internal and external developers, some of our customers have access to the source code as well. Therefore it is essential that all code is commented in line with Microsoft Best practices.

Please provide comments for:

1. Classes.
2. Individual Items in enumerations.
3. All Methods, including constructors and getters / setters.
4. Inline in code where necessary.
5. Anywhere else where it makes sense.

Some languages, such as C#, provide a native commenting system (see <http://msdn.microsoft.com/en-us/library/b2s063f7.aspx>), other languages and file formats require custom comments.

For example a JavaScript function header looks as follows:

|  |
| --- |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  FUNCTION : <function name>  PURPOSE : <brief description>  PARAMETERS: <param name1> - <description>  <param name1> - <description>  RETURNS: <details of what the function returns> \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |

***All comments should pass a spell checker. Even though software development can be frustrating at times, do not use any profanity or potentially insulting language.***

## Code Formatting

In addition to being commented well, properly formatted code is easier to maintain. Please take the following into account:

1. Indent your code properly.
2. Break up long statements.
3. Break up long methods.
4. Use C# ‘Regions’ to divide the code into sections.
5. Space your code: Don’t leave empty lines all over the place, a single line between code fragments and two lines between function definitions will do.
6. Watch you tab size. Try to use your development environment’s default tab size as this is most likely the tab size your fellow coders are using. Try not to mix tabs and spaces to indent your code. (Spaces are preferred)
7. Remove unused code. Don’t be afraid to delete someone else’s lines. If needed we can always get it back from the Source Code Control System.
8. Place the opening brace on a newline. Wars have been fought over this one, but agree with it or not, it is our standard. So use

private void SomeMethod()

{

DoSomething();

}

And not

private void SomeMethod() {

DoSomething();

}

1. Be consistent.

## Naming conventions

### C#

We use the standard naming conventions for C# as defined by Microsoft at <http://msdn.microsoft.com/en-us/library/xzf533w0.aspx>.

In Summary:

|  |  |  |
| --- | --- | --- |
| Identifier | Case | Example |
| Class | Pascal | AppDomain |
| Enum type | Pascal | ErrorLevel |
| Enum values | Pascal | FatalError |
| Event | Pascal | ValueChange |
| Exception class | Pascal | WebException  Note   Always ends with the suffix Exception. |
| Read-only Static field | Pascal | RedValue |
| Interface | Pascal | IDisposable  Note   Always prefix with I. |
| Method | Pascal | ToString |
| Namespace | Pascal | System.Drawing |
| Parameter | Camel | typeName |
| Property | Pascal | BackColor |
| Protected instance field | Camel | redValue  Note   Rarely used. A property is preferable to using a protected instance field. |
| Public instance field | Pascal | RedValue  Note   Rarely used. A property is preferable to using a public instance field. |

The same naming convention is used for JavaScript code.

### Namespaces

Our predefined namespaces are as follows:

1. Muhimbi.SharePoint: Root namespace for all our SharePoint applications.
2. Muhimb.SharePoint.*<SolutionName>*: Root namespace for all code specific to a solution.

Please apply some logic to any further namespace refinements. Don’t go overboard.

### Feature names

Feature names should be recognisable and to the point, generally:

Muhimbi.<Solution Name>

Don’t be tempted to use the entire namespace.

Please use the Feature Name to divide assets (images / layouts pages etc) in the Layouts directory.

# Exception Management & Logging

Proper exception handling and trace logging is essential for any application and SharePoint is no exception.

Although an exhaustive discussion about exception management is out of the scope of this document, some helpful pointers are provided below:

1. Errors displayed for the end user should always be user friendly. Business users don’t know how to decipher a stack trace, all they should see is a friendly, but relevant, error message *using proper spelling and grammar*.
2. Errors displayed for the end user should always be displayed using the correct Locale. For details about how to support multiple languages see chapter 4 .
3. To make it easy to detect what has gone wrong, always enrich Exceptions with as much local state as possible. For example:

|  |
| --- |
| try  {  // ... Actual implementation  }  catch(Exception ex)  {  // \*\* Enrich, log and rethrow it to the calling method.  String errorMessage = String.Format("An error occurred in DoWork.\n" +  " - customerID: {0}\n" +  " - sessionID: {1}\n" +  " - itemsFound: {2}\n",  customerID, sessionID, itemsFound);  SomeException newEx = new SomeException(errorMessage, ex)  // \*\* Handle the exception in a centralised exception handler  Muhimbi.SharePoint.Common.Diagnostics.EventLog.Write(newEx);  throw (newEx);  } |

1. At a minimum, every entry point of an asp.net page / user control should contain an exception handler and deal with the logic of converting an exception to a user friendly message. Common entry points are:
   1. Page Load event.
   2. Button Click events.
   3. Data grid paging, filtering and sorting events.

For details and additional help with Exceptions see the following section on the MSDN site: <http://msdn.microsoft.com/en-us/library/ms173160.aspx>.

## Writing to the SharePoint Trace log

The SharePoint Trace Log contains valuable information that allows System Administrators to identify and troubleshoot problems at a level that is generally more detailed than the Event Log.

To assist these Administrators all Muhimbi applications must write information to this log. At a minimum write the following information:

1. All exceptions, enriched with local state (TraceSeverity.Exception).
2. Unexpected events (TraceSeverity.WarningEvent).
3. Information Events (TraceSeverity.InformationEvent): Anything that may be of interest for troubleshooting purposes, e.g.:
   1. Feature Installed / uninstalled
   2. EventReceiver event triggered
   3. Some major functionality of the application accessed such as a page converter or audit viewer.

While debugging it may be useful to add debug (TraceSeverity.Verbose) messages.

The SharePoint Object Model does not allow entries to be written to the Trace Log. However, Muhimbi have implemented the TraceProvider class, located in Muhimbi.SharePoint.Common.Diagnostics. Use one of the following overloaded methods to write messages:

1. WriteTrace(Exception ex)
2. WriteTrace(string message, Exception ex)
3. WriteTrace(string message)
4. WriteTrace(string message, TraceProvider.TraceSeverity severity)
5. WriteTrace(string message, TraceProvider.TraceSeverity severity, Assembly assembly)
6. WriteTrace(uint tag, TraceSeverity level, Guid correlationGuid, string exeName, string productName, string categoryName, string message)

It is unlikely that you will ever need to use #5 and #6.

Useful Utilities:

1. SPTraceView: Real-time monitor for the SharePoint trace log, allows filtering and alerting for events. Very useful during development. http://hristopavlov.wordpress.com/sptraceview/
2. WSS Log File Reader: An add-in to access log files directly from Central Administration. Much friendlier than going through the logs using notepad. (http://www.codeplex.com/wssmosslogfilereader/). Make sure to download a release newer than the one published on the 15th of November 2008 as earlier versions don’t deal with log files that are stored in any location other than the default one.

Note that the trace logs can be configured using *Central Administration > Operations > Diagnostic Logging*.

Muhimbi’s TraceProvider class is based on code published by Microsoft. For more details see http://msdn.microsoft.com/hi-in/library/aa979590(en-us).aspx.

## Writing to the Event log

The Windows Event Log is often the first place administrators will look in order to troubleshoot a problem.

To assist these Administrators all Muhimbi applications must write information to this log. At a minimum all Exceptions should be written to the event log.

Muhimbi have created a custom EventLog class[[1]](#footnote-1), located in Muhimbi.SharePoint.Common.Diagnostics. Use one of the following overloaded methods to write messages:

1. WriteEntry(Exception ex)
2. WriteEntry(string message, Exception ex)
3. WriteEntry(string message)
4. WriteEntry(string message, EventLogEntryType eventLogEntryType)
5. WriteEntry(string message, EventLogEntryType eventLogEntryType, Assembly assembly, int? eventID)

Note that any events written to the event log are automatically written to the SharePoint Trace Log. Therefore it is not necessary to log Exceptions separately to the trace log.

For an example see the introduction of this chapter.

# SharePoint Development & Technologies

The majority of the topics in this document apply to both SharePoint as well as general purpose development. This chapter, however, provides helpful pointers and guidelines for a number of SharePoint specific technologies.

Note that this is not a Tutorial or training manual. Developers should already be familiar with SharePoint.

## Application Pages

Application pages are relatively static pages that cannot typically be modified by end users. An example of an Application page is the settings page (/\_layouts/settings.aspx).

Listed below are some helpful pointers

1. Do not use in-line server side code. All presentation code should go in a ‘code behind’ class and business logic should be placed in separate classes.
2. ‘Code behind classes’ inherit from *LayoutsPageBase*.
3. Application Pages must display a Breadcrumb. For details see 3.1.2.
4. The MasterPageFile is “~/\_layouts/application.master”
5. All files for custom Application Pages must be stored in 12\TEMPLATE\LAYOUTS\<Feature Name>.
6. All static text should be stored in a Resource file. For details see 4.1 *Multi lingual support*.

Some additional details specific to the user interface of Application pages are outlined in chapter 3.

Note that Application and Administration pages are closely related, however some differences are discussed in 7.2 Administration Pages.

## Administration Pages

Administration Pages are quite similar to the previously discussed Application Pages. However, they can only be accessed from the Central Administration site.

In addition to the guidelines specified in 7.1, the guidelines are:

1. The MasterPageFile is “~/\_admin/admin.master”. Quite often we see that an existing Application Page is re-used as an Administration page. If the MasterPageFile is not updated then breadcrumbs do not display properly.
2. All files for custom Administration Pages must be stored in 12\TEMPLATE\ADMIN\<Feature Name>.

## Event Receivers

Make sure that all entry points of an Event Receiver writes entries of type InformationEvent to the SharePoint TraceLog. For details see 6.1 .

## Web Parts

As Muhimbi develops commercial software, all Web Parts must be implemented using traditional methods. Although highly productive for enterprise development, tools such as SmartPart may not be used.

## Features & Custom Actions

When writing Custom Actions don’t just copy an existing one from a website without considering the following:

1. All static text should be loaded from Resource Files. See section 4.1.
2. Consider the Scope of the Feature
3. Does the Version number need to be updated? See section 8.1.
4. Should the Feature be hidden?
5. Has a relevant ImageURL been identified?
6. Use sensible values for ID fields, e.g. Muhimbi.SharePoint.Administration.  
   Operations.Licensing.LicenseManager
7. Don’t position them just anywhere, give sequence numbers some consideration.

For more details see:

1. Custom Actions: <http://msdn.microsoft.com/en-us/library/ms460194.aspx>
2. Action Locations: <http://msdn.microsoft.com/en-us/library/bb802730.aspx>
3. Feature Scopes and functionality allowed per scope: <http://weblogs.asp.net/soever/archive/2007/05/03/sharepoint-features-elements-scope-and-other-info.aspx>

## Maximum file and folder lengths in SharePoint

There are some restrictions in SharePoint with regards to the maximum file and folder name lengths as well as valid characters. For details see:

<http://blogs.msdn.com/joelo/archive/2007/06/27/file-name-length-size-and-invalid-character-restrictions-and-recommendations.aspx>

## SPGridview

To ensure that all tables automatically inherit the SharePoint look and feel as well as filtering functionality, use SPGridView when displaying tabular data.

For some excellent examples see:

1. Basics:  
   <http://blogs.msdn.com/powlo/archive/2007/02/25/displaying-custom-data-through-sharepoint-lists-using-spgridview-and-spmenufield.aspx>
2. Paging:  
   <http://blogs.msdn.com/powlo/archive/2007/03/23/adding-paging-to-spgridview-when-using-custom-data-sources.aspx>
3. Filtering: <http://www.sharepointblogs.com/bobsbonanza/archive/2007/07/02/filtering-with-spgridview.aspx>

Make sure the size of the ViewState does not become unmanageable when the grid contains a large amount of data.

For large grids make sure that calculating and displaying the filter options doesn’t take an unacceptable amount of time.

## Storing settings

SharePoint provides a number of different facilities for storing settings. There is no ‘one size fits all’ solution so use your judgement when selecting a technology:

1. **Hierarchical Object Store**: The HOS allows settings to be attached to virtually any SharePoint object (SPFarm, Web Application, List etc). For an example see <http://www.bluedoglimited.com/SharePointThoughts/ViewPost.aspx?ID=271>   
   It is particularly useful in combination with Custom Timer Jobs: <http://msdn.microsoft.com/en-us/library/cc406686.aspx>

*Please note that when a new version of a DLL that uses the HOS is released (with a new strong named version number) it cannot read entries written by a previous version, even when the implementation is exactly the same. Another reason to keep the version number in the Strong Name always the same (See 8.1)*

1. **SPPropertyBag**: Some of the key objects in SharePoint such as SPFarm, SPWebApplication and SPWeb contain a *Properties* collection that can be used to store name / value pairs. It is not as flexible as the HOS, but more often than not it is good enough and very easy to use. For an example see <http://sharepoint.nailhead.net/2008/06/managing-key-value-pair-settings-in.html>
2. **Web.config:** SharePoint 2007 makes it relatively simple to modify the web.config, however in our opinion it should be used sparingly as any modification to the web.config will restart the web application. Only use it when you really have to.
3. **Lists:** Sometimes the best option is to store settings in a standard SharePoint list. This may make sense if end users should be able to update the settings. It is not always the best option as Lists are tied to a Site Collection and need to be created in code.

Unless there is a good reason to use something else, our preference is to use options #2 and #1 (in this order).

## Modifying the web.config

Certain changes can only be made by modifying the web.config, e.g. adding authorizedTypes for custom workflow actions.

Changes can be made either declaratively or programmatically. For simple changes, which are unlikely to ever change, use the declarative approach outlined at <http://msdn.microsoft.com/en-us/library/ms439965.aspx>.

For more complex changes, or changes that will likely be updated in future versions of our software, use the programmatic approach.

## Elevating privileges

Sometimes a piece of code needs access to resources the current user may not have access to. In such cases the security context can be elevated using a call to *SPSecurity.RunWithElevatedPrivileges*.

Guidelines and tips:

1. Only use this when absolutely needed. It is extremely bad practice to just run everything in elevated mode.
2. Recreate new SPSite objects inside the elevated code. Don’t use the one that comes straight of SPContext.

For more information see:

<http://dhunter-thinkingoutaloud.blogspot.com/2007/07/sharepoint-2007-running.html>

## Preventing memory leaks

It is easy to leak a lot of memory in SharePoint, even when you think you are disposing your SPSite objects properly. Listed below are a number of resources that detail common memory leaks and their solutions:

1. SharePoint Dispose Checker Tool: <http://code.msdn.microsoft.com/SPDisposeCheck>
2. Various situations in SharePoint that require a Dispose: <http://blogs.msdn.com/rogerla/archive/2008/02/12/sharepoint-2007-and-wss-3-0-dispose-patterns-by-example.aspx>
3. Troubleshooting and detecting leaking SharePoint objects: <http://blogs.technet.com/stefan_gossner/archive/2008/05/07/troubleshooting-spsite-spweb-leaks-in-wss-v3-and-moss-2007.aspx>
4. Dealing with memory pressure problems in SharePoint: <http://blogs.technet.com/stefan_gossner/archive/2007/11/26/dealing-with-memory-pressure-problems-in-moss-wss.aspx>
5. Generic article on preventing memory leaks in .net: <http://msdn.microsoft.com/en-gb/magazine/cc163491.aspx>

# Dealing with version numbers

The .net framework provides a rich infrastructure for dealing with versioning of assemblies. It is tempting to follow best practices and automatically increase version numbers between releases. However, in the context of SharePoint this is usually not a good idea.

What follows is based on real world experience after releasing several incremental versions of our products. It is not pretty, but it works in the real world.

## Determining the ‘strong named’ version number

This section explains the guidelines for generating the version number that is part of the Assembly’s *strong name*.

This version number is determined once for a product and never updated, not even during subsequent releases. The reasons for this include, but are not limited to the following:

1. If the product is used from a SharePoint Workflow then all workflows using the product will be invalidated if the version number changes.
2. Any web parts used by the system will be invalidated if the version number changes.
3. Data written to the Hierarchical Object Store using one version cannot be read back using a different version of the product.

Assembly version numbers are normally specified as follows: *major.minor.build.revision*. This strategy does not work for SharePoint, which is why we generate the version number as follows:

1. Major: Always 1
2. Minor: Always 0
3. Build & Revision: Both the same and equal to the Product ID (See section 9 )

The reason for the Revision number being the same as the Build number is related to a bug in the SharePoint workflow engine. On some systems the build and revision number are sometimes switched around, resulting in SharePoint not being able to find the DLL with the specified version number.

For details see <http://social.msdn.microsoft.com/Forums/en-US/sharepointworkflow/thread/db99b3c0-e56f-41c1-8991-6e4a0cc33a1e>.

## Determining the real version number

The *real* version number, the number that really indicates which release you are running, is incremented every time a new version of our software is released.

Rather than storing this version number in the *AssemblyVersion* attribute (used as part of Assembly’s strong name), we store it in the *AssemblyFileVersion* attribute.

This attribute is queried at runtime and displayed in the user interface using the code in section 3.2.

# Sharing Libraries and DLLs between SharePoint solutions

Any software development company, or IT department, that develops solutions usually have one or more shared libraries that contain generic functionality shared by multiple products or projects.

Due to a flaw in the SharePoint deployment mechanism - the lack of reference counts for shared files - implementing shared libraries is troublesome, though not impossible.

## The problem

Under normal circumstances (products / projects that don’t use SharePoint) the process is as follows:

1. Maintain shared libraries somewhere in your source tree.
2. Periodically compile this code into a shared DLL.
3. Branch this DLL into your project.
4. Reference this DLL from your project.

Let’s illustrate why this doesn’t work in SharePoint using an example:

1. We have 2 different products named *A* and *B* that both use a shared DLL named *C*.
2. Both products are deployed as a SharePoint Solution (wsp) file. This has the following effect:
   1. When Product A is deployed, assemblies A and C are installed, e.g. in the GAC
   2. When Product B is deployed, assembly B is deployed and if you are lucky Assembly C is deployed as well. If you are not lucky Assembly C is in use by IIS and locked resulting in a deployment error.
3. It is decided that product B is really not needed and is retracted from SharePoint. Unfortunately this removes the shared Assembly C as well, leaving Product A broken.

Although you may be able to document your way around this in a typical enterprise IT department, there is no way we can afford these kinds of problems in commercial products.

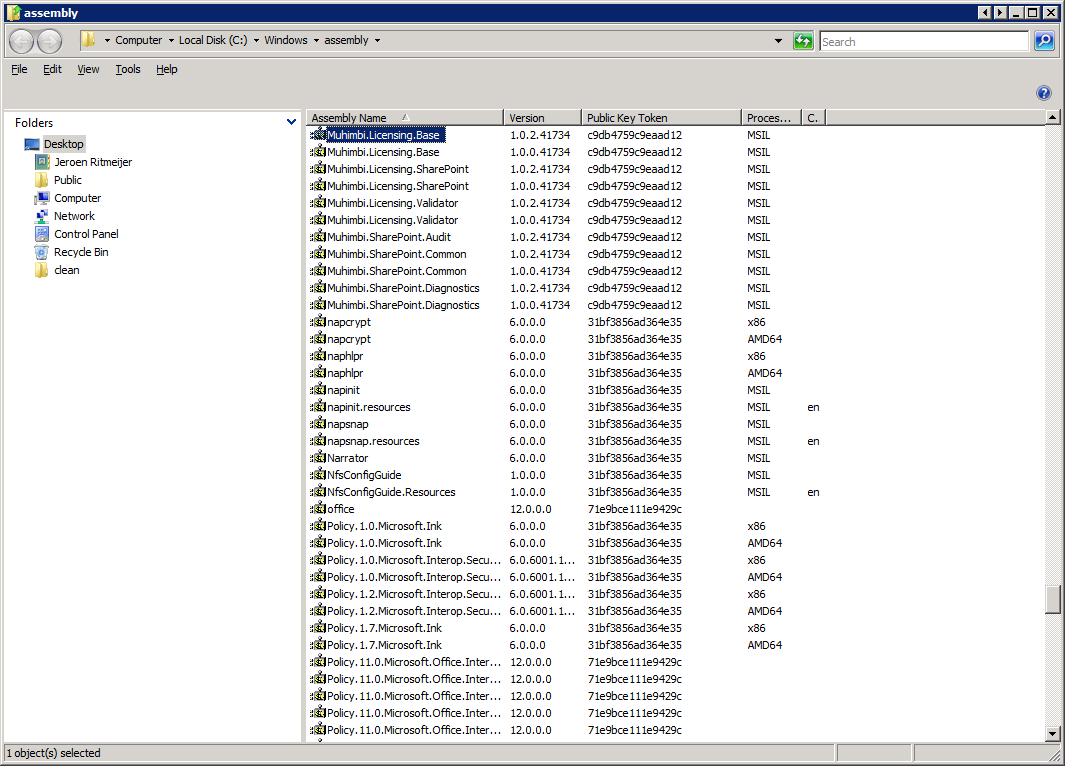
## The Solution for First party libraries

First Party Libraries, aka the libraries you write and control, can be shared between SharePoint solutions by branching the source code into the product and then making the version number the same as the product’s version number.

The process is as follows:

1. Maintain the source of the shared library somewhere central in your source control system.
2. When a project needs this library, the source is branched into the project.
3. The version number of the branched source code is synchronised with the project it is embedded in (we use a shared SolutionInfo file).
4. During the development of the project, changes can be made to the branched source code.
5. The build script compiles all assemblies, including this customised shared library and places it in the WSP file.
6. At the end of the project the branched code is merged back into the main repository.

This potentially leads to some duplication, but considering the average .net library is well below 100KB, this is generally not a problem.



*Different versions of the same assemblies in the GAC[[2]](#footnote-2)*

The following Product IDs (last 2 digits of strong named version number) are currently in use:

* License Manager: 0
* PDF Converter: 1
* SharePoint Audit: 2

## The Solution for Third party libraries

When third party libraries are shared between products then we don’t have the luxury of being able to synchronise all version numbers with the main assembly. We could jump through some hoops and ‘re-version’ someone else’s library, but that may lead to all kind of other problems.

Instead we use *ilmerge*, a command line utility from Microsoft research, to merge any third party libraries into our own DLLs as part of the build process.

ilmerge /ndebug /internalize /keyfile:muhimbi.snk /out:"OurProduct.dll" "OurProduct.dll" "Some Shared Lib.dll"

Full details can be found on the ilmerge website at:

<http://research.microsoft.com/en-us/people/mbarnett/ILMerge.aspx>.

# Scalability

As some SharePoint environments exceed 100.000 users it is essential that Scalability of any solution is taken into account.

A full discussion about writing scalable application is outside the scope of this document, but the key tips and guidelines are:

1. Always assume that your application will be deployed across multiple Web Front End servers. So be careful if and where to maintain state and make sure any binaries / configuration changes are automatically deployed to all machines.
2. Repeatedly accessing data that is expensive to retrieve and doesn’t change frequently is a prime candidate for caching.
3. When dealing with long running requests do not create new threads, but rather create a job and poll for the results.

# Security

SharePoint environments typically contain highly sensitive information. It is essential that security is taken into account while developing applications. *It is assumed that all Muhimbi developers are aware of the NSA’s list of the Top 25 Most Dangerous Programming errors (*[*http://www.sans.org/top25errors/*](http://www.sans.org/top25errors/)*).*

## SSL

Typically no work is required by the developer to support SSL. However take into account that sometimes your application may run in an environment where URLs start with ‘https’. Be careful when requesting the URL of a page or object as it may be returned with an http prefix.

## Anonymous Access

Take into account that sometimes your solution will run on a SharePoint site that allows anonymous access. As a result it may not be possible to tell users apart or request their credentials.

## Forms Authentication

Although generally SharePoint runs in an Active Directory environment, some Web Applications may have been configured to allow Forms Based Authentication.

Make sure your application is tested using this kind of Authentication.

## Audit log

SharePoint comes with a built in auditing facility. Not only can the audit log be queried from custom applications, it is also possible to write to it.

Whenever an event occurs that is worth auditing, and is not already audited by SharePoint, then make sure an entry is written to the audit log. Before writing such an event, check that auditing is enabled as SharePoint will not automatically perform this check for you.

Note that access to Application (\_layouts) and Administration (\_admin) screens is not logged by SharePoint, so make sure a ‘View’ audit entry is always written whenever any of these screens are accessed.

Do not confuse the Audit log with the SharePoint Trace log discussed in section 6.1.

For an example on writing data to the Audit log see: <http://msdn.microsoft.com/en-us/library/bb466221.aspx>.

## Validate user data / Prevent cross-site scripting attacks

Always make sure any data entered by users is validated on the Server. Prevent cross-site scripting attacks by filtering out any <script> tags and other executable content.

For more information on Cross Site Scripting and how to prevent it (including a .net library) see <http://msdn.microsoft.com/en-us/library/aa973813.aspx>.

## Non privileged users vs Administrators

During the development cycle the Windows account used by a developer is typically a local Windows Administrator as well as a SharePoint Farm Administrator. This gives the developer almost limitless access to all data in SharePoint and SharePoint will rarely put up any security barriers.

However, actual users of the application are typically non-privileged domain users who only have access to those parts of SharePoint they have been explicitly given access to. This may cause all kind of Security exceptions that need to be catered for by either displaying clear error messages or by elevating relevant parts of the application (See section 7.9).

In other words, before releasing the software to the testers, always make sure an initial test has been carried out using a low privileged test account.

## Trust levels

SharePoint makes extensive use of CAS (Code Access Security) to control which code is allowed to be executed.

This is an important topic. Make sure you familiarise yourself with it. For details see:

<http://daniellarson.spaces.live.com/blog/cns!D3543C5837291E93!1963.entry>

Some tips and guidelines:

1. Always develop using the WSS\_Minimal level as specified in the web.config.
2. Deploy custom CAS policies as part of the WSP file.
3. Files in the GAC are not subject to CAS and have full permissions. Don’t use this as an excuse to just throw everything in the GAC.

# Compatibility

SharePoint can be deployed on a diverse combination of hardware, software and at numerous patch levels. Listed below are some topics about compatibility to ensure our applications run on the majority of SharePoint deployments.

## 32/64 bit

SharePoint can run in both 32 and 64 bit environments. When doing ‘regular’ .net development there should be no need to do anything special unless you interact with any native code.

To ensure everything is working correctly, make sure to test in both 32 and 64 bit environments.

For some typical 64-bit ‘gotchas’ check out the following article:

<http://www.sharepointblogs.com/johnwpowell/archive/2007/07/19/64-bit-gotchas.aspx>

## Windows Server 2003 / 2008

SharePoint is typically installed on either Windows Server 2003 or 2008. Make sure to test in both environments.

## Different patch levels

Since SharePoint 2007 was released, a number of Service Packs and patches have been made available. Make sure that that any application is tested with the following versions:

1. RTM (Original release). Note that this one does not run on Windows Server 2008
2. SP1
3. SP2
4. Any other Service Packs.

For an overview of all service packs and patches see the following article:

<http://www.harbar.net/articles/postsp1.aspx>

## WSS3 vs MOSS

All solutions provided by Muhimbi support both WSS and MOSS. Therefore it is essential to test any solution on both WSS and MOSS.

## .net Framework version

SharePoint 2007 requires at a minimum version 3.0 of the .net Framework. Although newer versions of the framework support all kind of interesting toys such as Ajax and Linq, we have to make sure all our solutions work on ‘clean’ SharePoint deployments that may not have been upgraded to the latest and greatest framework.

So, make sure tests are carried out on systems that only have version 3.0 of the framework installed.

## International versions

Like many other Microsoft products, SharePoint is available in a large number of different languages, either as a fully localised version or via language packs.

As described in section 4 , all Muhimbi solutions support multiple locales. Therefore it is essential that all solutions are checked on:

1. A SharePoint installation with multiple Language Packs installed
2. A non English fully localized version of SharePoint.

For additional details see the following articles:

1. <http://blogs.msdn.com/sharepoint/archive/2007/04/27/additional-information-about-wss-3-0-and-moss-2007-language-packs.aspx>
2. Language pack installation guide: <http://technet.microsoft.com/en-us/library/cc288518.aspx>

# Creating a new project

Whenever a new project is created from scratch a number of steps need to be carried out.

## In TFS

The new Project needs to be created in TFS first. Follow the steps outlined below:

1. Login as a member of the TFS Administrators group
2. Open the Team Explorer in Visual Studio, right click on the root node and select ‘New Team Project’
3. Fill out the wizard and select the ‘Muhimbi Agile’ process template.
4. Assign the default work items to the relevant team members and action all items.
5. Copy the content of ‘$/TFS/Source code Folder Structure Template’ to the Source Control for the new project.
6. Check in the folder structure. Make sure to include all items market as ‘Excluded’.

## On the developer workstation

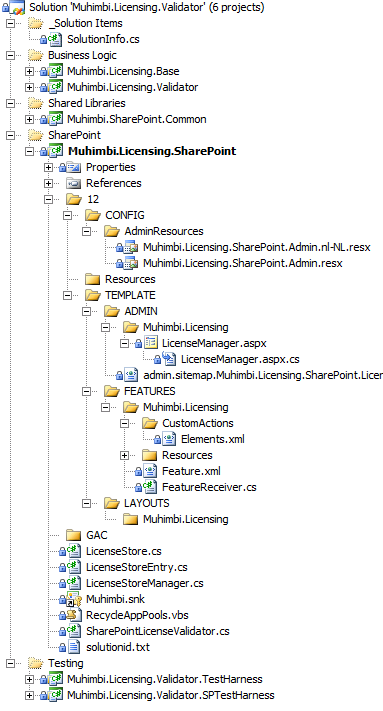
Each developer participating in the project will need to carry out the following steps.

1. The first time TFS is used by a developer the TFS server needs to be added to the Team Explorer.
2. Once the server has been added, click on the ‘Add Existing Team Project’ icon in the Team Explorer and select the relevant project.
3. Once the project has been loaded, double click the Source Control icon and map the root TFS node (not the new project) to f:\Projects\<username>. For details see 14.2 .
4. Setup alerts to track all check-ins and work items assigned to the user. The Project lead should subscribe to all events in the project to keep an overview of what is going on.
5. Create a sub folder for the project in Team Explorer under ‘My Favorites’ and drag ‘My Active Bugs’ and ‘My Active Work Items’ to it.

## Setting up a SharePoint project

1. Setup a TFS project and folder structure as outlined in section 13.1.
2. Create a new WSPBuilder project
3. Make sure required shared libraries are branched into the project and version numbers are synchronised.
4. Start building up the ‘12’ folder structure.

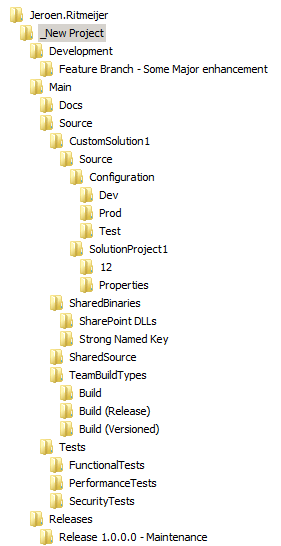
Listed below is the structure of a typical SharePoint solution.



# Source code control

The guidelines listed in this topic are based on Microsoft’s best practices (http://www.codeplex.com/TFSGuide). At times it seems to be over engineered due to deep nesting of folders, but this structure makes branching very easy and works quite well once you start using it.

## TFS Project Folder Structure

The source tree on the right hand side is for a typical TFS project as stored on a developer’s file system. The Project named ‘\_New Project’ is the name and root folder of the TFS project.

The Project folder contains 3 sub folders: Development, Main and Releases, each represents different branches.

### Main

This is the main branch where development normally takes place. New Projects / solutions are started here and under normal circumstances developers work here as well.

It contains the following sub folders:

1. **Docs:** Any documents that are part of a release and are shipped to customers / end users are stored here. An example of such a document is a User Guide or License Agreement. Do not store documents such as requirements or the High Level Design in here.   
   SharePoint is typically used to edit and store documents. When the project has been signed off copy the documents from SharePoint to the Source Tree.
2. **Source:** This is the root folder for all source code. It contains 1 or more solution files as well as the root folders for the SharedBinaries, SharedSource and Build scripts:
   1. **CustomSolution1:** This folder will need to be renamed to the actual solution name, e.g SharePointAudit. This solution may contain multiple sub folders, e.g. for solution specific documentation, but in general has only two sub folders ‘Configuration’ and ‘Source’. The ‘Source’ subfolder contains the actual Visual Studio SLN file and one or more sub projects, the ‘Configuration’ sub folder contains configuration files for Dev, Test and Production.
   2. **SharedBinaries:** Contains any binaries that may be shared between different TFS projects. For example the company’s SNK file, ilmerge, wspbuilder, SharePoint DLLs or other third party DLLs. Basically any dependency that is needed to perform a build on a clean build server. For details on sharing files between TFS projects see 14.4 .
   3. **SharedSource:** *Use of this folder is no longer encouraged; shared source is directly branched into the project so everything is at the same ‘level’.*
   4. **TeamBuildTypes:** Each TFS Solution can have one or more build scripts. These scripts can be assigned to a build definition, which is then used by the build server to create a ‘build’.
3. **Tests:** Test results, not the unit tests, are stored in here.

### Development

The Development Branch can be used for parallel development. This should be an exception rather than the rule (see 14.3 ). A typical scenario is the development of a major new feature or breaking change while development is also going on in the Main Branch.

### Releases

Release branches are generally used by companies who need to maintain and support several different product versions that are in use by customers. For example Microsoft would have a Windows XP and a Windows Vista release branch while Windows 7 is in development in Main. If a security bug is found in XP then this can be fixed in the Windows XP release branch. Obviously this is a simplified view, Microsoft has hundreds of branches.

In house development departments or professional services companies who only look after one version of a production system generally have little use for Release Branches with one major exception, which is the ability to patch the current production software while a new version is being developed on the Main branch.

## Client Folder structure

The folder structure on the developer’s workstation is identical to the structure in TFS. The only difference is that the root folder for all projects is as follows:

F:\Projects\<User name>

This allows multiple developers to work on the same machine without interfering with each other’s source tree. In general all developers have their own PC, but sometimes PCs are temporarily shared for troubleshooting or emergency purposes.

## Branching guidelines

TFS follows a similar branching strategy as VSS. Branching can be very useful, but as a general rule ***don’t branch unless you have a good reason to do so*** and consult your team’s tech lead for advice.

### Typical branching scenarios

1. **No Branch:** The most common scenario is to work on the Main branch during day to day development. This prevents the administrative overhead of periodically merging changes and resolving conflicts.
2. **Maintenance releases:** When a change is required to a version of the software that was previously released and a new version is under development then a branch can be created from the previously released version number into the *Releases* Branch folder. Once the changes have been made it is worth considering merging the changes into the Main branch to ensure any future releases automatically benefit from the change. There is no need to automatically create a Release branch after delivery of each version, only branch when changes are required.
3. **Feature branches:** When a potential large change is about to be made, which may break the main build, while parallel development is also ongoing on the Main branch then it may be a good idea to create a ‘Feature branch’ under the ‘Development’ Branch folder. Once development on the feature branch is complete changes can be merged into the Main Branch.

For details on the various available branching folders see 14.1 .

### How to branch

A full discussion of how to branch and merge is out of the scope of this document. Listed below is a short overview of how to create a Feature Branch for a new Reporting Engine.

1. Use the Team Explorer to open the Source Control view for the project.
2. Create a new folder under *Development* named ‘Feature Branch – Report Engine’
3. Generally we want to branch the Source only, so right-click on ‘$/TFS ProjectName/Main/Source’ and select Branch.
4. Use the Browse Button to select the destination branch folder ‘$/TFS ProjectName/Development/Feature Branch – Report Engine’
5. Select a specific version or accept the default option of ‘Latest Version’. Click OK to create out the branch.
6. The branch is not checked in by default, so check it in.

## Working with shared libraries and files

Many projects share, or at least should share, a number of common source code files as well as binary files. This section describes how we deal with this, which is in line with TFS best practices.

As mentioned in 14.1.1 we distinguish between two types of Shared content; *SharedSource* and *SharedBinaries*. Each is maintained in its own TFS project with its own source code and work items.

Shared content can be selectively branched into a project. One of the key benefits for using branching over a shared folder approach is that changes in the shared library can be merged into a project at a convenient time to prevent breaking changes in a shared library just before a project is released.

Branching shared content, e.g. ‘wspbuilder’ works as follows:

1. Right-click $/SharedBinaries/Main/Source/wspbuilder and select ‘Branch’.
2. As the target select $/<Project Name>/Main/Source/SharedBinaries
3. Repeat the same for other shared binaries and check in all changes.

Branching shared source code works in the same way.

When changes are made to the shared files in the future then these changes can be re-merged into individual projects using the *merge* command.

If during the course of the development of a project changes are made to the branched libraries, that are now part of the project, then it is worth considering merging those changes back.

# Appendix - Sample files

## Feature.XML

A typical feature.xml file looks as follows. Note that the translations are loaded from a resource file.

|  |
| --- |
| **<?xml version="1.0" encoding="utf-8" ?>**  **<!--**  **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***  **Muhimbi Licensing for SharePoint**  **Copyright 2008, Muhimbi Ltd**  **www.muhimbi.com**  **All rights reserved**  **Original Document author: jeroen.ritmeijer**  **Filename: Feature.xml**  **Description: Enble Farm scoped functionality such as**  **central administration screens**  **To do: -**  **Known issues: -**  **History: 20/11/08 - JR - Initial version**  **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***  **-->**  **<Feature Id="33F332C8-6236-4c0e-A3F9-4C3765475F63"**  **Title="$Resources:FeatureTitle;"**  **Description="$Resources:FeatureDescription;"**  **ImageUrl="MUHIMBI.GIF"**  **Version="1.0.0.0"**  **Scope="Farm"**  **Hidden="FALSE"**  **RequireResources="FALSE"**  **ReceiverAssembly="Muhimbi.Licensing.SharePoint, Version=1.0.0.0, Culture=neutral, PublicKeyToken=c9db4759c9eaad12"**  **ReceiverClass="Muhimbi.Licensing.SharePoint.FeatureReceiver"**  **xmlns="http://schemas.microsoft.com/sharepoint/">**  **<ElementManifests>**  **<ElementManifest Location="CustomActions\Elements.xml" />**  **</ElementManifests>**  **</Feature>** |

## Elements.xml

A typical Elements.xml file. Note the hierarchically named ID attribute.

|  |
| --- |
| **<?xml version="1.0" encoding="utf-8" ?>**  **<!--**  **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***  **Muhimbi Licensing for SharePoint**  **Copyright 2008, Muhimbi Ltd**  **www.muhimbi.com**  **All rights reserved**  **Original Document author: jeroen.ritmeijer**  **Filename: Elements.xml**  **Description: Contains custom actions for the various**  **user interface elements in the central**  **administration pages**  **To do: -**  **Known issues: -**  **History: 21/11/08 - JR - Initial version**  **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***  **-->**  **<Elements xmlns="http://schemas.microsoft.com/sharepoint/">**    **<CustomAction**  **Id="Muhimbi.SharePoint.Administration.Operations.Licensing.LicenseManager"**  **Title="$Resources:LicenseManagerCustomActionTitle;"**  **Description="$Resources:LicenseManagerCustomActionDescription;"**  **Location="Microsoft.SharePoint.Administration.Operations"**  **GroupId="GlobalConfiguration"**  **RequireSiteAdministrator="FALSE"**  **Sequence="60"**  **ImageUrl="/\_layouts/images/muhimbi.GIF">**  **<UrlAction Url="~site/\_admin/Muhimbi.Licensing/LicenseManager.aspx"/>**  **</CustomAction>**  **</Elements>** |

1. For general enterprise development we would normally recommend the ExceptionHandling Application Block, part of Microsoft’s Enterprise Libraries (http://msdn.microsoft.com/en-us/library/cc309505.aspx), however this makes deployment for the kind of commercial applications that Muhimbi develops rather impractical. Therefore a more lightweight approach is used with minimal dependencies. [↑](#footnote-ref-1)
2. We did not follow our own advice and make the last 2 digits of the version number the same. These products were released before we settled on that standard. Some day we may pay the price. [↑](#footnote-ref-2)