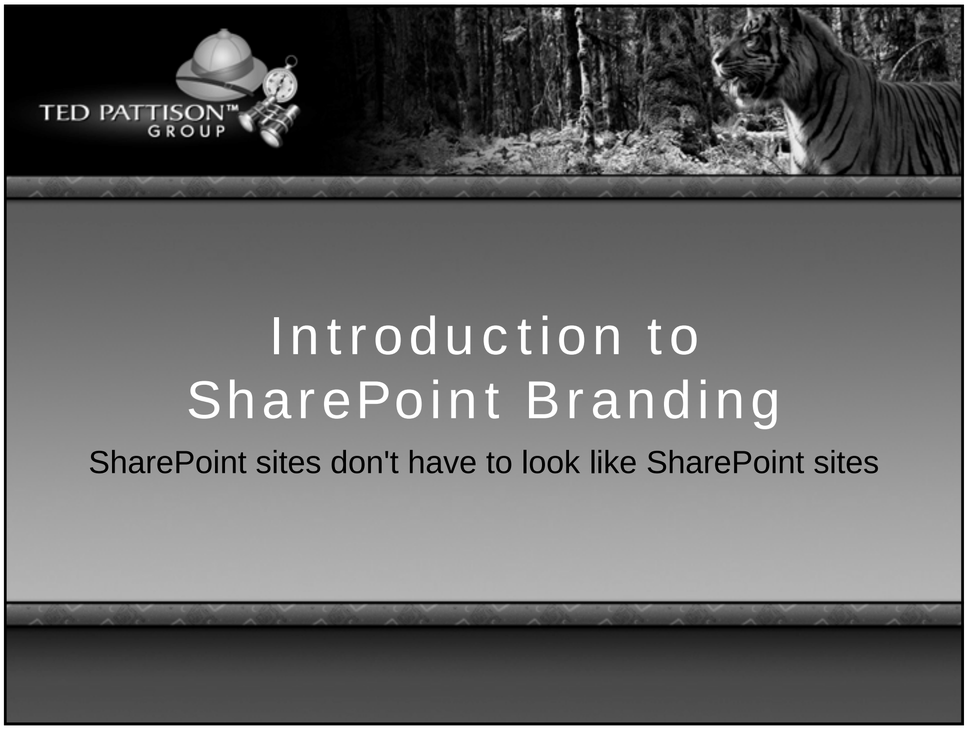
SCB301: SharePoint Brand Camp

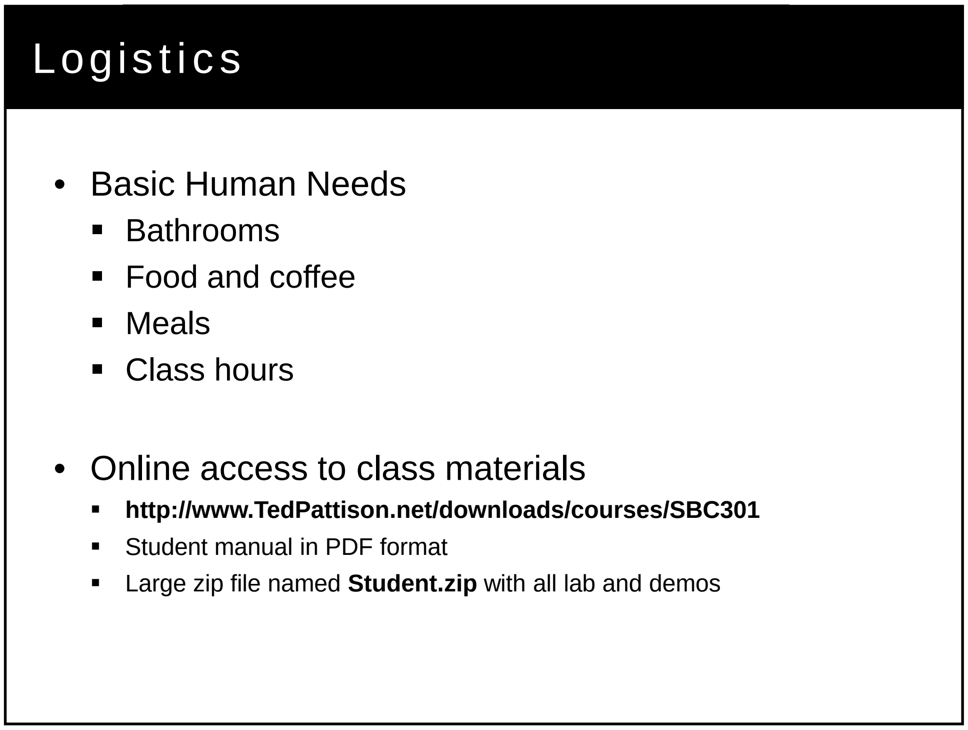
Schedule of lectures

Revision Date: December 16th, 2008



Introduction to SharePoint Branding

SharePoint sites don't have to look like SharePoint sites



Logistics

- Basic Human Needs
  - Bathrooms
  - Food and coffee
  - Meals
  - Class hours
- Online access to class materials
  - **http://www.TedPattison.net/downloads/courses/SBC301**
  - Student manual in PDF format
  - Large zip file named **Student.zip** with all lab and demos

Student Questionnaire

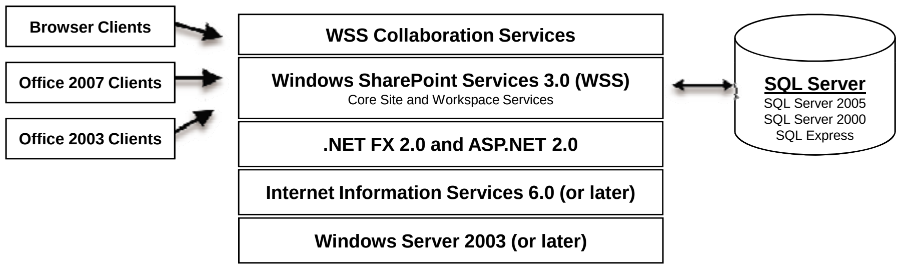
- What's Your Name and Company?
- What languages have you used?
  - HTML, XHTML, CSS and JavaScript
  - XML, XSD, XPath and XSLT
  - C# or VB.NET
- What design/branding tools have you used?
  - SharePoint Designer
  - Visual Studio and ASP.NET
  - Adobe Photoshop
  - Legacy toolsets
    DreamWeaver, Cold Fusion, FrontPage, Visual InterDev, Visual Notepad, etc

Agenda

- WSS Fundamentals
- Anatomy of a Team Site
- Understanding the role of CSS and Themes
- SharePoint Integration with Master Pages
- Working with the SharePoint Designer
- Understanding Features
- WCM using the MOSS Publishing Features

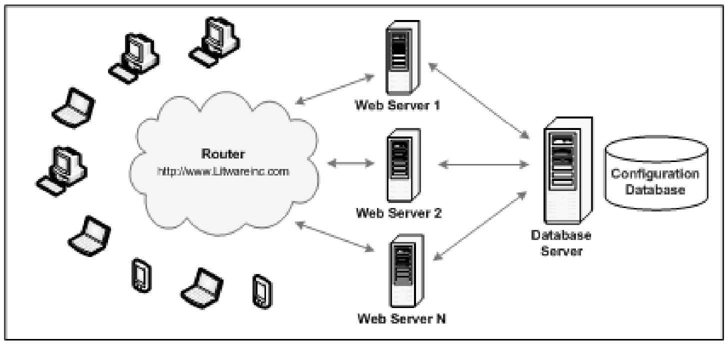
What is WSS?

- Windows SharePoint Services 3.0 (WSS)
  - An engine for creating/running/managing sites
  - Architecture designed to scale to 10,000s of sites
  - Platform for building Web application and solutions
  - Collaboration services included out-of-the-box


Browser Clients
Office 2007 Clients
Office 2003 Clients
WSS Collaboration Services
Windows SharePoint Services 3.0 (WSS)
Core Site and Workspace Services
.NET FX 2.0 and ASP.NET 2.0
Internet Information Services 6.0 (or later)
Windows Server 2003 (or later)
SQL Server
SQL Server 2005
SQL Server 2000
SQL Express

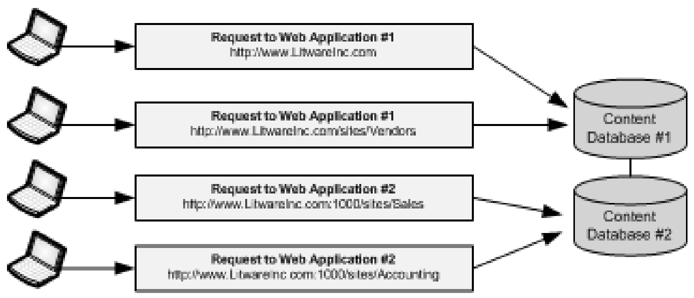
The WSS Farm

- WSS deployment based on a farm
  - Farm requires Web server(s) and database server
  - Farm can be single server or multi-server
  - Each farm has exactly one configuration database



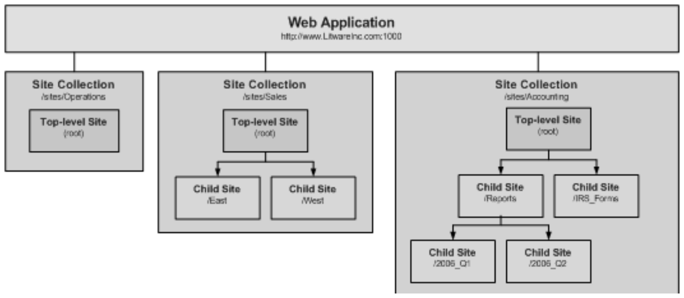
Web Applications

- Web Applications provide HTTP entry points
  - Web Applications based on IIS Web sites
  - Web Application defines one or more URL spaces
  - Web Application security configured independently


Request to Web Application #1
http://www.LitwareInc.com
Request to Web Application #1
http://www.LitwareInc.com/sites/Vendors
Request to Web Application #2
http://www.LitwareInc.com:1000/sites/Sales
Request to Web Application #2
http://www.LitwareInc.com:1000/sites/Accounting
Content Database #1
Content Database #2

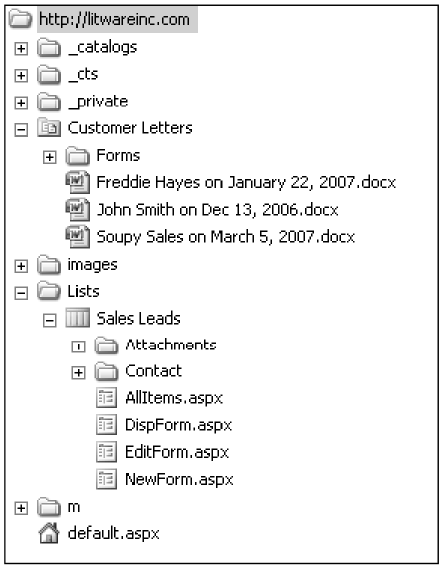
Site Collections and Sites

- Sites are partitioned using Site Collections
  - Site collection is scope for administrative privileges
  - Site collection always contains top-level site
  - Site collection may contain hierarchy of child sites
  - Web application can support 1000s of site collections



The Virtual File System of a Site

- Site is a virtual file system
  - made up of folders and files
  - Pages are files
  - Documents are files
  - Stored in content database
- How can you look at it?
  - SharePoint Designer
  - Windows Explorer (WebDav)



```
http://litwareinc.com
+ _catalogs
+ _cts
+ _private
- Customer Letters
    + Forms
      Freddie Hayes on January 22, 2007.docx
      John Smith on Dec 13, 2006.docx
      Soupy Sales on March 5, 2007.docx
+ images
- Lists
    - Sales Leads
        + Attachments
        + Contact
          AllItems.aspx
          DispForm.aspx
          EditForm.aspx
          NewForm.aspx
+ m
  default.aspx
```

The _layouts Virtual Directory

- Files in `_layouts` directory accessible to all sites
  - _layouts provides access to common resources
  - _layouts contains files for images, CSS and JavaScript
  - _layouts contains Application Pages
- All these URLS resolve to the same page

```
http://Litwareinc.com/_layouts/settings.aspx
http://Litwareinc.com/sites/Vendors/_layouts/settings.aspx
http://Litwareinc.com:1001/sites/Accounting/_layouts/settings.aspx
```

Site Pages Versus Application Pages

- Site Pages exist within virtual file system of site
  - They may or may not be ghosted
  - They support customization via Web Parts
  - They support customization via SharePoint Designer
  - Customized pages impact performance and security
- Application Pages are deployed once per farm
  - They do not support customization or Web Parts
  - They are parsed/compiled as classic ASP.NET pages
  - They run faster than Site Pages
  - They always support code behind

SharePoint Integration with CSS

- WSS supplies core.css
- Several approaches to extend & override core.css

Understanding core.css

- All styles in WSS initially defined by core.css
  - Located in `\TEMPLATE\LAYOUTS\1033\STYLES`
  - Contains over 4000 lines of CCS class definitions
  - Classes used throughout standard WSS UI elements
- Extending core.css
  - Applying WSS styles (meant for end users)
  - Applying custom CCS files (meant for developers)

SharePoint Themes

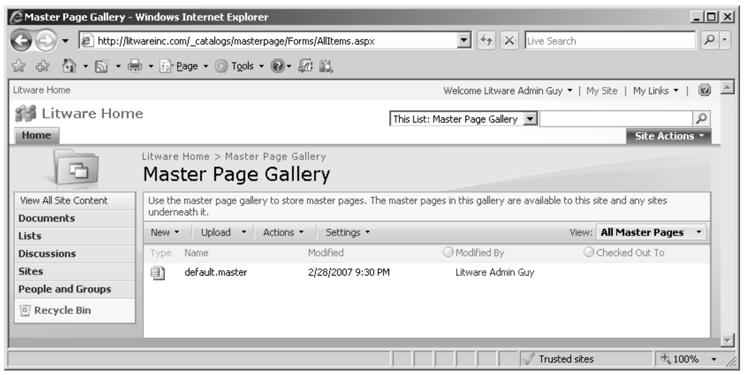
- WSS provides set of out-of-box themes
  - Collection of CSS file and graphics
  - Theme files deployed to front-end Web server
  - WSS provides UI for users to apply themes
  - Themes can be customized using SharePoint Designer

Master Pages in WSS

- Application pages use application.master
  - Farm-wide master page
  - Cannot be customized on per-site basis
- Site Pages use default. master by default
  - default.master is a page template
  - default.master instance created in Master Page Gallery
  - default.master can be customized on per-site basis
  - default.master can be replaced with different template

The Master Page Gallery

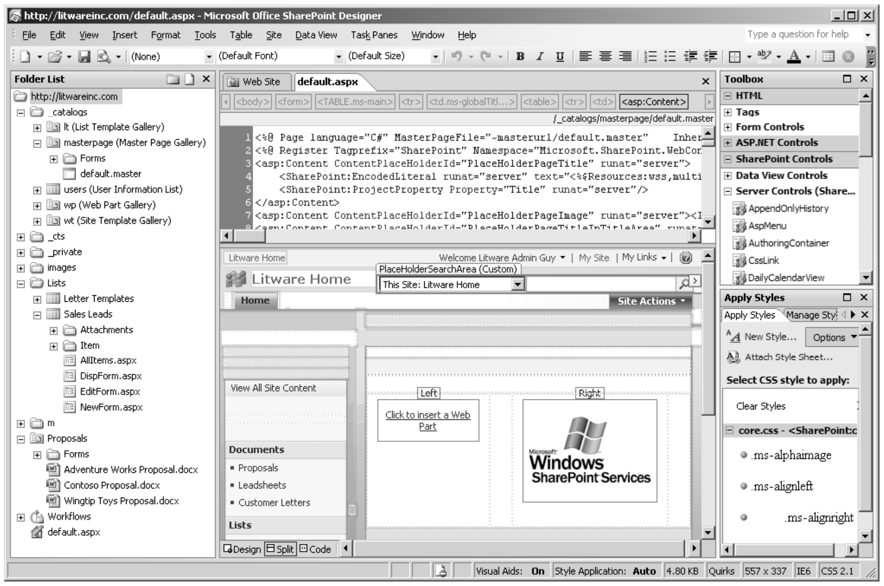
- Each site has a Master Page Gallery
  - Instance of default.master automatically provisioned
  - default.master can be customized on per-site basis



Use the master page gallery to store master pages. The master pages in this gallery are available to this site and any sites underneath it.

Type	Name	Modified	Modified By	Checked Out To
	default.master	2/28/2007 9:30 PM	Litware Admin Guy	

Customization with the SharePoint Designer


http://litwareinc.com/default.aspx - Microsoft Office SharePoint Designer
Folder List
Web Site
default.aspx
/_catalogs/masterpage/default.master
Toolbox
Apply Styles
Litware Home
Site Actions
Design
Split
Code

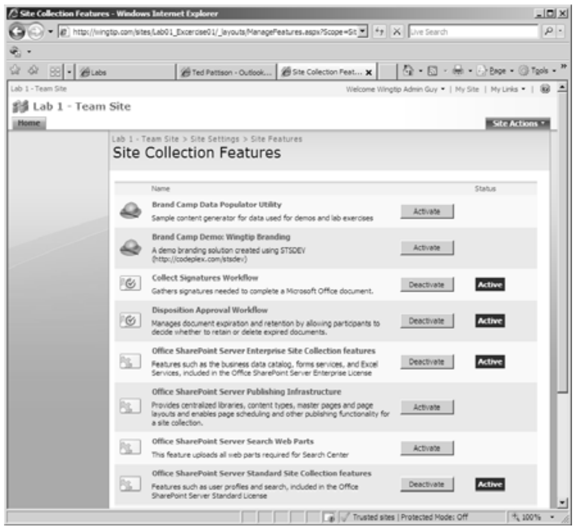
```
<%@ Page language="C#" MasterPageFile="~masterurl/default.master"   Inher
<%@ Register Tagprefix="SharePoint" Namespace="Microsoft.SharePoint.WebCor
<asp:Content ContentPlaceHolderId="PlaceHolderPageTitle" runat="server">
    <SharePoint:EncodedLiteral runat="server" text="<%$Resources:wss,multi
    <SharePoint:ProjectProperty Property="Title" runat="server"/>
</asp:Content>
<asp:Content ContentPlaceHolderId="PlaceHolderPageImage" runat="server"><I
```

What is a Feature?

- A building block for creating SharePoint solutions
  - A unit of design, implementation and deployment
- Features can contain elements
  - e.g. menu items, links, list types and list instances
  - Many other element types possible
- Features can contain event handlers
  - You can add any code which used WSS object model

User's View of Features

- Features support concept of activation/deactivation



Site Collection Features - Windows Internet Explorer
http://wingtip.com/sites/Lab01_Excercise01/_layouts/ManageFeatures.aspx?Scope=Sit
Live Search
Labs
Ted Pattison - Outlook...
Site Collection Feat...
Page
Tools
Lab 1 - Team Site
Welcome Wingtip Admin Guy | My Site | My Links
Lab 1 - Team Site
Home
Site Actions
Lab 1 - Team Site > Site Settings > Site Features

Site Collection Features

Name		Status
Brand Camp Data Populator Utility Sample content generator for data used for demos and lab exercises	Activate	
Brand Camp Demo: Wingtip Branding A demo branding solution created using STSDEV (http://codeplex.com/stsdev)	Activate	
Collect Signatures Workflow Gathers signatures needed to complete a Microsoft Office document.	Deactivate	Active
Disposition Approval Workflow Manages document expiration and retention by allowing participants to decide whether to retain or delete expired documents.	Deactivate	Active
Office SharePoint Server Enterprise Site Collection features Features such as the business data catalog, forms services, and Excel Services, included in the Office SharePoint Server Enterprise License	Deactivate	Active
Office SharePoint Server Publishing Infrastructure Provides centralized libraries, content types, master pages and page layouts and enables page scheduling and other publishing functionality for a site collection.	Activate	
Office SharePoint Server Search Web Parts This feature uploads all web parts required for Search Center	Activate	
Office SharePoint Server Standard Site Collection features Features such as user profiles and search, included in the Office SharePoint Server Standard License	Deactivate	Active

This is the site-level feature management page in a WSS farm where MOSS has been installed.

Much of the functionality of MOSS is enabled and disable by activating and deactivating features that have been developed by the MOSS team.

MOSS Support for WCM

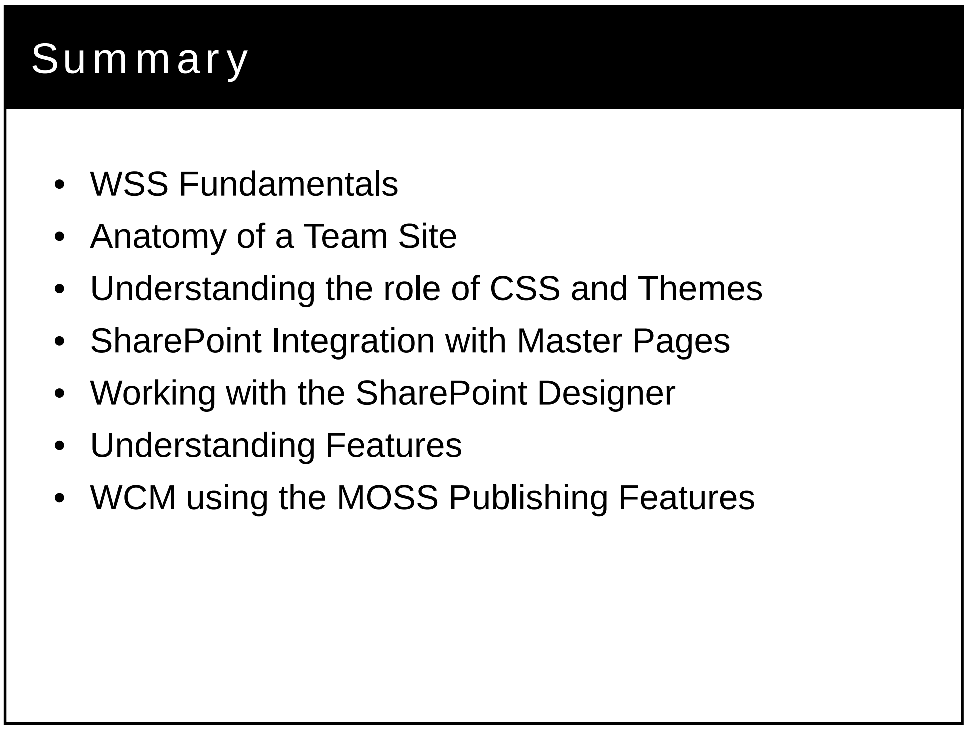
- MOSS supports WCM via its Publishing features
  - One feature is scoped at site-collection level
  - Other feature scoped to site
- What do the MOSS Publishing features do?
  - Enable uploading and editing of custom CCS files
  - Enable uploading and editing of Master Pages
  - Provide framework for content approval

The MOSS Approval Process

- How does the MOSS approval process work?
  - Publishing Pages created on demand by users
  - Publishing Pages are created from Page Layouts
  - Page Layouts are templates stored in MPG
  - Publishing Pages are instances stored in Pages library
  - Styles Library enabled approval for other types of files
    e.g. GIF files, JPG files, CSS files, JavaScript files, etc

Anatomy of a MOSS Publishing site

- Publishing feature (Site Collection Level)
  - Activated once per site collection
- Publishing feature (Site Level)
  - Activated once for each site with publishing pages

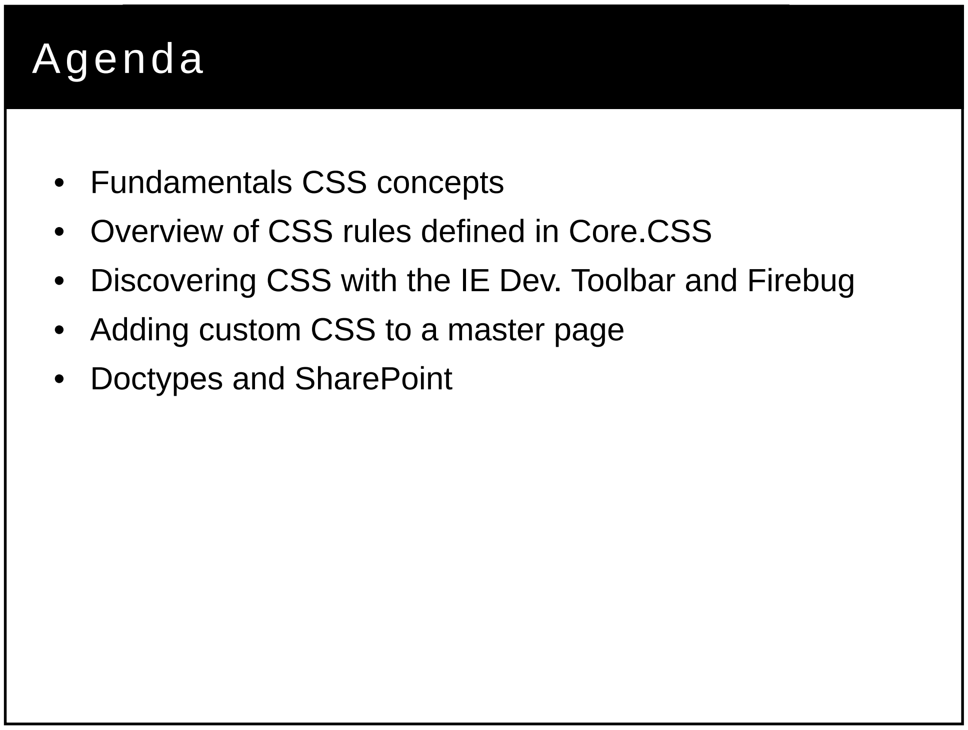


Summary

- WSS Fundamentals
- Anatomy of a Team Site
- Understanding the role of CSS and Themes
- SharePoint Integration with Master Pages
- Working with the SharePoint Designer
- Understanding Features
- WCM using the MOSS Publishing Features

CSS Primer for SharePoint Designers

Working together with CSS and SharePoint



Agenda

- Fundamentals CSS concepts
- Overview of CSS rules defined in Core.CSS
- Discovering CSS with the IE Dev. Toolbar and Firebug
- Adding custom CSS to a master page
- Doctypes and SharePoint

Fundamentals CSS concepts

- Why is CSS important for web design?
  - Table and Font tags were popular a few years ago
  - Web design has moved to a standards based separation of design and functionality
  - Tables are better suited to tabular data
  - Does this mean that we will never use Table and Font tags?
- Why is CSS important with SharePoint?
  - CSS is the primary way that Microsoft lets you override their default branding
  - OOTB SharePoint uses a LOT of CSS… 100’s of lines of code

How is CSS applied to HTML?

- Inline Style

```
<p style="color: red;">The text is red</p>
```

- Internal Style Sheets

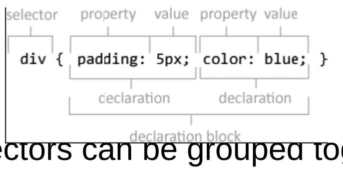
```
<style type="text/css">
  h1 { color: blue; }
</style>
```

- External Style Sheets

```
<link href="customStyle.css" rel="stylesheet" type="text/css">
```

Style sheets are made up of Rules

- Rules consist of
  - Selector – What HTML element is being styled
  - Declaration block - Sets of style properties and values
- Each selector may have many sets of properties and values in its declaration block



- Several selectors can be grouped together with one common declaration (example: `div, p, h1 { color: blue; }`)

Types of Selectors

- Type Selectors
```
p { font-family: arial; }
```
- Class Selectors
```
.myClass { color: red; }
p.myClass { color: blue; }
```
- ID Selectors
```
<div id="myID">
#myID { color: green; }
```
- Descendent Selectors
```
<div>
  <p>Hello World</p>
  Goodbye World
</div>

<style>
div p {
  color: purple;
}
</style>
```

Types of Selectors

- Pseudo-Classes

```
a:link {
  color: red;
}
a:visited {
  color: blue;
}
a:hover {
  color: green;
}
```

- The Universal Selector

```
* {
  margin: 0px;
  padding: 0px;
}
.myClass * {
  padding: 10px;
}
```

CSS Properties

- Fonts
  - Font Family
    ```
    font-family: Cambria, 'Times New Roman', serif;
    ```
  - Font Size
    ```
    font-size: 12px;
    font-size: 1em;
    ```
  - Color
    ```
    color: blue;
    color: rgb(0,0,255);
    color: #0000FF;
    ```
- Text Decoration can be used to create interactivity on links

```
a {
  text-decoration: none;
}
a:hover {
  text-decoration: underline;
}
```

- Wrapping can be controlled with CSS

```
white-space: nowrap;
```

- Display can be used control spacing and to hide elements

```
display: block;
display: none;
```

CSS Properties

- Float
  - The normal flow of a web page is from top to bottom
  - Float can remove an element from this normal flow and move it to the far left or far right

    `float: right;`
- Position
  - By default elements have a position of static, and flow normally down the page

    Relative Position - Causes the element to be positioned relatively based on where it normally would be in the flow

    Absolute Position - Causes the element to be positioned absolutely based on its parent.
  - When position is set to relative or absolute you will want to set a top, bottom, left, or right value to control where the element moves.
- Backgrounds can be used to impart a lot of design change with little effort
  - Background Color

    `background-color: blue;`
  - Background Image

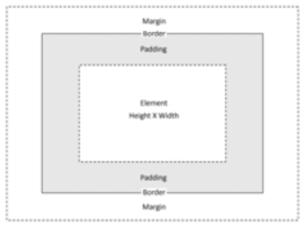
    `background-image: url('/images/myimage.gif');`
  - Background Repeat

    `background-repeat: repeat-x;`
- Borders can be set all in one shortcut property or they can be set per side

```
border: 1px solid green;

border-left: 2px solid red;
border-right: 2px solid green;
border-top: 2px solid yellow;
border-bottom: 2px solid blue;
```

CSS Properties

- Padding and Margin are both very important design concepts, without them your text and elements will run together without white space
- Padding is the spacing between an element and its border, background colors and images show in this area
  - `padding-top: 1px;`
- Margin is the spacing past the border and does not show background colors or images for the specific element (though parent backgrounds will show through)
  - `margin-top: 1px;`



Inheritance in CSS

- Inheritance refers to the ability of some CSS values to be passed along to all of their descendant elements
- Not all of the CSS properties behave this way. Typically, the ones that do are the ones that affect text styling
- *For a list of all the CSS properties that have inheritance, see the W3C's Full Property Table at*
  - http://www.w3.org/TR/CSS21/propidx.html
- "!important" is really important!
  - When a CSS rule is declared as `!important`, the usual cascade rules do not apply and the !important rule takes precedence.

```
div {
  color: red !important;
}
```

Understanding the Cascade

1. Find all style declarations applied to the particular element
2. Sort by origin and importance (including the !important tag)
3. If origin and importance is the same, use specificity to determine what wins the cascade

- Specificity is determined via the following equation:

  ([The number of ID selectors] x 100) + ([The number of Class selectors] x 10) + ([The number of HTML selectors] x 1)

4. Finally, if all other steps are equivalent, the rule that is declared last is the winner

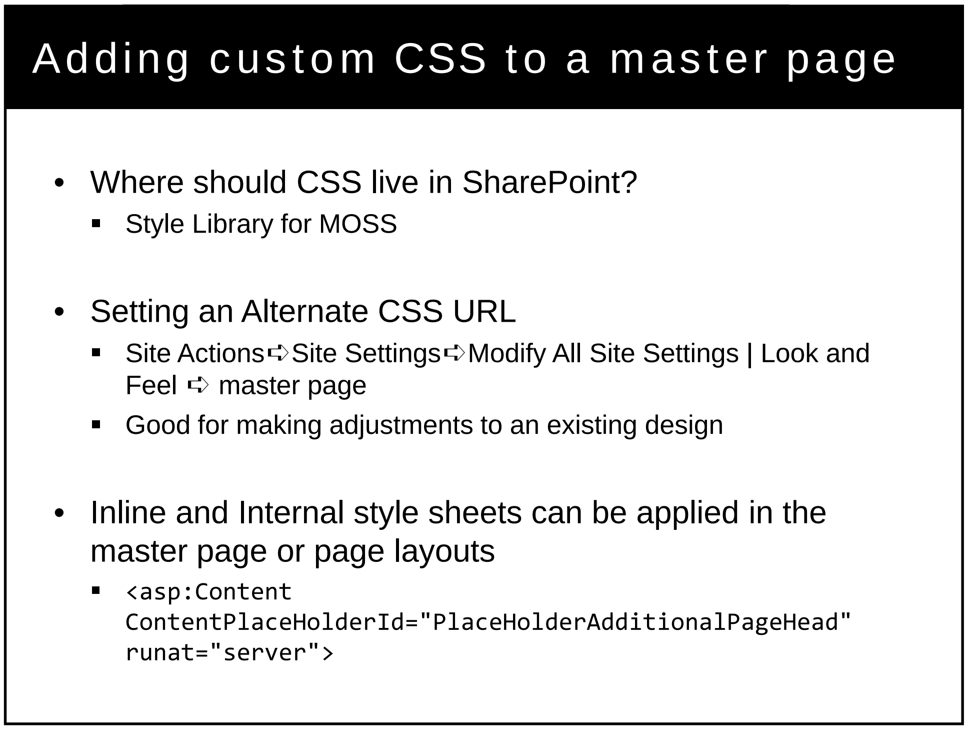
Overview of CSS rules defined in Core.CSS

- Demo

  http://msdn.microsoft.com/en-us/library/ms438349.aspx

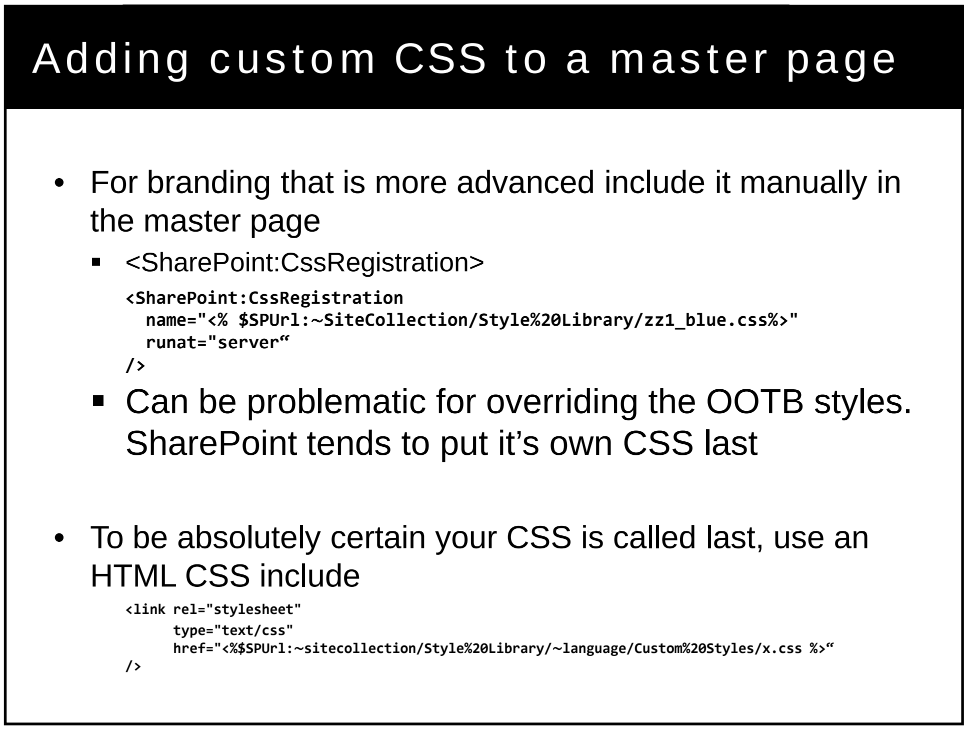
Discovering CSS with the IE Dev. Toolbar and Firebug

- IE Developers Toolbar
  - Visually explore CSS class names, IDs, image alt information, image sizes, etc.
  - Disable JavaScript and CSS, clear cache
  - Resize the browser to common monitor resolutions
  - Hex color picker & on-screen ruler for measuring pixels
  - http://www.microsoft.com/downloads/details.aspx?FamilyId=E59C3964-672D-4511-BB3E-2D5E1DB91038&displaylang=en
- Firebug
  - Performs many of the same functions as IE Dev. Toolbar + ability to manipulate CSS and see the changes in real-time
  - http://getfirebug.com



Adding custom CSS to a master page

- Where should CSS live in SharePoint?
  - Style Library for MOSS
- Setting an Alternate CSS URL
  - Site Actions➪Site Settings➪Modify All Site Settings | Look and Feel ➪ master page
  - Good for making adjustments to an existing design
- Inline and Internal style sheets can be applied in the master page or page layouts
  - `<asp:Content ContentPlaceHolderId="PlaceHolderAdditionalPageHead" runat="server">`



Adding custom CSS to a master page

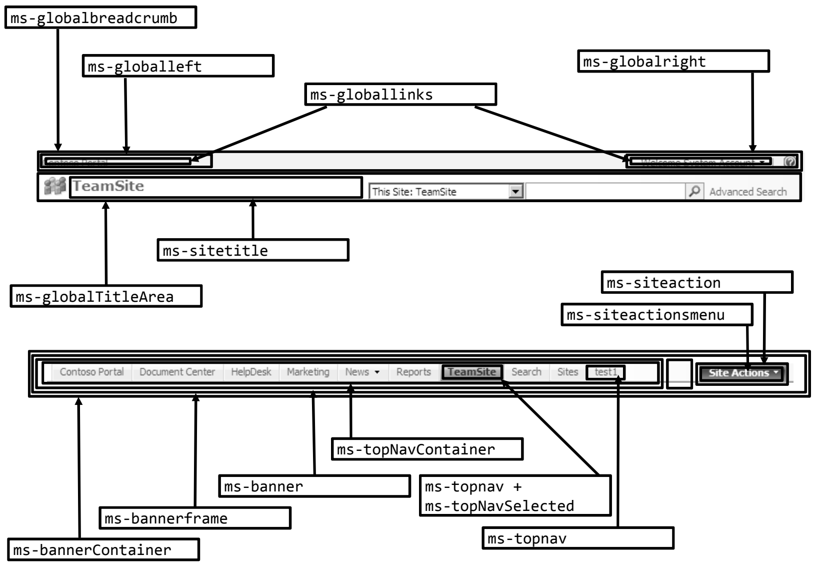
- For branding that is more advanced include it manually in the master page
  - <SharePoint:CssRegistration>

```
<SharePoint:CssRegistration
  name="<% $SPUrl:~SiteCollection/Style%20Library/zz1_blue.css%>"
  runat="server“
/>
```

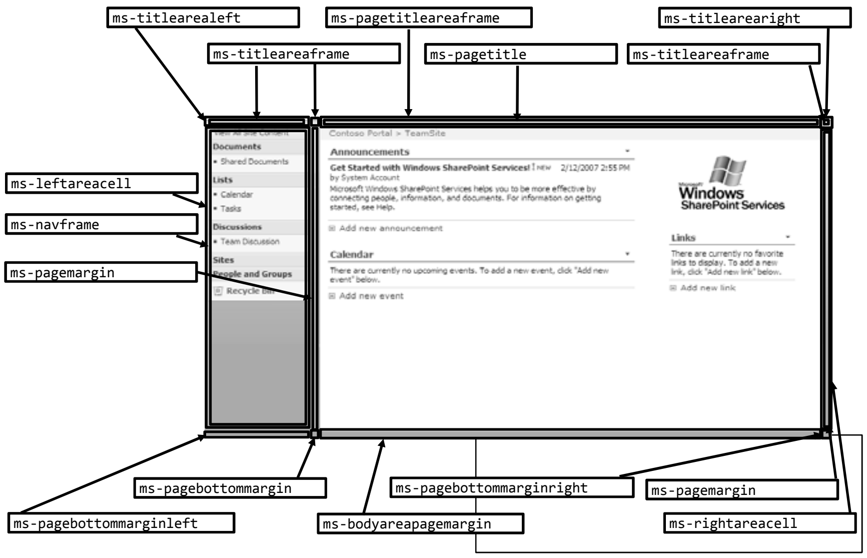
  - Can be problematic for overriding the OOTB styles. SharePoint tends to put it’s own CSS last
- To be absolutely certain your CSS is called last, use an HTML CSS include

```
<link rel="stylesheet"
      type="text/css"
      href="<%$SPUrl:~sitecollection/Style%20Library/~language/Custom%20Styles/x.css %>“
/>
```

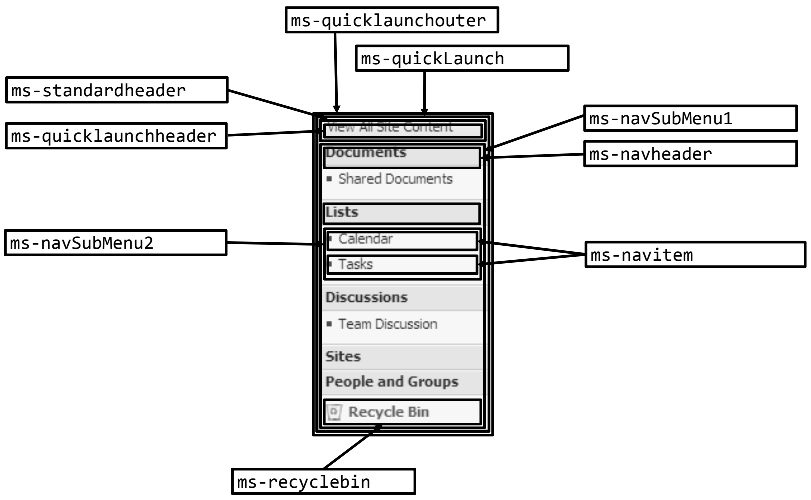
Top Navigation Areas



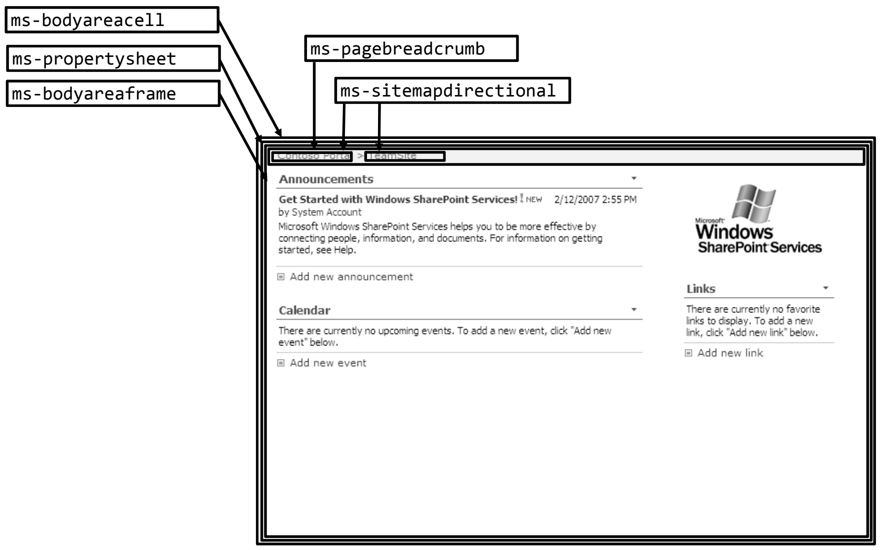
Body Structure



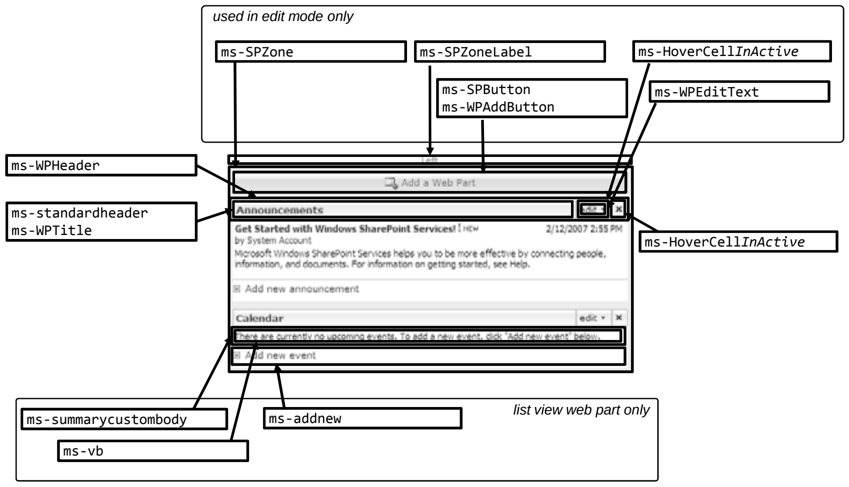
Quick Launch



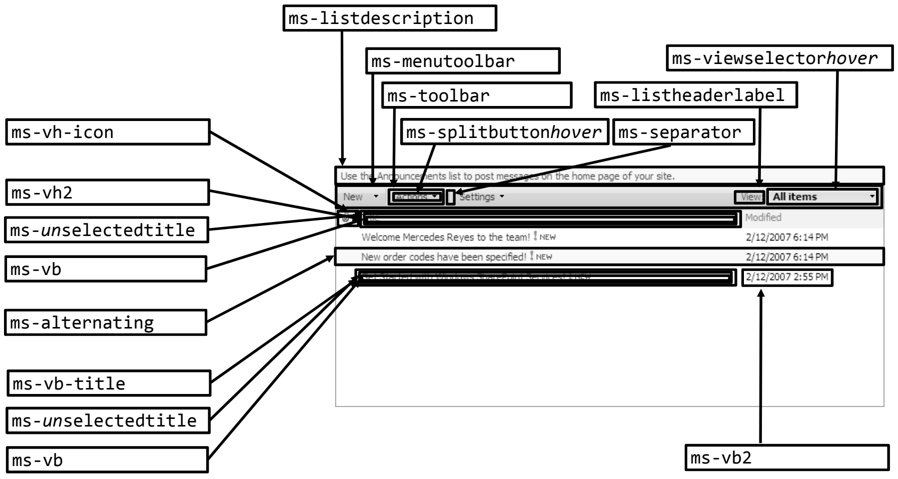
Page Body


ms-bodyareacell
ms-propertysheet
ms-bodyareaframe
ms-pagebreadcrumb
ms-sitemapdirectional
Contoso Portal > TeamSite
Announcements
Get Started with Windows SharePoint Services! NEW 2/12/2007 2:55 PM
by System Account
Microsoft Windows SharePoint Services helps you to be more effective by connecting people, information, and documents. For information on getting started, see Help.
Add new announcement
Calendar
There are currently no upcoming events. To add a new event, click "Add new event" below.
Add new event
Windows SharePoint Services
Links
There are currently no favorite links to display. To add a new link, click "Add new link" below.
Add new link

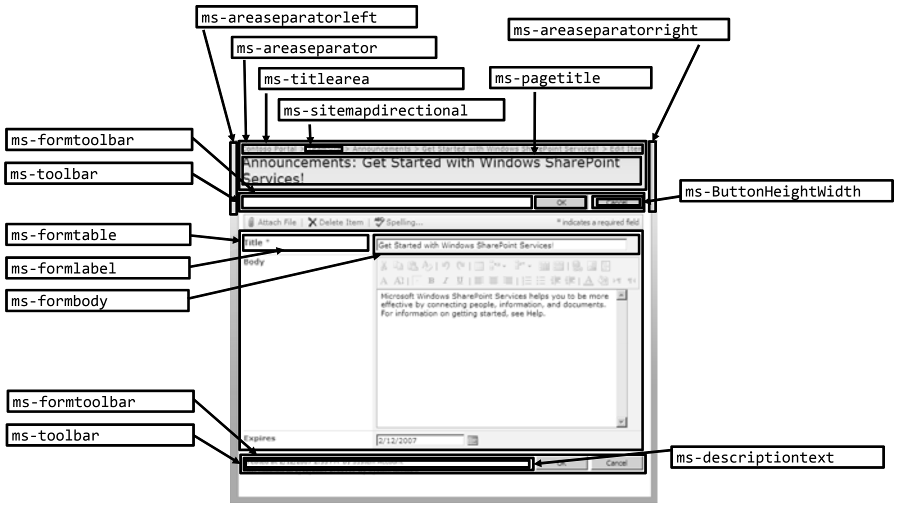
Web Parts



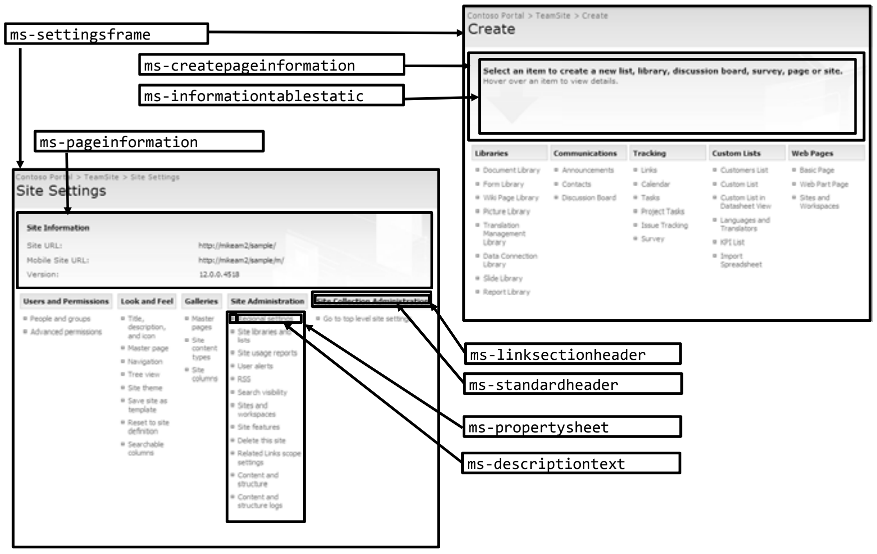
List Views



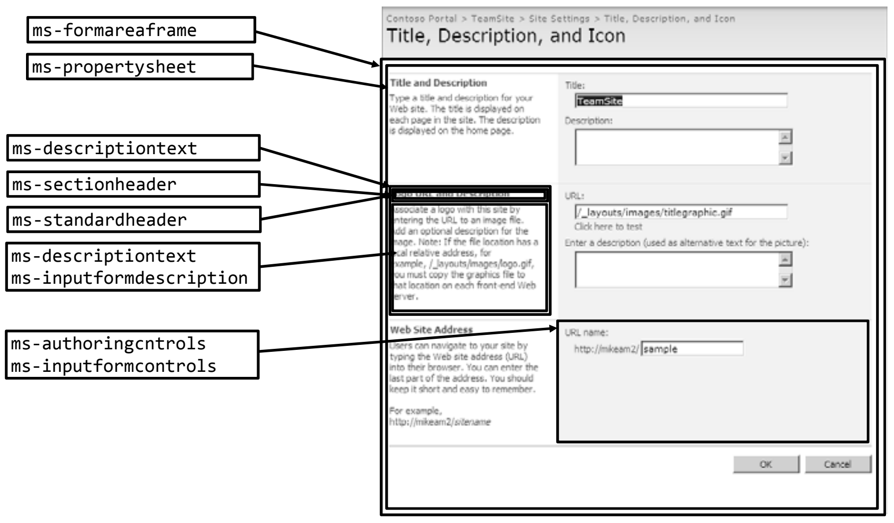
Forms



Application Pages (Landing Pages)



Application Pages (Form Pages)



Contoso Portal > TeamSite > Site Settings > Title, Description, and Icon

Title, Description, and Icon

Title and Description

Type a title and description for your Web site. The title is displayed on each page in the site. The description is displayed on the home page.

Title: TeamSite

Description:

ssociate a logo with this site by ntering the URL to an image file. dd an optional description for the nage. Note: If the file location has a cal relative address, for xample, /_layouts/images/logo.gif, ou must copy the graphics file to hat location on each front-end Web erver.

URL: /_layouts/images/titlegraphic.gif

Click here to test

Enter a description (used as alternative text for the picture):

Web Site Address

Users can navigate to your site by typing the Web site address (URL) into their browser. You can enter the last part of the address. You should keep it short and easy to remember.

For example, http://mikeam2/*sitename*

URL name: http://mikeam2/ sample

DocTypes

- What is a DocType?
  - Standard W3C instruction for browsers
  - tells browser to use specific language to interpret HTML/XML
  - pages without DocType render in Quirks mode (e.g. IE4)
- Strict versus Transitional
  - Strict prohibits legacy code (e.g. no Font tags, tags must be closed)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN”
                      "http://www.w3.org/TR/html4/strict.dtd">
```

  - Transitional allows pages with legacy code to be W3C compliant

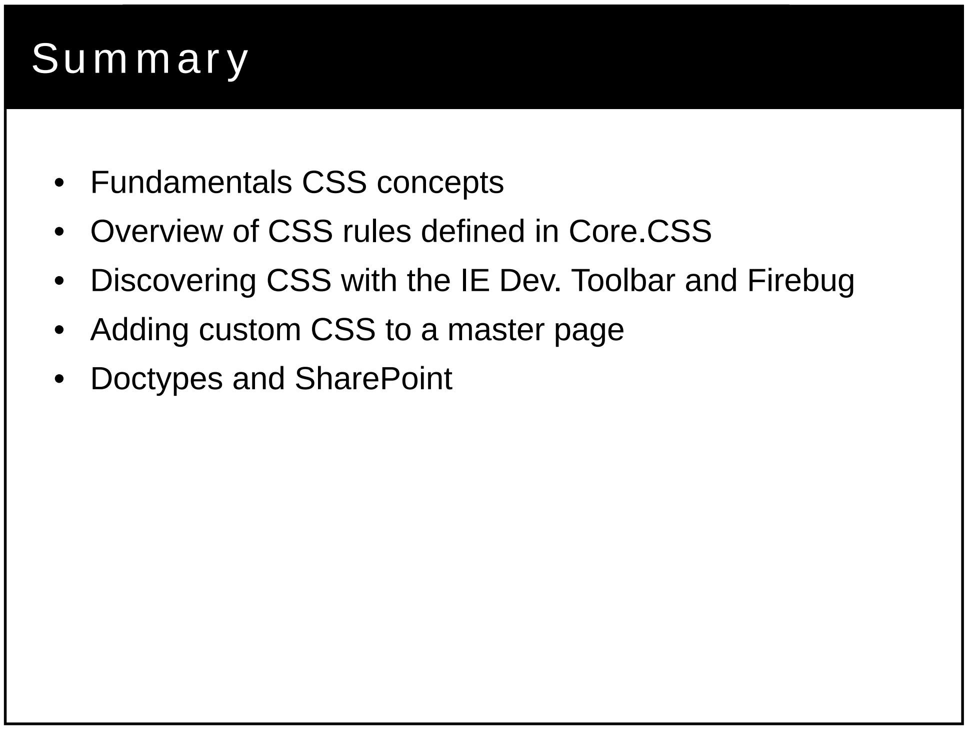
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN”
                      "http://www.w3.org/TR/html4/loose.dtd">
```

DocTypes and SharePoint

- What DocType does SharePoint use
  - default.master does not define a DocType
  - BlueBand.master (and its relatives) use HTML 4.01 Transitional
- SharePoint not W3C compliant with HTML4 or XML
- Tools for checking W3C compliance
  - https://addons.mozilla.org/en-US/firefox/addon/249

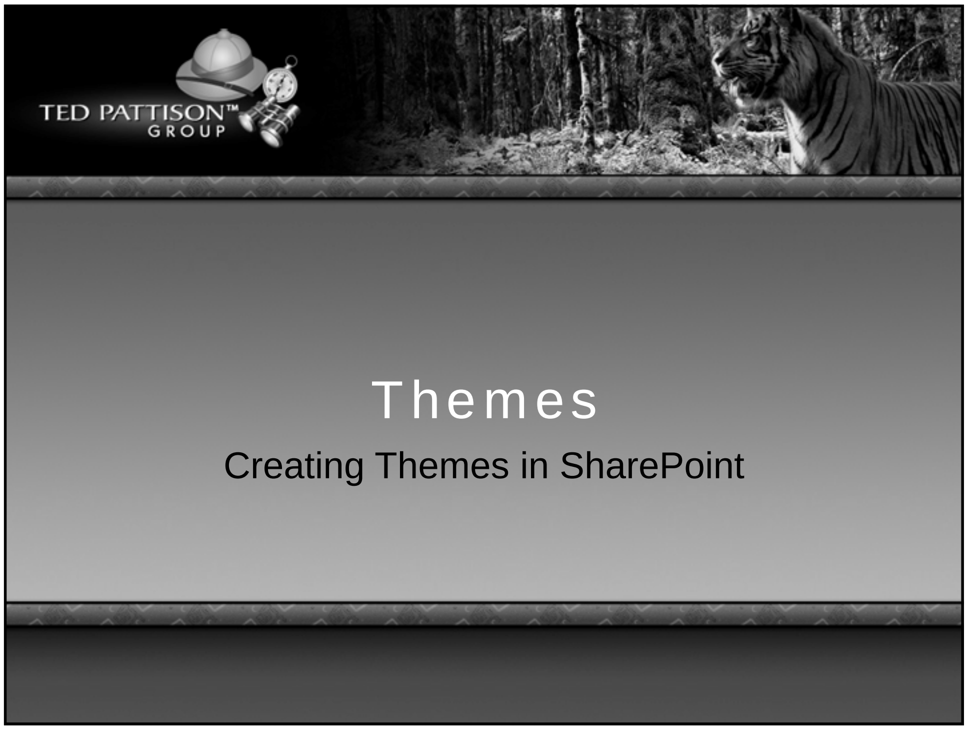
Browsers and Support

- All modern browsers support CSS styling though they sometimes handle CSS slightly differently
- Testing in all of your target browsers is crucial
  - For testing old versions of IE on Vista use IETester
    http://www.my-debugbar.com/wiki/IETester/HomePage



Summary

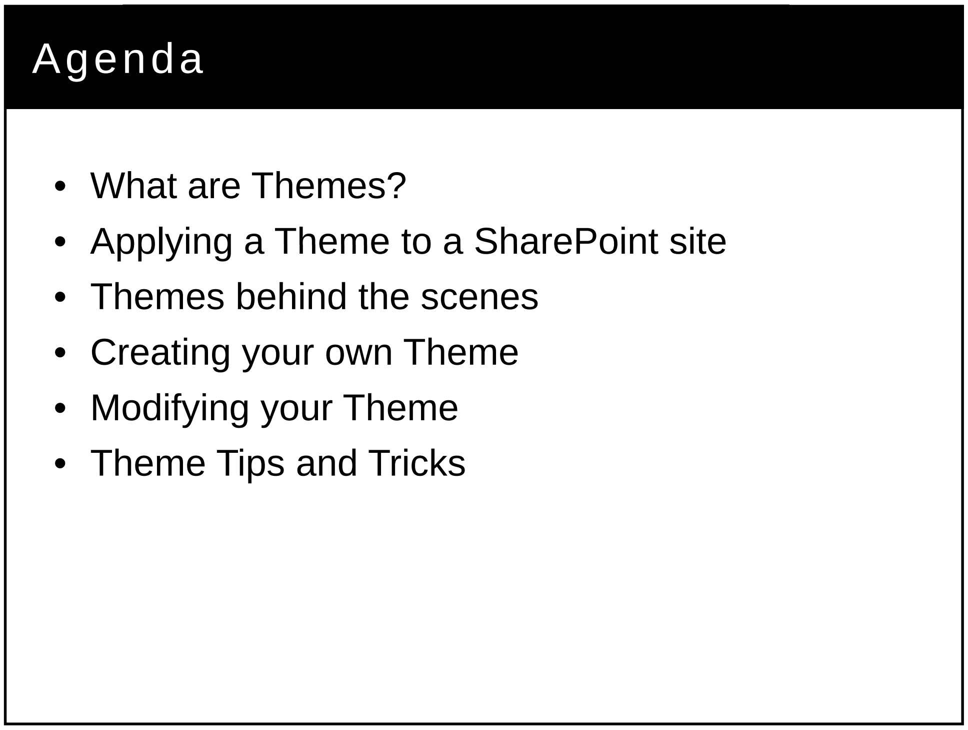
- Fundamentals CSS concepts
- Overview of CSS rules defined in Core.CSS
- Discovering CSS with the IE Dev. Toolbar and Firebug
- Adding custom CSS to a master page
- Doctypes and SharePoint



TED PATTISON™ GROUP

Themes

Creating Themes in SharePoint



Agenda

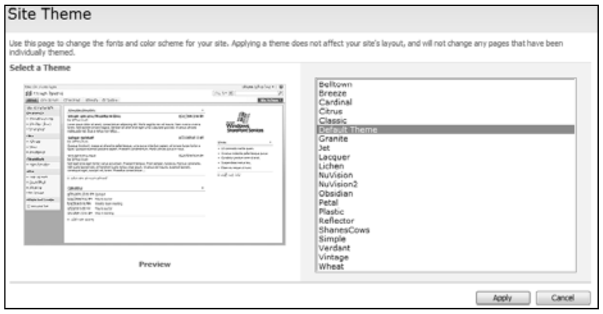
- What are Themes?
- Applying a Theme to a SharePoint site
- Themes behind the scenes
- Creating your own Theme
- Modifying your Theme
- Theme Tips and Tricks

What are Themes?

- Simple styling and branding for both WSS and MOSS sites
  - While master pages can change the actual “layout” of a site, Themes can only change CSS and background images
- Apply styling to all of one particular SharePoint site
  - Including Application pages (/_layouts)
  - Master pages do not style Application pages
- Can be used along with master pages for even more customization

Applying a Theme to a SharePoint site

- MOSS
  - Site Actions > Site Settings > Modify All Site Settings Look and Feel > Site Theme
- WSS
  - Site Settings > Site Theme



Site Theme

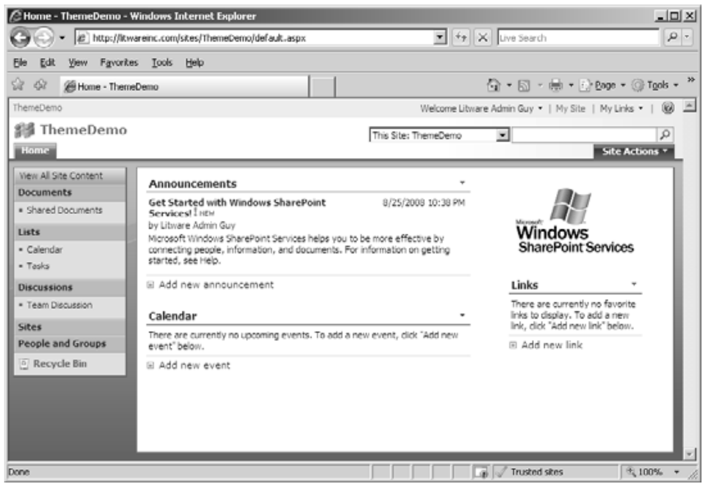
Use this page to change the fonts and color scheme for your site. Applying a theme does not affect your site's layout, and will not change any pages that have been individually themed.

Select a Theme

Belltown
Breeze
Cardinal
Citrus
Classic
Default Theme
Granite
Jet
Lacquer
Lichen
NuVision
NuVision2
Obsidian
Petal
Plastic
Reflector
ShanesCows
Simple
Verdant
Vintage
Wheat

Applying a Theme to a SharePoint site

Site with Lacquer Theme applied



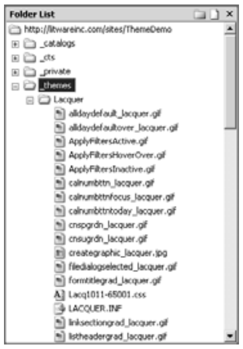
Get Started with Windows SharePoint Services! NEW 8/25/2008 10:38 PM
by Litware Admin Guy
Microsoft Windows SharePoint Services helps you to be more effective by connecting people, information, and documents. For information on getting started, see Help.

There are currently no upcoming events. To add a new event, click "Add new event" below.

There are currently no favorite links to display. To add a new link, click "Add new link" below.

Applying a Theme to a SharePoint site

- What happens when a Theme is applied?
  - Files provision in site at "_themes/Lacquer"
  - mossExtension.css is added to the end of theme.css and a new Theme file is created "Lacq1011-65001.css"



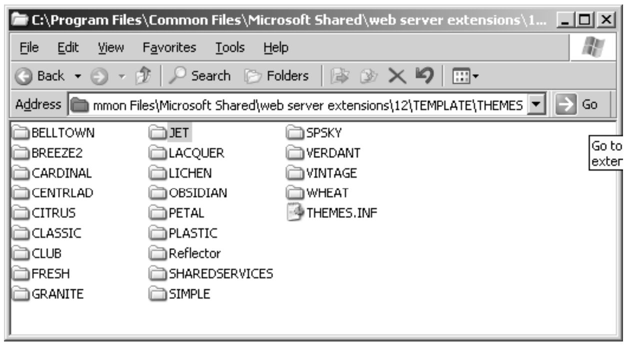
```
http://litwareinc.com/sites/ThemeDemo
  _catalogs
  _cts
  _private
  _themes
    Lacquer
      alldaydefault_lacquer.gif
      alldaydefaultover_lacquer.gif
      ApplyFiltersActive.gif
      ApplyFiltersHoverOver.gif
      ApplyFiltersInactive.gif
      calnumbttn_lacquer.gif
      calnumbttnfocus_lacquer.gif
      calnumbttntoday_lacquer.gif
      cnspgrdn_lacquer.gif
      cnsugrdn_lacquer.gif
      creategraphic_lacquer.jpg
      filedialogselected_lacquer.gif
      formtitlegrad_lacquer.gif
      Lacq1011-65001.css
      LACQUER.INF
      linksectiongrad_lacquer.gif
      listheadergrad_lacquer.gif
```

How Master Pages Refer to Themes

- Master pages refer to Themes in the <head>
  `<SharePoint:Theme runat="server"/>`
- If that tag is excluded, MOSS will add a meta tag to the end of the <head>
  `<meta name="Microsoft Theme" content=?Lacquer 1011, default">`
- Forcing a master page to NOT load Themes
  - http://blog.drisgill.com/2008/08/disabling-themes-on-custom-master-pages.html

The Themes Folder

- Themes Folder is located in the 12 Folder
  - `C:\Program Files\Common Files\Microsoft Shared\web server extensions\12\TEMPLATE\THEMES`


C:\Program Files\Common Files\Microsoft Shared\web server extensions\1...
File Edit View Favorites Tools Help
Back Search Folders
Address mmon Files\Microsoft Shared\web server extensions\12\TEMPLATE\THEMES Go
Go to exter

```
BELLTOWN    JET            SPSKY
BREEZE2     LACQUER        VERDANT
CARDINAL    LICHEN         VINTAGE
CENTRLAD    OBSIDIAN       WHEAT
CITRUS      PETAL          THEMES.INF
CLASSIC     PLASTIC
CLUB        Reflector
FRESH       SHAREDSERVICES
GRANITE     SIMPLE
```

The Theme INF File

- LACQUER.INF

```
[info]
title=Lacquer
codepage=65001
version=3.00
format=2.00
readonly=true
refcount=0

[titles]
1031=Lacquer
...
```

Other Files in the Theme Folder

- theme.css
  - Most of the CSS for the Theme
- mossExtension.css
  - Extra CSS applied to the end of the Theme CSS for MOSS sites
- Supporting images and CSS

SPTHEMES.XML

- Contains references to all Themes for SharePoint
- Located in the 12 Folder:
  - `C\Program Files\Common Files\Microsoft Shared\web server extensions\12\TEMPLATE\LAYOUTS\1033\SPTHEMES.XML`

```
<Templates>
   <TemplateID>Lacquer</TemplateID>
   <DisplayName>Lacquer</DisplayName>
   <Description>Lacquer has a gray background with gray control areas and
orange highlights.</Description>
   <Thumbnail>images/thlacquer.gif</Thumbnail>
   <Preview>images/thlacquer.gif</Preview>
</Templates>
```

Thumbnail / Preview Images

- Thumbnail and preview images are located in the 12 Folder:
  - `C:\Program Files\Common Files\Microsoft Shared\web server extensions\12\TEMPLATE\IMAGES`
- OOTB Thumbnails are named “th*.gif”
- Can be GIF, JPG, or PNG

Creating your own Theme

- Copy an OOTB Theme that is a similar design
  - Make a copy of the Lacquer folder
- Name the folder and .INF file the same ("WingTip" your new Theme name)
  - Alpha-Numeric only, no special characters or spaces

Creating your own Theme

- Edit WINGTIP.INF file and replace all instances of OOTB Theme name (Lacquer) with your Theme name (WingTip)

```
[info]
title=WingTip
codepage=65001
version=3.00
format=2.00
readonly=true
refcount=0

[titles]
1031=WingTip
...
```

Creating your own Theme

- Edit SPTHEMES.XML, add an entry for your new Theme before `</SPThemes>`

```
<Templates>
   <TemplateID>WingTip</TemplateID>
   <DisplayName>WingTip</DisplayName>
   <Description>Theme for WingTip Inc.</Description>
   <Thumbnail>images/th_wingtip.jpg</Thumbnail>
   <Preview>images/th_wingtip.jpg</Preview>
</Templates>
```

Creating your own Theme

- Add a placeholder preview thumbnail to `\12\TEMPLATE\IMAGES`
  - Replace later with an actual screenshot
- Make a small change to theme.css

Creating your own Theme

- Run iisreset /noforce
  - Obviously your will want to not do this on production until you are ready
- Browse to your SharePoint site and select and apply the new Theme
  - Site Actions > Site Settings > Modify All Site Settings | Look and Feel > Site Theme

Modifying your Theme

- Open the site in SharePoint Designer
- Navigate to _themes > WingTip
- Open Wing1011-65001.css and make some changes
- Add or change images in the _themes > WingTip folder
- Your changes appear immediately after browser refresh
  - Much nicer than changing the Theme folder in the 12 and doing an IISRESET and re-applying the Theme

Modifying your Theme

- When you are happy with your changes move your CSS and images back to the 12 Folder in order to make them permanent
  - Do not run IISRESET or switch the Theme beforehand!
- Run iisreset /noforce
- Reselect and apply your Theme from the browser

Theme Tips and Tricks

- Strange errors
  - "Write Error on file _themes\“

    Check for non typical file or folder names in the Theme folder (look for backup folders)
- Importing Theme CSS
  - Allows for easier maintenance, no need to ResetIIS or re-apply Theme
  - Slightly more complicated structure and references
  - Add an import statement to the theme.css file

    ```
    @import "/_layouts/1033/styles/WingTip/theme.css";
    ```

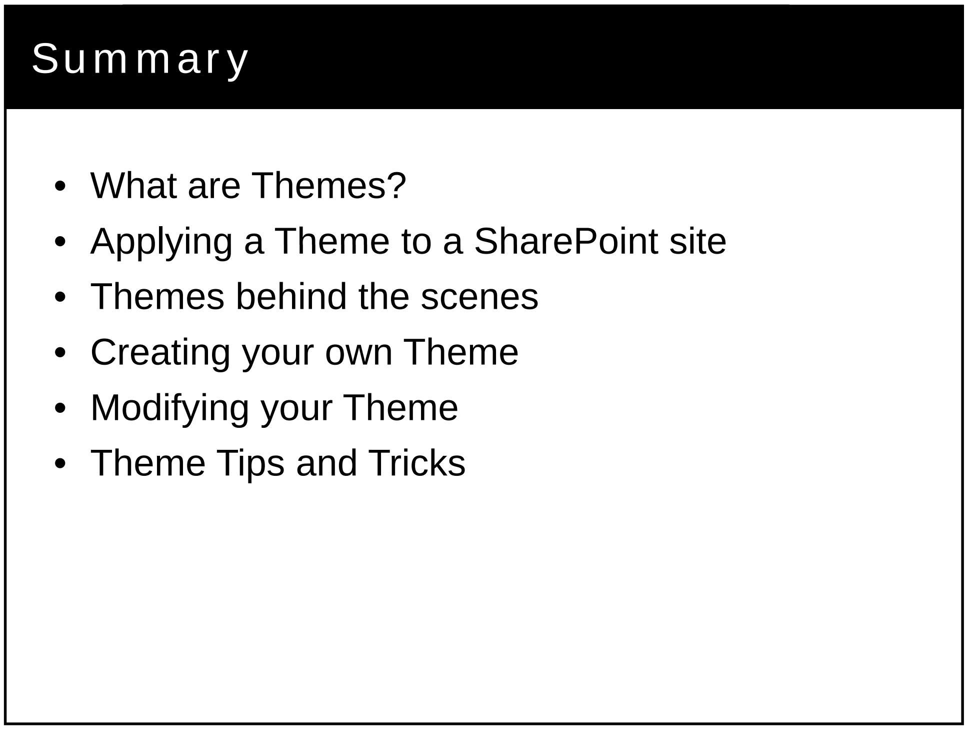
    http://www.heathersolomon.com/blog/archive/2008/01/30/SharePoint-2007-Design-Tip-Import-your-CSS-for-SharePoint-Themes.aspx

Theme Tips and Tricks

- IE Developers Toolbar http://www.microsoft.com/downloads/details.aspx?FamilyId=E59C3964-672D-4511-BB3E-2D5E1DB91038&displaylang=en
- Firebug - http://getfirebug.com
- Both allow visual inspection of applied CSS, Firebug allows you to manipulate the CSS in real time

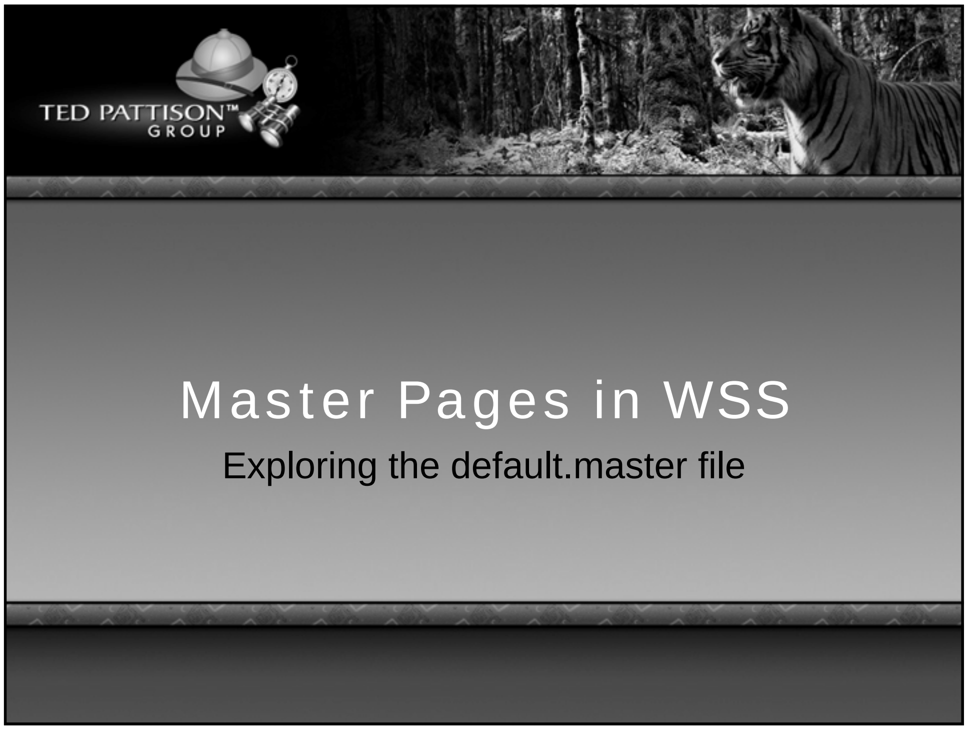
Deploying Themes

- Automatic deployment http://solutionizing.net/2008/07/09/updating-spthemesxml-through-a-feature/
- Feature stapling adds the Theme to the Site Definition so that the Theme is added when a new site is created http://www.grahamzero.com/thoughtfactory/2007/06/automatically_a.html



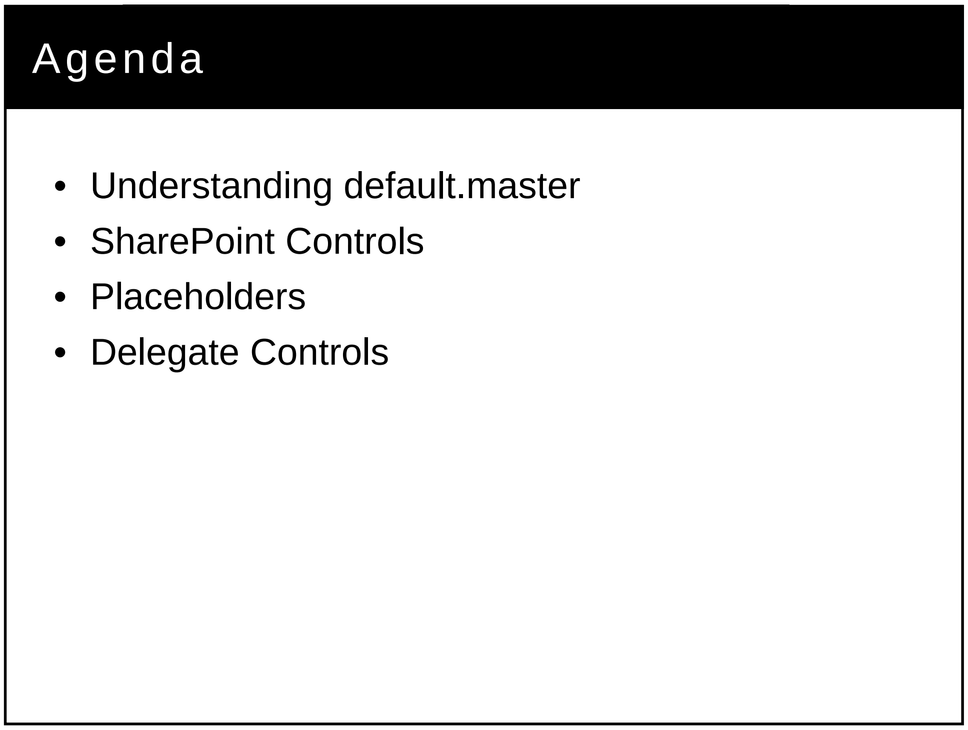
Summary

- What are Themes?
- Applying a Theme to a SharePoint site
- Themes behind the scenes
- Creating your own Theme
- Modifying your Theme
- Theme Tips and Tricks



Master Pages in WSS

Exploring the default.master file

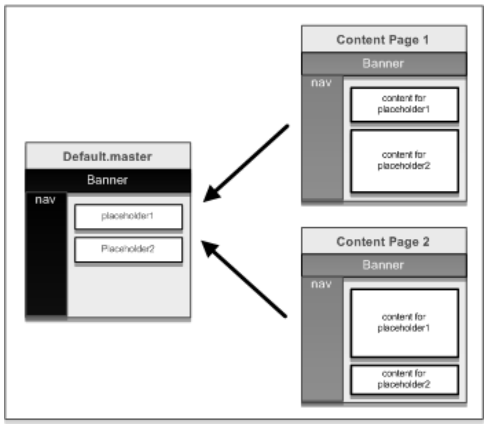


Agenda

- Understanding default.master
- SharePoint Controls
- Placeholders
- Delegate Controls

ASP.NET Background: Master Pages

- ASP.NET 2.0 introduces Master Pages
  - Defines common layouts used across content pages

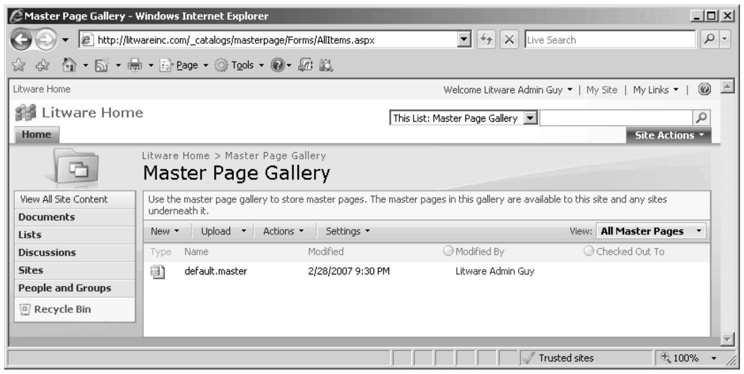


Master Pages in WSS

- Application pages use application.master
  - Farm-wide master page
  - Cannot be customized on per-site basis
- Site Pages use default. master by default
  - default.master is a page template
  - default.master instance created in Master Page Gallery
  - default.master can be customized on per-site basis
  - default.master can be replaced with different template

The Master Page Gallery

- Each site has a Master Page Gallery
  - Instance of default.master automatically provisioned
  - default.master can be customized on per-site basis

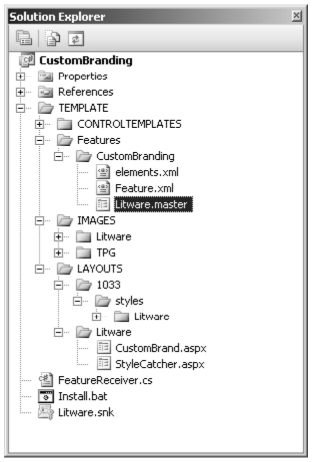


Use the master page gallery to store master pages. The master pages in this gallery are available to this site and any sites underneath it.

Type	Name	Modified	Modified By	Checked Out To
	default.master	2/28/2007 9:30 PM	Litware Admin Guy	

Demo: CustomBranding

- Important Concepts
  - Custom Master Page Templates
  - Custom CCS File
  - Custom Site Logo



```
CustomBranding
  Properties
  References
  TEMPLATE
    CONTROLTEMPLATES
    Features
      CustomBranding
        elements.xml
        Feature.xml
        Litware.master
    IMAGES
      Litware
      TPG
    LAYOUTS
      1033
        styles
          Litware
      Litware
        CustomBrand.aspx
        StyleCatcher.aspx
  FeatureReceiver.cs
  Install.bat
  Litware.snk
```

Custom Master Page Templates

- Creating a Master Page Template
  - Use default.master as a starting point
  - Make changes to suit your tastes
- Master Page templates like site page template
  - Support ghosting and unghosting
  - Provisioned using a File element within a Module

```
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
  <Module Name="MasterPages" List="116" Url="_catalogs/masterpage">
    <File Url="Litware.master" Type="GhostableInLibrary" />
  </Module>
</Elements>
```

Master Page Elements

```
<%@Master language="C#"%>
<%@ Register Tagprefix="SharePoint"
    Namespace="Microsoft.SharePoint.WebControls"
    Assembly="Microsoft.SharePoint, …" %>

<HTML id="HTML1" runat="server">
<HEAD id="HEAD1" runat="server">

  <!-- SharePoint Utility Controls -->
  <SharePoint:CssLink ID="CssLink1" runat="server"/>
  <SharePoint:Theme ID="Theme1" runat="server"/>

  <!-- Named Placeholders -->
  <Title ID=onetidTitle>
    <asp:ContentPlaceHolder id=PlaceHolderPageTitle runat="server"/>
  </Title>
  <asp:ContentPlaceHolder id="PlaceHolderAdditionalPageHead" runat="server"/>

    <!-- Named Delegate Control -->
    <SharePoint:DelegateControl
      ID="DelegateControl1" runat="server"
      ControlId="AdditionalPageHead" AllowMultipleControls="true"/>

</HEAD>
```

Updating the MasterUrl Property

- Update MasterUrl to redirect site pages
  - Child site can reference Master Page in top-level site

```
protected void cmdApplyCustomBrand_Click(object sender, EventArgs e) {
  SPWeb site = SPContext.Current.Site.RootWeb
  string MasterUrlPath = site.ServerRelativeUrl;
  if (!MasterUrlPath.EndsWith(@"/"))
    MasterUrlPath += @"/";
  MasterUrlPath += @"_catalogs/masterpage/Litware.master";
  ApplyCustomBrand(MasterUrlPath, site);
}

protected void ApplyCustomBrand(string MasterUrlPath, SPWeb site) {
  site.MasterUrl = MasterUrlPath;
  site.Update();
  // use recusion to update all child sites in site collection
  foreach (SPWeb child in site.Webs) {
    ApplyCustomBrand(MasterUrlPath, child);
  }
}
```

Applying Custom Branding

```
protected void cmdApplyCustomBrand_Click(object sender, EventArgs e) {

  SPWeb site = SPContext.Current.Web;

  string MasterUrlPath = site.ServerRelativeUrl;
  if (!MasterUrlPath.EndsWith(@"/"))
    MasterUrlPath += @"/";
  MasterUrlPath += @"_catalogs/masterpage/Litware.master";
  ApplyCustomBrand(MasterUrlPath, site);

  Response.Redirect(Request.RawUrl);
}

protected void ApplyCustomBrand(string MasterUrlPath, SPWeb site) {
  site.ApplyTheme("");
  site.MasterUrl = MasterUrlPath;
  site.AlternateCssUrl = "/_layouts/1033/STYLES/Litware/LitwareBrand.css";
  site.SiteLogoUrl = "/_layouts/images/Litware/LitwareFullLogo.png";
  site.Update();

  foreach (SPWeb child in site.Webs) {
    ApplyCustomBrand(MasterUrlPath, child);
  }
}
```

Creating Pages using default.master

- Commonly-used placeholders
  - PlaceHolderMain
  - PlaceHolderPageTitle
  - PlaceHolderPageTitleInTitleArea
  - PlaceHolderAdditionalPageHead
- Less commonly used placeholder

PlaceHolderGlobalNavigation	PlaceHolderGlobalNavigationSiteMap	PlaceHolderSiteName
PlaceHolderSearchArea	PlaceHolderTopNavBar	PlaceHolderHorizontalNav
WSSDesignConsole	SPNavigation	PlaceHolderPageImage
PlaceHolderTitleLeftBorder	PlaceHolderTitleBreadcrumb	PlaceHolderMiniConsole
PlaceHolderTitleRightMargin	PlaceHolderTitleAreaSeparator	PlaceHolderLeftNavBarDataSource
PlaceHolderCalendarNavigator	PlaceHolderLeftNavBarTop	PlaceHolderLeftNavBar
PlaceHolderLeftActions	PlaceHolderNavSpacer	PlaceHolderLeftNavBarBorder
PlaceHolderBodyLeftBorder	PlaceHolderPageDescription	PlaceHolderBodyRightMargin
PlaceHolderFormDigest"	PlaceHolderUtilityContent	PlaceHolderBodyAreaClass
PlaceHolderTitleAreaClass		

Demo

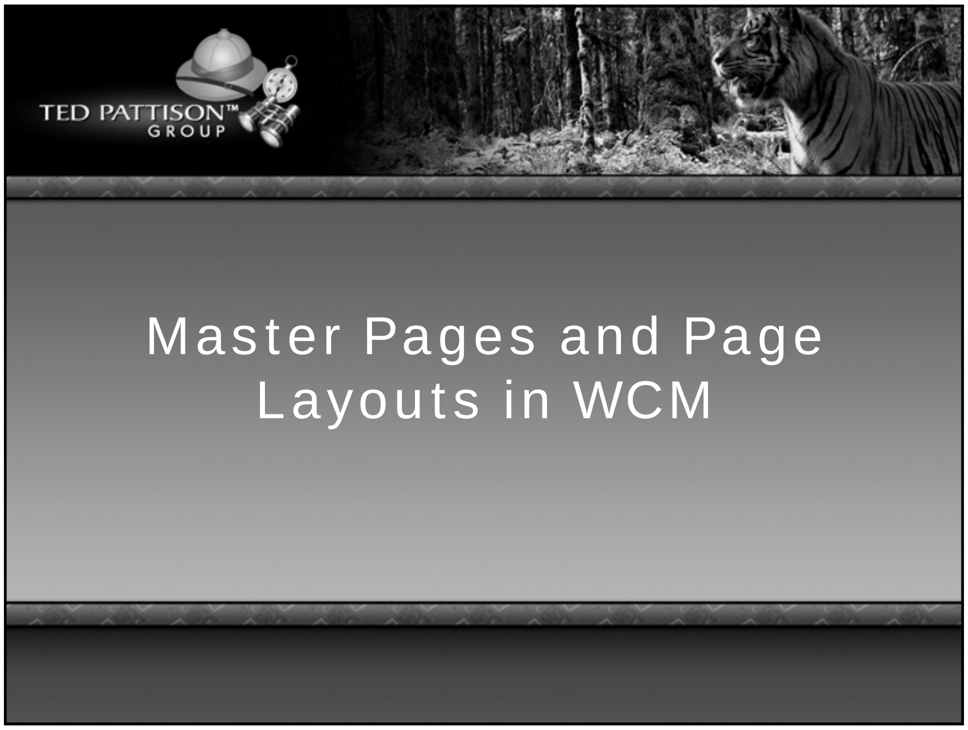
- Learn how to…
  - Begin with minimal WSS master page
  - Use SharePoint controls
  - Use PlaceHolders from default.master
  - Use delegate controls

Adding Web Part Zones to a Page

- Web Parts must go inside Web Part Zones
  - SPD allows you to add zones to pages
  - SharePoint uses zone ID to remember Web Part instances and their customization data

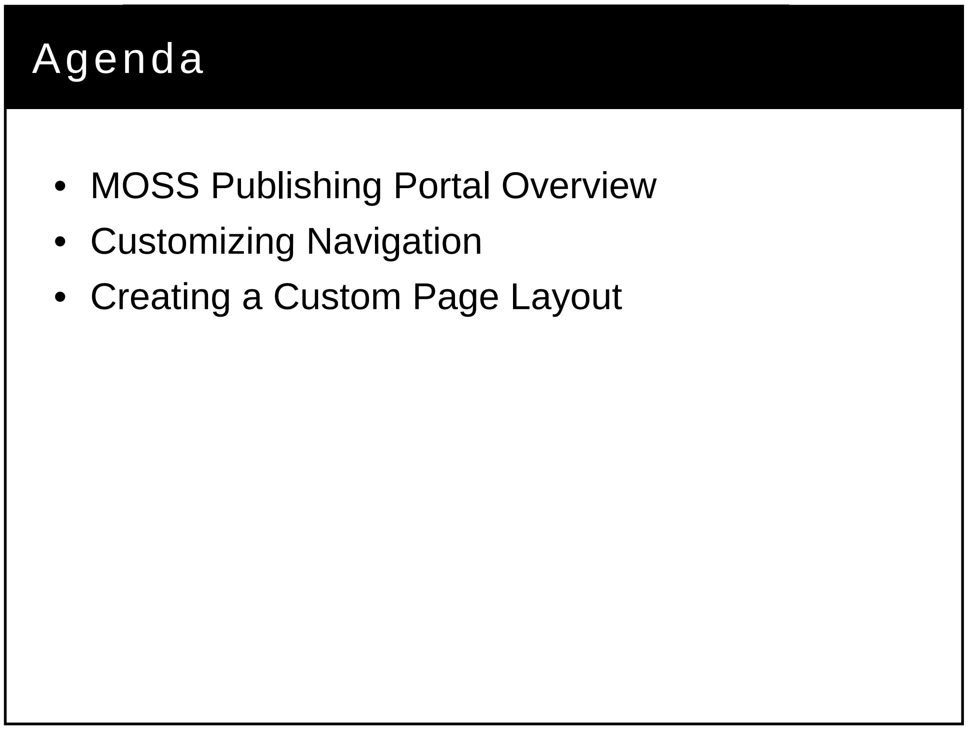
Summary

- Understanding default.master
- SharePoint Controls
- Placeholders
- Delegate Controls



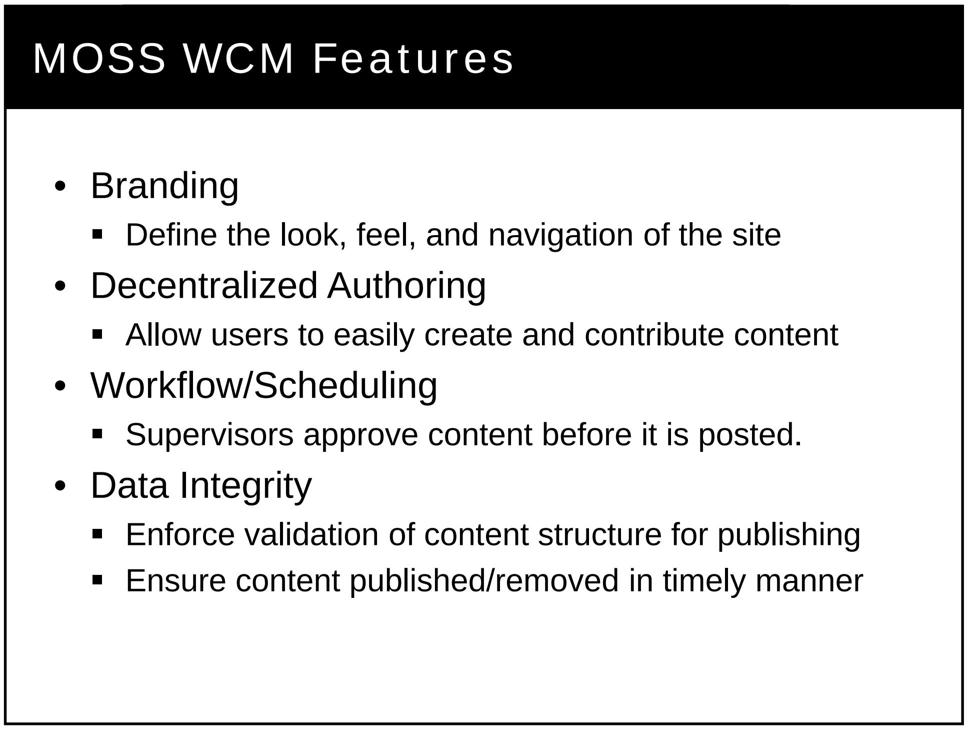
TED PATTISON™ GROUP

Master Pages and Page Layouts in WCM



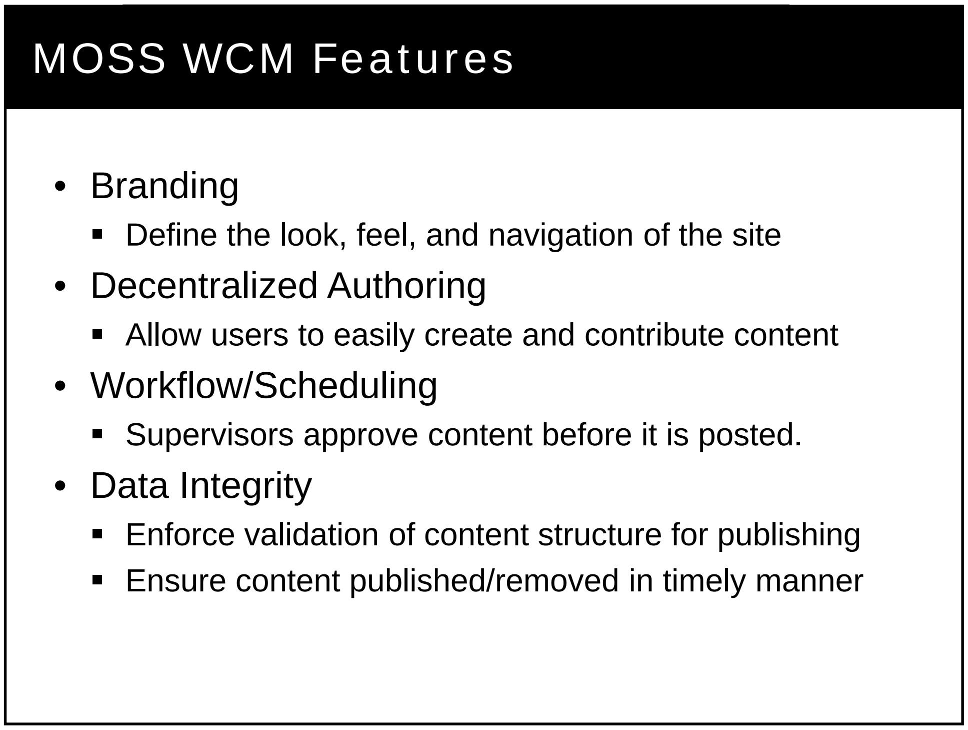
Agenda

- MOSS Publishing Portal Overview
- Customizing Navigation
- Creating a Custom Page Layout



MOSS WCM Features

- Branding
  - Define the look, feel, and navigation of the site
- Decentralized Authoring
  - Allow users to easily create and contribute content
- Workflow/Scheduling
  - Supervisors approve content before it is posted.
- Data Integrity
  - Enforce validation of content structure for publishing
  - Ensure content published/removed in timely manner

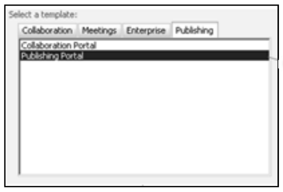


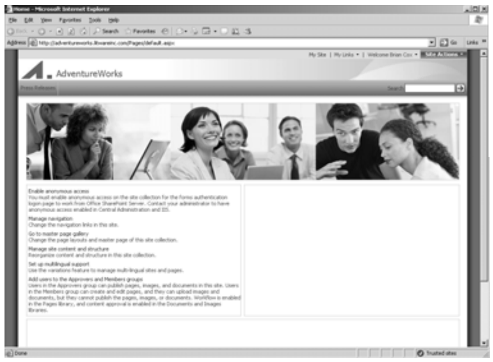
MOSS WCM Features

- Branding
  - Define the look, feel, and navigation of the site
- Decentralized Authoring
  - Allow users to easily create and contribute content
- Workflow/Scheduling
  - Supervisors approve content before it is posted.
- Data Integrity
  - Enforce validation of content structure for publishing
  - Ensure content published/removed in timely manner

Creating A Publishing Portal

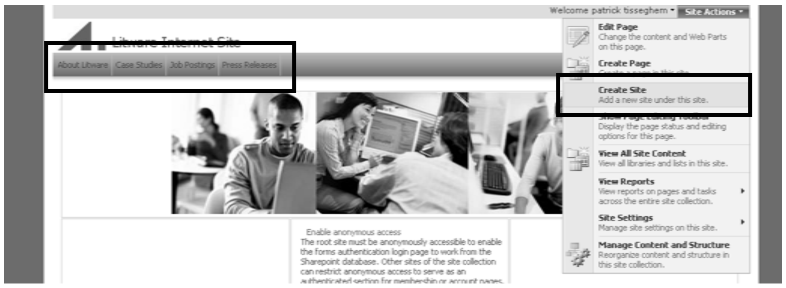
- Creating with WSS Central Administration
  - Create a site collection based on Publishing Portal





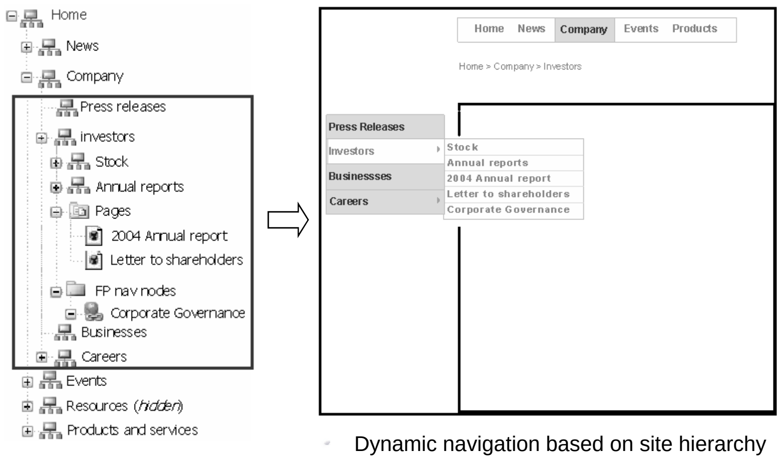
Site Hierarchy

- In the past a lot of confusion
  - Windows SharePoint Services 2003 → sites
  - SharePoint Portal Server 2003 → areas
  - Content Management Server 2002 → channels
- In SharePoint 2007 everything is a site


Welcome patrick tisseghem
Site Actions
Litware Internet Site
About Litware
Case Studies
Job Postings
Press Releases
Edit Page
Change the content and Web Parts on this page.
Create Page
Create Site
Add a new site under this site.
Display the page status and editing options for this page.
View All Site Content
View all libraries and lists in this site.
View Reports
View reports on pages and tasks across the entire site collection.
Site Settings
Manage site settings on this site.
Manage Content and Structure
Reorganize content and structure in this site collection.

Enable anonymous access
The root site must be anonymously accessible to enable the forms authentication login page to work from the Sharepoint database. Other sites of the site collection can restrict anonymous access to serve as an authenticated section for membership or account pages.

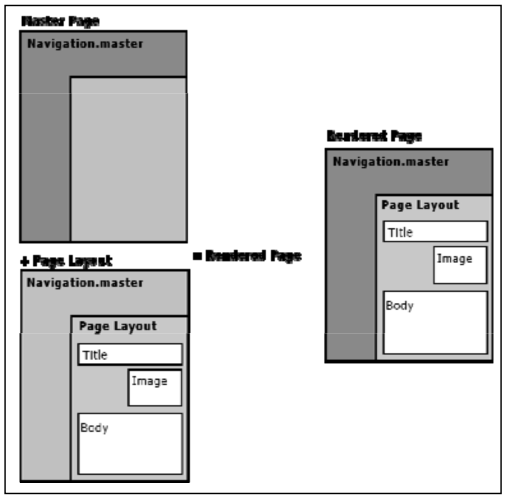
Navigation



- Dynamic navigation based on site hierarchy
- Includes webs, pages and authored links
- Navigation links trimmed based on security, workflow state and publishing schedule

Page = Master Page + Page Layout

- Master page defines banner and navigation
- Page layout ASPX defines how page content is rendered
- Possible scenario
  - 1-3 Master pages
  - 10-25 Page Layouts
  - 10s of 1000s of Content Pages

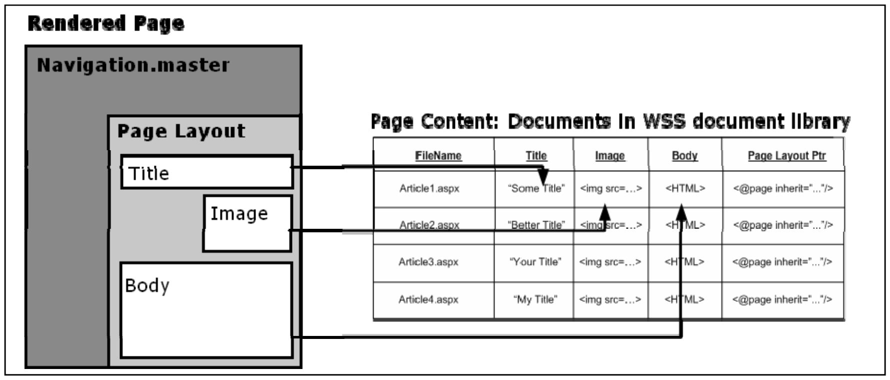


Page Layouts

- Page execution:
- Page URL requested
- Page layout executed in content of page
- Content server controls bind to page fields
- Rendered page returned

Inherited from WSS:

- Versioning,
- Check-in/Check-out
- Content types
- Access control
- Workflow



Page Content: Documents in WSS document library

FileName	Title	Image	Body	Page Layout Ptr
Article1.aspx	"Some Title"	<img src=...>	<HTML>	<@page inherit="..."/>
Article2.aspx	"Better Title"	<img src=...>	<HTML>	<@page inherit="..."/>
Article3.aspx	"Your Title"	<img src=...>	<HTML>	<@page inherit="..."/>
Article4.aspx	"My Title"	<img src=...>	<HTML>	<@page inherit="..."/>

Steps to Create a New Page Layout

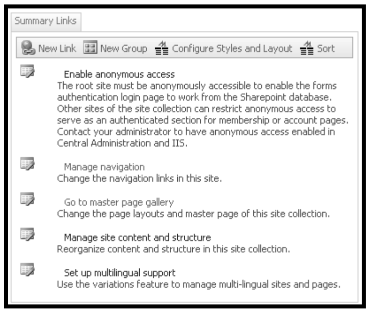
- Create shared columns
- Create content type
- Add created site columns to content type
- In the Master Page Gallery
  - Create new Page Layout file
  - Check-out file and edit in SharePoint Designer
  - Populate the file with content fields
  - Check-in and approve
- Use the new page layout file

Publishing Cycle

- Workflow based on Windows Workflow Foundation
- Light-weight approval workflow is active OOB
  - Based on approval
  - Minor versions need to be approved to become major versions
  - Visitors only see the major (published) versions
- Workflow can be replaced by custom workflow
  - OOB delivered with MOSS 2007
  - Designed using SharePoint Designer 2007
  - Created using Visual Studio.NET 2005

WCM Web Parts

- Summary Links Web Part
  - Custom annotated, stylized links
- Table of Contents Web Part
  - Displays navigation information of your site
- Content Query Web Part
  - Displays a dynamic view of the content in your site



Enable anonymous access
The root site must be anonymously accessible to enable the forms authentication login page to work from the Sharepoint database. Other sites of the site collection can restrict anonymous access to serve as an authenticated section for membership or account pages. Contact your administrator to have anonymous access enabled in Central Administration and IIS.

Manage navigation
Change the navigation links in this site.

Go to master page gallery
Change the page layouts and master page of this site collection.

Manage site content and structure
Reorganize content and structure in this site collection.

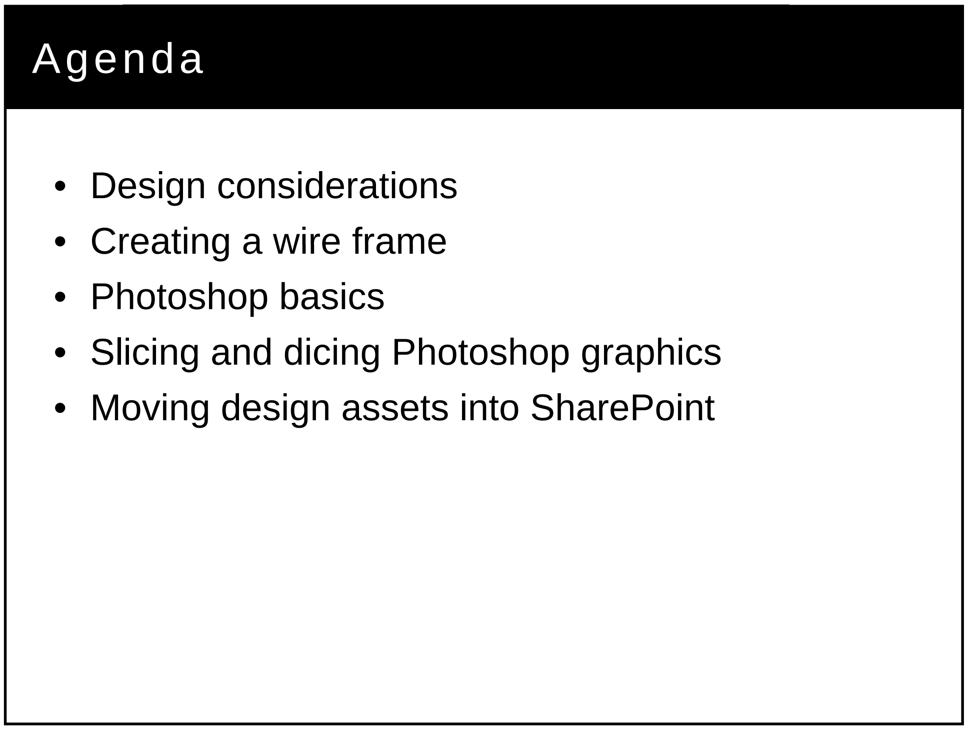
Set up multilingual support
Use the variations feature to manage multi-lingual sites and pages.

Summary

- MOSS Publishing Portal Overview
- Customizing Navigation
- Creating a Custom Page Layout

Photoshop and SharePoint

Designing for SharePoint



Agenda

- Design considerations
- Creating a wire frame
- Photoshop basics
- Slicing and dicing Photoshop graphics
- Moving design assets into SharePoint

Design Considerations

- OOTB SharePoint Site
  - Default.master
  - Blueband.maste
- How far can you go with SharePoint design
  - http://www.ocps.net
  - http://www.kroger.com
  - http://www.cps.edu
  - http://biz.viacom.com/tvlandpress
  - http://www.westrac.com.au
  - List of Many MOSS Sites: http://www.wssdemo.com/Pages/websites.aspx

Design Considerations

- Differences between Intranet and Internet
  Internet:
  - Internet sites are usually focused on publishing information
  - Few content authors / many consumers
  - Usually highly customized UI

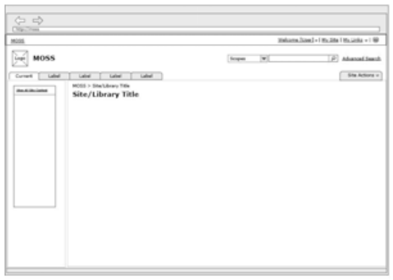
  Intranet:
  - Intranet sites are usually focused on collaboration
  - Many content authors and consumers
  - Usually a more simple UI with consistent branding
- Planning for “self service”
  - When creating a design for SharePoint, you must consider that areas of the design will be completely out of your control once the site goes live
  - Have policy in place for making sure corporate standards are met
  - Plan for change, once you empower your users to edit content you will quickly learn their needs

Design Considerations

- Taxonomy
  - Create a hierarchical flow chart of how your site will flow
  - Will each department get its own SharePoint site? Or will they just get pages?
  - With a sound taxonomy you can start to think about navigation needs
- Navigation
  - Do you need top and side navigation? Do you need multiple levels of navigation?
  - Will the navigation be constant across all pages or will it change per section
  - Will navigation be managed manually using the MOSS web UI
    `Site Actions > Site Settings > Modify Navigation`
  - Some of the navigation features can be controlled from the MOSS web UI others need to be controlled in the master page
  - http://blogs.msdn.com/ecm/archive/2007/02/10/moss-navigation-deep-dive-part-1.aspx

Creating a Wire Frame

- Wire frames can be created using any tool you would like
- They help you visualize where design elements will be located before the creative process begins



- http://www.siolon.com/blog/sharepoint-wireframes/

Design Considerations

- Functionality
  - Will your design require all of the OOTB features?
  - Search? Breadcrumbs? Top navigation? Left Navigation? Tree View? My Sites? My Links? Help? Login?
- Ease of Use
  - Clean / Simple UI (white space)
  - Keep the design consistent – Do the users feel like the entire site is one cohesive unit?
  - Make sure common functions are easily found – Multiple forms of navigation are ok
- Best Practices
  - Choose a maximum height and width in accordance with end user capabilities

    TheCounter.com lists 1024 × 768 and 1280 × 1024 as the most common resolutions
  - Image sizes
  - W3C Compliance

Photoshop Basics

- Overview of the Photoshop UI
  - Demo

Photoshop Basics

- Working with PSD's
  - Photoshop's native format
  - Supports layers and editable text
- Working with Layers and Groups
  - Create as many layers as needed to make sure each element is editable by itself
  - Group common layers for easy maintenance
  - Use layer effects to make color and gradient changes, as well as for other effects like bevels and drop shadows

Photoshop Basics

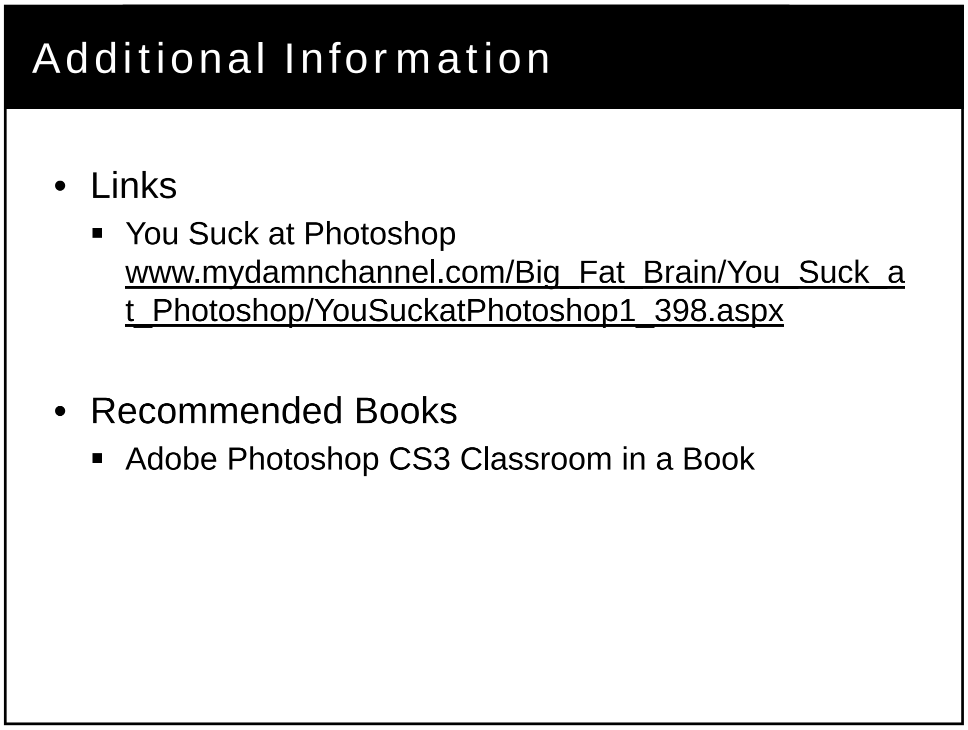
- Working with text
  - Be careful choosing body or link fonts that are unavailable on user machines
  - Be careful with anti-aliasing, sometimes text will look better or worse in Photoshop than it will on the web
  - Large text should probably be converted to images before exporting to SharePoint
  - Small text will usually not be converted to images, instead you are simply mockup up how dynamic text will look in SharePoint
- Working with colors
  - The eyedropper is good for getting hexadecimal values of colors to transfer to CSS
  - Use the Hue and Saturation menu to change hues of images
  - Image > Adjustments > Hue and Saturation

Slicing and dicing Photoshop graphics

- Creating repeating backgrounds
  - Often you will want to just save a small area of an image so that it can be repeated horizontally or vertically
- Saving for the web
  - File > Save for Web and Devices
  - Using GIF, JPG, and PNG formats
  - Compression, choose the smallest size without compromising the look
  - Creating transparent images
  - Transparent PNG's not fully supported in all browsers

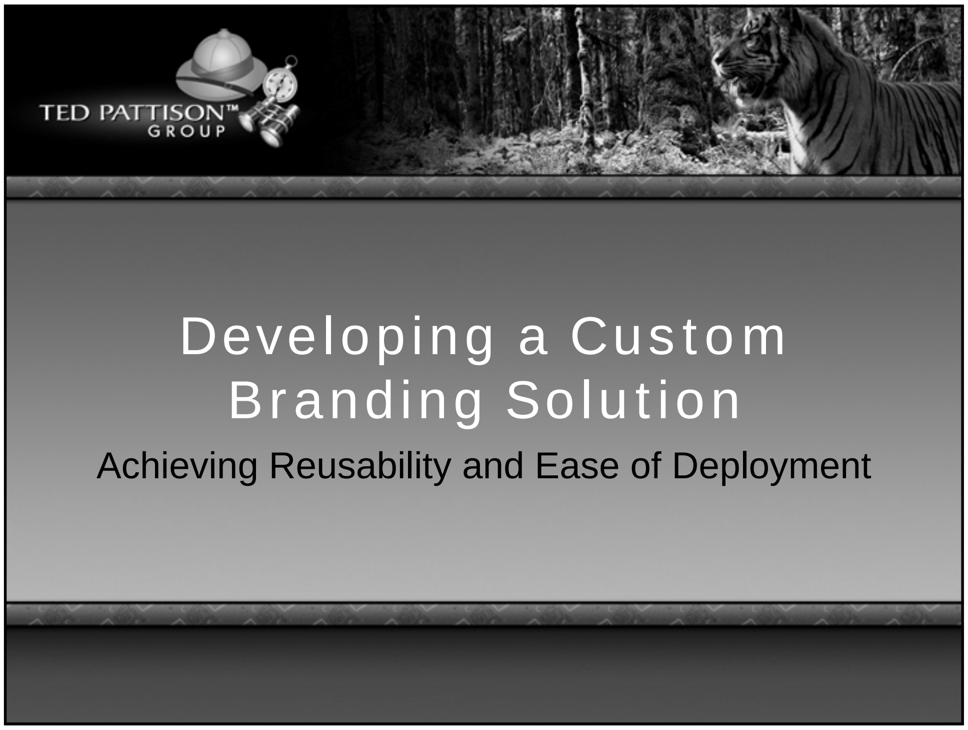
Moving design assets into SharePoint

- Where should everything live?
  - _layouts folder in the 12 folder
  - /style library/images
  - /images
- Uploading assets with the SharePoint web UI
  - Site Actions > Manage Content and Structure
  - Navigate to /style library/images
  - New > Item
- Uploading assets with SharePoint Designer
  - Navigate to /style library/images
  - Simply drag images from your computer into the Folder pane



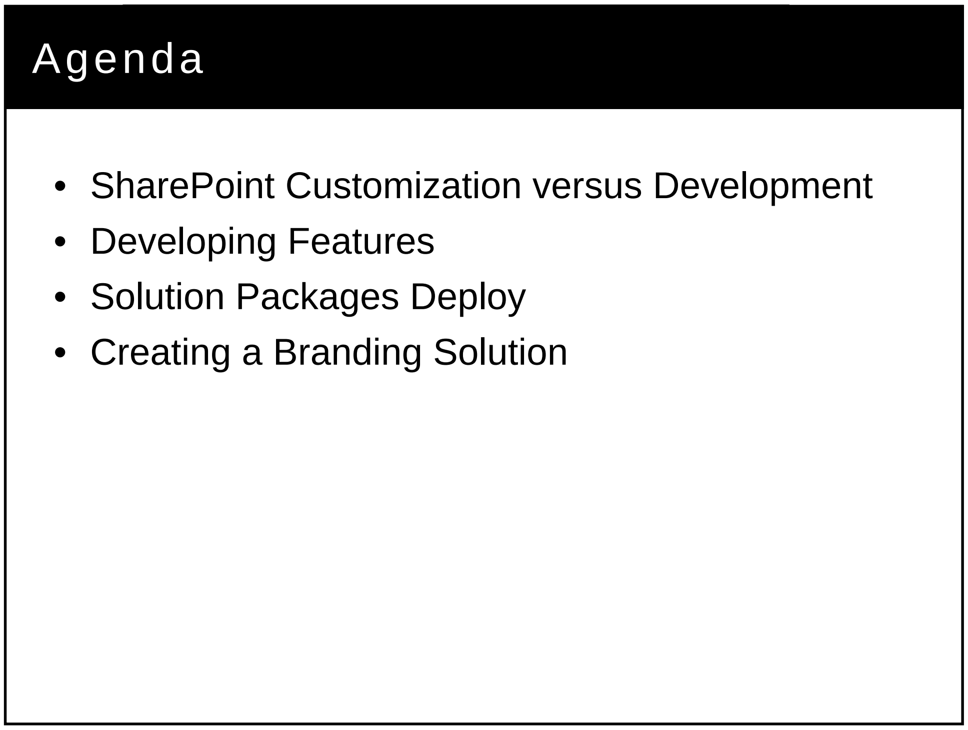
Additional Information

- Links
  - You Suck at Photoshop
    www.mydamnchannel.com/Big_Fat_Brain/You_Suck_at_Photoshop/YouSuckatPhotoshop1_398.aspx
- Recommended Books
  - Adobe Photoshop CS3 Classroom in a Book



Developing a Custom Branding Solution

Achieving Reusability and Ease of Deployment



Agenda

- SharePoint Customization versus Development
- Developing Features
- Solution Packages Deploy
- Creating a Branding Solution

Customization Versus Development

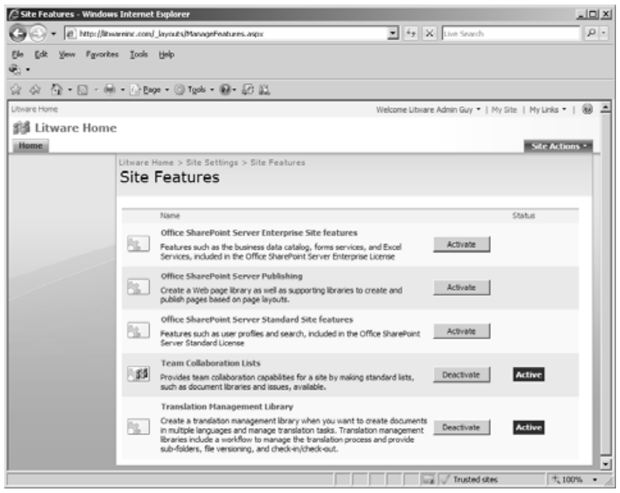
- Site Customizations
  - Changes to one particular site
  - Done using the browser or the SharePoint Designer
  - Changes recorded in content database
  - Easy to do but hard to reuse
- WSS Development
  - Creation of reusable templates/components
  - Templates/components installed on Web server
  - Development based on Visual Studio projects
  - Project source files checked into source code control
  - Projects can be moved through staging to production

What is a Feature?

- A building block for creating SharePoint solutions
  - A unit of design, implementation and deployment
- Features can contain elements
  - e.g. menu items, links, list types and list instances
  - Many other element types possible
- Features can contain event handlers
  - You can add any code which used WSS object model

User’s View of Features

- Features support concept of activation/deactivation



Litware Home > Site Settings > Site Features

Site Features

Name		Status
Office SharePoint Server Enterprise Site features Features such as the business data catalog, forms services, and Excel Services, included in the Office SharePoint Server Enterprise License	Activate	
Office SharePoint Server Publishing Create a Web page library as well as supporting libraries to create and publish pages based on page layouts.	Activate	
Office SharePoint Server Standard Site features Features such as user profiles and search, included in the Office SharePoint Server Standard License	Activate	
Team Collaboration Lists Provides team collaboration capabilities for a site by making standard lists, such as document libraries and issues, available.	Deactivate	Active
Translation Management Library Create a translation management library when you want to create documents in multiple languages and manage translation tasks. Translation management libraries include a workflow to manage the translation process and provide sub-folders, file versioning, and check-in/check-out.	Deactivate	Active

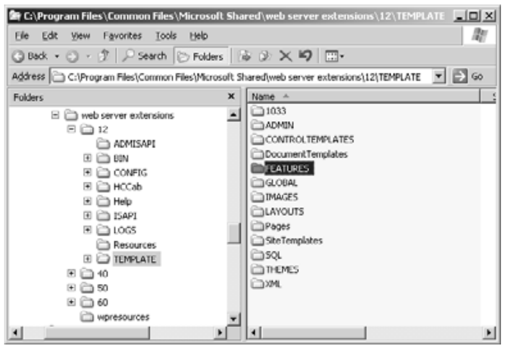
This is the site-level feature management page in a WSS farm where MOSS has been installed.

Much of the functionality of MOSS is enabled and disable by activating and deactivating features that have been developed by the MOSS team.

The WSS System Directories

- Developers must learn WSS system directories

`\12\TEMPLATE`
`\12\TEMPLATE\FEATURES` ← **`This is the one we care about in this lecture`**
`\12\TEMPLATE\IMAGES`
`\12\TEMPLATE\LAYOUTS`



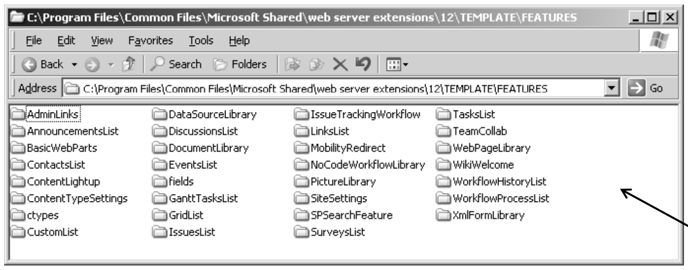
C:\Program Files\Common Files\Microsoft Shared\web server extensions\12\TEMPLATE

```
Folders                        Name
web server extensions          1033
  12                           ADMIN
    ADMISAPI                   CONTROLTEMPLATES
    BIN                        DocumentTemplates
    CONFIG                     FEATURES
    HCCab                      GLOBAL
    Help                       IMAGES
    ISAPI                      LAYOUTS
    LOGS                       Pages
    Resources                  SiteTemplates
    TEMPLATE                   SQL
  40                           THEMES
  50                           XML
  60
  wpresources
```

The Features Directory

- Functionality in WSS based on Features
  - Features are installed at farm level
  - Feature activation makes functionality available
  - WSS supports for different feature activation scopes

    `(1) Site (2) Site Collection (3) Web Application (4) Farm`



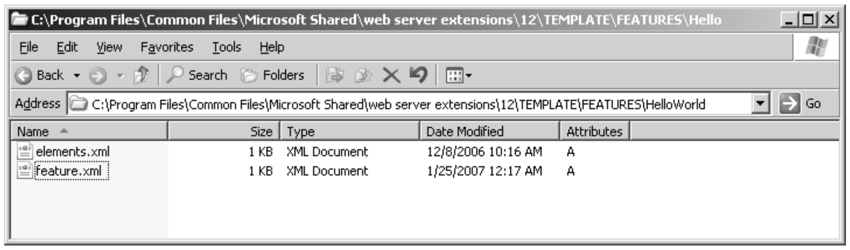
C:\Program Files\Common Files\Microsoft Shared\web server extensions\12\TEMPLATE\FEATURES

AdminLinks	DataSourceLibrary	IssueTrackingWorkflow	TasksList
AnnouncementsList	DiscussionsList	LinksList	TeamCollab
BasicWebParts	DocumentLibrary	MobilityRedirect	WebPageLibrary
ContactsList	EventsList	NoCodeWorkflowLibrary	WikiWelcome
ContentLightup	fields	PictureLibrary	WorkflowHistoryList
ContentTypeSettings	GanttTasksList	SiteSettings	WorkflowProcessList
ctypes	GridList	SPSearchFeature	XmlFormLibrary
CustomList	IssuesList	SurveysList	

Your custom features will each need a directory created here

Developer's View of a Feature

- Each feature requires its own directory
  - Directory must contain feature.xml file
  - Directory often contains other files definition elements

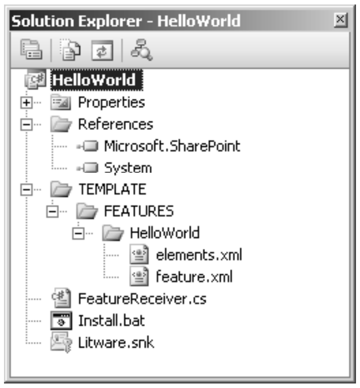


C:\Program Files\Common Files\Microsoft Shared\web server extensions\12\TEMPLATE\FEATURES\HelloWorld

Name	Size	Type	Date Modified	Attributes
elements.xml	1 KB	XML Document	12/8/2006 10:16 AM	A
feature.xml	1 KB	XML Document	1/25/2007 12:17 AM	A

Creating the 'Hello World' Feature

- Create a new Visual Studio Class Library project
  - Create XML files which define feature
  - Add a FeatureActivated event handler



```
Solution Explorer - HelloWorld
HelloWorld
  Properties
  References
    Microsoft.SharePoint
    System
  TEMPLATE
    FEATURES
      HelloWorld
        elements.xml
        feature.xml
  FeatureReceiver.cs
  Install.bat
  Litware.snk
```

The feature.xml file

- Feature.xml file serves as feature manifest
  - Features defined in declarative fashion using CAML

    `CAML = Collaborative Application Markup Language`

```
<Feature
  Id="B2CB42E2-4F0A-4380-AABA-1EF9CD526F20"
  Title="A Sample Feature: Hello World"
  Description=“Hi mom, class is fun. I am doing great"
  Scope="Web"
  Hidden="FALSE"
  ImageUrl="TPG\WhitePithHelmet.gif"
  xmlns="http://schemas.microsoft.com/sharepoint/">

  <ElementManifests>
    <ElementManifest Location="elements.xml" />
  </ElementManifests>

</Feature>
```

Elements.xml

- Feature includes elements defined using CAML
  - This element defines a Site Actions menu item
  - There are many other types of elements

```
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">

  <CustomAction
    Id="SiteActionsToolbar"
    GroupId="SiteActions"
    Location="Microsoft.SharePoint.StandardMenu"
    Sequence="100"
    Title="Hello World"
    Description="A custom menu item added using a feature"
    ImageUrl="_layouts/images/crtsite.gif" >

      <UrlAction Url="http://msdn.microsoft.com"/>

  </CustomAction>

</Elements>
```

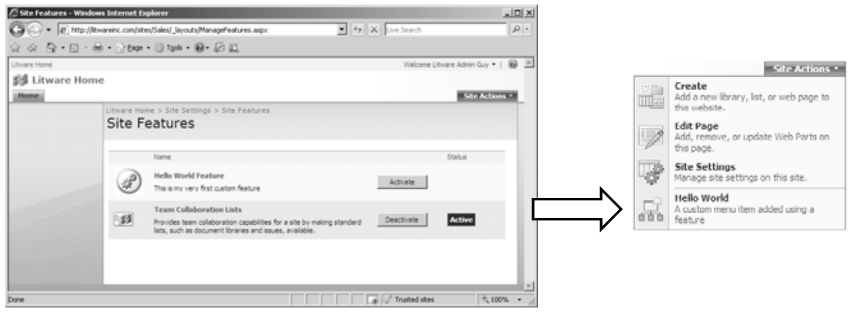
Install.bat

- Visual Studio supports post-build events
  - Can be used to run batch file to deploy components
  - Used on development machines
  - Should not be used on staging/production machines

```
@SET TEMPLATEDIR="c:\program files\common files\microsoft shared\web server extensions\12\Template"
@SET STSADM="c:\program files\common files\microsoft shared\web server extensions\12\bin\stsadm"
@SET GACUTIL="c:\Program Files\Microsoft Visual Studio 8\SDK\v2.0\Bin\gacutil.exe"

Echo Installing HelloWorld.dll in GAC
%GACUTIL% -if bin\debug\HelloWorld.dll

Echo Copying files to TEMPLATE directory
xcopy /e /y TEMPLATE\* %TEMPLATEDIR%

Echo Installing feature
%STSADM% -o installfeature -filename  HelloWorld\feature.xml -force

Echo Restart all IIS worker processes
IISRESET

Echo Restart just the IIS worker process for a particular Application Pool
REM cscript c:\windows\system32\iisapp.vbs /a "SharePointDefaultAppPool" /r
```

Feature Activation

- Steps to testing Feature
  - Copy Feature files to FEATURES directory
  - Install feature with WSS
  - Activate Feature within a specific site

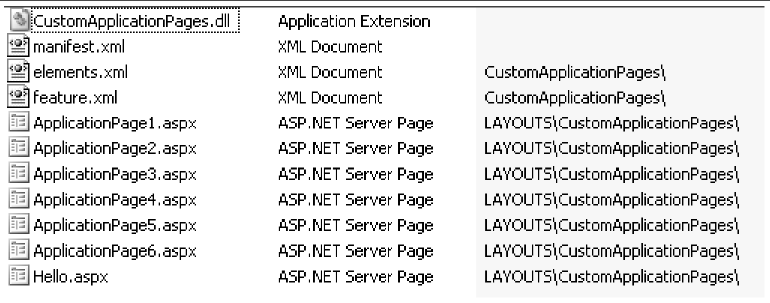

Site Features - Windows Internet Explorer
http://litwareinc.com/sites/Sales/_layouts/ManageFeatures.aspx
Litware Home
Welcome Litware Admin Guy
Home
Site Actions
Litware Home > Site Settings > Site Features
Site Features
Name
Status
Hello World Feature
This is my very first custom feature
Activate
Team Collaboration Lists
Provides team collaboration capabilities for a site by making standard lists, such as document libraries and issues, available.
Deactivate
Active
Done
Trusted sites
100%
Site Actions
Create
Add a new library, list, or web page to this website.
Edit Page
Add, remove, or update Web Parts on this page.
Site Settings
Manage site settings on this site.
Hello World
A custom menu item added using a feature

Deployment using Solution Packages

- Evolution of Web Part Packages from WSS 2.0
  - Solution Package is a CAB file with .wsp extension
  - Solution Package contains a manifest
  - Solution Package contains files required on Web server
- What can be deployed via a Solution Package
  - Feature definitions
  - Application Pages
  - Assembly DLLs
  - And much more…

Deployment using Solution Packages

- WSS Deployment done with Solution Packages
  - Solution Package is CAB file with .wsp extension
  - Created using DDF file and MAKECAB.EXE
  - Deployed using STSADM.EXE or WSS Central Admin



CustomApplicationPages.dll	Application Extension	
manifest.xml	XML Document	
elements.xml	XML Document	CustomApplicationPages\
feature.xml	XML Document	CustomApplicationPages\
ApplicationPage1.aspx	ASP.NET Server Page	LAYOUTS\CustomApplicationPages\
ApplicationPage2.aspx	ASP.NET Server Page	LAYOUTS\CustomApplicationPages\
ApplicationPage3.aspx	ASP.NET Server Page	LAYOUTS\CustomApplicationPages\
ApplicationPage4.aspx	ASP.NET Server Page	LAYOUTS\CustomApplicationPages\
ApplicationPage5.aspx	ASP.NET Server Page	LAYOUTS\CustomApplicationPages\
ApplicationPage6.aspx	ASP.NET Server Page	LAYOUTS\CustomApplicationPages\
Hello.aspx	ASP.NET Server Page	LAYOUTS\CustomApplicationPages\

CustomApplicationPages.wsp

Solution Package Manifest

- Solution Manifest read by WSS installer

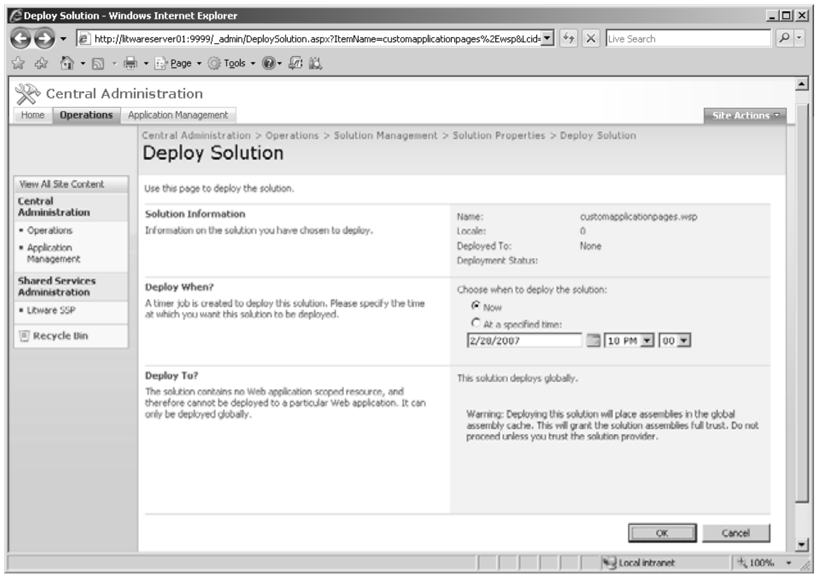
```
<Solution SolutionId="9EFFE92B-781D-4c99-BBCC-432D248B899D"
                    xmlns="http://schemas.microsoft.com/sharepoint/">

  <FeatureManifests>
    <FeatureManifest Location="CustomApplicationPages\feature.xml" />
  </FeatureManifests>

  <TemplateFiles>
    <TemplateFile Location="LAYOUTS\CustomApplicationPages\Hello.aspx"/>
    <TemplateFile Location="LAYOUTS\CustomApplicationPages\ApplicationPage1.aspx"/>
    <TemplateFile Location="LAYOUTS\CustomApplicationPages\ApplicationPage2.aspx"/>
    <TemplateFile Location="LAYOUTS\CustomApplicationPages\ApplicationPage3.aspx"/>
    <TemplateFile Location="LAYOUTS\CustomApplicationPages\ApplicationPage4.aspx"/>
    <TemplateFile Location="LAYOUTS\CustomApplicationPages\ApplicationPage5.aspx"/>
    <TemplateFile Location="LAYOUTS\CustomApplicationPages\ApplicationPage6.aspx"/>
  </TemplateFiles>

  <Assemblies>
    <Assembly Location="CustomApplicationPages.dll"
              DeploymentTarget="GlobalAssemblyCache" />
  </Assemblies>

</Solution>
```

Solution Package: install vs. deploy

- Solution Package Installation
  - WSP file copied into configuration database
  - Done using **addsolution** operation of STSADM.EXE
- Solution Package Deployment
  - WSP files copied to each FE Web Server and deployed
  - Done using **deploysolution** operation of STSADM.EXE

```
REM – a batch file named DeploySolutionPackage.cmd from CustomApplicationPage project
Echo Generating Solution Package CustomApplicationPages.wsp
if EXIST CustomApplicationPages.wsp del CustomApplicationPages.wsp
cd ..
makecab /f Solution\cab.ddf
cd package

Echo Installing CustomApplicationPages.wsp in WSS Solution Package Store
%STSADM% -o addsolution -filename CustomApplicationPages.wsp
%STSADM% -o execadmsvcjobs

Echo Deploying Solution Package CustomApplicationPages.wsp
%STSADM% -o deploysolution -name CustomApplicationPages.wsp -immediate -allowGacDeployment
%STSADM% -o execadmsvcjobs
```

Deploying Solution Packages



Central Administration > Operations > Solution Management > Solution Properties > Deploy Solution

Deploy Solution

Use this page to deploy the solution.

Solution Information
Information on the solution you have chosen to deploy.

Name: customapplicationpages.wsp
Locale: 0
Deployed To: None
Deployment Status:

Deploy When?
A timer job is created to deploy this solution. Please specify the time at which you want this solution to be deployed.

Choose when to deploy the solution:
[✓ Now]
At a specified time:
2/28/2007 10 PM 00

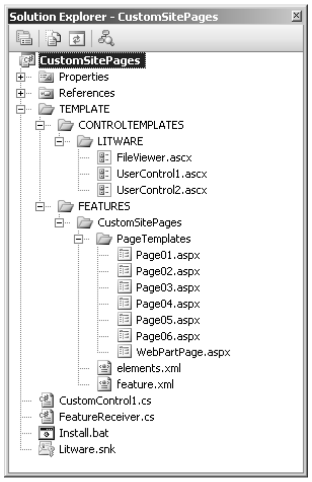
Deploy To?
The solution contains no Web application scoped resource, and therefore cannot be deployed to a particular Web application. It can only be deployed globally.

This solution deploys globally.

Warning: Deploying this solution will place assemblies in the global assembly cache. This will grant the solution assemblies full trust. Do not proceed unless you trust the solution provider.

Demo: CustomSitePages

- Important Concepts
  - Page template vs. page instance
  - Page customization
  - SafeMode processing



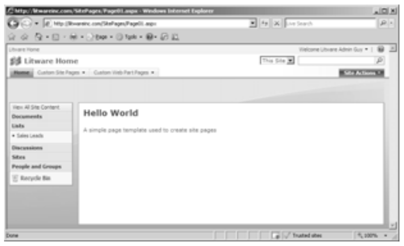
'Hello World' Page Template

- Page Template can be added to feature
  - MasterPageFile points to ~**`masterurl/default.master`**
  - **`progid`** adds support for SharePoint Designer

```
<%@ Page MasterPageFile="~masterurl/default.master"
    meta:progid="SharePoint.WebPartPage.Document"  %>

<asp:Content runat="server" ContentPlaceHolderID="PlaceHolderMain">

  <h3>Hello World</h3>

  A simple page template used to create site pages

</asp:Content>
```

Provisioning a Page Instance

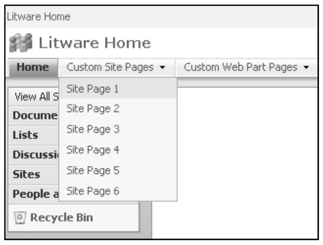
- Module element used to provision page instance
  - File element per page instance
  - Supports page ghosting



```
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">

  <Module Path="PageTemplates" Url="SitePages" >
    <File Url="Page01.aspx" Type="Ghostable" />
  </Module>

</Elements>
```

Adding Navigation Support for Pages

- Navigation nodes can be added
  - Can be added during feature activation
  - Can be added to top-link bar
  - Can be added to QuickLaunch
  - Nodes created as SPNavigationNode



```
public class FeatureReceiver : SPFeatureReceiver {
  public override void FeatureActivated(SPFeatureReceiverProperties properties) {
    // get a hold off current site in context of feature activation
    SPWeb site = (SPWeb)properties.Feature.Parent;
    SPNavigationNodeCollection topNav = site.Navigation.TopNavigationBar;

    // create dropdown menu for custom site pages
    SPNavigationNode DropDownMenu1 =
                     new SPNavigationNode("Custom Site Pages", "", false);
    topNav[0].Children.AddAsLast(DropDownMenu1);
    DropDownMenu1.Children.AddAsLast(
      new SPNavigationNode("Site Page 1", "SitePages/Page01.aspx"));
  }
}
```

Custom Master Page Templates

- Creating a Master Page Template
  - Use default.master as a starting point
  - Make changes to suit your tastes
- Master Page templates like site page template
  - Support ghosting and unghosting
  - Provisioned using a File element within a Module

```
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
  <Module Name="MasterPages" List="116" Url="_catalogs/masterpage">
    <File Url="Litware.master" Type="GhostableInLibrary" />
  </Module>
</Elements>
```

Master Page Elements

```
<%@Master language="C#"%>
<%@ Register Tagprefix="SharePoint"
    Namespace="Microsoft.SharePoint.WebControls"
    Assembly="Microsoft.SharePoint, …" %>

<HTML id="HTML1" runat="server">
<HEAD id="HEAD1" runat="server">

  <!-- SharePoint Utility Controls -->
  <SharePoint:CssLink ID="CssLink1" runat="server"/>
  <SharePoint:Theme ID="Theme1" runat="server"/>

  <!-- Named Placeholders -->
  <Title ID=onetidTitle>
    <asp:ContentPlaceHolder id=PlaceHolderPageTitle runat="server"/>
  </Title>
  <asp:ContentPlaceHolder id="PlaceHolderAdditionalPageHead" runat="server"/>

    <!-- Named Delegate Control -->
    <SharePoint:DelegateControl
      ID="DelegateControl1" runat="server"
      ControlId="AdditionalPageHead" AllowMultipleControls="true"/>

</HEAD>
```

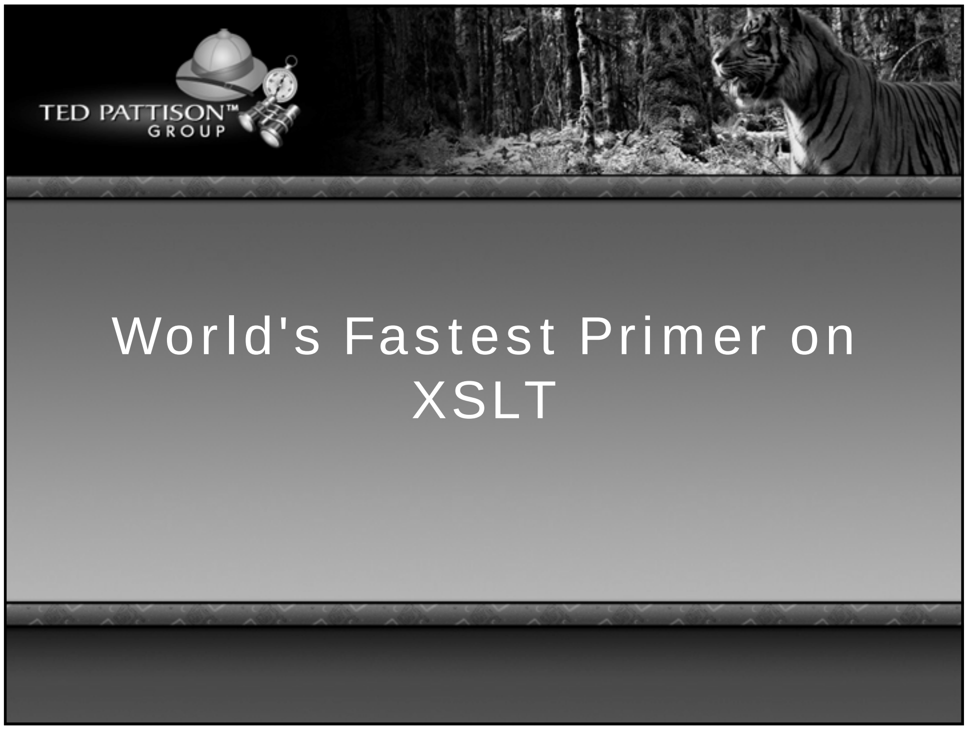
Updating the MasterUrl Property

- Update MasterUrl to redirect site pages
  - Child site can reference Master Page in top-level site

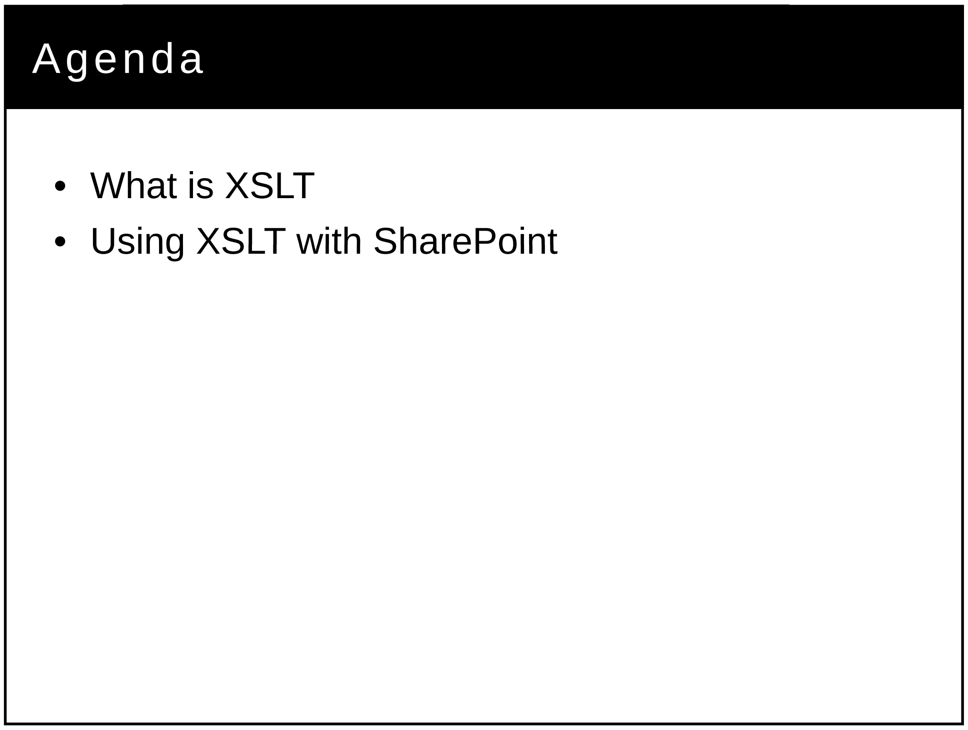
```
protected void cmdApplyCustomBrand_Click(object sender, EventArgs e) {
  SPWeb site = SPContext.Current.Site.RootWeb
  string MasterUrlPath = site.ServerRelativeUrl;
  if (!MasterUrlPath.EndsWith(@"/"))
    MasterUrlPath += @"/";
  MasterUrlPath += @"_catalogs/masterpage/Litware.master";
  ApplyCustomBrand(MasterUrlPath, site);
}

protected void ApplyCustomBrand(string MasterUrlPath, SPWeb site) {
  site.MasterUrl = MasterUrlPath;
  site.Update();
  // use recusion to update all child sites in site collection
  foreach (SPWeb child in site.Webs) {
    ApplyCustomBrand(MasterUrlPath, child);
  }
}
```

Agenda

- SharePoint Customization versus Development
- Developing Features
- Solution Packages Deploy
- Creating a Branding Solution



World's Fastest Primer on XSLT



Agenda

- What is XSLT
- Using XSLT with SharePoint

What is XSLT

- XSLT = Extensible Stylesheet Language Transformations
- What does that mean?
  - An XML language that converts another XML document to various Human Readable formats
  - One common use is taking raw XML in and sending styled HTML out
- Why XSLT?
  - Separation of data from the display
  - Same XML could be seen as HTML or WAP or whatever

What is XSLT

- Sample XML:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="style.xsl"?>
<books>
   <book>
        <title>Professional SharePoint 2007 Design</title>
        <url>http://www.amazon.com/exec/obidos/ASIN/047028580X</url>
   </book>
   <book>
        <title>Real World SharePoint 2007</title>
        <url>http://www.amazon.com/exec/obidos/ASIN/0470168358</url>
   </book>
</books>
```

What is XSLT

- Sample XSLT:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
     <xsl:output method="html"/>
     <xsl:template match="/books">
      <table>
       <xsl:for-each select="book">
        <tr>
         <td><a><xsl:attribute name="href"><xsl:value-of select="url"/></xsl:attribute>Link</a></td>
         <td><xsl:value-of select="title"/></td>
        </tr>
       </xsl:for-each>
      </table>
     </xsl:template>
</xsl:stylesheet>
```

Would look like this in the browser:

Link Professional SharePoint 2007 Design
Link Real World SharePoint 2007

Using XSLT with SharePoint

- Many of the SharePoint Web Parts use XSLT for styling
- Content Query Web Part uses styles in the Style Library > XSL Style Sheets (Header.xsl and ItemStyle.xsl)
  - You have to edit CommonViewFields in the Web Part to see custom columns
  - Enhanced Content Query Web Part: http://www.codeplex.com/ECQWP

Using XSLT with SharePoint

- XML Web Part is built for styling XML with XSL
- SharePoint Designer can use the Data View Web Part to build XSLT easily
- Customizing the Search Results Web Part with XSLT:
  - http://msdn.microsoft.com/en-us/library/ms584121.aspx

Summary

- What is XSLT
- Using XSLT with SharePoint

Best Practices in Branding SharePoint 2007 Sites

#Introduction

Ted Pattison, Ted Pattison Group

David Mann, Mann Software

May 2008

Applies to: Windows SharePoint Services 3.0, Microsoft Office SharePoint Server 2007

Summary: SharePoint

Contents

#Introduction to Branding with SharePoint Technologies

"I want to use SharePoint, but I don't want it to **look** like SharePoint."

How many times have you heard that?

Or this:

"SharePoint sites all look the same. They are all so *boxy* "

Unfortunately, SharePoint is guilty as charged. Out of the box, all SharePoint sites look, well, like SharePoint.

This is problematic because most companies want to have their SharePoint environment use the same colors, fonts, images, layouts, etc. (hereafter called the *brand*), as the rest of their web-based initiatives. They have likely invested significant resources in developing that brand and applying it to other initiatives and so for SharePoint not to adhere to it would be a problem.

Fortunately, the 2007 release of SharePoint Products and Technologies makes it easy to apply a custom brand to SharePoint sites. This paper will demonstrate how to take your SharePoint environment from the default brand:

# ProductVersionKeyword_Introduction

# ProductVersionKeyword_MainSubhead

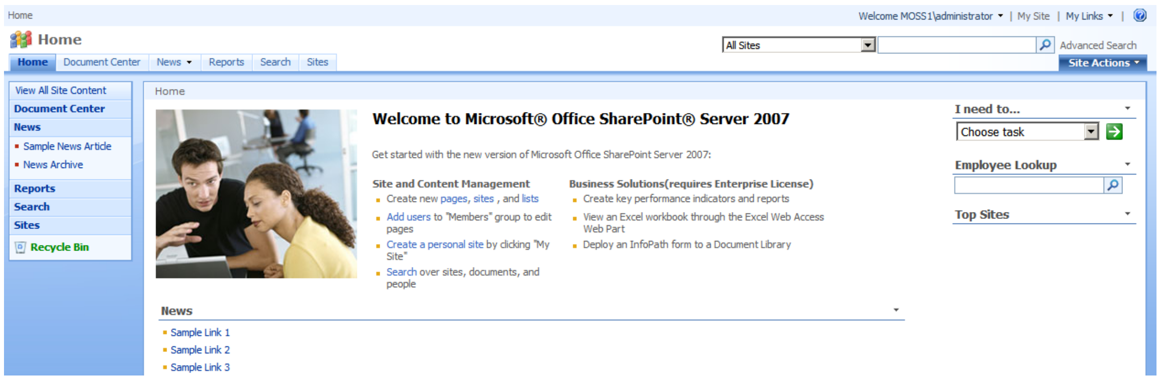

Home
Welcome MOSS1\administrator | My Site | My Links
Home
All Sites
Advanced Search
Home Document Center News Reports Search Sites
Site Actions
View All Site Content
Document Center
News
Sample News Article
News Archive
Reports
Search
Sites
Recycle Bin
Welcome to Microsoft® Office SharePoint® Server 2007
Get started with the new version of Microsoft Office SharePoint Server 2007:
Site and Content Management
Create new pages, sites, and lists
Add users to "Members" group to edit pages
Create a personal site by clicking "My Site"
Search over sites, documents, and people
Business Solutions(requires Enterprise License)
Create key performance indicators and reports
View an Excel workbook through the Excel Web Access Web Part
Deploy an InfoPath form to a Document Library
News
Sample Link 1
Sample Link 2
Sample Link 3
I need to...
Choose task
Employee Lookup
Top Sites

Figure 1: The default brand - boring and boxy

to, for example, something with a little more pizzazz:

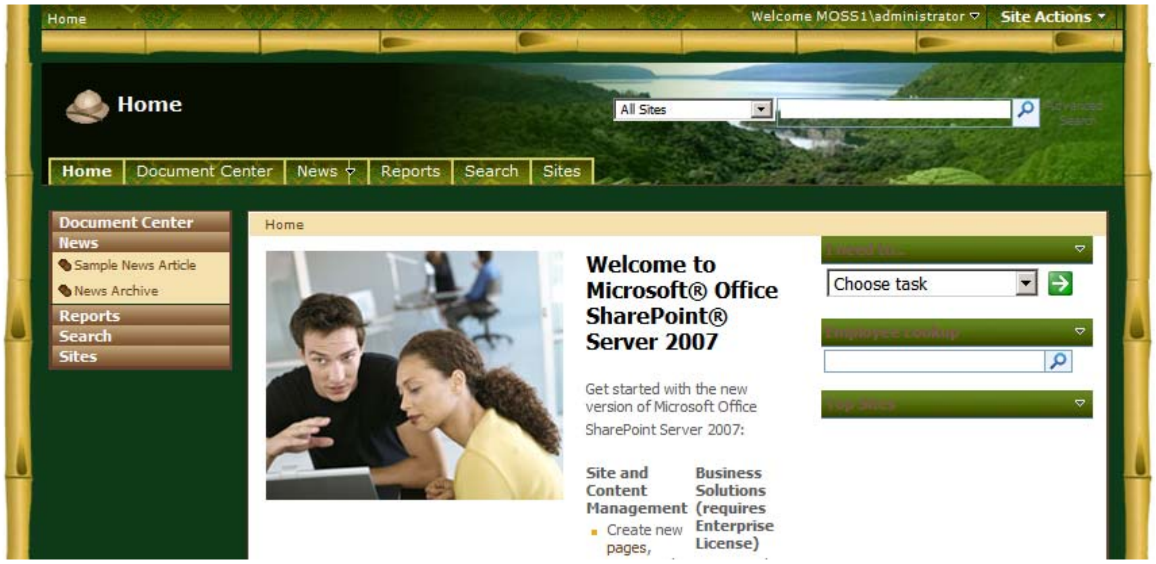

Home
Welcome MOSS1\administrator
Site Actions
Home
All Sites
Home Document Center News Reports Search Sites
Document Center
News
Sample News Article
News Archive
Reports
Search
Sites
Welcome to Microsoft® Office SharePoint® Server 2007
Get started with the new version of Microsoft Office SharePoint Server 2007:
Site and Content Management
Create new pages,
Business Solutions (requires Enterprise License)
Choose task

Figure 2: The sample Liteware brand - just a little more adventuresome...

NOTE: The intent of this article is not to go through the specific details of how to build a new brand (creating MasterPages, modifying style sheets, etc.) but rather to show how to manage, deploy and apply the new brand to any or all sites in your SharePoint farm. Please see the Additional Resources section at the end for details on how to build the MasterPage, CSS and other elements necessary to create your brand.

The ultimate goal of a fully-realized branding implementation is to have a solution that delivers the brand elements you need in a way that:

- Follows best practices for SharePoint development and packaging
- Is deployable across site collections, web applications and farms
- Is manageable

- Integrates with standard source control management systems for versioning, check-in/check-out, etc.
- Is loosely coupled with the implementation (meaning it can be applied equally well to any implementation)

We'll make our way to this branding utopia through the course of this article.

Getting Started

Depending on your goals, which edition of SharePoint Products and Technologies you have installed, and which tools you have available, there are several options available to you to help reach your goals. Ranging from simple browser-based customizations to SharePoint Designer customizations to custom programming with Visual Studio, and covering both Windows SharePoint Services (WSS) and Microsoft Office SharePoint Server (MOSS), Table 1 lists your options.

Table 1. Options for changing a site's look and feel

Technique	Scope	Application	Deployment
Apply theme on team site	Site	Browser	WSS, MOSS
Customize theme on a team site	Site	SharePoint Designer	WSS, MOSS
Customizing master page of team site	Site	SharePoint Designer	WSS, MOSS
Customize master page of Publishing portal	Site Collection	SharePoint Designer	MOSS
Upload custom CSS to Publishing portal	Site Collection	SharePoint Designer	MOSS
Develop custom solution	Farm	Visual Studio	WSS, MOSS

Themes are the simplest, but least powerful option. They are, however, an option, so let's take a quick peak at what they offer.

Working with Themes

Windows SharePoint Services 3.0 (WSS) provides for the option of branding through themes. Themes are a collection of Cascading Style Sheet (CSS) files that can be used to change the presentation of a site. Unlike the branding solutions we will talk about shortly, themes cannot change anything but colors, fonts and other aspects that can be manipulated via CSS. Essentially this means that themes can only change existing elements, they cannot add new elements.

Note that themes in WSS are entirely different than themes in ASP.NET. WSS implements its infrastructure for themes by copying a CSS file from the file system of the Web server into the context of the current site and dynamically linking to this CSS file from each page.

A user (with proper permissions) can apply a WSS theme to a site to change the look and feel of that particular site. This change does not require that the users know anything about web programming, CSS, etc. All they need to do is pick the theme they wish to apply from a list. Some of the default themes are shown below in Figure 3. As you can see, they change the look of the site but none of the structure.

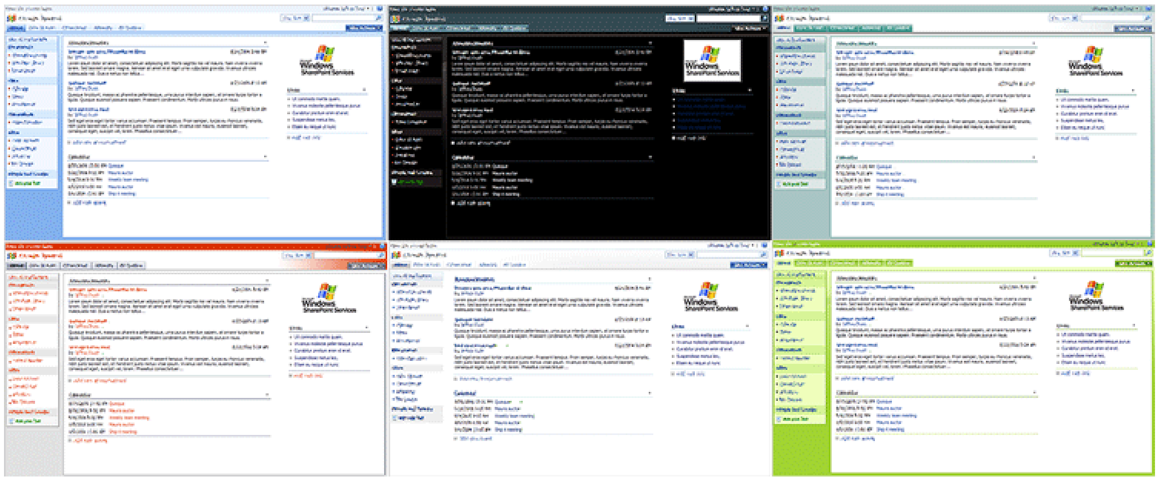


Figure 3: Themes can change the look of a site

A new theme can be applied to a site from the *Site Settings* page by clicking on the *Site Themes* link within the *Look and Feel* section.

As mentioned previously, themes are the least powerful option. This is because themes have limitations that make them less useful in an enterprise implementation:

- Themes cannot integrate a custom master page to customize the page layouts used within a site.
- Themes cannot add new CSS files, only make changes to the theme CSS files
- A theme can only be applied on a per-site basis as opposed to a wider scope. There is no support anywhere in SharePoint to apply a theme to all the sites within a site collection in a single action. Instead, a user would have to go to each site separately and apply the same theme to maintain a consistent look and feel across a site collection with multiple sites.

Each of these is problematic for the usefulness of themes for any significant deployment of SharePoint. Lets move on to some options that begin to overcome these issues and also present some additional benefits.

Working with SharePoint Designer

Like themes, Microsoft Office SharePoint Designer 2007 (SPD) provides another means for branding sites in WSS or MOSS. SPD overcomes three of the problems with themes in that you can:

- customize a site's master page
- add a new master page to change page layouts
- add and integrate new CSS files.

However, SPD doesn't provide a way to apply a brand across an entire site collection. This means that we're not quite to the point we need to be. We can do more with regard to branding from SPD, but we are still hampered by our deployment options. We are still limited to working with one site at a time with no way to move changes across multiple sites, site collections, web applications or farms . This can lead to scenarios where you are forced to copy and paste your edits across multiple sites.

For WSS-only implementations, the story stops there. We can go no further with SPD and WSS.

For MOSS implementations, the SPD story gets a little better because MOSS provides a Web Content Management (WCM) infrastructure through the use of Publishing portals. A MOSS Publishing portal is a site collection containing Publishing sites which can be configured to have all sites within the collection use the same Master Page and CSS files.

Within the confines of a single site collection, this solves the problem for all site pages. However, across site collections, web applications or farms, we're still stuck with the same deployment problem. We also run into a problem when we attempt to apply a brand to application pages.

One other issue presented with SPD (that can't even come into play with themes) is that it's impractical to integrate site customizations made with the SharePoint Designer into a source code management system such as Team System.

All of this makes SPD more attractive for branding than themes, but it means that we're still only part way to the goal we laid out at the beginning of this article. To arrive at our goal, we need to leave SPD behind and step into Visual Studio.

Developing A Branding Solution

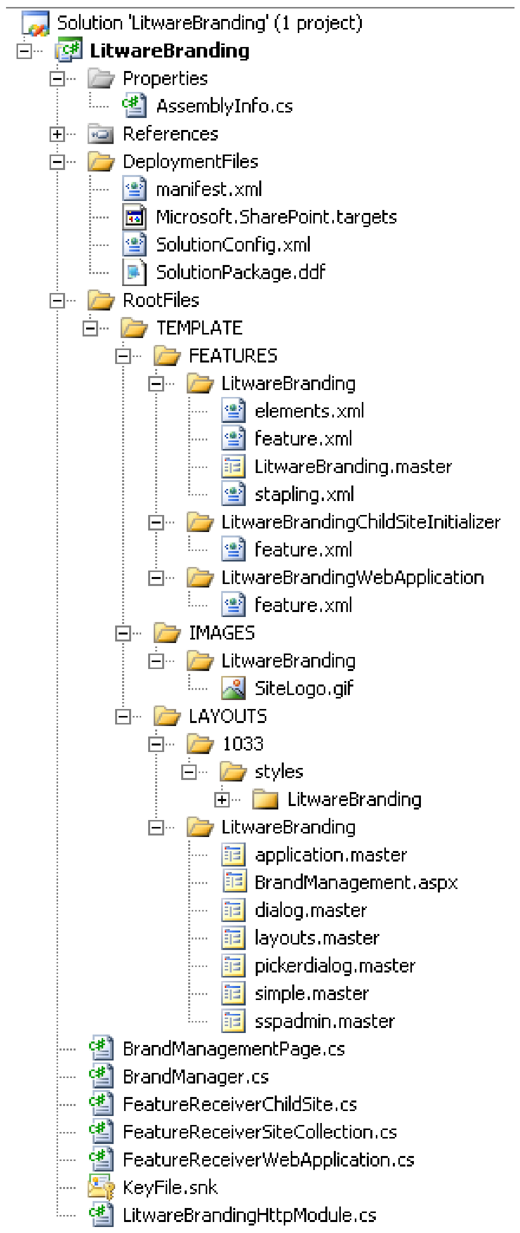
The rest of this whitepaper is going to walk through developing a custom branding solution for SharePoint 2007 sites in Visual Studio. We will walk through the code within a sample branding solution named LitwareBranding that has been developed using a Visual Studio project and that is deployed using a solution package.

Right from the start, generating a solution package makes it possible to deploy our branding solution to any farm running WSS 3.0 or MOSS. This means that when we're done, we'll have achieved our deployability goal. We'll have a single package (WSP file) that contains a single source file for each master page and CSS file which can be reused across multiple site collections or across farms.

Furthermore, the Visual Studio approach can be far more appealing than using the SharePoint Designer because it allows you to check all your source files into a source code management system and also to move your development efforts from a development farm to a staging farm for quality assurance testing and then into production.

Introducing the LitwareBranding solution

This whitepaper is accompanied by a sample Visual Studio project named LitwareBranding. Figure 4 provides a screenshot which shows the structure of the LitwareBranding project within Visual Studio.



```
Solution 'LitwareBranding' (1 project)
  LitwareBranding
    Properties
      AssemblyInfo.cs
    References
    DeploymentFiles
      manifest.xml
      Microsoft.SharePoint.targets
      SolutionConfig.xml
      SolutionPackage.ddf
    RootFiles
      TEMPLATE
        FEATURES
          LitwareBranding
            elements.xml
            feature.xml
            LitwareBranding.master
            stapling.xml
          LitwareBrandingChildSiteInitializer
            feature.xml
          LitwareBrandingWebApplication
            feature.xml
        IMAGES
          LitwareBranding
            SiteLogo.gif
        LAYOUTS
          1033
            styles
              LitwareBranding
          LitwareBranding
            application.master
            BrandManagement.aspx
            dialog.master
            layouts.master
            pickerdialog.master
            simple.master
            sspadmin.master
    BrandManagementPage.cs
    BrandManager.cs
    FeatureReceiverChildSite.cs
    FeatureReceiverSiteCollection.cs
    FeatureReceiverWebApplication.cs
    KeyFile.snk
    LitwareBrandingHttpModule.cs
```

Figure 4: The high-level structure of the LitwareBranding solution

As you can see, the LitwareBranding solution has been designed using 3 different Features, as summarized in Table 2.

Feature	**Scope**	**Description**
LitwareBranding	Site	This is the main Feature for the solution. It is responsible for applying all of the branding techniques we will discuss to existing sites
LitwareBrandingChildSiteInitializer	Web	Applies custom brand to new child sites when they are

		created
LitwareBrandingWebApplication	WebApplication	Manages changes required to the web.config file for the web application that contains the site collection(s) to which our brand will be applied

Table 2: The Features that make up our solution

Each of these Features will be discussed in more detail in the following sections.

At a very high level, programmatically applying a brand to a site requires the following changes:

1. Using the proper master page
2. Setting the AlternateCssUrl property
3. Updating the SiteLogoUrl property

To apply the brand to all sites within a site collection simply means iterating through the child sites and making the above changes. There are some other requirements related to placing files within the environment and modifying the web.config file which we'll cover as we examine each Feature.

Exploring the LitwareBranding Feature

Let's begin our code walkthrough by looking at the feature.xml file for the central Feature named LitwareBranding.

```
<Feature
  Id="065E2243-B968-4F14-BAAE-610BB975EFB7"
  Title="A sample feature: LitwareBranding"
  Description="Demoware created to demonstrate branding solution"
  Hidden="FALSE"
  Scope="Site"
  ImageUrl="LitwareBranding\AfricanPith32.gif"
  ReceiverAssembly="LitwareBranding, [4-part assembly name]"
  ReceiverClass="LitwareBranding.FeatureReceiver"
  xmlns="http://schemas.microsoft.com/sharepoint/">

  <ElementManifests>
    <ElementManifest Location="elements.xml"/>
    <ElementManifest Location="stapling.xml" />
    <ElementFile Location="LitwareBranding.master"/>
  </ElementManifests>

</Feature>
```

The LitwareBranding Feature has been defined with Scope attribute value of Site which means it is scoped to the site collection level. The idea is that a site collection owner should be able to activate a single Feature to apply the Litware corporate branding to all the sites within the current site collection. The LitwareBranding Feature contains a master page template named LitwareBranding.master, and two element manifests: elements.xml and stapling.xml.

Within elements.xml there is declarative XML provisioning logic to create an instance of this master page template. This logic is defined by a Module element with an inner File element.

```
<Module Name="MasterPages" Path="" List="116"
        Url="_catalogs/masterpage" >
  <File Url="LitwareBranding.master" Type="GhostableInLibrary" />

</Module>
```

During Feature activation, this Module element causes WSS to provision an instance of a master page template named LitwareBranding.master in the master page gallery of the top-level site. After this master page instance has been provisioned, we can programmatically force pages throughout the site collection to utilize it.

The last element manifest from the Litwarebranding Feature, stapling.xml, makes use of a capability of WSS, known as *Feature stapling*, to attach the LitwareBrandingChildSiteInitializer Feature to the Global and Blank site definitions. We'll cover the details of that Feature later, but this causes any site created within the site collection to have the LitwareBrandingChildSiteInitializer Feature attached to it.

Creating a utility class for brand management

The LitwareBranding project contains a class named BrandManager (within the BrandManager.cs file). This utility class encapsulates all of the code that programs against the WSS object model to apply and remove various branding elements. There are static methods exposed by the BrandManager class that accomplishes the following tasks.

- Synchronizing site pages to link to LitwareBranding.master
- Synchronizing all pages to use an alternate CSS file
- Synchronizing all pages to use a custom graphic for the site logo
- Adding support to swap out the master page for application pages

Before drilling down in specific members, you should examine this high-level listing which shows all of the members of the BrandManager class:

```
public class BrandManager {
 // read-only properties
 public static string SiteCollectionUrl
 public static string DefaultMasterPageUrl
 public static string CustomMasterPageUrl
 public static string CustomCssUrl
 public static string CustomSiteLogoUrl
 // utility branding methods
 public static void ConfigureMasterUrl(bool ApplyMasterUrl) {}
 public static void ConfigureCustomMasterUrl(bool ApplyCustomMasterUrl)
{}
 public static void ConfigureAlternateCss(bool ApplyCustomCss) {}
 public static void ConfigureSiteLogo(bool ApplySiteLogo) {}
 public static void ConfigureApplicationPageMaster(bool
ApplyApplicationPageMaster) {}
}
```

As you can see, the BrandManager class contains several static properties. These properties provide URLs which point to various resources such as master pages and a CSS file. For example, the SiteCollectionUrl property returns the Web Application-relative path to the current site collection. This serves as the base for two other properties: DefaultMasterPageUrl and CustomMasterPageUr.

The DefaultMasterPageUrl property returns a Web Application-relative path to the default master page (named default.master) in the Master Page Gallery of the current site collection's top-level site. This path is put together by combining the SiteCollectionUrl property with the following site-relative path:

```
_catalogs/masterpage/default.master
```

The CustomMasterPageUrl property returns a Web Application-relative path to the master page instance named LitwareBranding.master in the Master Page Gallery of the current site collection's top-level site. This path is put together using the SiteCollectionUrl property together with the following site-relative path:

```
_catalogs/masterpage/litwarebranding.master
```

The other two properties, CustomCssUrl and CustomSiteLogoUrl, simply return fixed strings pointing to other files deployed as part of our solution: /_layouts/1033/STYLES/LitwareBranding/styles.css and /_layouts/images/LitwareBranding/SiteLogo.gif respectively. As you can see, we're instructing WSS to look for each of these files inside a *LitwareBranding* folder within the appropriate parent folder of the WSS Root (\Program Files\Common Files\Microsoft Shared\Web Server Extensions\12) file system structure.

Now, let's examine the ConfigureMasterUrl method which enumerates through every site in the current site collection and updates each site's MasterUrl property to point to either the custom master page or the default master page. The following listing shows this functionality:

```
public static void ConfigureMasterUrl(bool ApplyMasterUrl) {
  // determine MasterUrl property setting
  string MasterUrlPath = (ApplyMasterUrl ?
                          CustomMasterPageUrl :
                          DefaultMasterPageUrl);

  // update MasterUrl property for all sites
  foreach (SPWeb site in SPContext.Current.Site.AllWebs) {
    site.MasterUrl = MasterUrlPath;
    site.Update();
  }
}
```

Note that changing the MasterUrl property affects all site pages that have been created with a MasterPageFile property setting of ~masterurl\default.master. This includes the standard default.aspx page template that provides the home page for many SharePoint site templates including Team Site and Blank site. It also includes the view and form pages such as AllItems.aspx and NewItem.aspx that are used by standard WSS lists.

As mentioned previously, we are updating both the MasterUrl property, which we have just seen, as well as the CustomMasterUrl property. This is because page layouts in a MOSS Publishing site will not be affected when you update only the MasterUrl property. Instead, these content pages are designed to use a dynamic token for the MastePageFile attribute which follows the form of ~masterurl\custom.master. This token is different from the ~masterurl\default.master token because it is switched out using the CustomMasterUrl property instead of the MasterUrl property. This effects all of the content pages within the Pages library of a MOSS publishing site. Therefore, the BrandManager class provides a second method named ConfigureCustomMasterUrl which can be used to switch out the master page for content pages created in the Pages library of MOSS Publishing sites.

```
public static void ConfigureCustomMasterUrl(bool ApplyCustomMasterUrl)
{

  // determine MasterUrl property setting
  string CustomMasterUrlPath = (ApplyCustomMasterUrl ?
                                CustomMasterPageUrl :
                                DefaultMasterPageUrl);

  // update MasterUrl property for all sites
  foreach (SPWeb site in SPContext.Current.Site.AllWebs) {
    site.CustomMasterUrl = CustomMasterUrlPath;
    site.Update();
  }
}
```

Modifying either MasterUrl property has no effect if the corresponding page type (standard WSS or publishing layouts) are not in use on the site.

In addition to swapping out master pages, the LitwareBranding solution also integrates a custom CSS file named styles.css. Following best practices, the LitwareBranding solution uses a strategy of deploying styles.css to a directory nested inside the LAYOUTS directory on the file system of each front end Web Server. The value of this deployment model is that the file can be deployed once on the Web server's file system and yet still be accessible from any site within the current farm using a fixed URL. If you remember back to the static properties we discussed previously, this URL is returned by the CustomCssUrl property.

```
public static string CustomCssUrl {
  get {
    return "/_layouts/1033/STYLES/LitwareBranding/styles.css";
  }
}
```

The BrandManager class provides a method named ConfigureAlternateCss that uses the CustomCssUrl property to assign the URL to the AlternateCSS property of each site within the current site collection. Passing `true` to this method will cause the CustomCssUrl property value to be written to the AlternateCssUrl property of the SPWeb. Passing `false` will cause the AlternateCssUrl property value to be cleared out.

```
public static void ConfigureAlternateCss(bool ApplyCustomCss) {

 // determine MasterUrl property setting
  string AlternateCssUrl = (ApplyCustomCss ?
                            CustomCssUrl :
                            string.Empty);

  // update AlternateCssUrl for all sites
  foreach (SPWeb site in SPContext.Current.Site.AllWebs) {
    // make sure no theme is enabled
    site.ApplyTheme(string.Empty);
    // apply custom CSS file
    site.AlternateCssUrl = AlternateCssUrl;
    site.Update();
  }
}
```

The branding capabilities built into WSS also provides an easy way for you to replace the standard WSS site logo that it shows (by default) on the top left portion of the page. The LitwareBranding solution takes advantage of this capability by including a custom graphic named SiteLogo.gif which is deployed in a directory nested inside the IMAGES directory. Like the styles.css file, SiteLogo.gif is also deployed in such a way that it is accessible from any site in the current farm using the URL returned by the CustomSiteLogoUrl property.

```
public static string CustomSiteLogoUrl {
  get {
    return "/_layouts/images/LitwareBranding/SiteLogo.gif";
  }
}
```

The ConfigureSiteLogo method of the BrandManager class has been written to enumerate through each site of the current site collection and to update the SiteLogoUrl property. As before, passing `true` will cause the CustomSiteLogoUrl property to be used; passing `false` will cause the property to be cleared.

```
public static void ConfigureSiteLogo(bool ApplySiteLogo) {

  // determine SiteLogoUrl property setting
  string SiteLogoUrl = (ApplySiteLogo ?
                        CustomSiteLogoUrl :
                        string.Empty);

  // update SiteLogoUrl for all sites
  foreach (SPWeb site in SPContext.Current.Site.AllWebs) {
    site.SiteLogoUrl = SiteLogoUrl;
    site.Update();
  }
}
```

Swapping out the Master Page for application pages

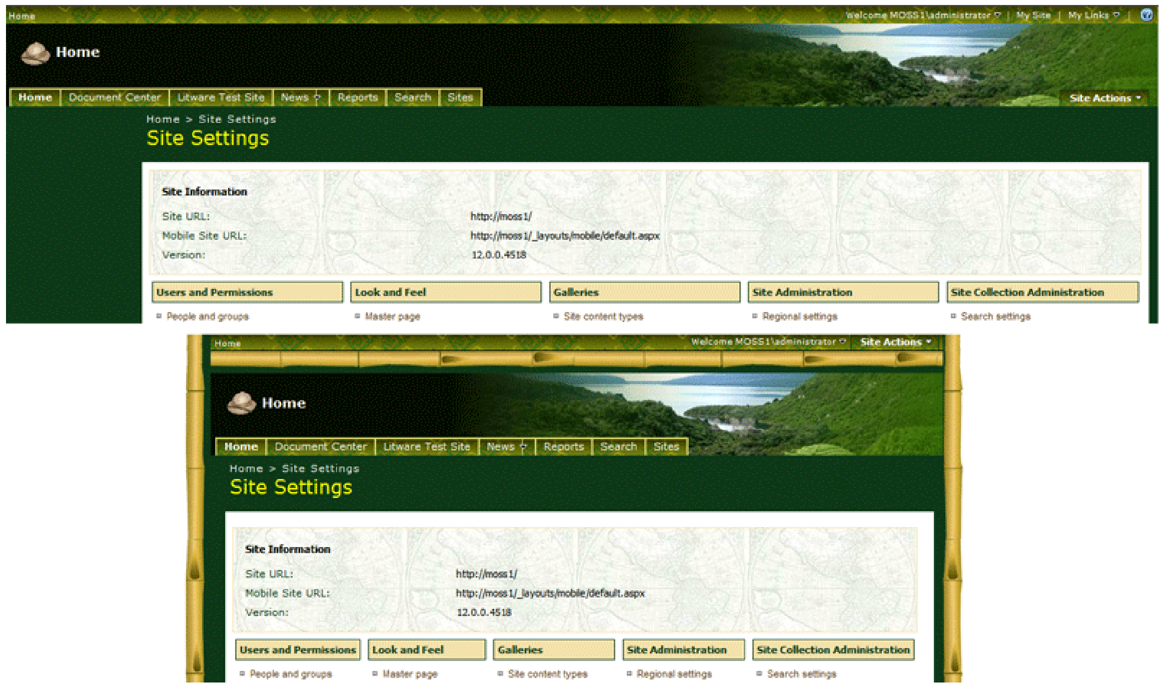
While SharePoint 2007 makes it easy (as we've seen) to swap out the master page for site pages, it does not provide an equivalent way to switch to a new master page for application pages. This is because the two different types of pages (see definitions, below) make use of different master pages and only one (site pages) has support for customization. This is not to say that it is impossible to customize the master page for application pages, just that it is more difficult and requires a different approach. We'll cover all of the steps required in this section.

Definitions

Site page: ASPX pages that are part of the regular, user-browsable site. Includes Default.aspx, AllItems.aspx and other view pages as well as NewItem.aspx and other form pages. As the name implies, site pages belong to one and only one site.

Application page: ASPX pages typically used for site administration and other common functionality. These pages are located in the LAYOUTS directory of each web front end server and include pages such as the standard WSS Site Settings page (/_layouts/settings.aspx). Application pages are addressable from any site in the farm.

Depending on the nature of your branding customizations, it is important to know that **some** changes will be reflected in application pages even without the steps we will cover in this section. These changes will be those that are handled by modifying CSS classes to update colors, fonts, and some existing images. Figure 5 shows an application page partially updated with only CSS changes (top) and the same application page fully updated with the new brand (bottom).



Site Information
Site URL: http://moss1/
Mobile Site URL: http://moss1/_layouts/mobile/default.aspx
Version: 12.0.0.4518

Figure 5: A partially branded application page (top) and a fully branded application page (bottom)

The changes from the partially-branded to the fully-branded example range from the obvious: the page width and the addition of the bamboo framing, to the subtle: the removal of the "My Site", "My Links" and Help icon links from the top of the page and the relocation of the "Site Actions" menu. This example gives a good idea of what is possible with each piece of the solution. If all that your brand requires for application pages can be achieved with CSS changes, then you do not need to implement the rest of the solution covered in this section.

Before going any further, we should discuss a little bit about why we are going to all of this trouble. If application pages are mostly used by administrators, can't we just give them the out of the box brand or get "pretty close" with CSS-only changes and not worry that their experience is different? After all, as someone with adminstrative access, they are a higher level of user and will probably be OK with a different user interface.

The short answer is that administrators are users, too, and you want to make their experience as consistent and enjoyable as possible. The long answer is that not all application pages are used only by administrators. Remember, I've always said that application pages are "typically" or "mostly" used by administrators. There are some notable exceptions. For example, the following application pages are just a few of the pages that are fully accessible to and will be used by visitors with less than administrator access to the site:

- Add New Alert
- User Information
- Recycle Bin
- Online Help
- View All Site Content
- Document Upload
- Workflows

There are plenty of other application pages that regular users will visit as well. For that reason alone, it is important to apply any custom brand fully across all application pages in your site, so let's get started.

By default, application pages are linked to a standard WSS master page named application.master that WSS deploys within the LAYOUTS directory. Unlike the way that site pages are processed, this entry is not used as a token to be swapped out with the value of an SPWeb property. (Remember, application pages are not associated with a particular SPWeb, so there is nowhere to go for a value to replace, even if the processing called for a replacement).

So what can we do?

The LitwareBranding solution included with this article demonstrates a technique for swapping out the master page for all application pages that link to the standard application.master. This technique involves implementing a custom HttpModule which swaps out the master page during the processing of a page request.

Examine the following skeleton listing for the class named LitwareBrandingHttpModule which provides the HttpModule used in the LitwareBranding solution to swap out the master page for application pages. We'll add the functionality to this class as we proceed.

```
using System;
using System.Web;
using System.Web.UI;
using Microsoft.SharePoint;

namespace LitwareBranding {
  public class LitwareBrandingHttpModule : IHttpModule {

    public void Init(HttpApplication context) {
      context.PreRequestHandlerExecute
        += new EventHandler(context_PreRequestHandlerExecute);
    }

    void context_PreRequestHandlerExecute(object sender, EventArgs e) {
      Page page = HttpContext.Current.CurrentHandler as Page;
      if ( (page != null ) &&
           (page.Request.Url.AbsolutePath.Contains("_layouts"))) {
        // register handler for PreInit event
        page.PreInit += new EventHandler(page_PreInit);
      }
    }

    void page_PreInit(object sender, EventArgs e) {
      // if requested page links to _layouts\application.master
      // then modify Page to link to custom master page instead
    }

    public void Dispose() {}
  }
}
```

As you can see, this HttpModule class registers a handler for the PreRequestHandlerExecute event inside the Init method. Within the method implementation of the PreRequestHandlerExecute event handler, the HttpModule class determines whether the request is based on an HttpHandler object that derives from the ASP.NET Page class. Only in cases where the request is based on a Page-derived object will the HttpModule class register an event handler from the PreInit event. The code in the PreRequestHandlerExecute event handler also checks to make sure the incoming request targets a page inside the _layouts directory which will always be the case when processing an application page in WSS.

It’s important to remember that an HttpModule cannot be deployed in a WSS farm for an individual site collection. Instead, an HttpModule must be configured as an all-or-nothing proposition at the Web Application level. However, a Web Application may contain hundreds of site collections and only certain site collection might have enabled the LitwareBranding Feature. Therefore, the PreInit event handler for the HttpModule class must be able to determine whether the current site collection has been configured with the behavior to swap out the master page for its application pages.

The LitwareBranding solution solves this problem by creating a custom property in the site collection's top-level site to indicate that swapping out the master page for application pages should be enabled. Examine the implementation of the ConfigureApplicationPageMaster method defined within the BrandManager class.

```
public static void ConfigureApplicationPageMaster(bool
ApplyApplicationPageMaster) {
  SPWeb TopLevelSite = SPContext.Current.Site.RootWeb;
  if (ApplyApplicationPageMaster) {
    TopLevelSite.Properties["UseCustomApplicationPageMaster"] = "True";
  }
  else {
    TopLevelSite.Properties["UseCustomApplicationPageMaster"] =
"False";
  }
  TopLevelSite.Properties.Update();
}
```

As you can see, this method creates a custom property named UseCustomApplicationPageMaster on the top-level site and assigns the property a value of True. Now, back in the HttpModule, the method implementation for the PreInit event handler can look for this property within the current site collection to see whether it has been configured to enable swapping out the application page master. Note that the PreInit event handler also performs several other checks to determine whether it is appropriate to swap out the master page within the context of the current request. Examine the following code snippet for the fully functional page_PreInit method.

```
void page_PreInit(object sender, EventArgs e) {
  Page page = sender as Page;
  if ((page != null) &&
      (page.MasterPageFile != null) &&
      (SPContext.Current != null)) {

    // inspect UseCustomApplicationPageMaster property
    SPWeb site = SPContext.Current.Site.RootWeb;
    string UseCustomApplicationPageMaster =
      site.Properties["UseCustomApplicationPageMaster"];
    if ((!string.IsNullOrEmpty(UseCustomApplicationPageMaster)) &&
       (UseCustomApplicationPageMaster.Equals("True"))) {
      // now replace application.master with customized version
      if (page.MasterPageFile.Contains("_layouts/application.master "))
```

```
        {
          page.MasterPageFile =
            "/_layouts/LitwareBranding/application.master";
        }
      }
    }
  }
```

Once the PreInit event handler determines the current site collection has been configured to enable swapping out the master page for application pages and also that the current page links to application.master, it modifies the MasterPageFile property of the current ASP.NET Page object to use a custom master page located within a solution-specific directory within the LAYOUTS directory at the following path:

```
/_layouts/LitwareBranding/application.master
```

Swapping Out the Master Page for Other Pages

If all we needed to do was change site pages and application pages that used application.master we'd be done now. Unfortunately, SharePoint makes use of quite a number of different master pages to cover the various scenarios and types of pages it serves up. A quick search of the TEMPLATE directory for *.master files reveals ten master pages deployed out of the box:

- Application.master
- Default.master
- Admin.master
- Popup.master
- MWSDefault.master
- Dialog.master
- Layouts.master
- PickerDialog.master
- Simple.master
- SSPAdmin.master

This does not include the eight deployed by the MOSS Publishing functionality. The purpose of some of these is fairly obvious - popup.master is used for popup pages and dialog.master for pages shown as dialog boxes. But what about the others, and what if you need to make changes to them? How do you make sure that they are served up correctly?

Whether you need to create a custom copy of each of these master pages is entirely dependent upon the needs of your brand. How you serve them up, though, is a matter of a simple change to the Http odule we introduced earlier.

Simpy replace the "if" block we used in the HttpModule above:

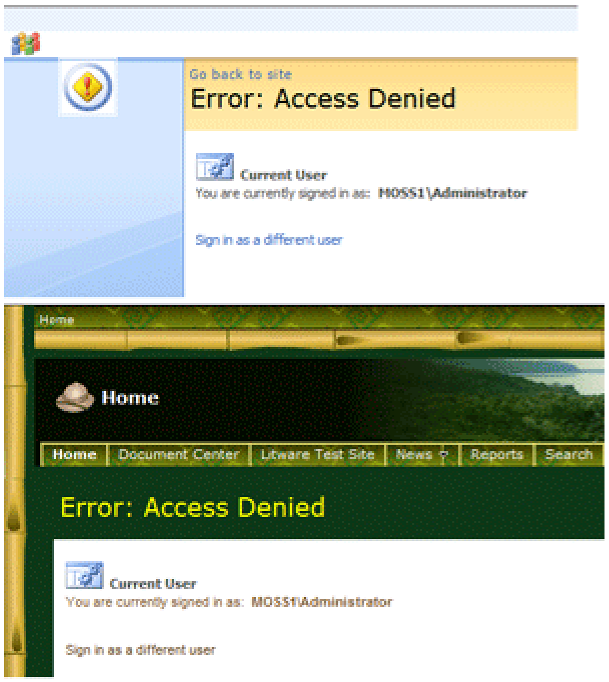
```
if (page.MasterPageFile.Contains("_layouts/application.master "))
{
  page.MasterPageFile =
    "/_layouts/LitwareBranding/application.master";
}
```

with the following code:

```
switch (page.MasterPageFile.ToLower())
{
    case "/_layouts/application.master":
        page.MasterPageFile =
"/_layouts/LitwareBranding/application.master";
        break;
    case "/_layouts/simple.master":
        page.MasterPageFile =
"/_layouts/LitwareBranding/simple.master";
        break;
    default:
        break;
}
```

In this case, we are only handling one other master page - simple.master - but you can easily expand upon this model to handle the other master pages if you need to.

This simple change allows us to see our brand applied to more pages. In the case of simple.master, it is used for the default Access Denied page so we see a page that matches our brand. Figure 6 shows the default Access Denied page (top) and our properly branded Access Denied page (bottom).


Go back to site

Error: Access Denied

Current User

You are currently signed in as: MOSS1\Administrator

Sign in as a different user

Error: Access Denied

Current User

You are currently signed in as: MOSS1\Administrator

Sign in as a different user

Figure 6: Our branding applied to other master pages (simple.master shown here)

Registering an HttpModule in the web.config file

When deploying a business solution with WSS and MOSS, it is a best practice to distribute your development efforts within a solution package. For example, the Visual Studio project for LitwareBranding builds a single solution package named LitwareBranding.wsp which makes it easy and reliable to deploy this solution out to any farm running WSS 3.0 or MOSS. However, the LitwareBranding solution requires adding an HttpModule entry to the web.config file for each Web Application which will be running site collection that activate the LitwareBranding Feature.

In order to update the web.config file within specific Web Applications, the LitwareBranding solution uses a second Feature named LitwareBrandingWebApplication. This Feature is scoped to the Web Application level and is configured with a Feature receiver class to fire event handlers as it is activated and deactivated within the scope of a particular Web Application.

```
<Feature
  Id="FF739C76-0B08-4bc2-A3A2-F61524B492D8"
  Title="Litware Branding Support Feature (WebApplication)"
  Scope="WebApplication"
  Hidden="False"
  ReceiverClass="LitwareBranding. FeatureReceiverWebApplication"
  ReceiverAssembly="LitwareBranding, [4-part assembly name]"
  xmlns="http://schemas.microsoft.com/sharepoint/">

  <!-- no declarative elements -->
  <ElementManifests />
```

```
</Feature>
```

Like any other Feature receiver class, the FeatureReceiverWebApplication class inherits from the SPFeatureReceiver class and it overrides the four handler methods named FeatureInstalled, FeatureActivated, FeatureDecactivating and FeatureUninstalling. The implementation for FeatureActivated adds the HttpModule entry to the web.config file for the current Web Application. The implementation for FeatureDeactivating reverses that operation by removing the HttpModule entry. The other two events must be implemented but they do not need to do anything.

When you need to add an entry the web.config file, it is a best practice to use the SPWebConfigModification class from the WSS object model. The FeatureReceiverWebApplication class contains a utility method named CreateHttpModuleModification which creates and initializes an instance of SPWebConfigModification and passes it back as its return value.

```
public SPWebConfigModification CreateHttpModuleModification() {
  SPWebConfigModification modification;
  string ModName = "add[@name='LitwareBrandingModule']";
  string ModXPath = "configuration/system.web/httpModules";
  modification = new SPWebConfigModification(ModName, ModXPath);
  modification.Owner = "LitwareBranding";
  modification.Sequence = 0;
  modification.Type =

SPWebConfigModification.SPWebConfigModificationType.EnsureChildNode;
  modification.Value =
   @"<add name=""LitwareBrandingModule"" " +
       @ "type=""LitwareBranding.LitwareBrandingHttpModule, [4-part
assembly name]"" />";
  return modification;
}
```

Once you have a utility method such as CreateHttpModuleModification which returns an initialized SPWebConfigModification object, you can simply call this method from event handlers such as FeatureActivated and FeatureDeactivating to add or remove the required HttpModule entry to or from the web.config file for the current Web Application.

```
public override void FeatureActivated(SPFeatureReceiverProperties
properties) {
  SPWebApplication WebApp =
(SPWebApplication)properties.Feature.Parent;
  WebApp.WebConfigModifications.Add(CreateHttpModuleModification());
  WebApp.WebService.ApplyWebConfigModifications();
  WebApp.WebService.Update();
}
```

```
public override void FeatureDeactivating(SPFeatureReceiverProperties
properties) {
  SPWebApplication WebApp =
(SPWebApplication)properties.Feature.Parent;
  WebApp.WebConfigModifications.Remove(CreateHttpModuleModification());
  WebApp.WebService.ApplyWebConfigModifications();
  WebApp.WebService.Update();
}
```

Initializing Branding During Feature Activation

Now, it’s time to put all of these pieces together. When a user activates the LitwareBranding Feature within a specific site collection, there is a Fseature receiver with a FeatureActivated event handler that uses the BrandManager class to apply all the various branding elements. There is also a FeatureDeactivating event handler that removes all the branding elements during Feature deactivation.

```
public override void FeatureActivated(SPFeatureReceiverProperties
properties) {
  EnsureWebApplicationFeatureEnabled();
  BrandManager.ConfigureMasterUrl(true);
  BrandManager.ConfigureCustomMasterUrl(true);
  BrandManager.ConfigureAlternateCss(true);
  BrandManager.ConfigureSiteLogo(true);
  BrandManager.ConfigureApplicationPageMaster(true);
}

public override void FeatureDeactivating(SPFeatureReceiverProperties
properties) {
  BrandManager.ConfigureMasterUrl(false);
  BrandManager.ConfigureCustomMasterUrl(false);
  BrandManager.ConfigureAlternateCss(false);
  BrandManager.ConfigureSiteLogo(false);
  BrandManager.ConfigureApplicationPageMaster(false);
}
```

Note the call to the EnsureWebApplicationFeatureEnabled method at the beginning of the FeatureActivated event handler. This method has been written ensure that the Web Application-level Feature named LitwareBrandingWebApplication has been activated so that the HttpModule which swaps out the master pages for application pages is properly registered with ASP.NET.

```
public void EnsureWebApplicationFeatureEnabled() {
  // make sure feature which adds HttpModule to web.config is active
  SPSecurity.RunWithElevatedPrivileges(delegate() {
    using (SPSite siteCollection = new
SPSite(SPContext.Current.Site.ID)) {
      try {
        Guid FeatureId = new Guid("FF739C76-0B08-4bc2-A3A2-
F61524B492D8");
        siteCollection.WebApplication.Features.Add(FeatureId);
      }
      catch { }
    }
  });
}
}
```

It makes use of the GUID from the LitwareBrandingWebApplication's feature.xml file to add that Feature to the site collection. If you're following along at home, make sure you use the right value for your Feature. If the Feature is already added to the site collection, an error will be thrown, which the empty `catch` statement will trap and swallow so as to not halt our processing.

Auto-initializing Branding in Child Sites Using Feature Stapling

So far, we've applied our branding to both site pages and application pages for every site within a site collection where our Feature is activated. Great, we're done right? Unfortunately, no. WSS and MOSS pose another design problem when creating a site collection-scoped branding solution. You must decide how to deal with properly branding new child sites as they are created. As we saw in looking at the code, the BrandManager class only affects the branding properties of existing sites. Therefore, the LitwareBranding solution includes a third Feature named **LitwareBrandingChildSiteInitializer** which handles applying the custom branding elements to new sites as they are created anywhere in the site collection where our main Feature is activated. The LitwareBrandingChildSiteInitializer Feature is very basic, containing only a Feature receiver and an event handler for the FeatureActivated event:

```
// fired whenever a new site is created
public override void FeatureActivated(SPFeatureReceiverProperties
properties) {
  SPWeb ChildSite = (SPWeb)properties.Feature.Parent;
  SPWeb TopLevelSite = ChildSite.Site.RootWeb;
  ChildSite.MasterUrl = TopLevelSite.MasterUrl;
  ChildSite.CustomMasterUrl = TopLevelSite.CustomMasterUrl;
  ChildSite.AlternateCssUrl = TopLevelSite.AlternateCssUrl;
  ChildSite.SiteLogoUrl = TopLevelSite.SiteLogoUrl;
  ChildSite.Update();
}
```

As you can see, this event handler fires during Feature activation and copies the top level site's branding properties into the current child site. Now you must figure out how to get this Feature to activate automatically whenever a new child site is created inside a site collection in which the LitwareBranding Feature has been activated. We looked at this breifly earlier, but the answer is Feature stapling. You'll recall from our discussion of the LitwareBranding Feature that it provides a FeatureSiteTemplateAssociation element in its feature.xml which staples the LitwareBrandingChildSiteInitializer Feature to the GLOBAL site definition.

```
<!-- staple GLOBAL site defition to LitwareBrandingChildSiteInitializer
-->
<FeatureSiteTemplateAssociation
  Id="1204A425-D105-46c5-BB2C-473A2F27B563"
  TemplateName="GLOBAL" />
```

Notice that we are attaching, or stapling, a Feature with an ID of **1204A425-D105-46c5-BB2C-473A2F27B563**, which happens to be the ID of our LitwareBrandingChildSiteInitializer Feature, to the GLOBAL site definition. If you're following along at home, make sure you use the right value for your Feature.

GLOBAL is a special site definition that is used automatically for any site created, regardless of what template or site definition the user selects. GLOBAL is applied first, and then the specific definition chosen by the user is applied. This allows developers to easily set up elements that are used across WSS, regardless of what definition is used.

This stapling technique is what forces automatic Feature activation on a newly created child site. By stapling the LitwareBrandingChildSiteInitializer Feature to the GLOBAL site definition, you are in effect configuring the Feature to activate automatically whenever a new site is created no matter what site template or definition has been used.

Note that there is one exception to the GLOBAL rule. Sites created from the standard WSS site definition named **Blank Site** do not get GLOBAL elements applied to them. This behavior was added to SharePoint 2007 so that sites created from the Blank Site template will work with the MOSS content deployment strategy. If you want to automate the activation of the LitwareBrandingChildSiteInitializer Feature in sites created from the Blank Site template, you must add explicit stapling instructions for that Site Definition and configuration as well, as we have in our sample solution.

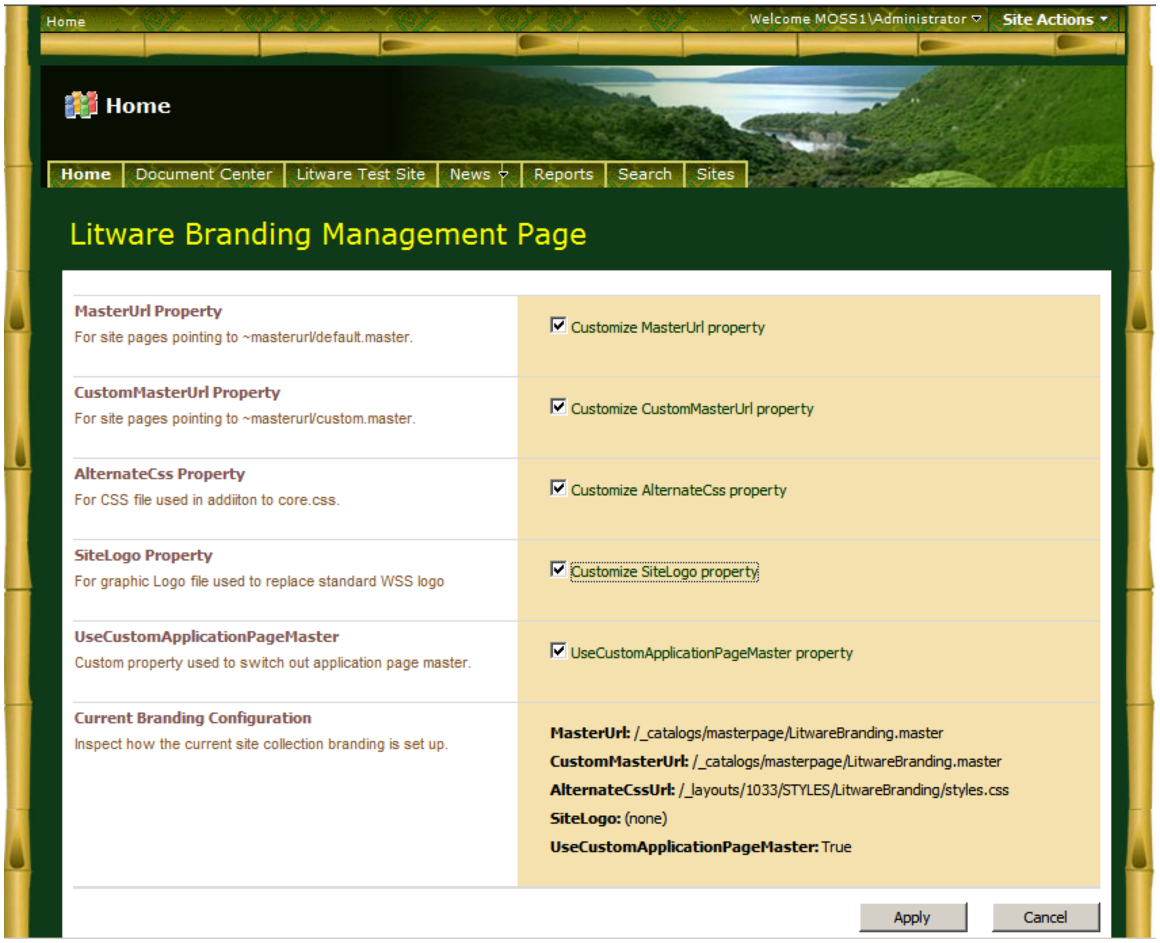
```
<!-- staple blank site template to LitwareBrandingChildSiteInitializer
-->
<FeatureSiteTemplateAssociation
  Id="1204A425-D105-46c5-BB2C-473A2F27B563"
  TemplateName="STS#1" />
```

As before, if you're following along at home, make sure you use the right ID value for your Feature.

Managing Branding Elements

The solution we have covered so far is an all-or-nothing proposal. This works great for single, monolithic implementations where a single brand is being centrally developed, managed and applied in one piece. However, what if you need more flexibility? What if you need to be able to turn on and off individual elements of your brand?

In that case, having all branding elements applied or removed in a Feature receiver is not the best option. Instead, you will need an application page that can turn on or off each specific element. The Visual Studio solution included with this article includes a rudimentary version of just such a page - BrandManagement.aspx, shown in Figure 7.



Litware Branding Management Page

MasterUrl Property For site pages pointing to ~masterurl/default.master.	☑ Customize MasterUrl property
CustomMasterUrl Property For site pages pointing to ~masterurl/custom.master.	☑ Customize CustomMasterUrl property
AlternateCss Property For CSS file used in addiiton to core.css.	☑ Customize AlternateCss property
SiteLogo Property For graphic Logo file used to replace standard WSS logo	☑ Customize SiteLogo property
UseCustomApplicationPageMaster Custom property used to switch out application page master.	☑ UseCustomApplicationPageMaster property
Current Branding Configuration Inspect how the current site collection branding is set up.	**MasterUrl:** /_catalogs/masterpage/LitwareBranding.master **CustomMasterUrl:** /_catalogs/masterpage/LitwareBranding.master **AlternateCssUrl:** /_layouts/1033/STYLES/LitwareBranding/styles.css **SiteLogo:** (none) **UseCustomApplicationPageMaster:** True

Figure 7: The Liteware Brand Management page allows for more granular control over branding elements

The code for this page is extremely simple. It makes use of the same BrandManager utility used in our Feature receiver, as seen here in the handler for the Apply button click event:

```
protected void cmdApplyCustomBrand_Click(object sender, EventArgs e) {
      bool ApplyMasterUrl = chkCustomizeMasterUrl.Checked;
      bool ApplyCustomMasterUrl = chkCustomizeCustomMasterUrl.Checked;
      bool ApplyAlternateCss = chkCustomizeAlternateCss.Checked;
      bool ApplySiteLogo = chkCustomizeSiteLogo.Checked;
      bool ApplyCustomApplicationPageMaster =
chkUseCustomApplicationPageMaster.Checked;

      BrandManager.ConfigureMasterUrl(ApplyMasterUrl);
      BrandManager.ConfigureCustomMasterUrl(ApplyCustomMasterUrl);
      BrandManager.ConfigureAlternateCss(ApplyAlternateCss);
      BrandManager.ConfigureSiteLogo(ApplySiteLogo);

BrandManager.ConfigureApplicationPageMaster(ApplyCustomApplicationPageM
aster);

      SPUtility.Redirect(Request.RawUrl,
                         SPRedirectFlags.Default,
                         HttpContext.Current);
    }
```

While the sample page included with the solution is very basic, it hints at some of the extensibility points available for this solution. Using this page as a launchpad, it would be very easy to:

- Allow administrators to specify a master page (or a CSS file, site logo, or any combination of these) to be used rather than having it hardcoded in our solution by presenting textboxes on this page to collect the information
- Allow administrators to select which elements of our brand would be applied
- Build our own skinning engine that would allow administrators or users to specify which brand, or which elements of a brand, to apply from a list of approved choices. This would be similar to themes, but would maintain our brand and could be deployed across environments and applied to all page types.

The last element to know if you are making use of a brand management page such as this one is how to provide access to it. In our sample solution we do this by simply adding a new menu item to the *Look and Feel* section of the Site Settings page, as shown in 8.

We do this with a simple entry in the elements.xml file of our LitwareBranding Feature, as shown below.

```
<CustomAction
        Id="SiteActionsToolbar"
        GroupId="Customization"
        Location="Microsoft.SharePoint.SiteSettings"
        Sequence="30"
        Rights="ManageWeb"
        Title="Litware Branding Manager" >
            <UrlAction
Url="~sitecollection/_layouts/LitwareBranding/BrandManagement.aspx"/>
      </CustomAction>
```

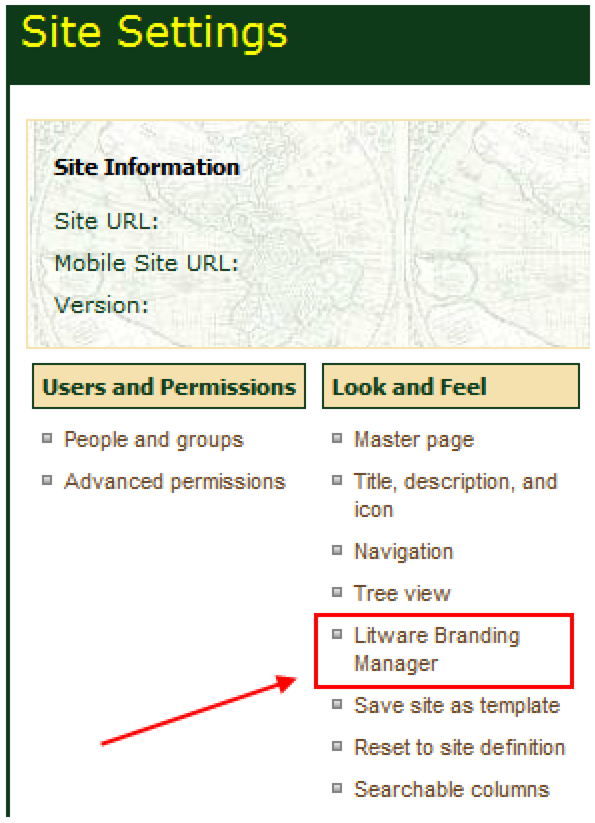


Figure 8: Providing access to our brand management page

One thing that is important to note is that we do not want just anyone who happens to make their way to the Site Settings page (assuming they have some level of permissions to the Site Settings page) to be able to click on the link and be able to change the branding on our site. What we need is some way to secure the branding page.

If you look at the `CustomAction` entry above, you'll see a `Rights` attribute which has a value of *ManageWeb*. This does exactly what you would expect - it only shows the menu to users who have the *ManageWeb* permission for the current site. Valid values for this attribute are any elements from the SPBasePermissions enumeration. That takes care of one part - it hides the menu option if users don't have the proper permissions; but it doesn't actually secure the page itself. If someone happens to know the URL, they can still navigate to it and change the branding.

To secure the page itself, we need to make some changes to the code file. If you look at the BrandManagementPage.cs file, you'll see the following code:

```
protected override SPBasePermissions RightsRequired
{
    get
    {
        return SPBasePermissions.ManageWeb;
    }
}
```

Like the Rights attribute in the elements.xml file, this code will check that users have the proper permissions before showing them the page. User who navigate directly to the page but do not have the proper permissions will see the access denied page - properly branded because of the changes we made earlier to the simple.master page.

Disabling Themes

If you've gone to all of the trouble of creating a custom brand and applying it to your sites, you almost certainly don't want people to muck about with it by applying their own theme. It is also highly likely that if your custom master page has changed some of the page elements, that the default themes would not render properly anyway. For these reasons it is advisable to disable theme support if you have applied a custom brand. There are two pieces to this:

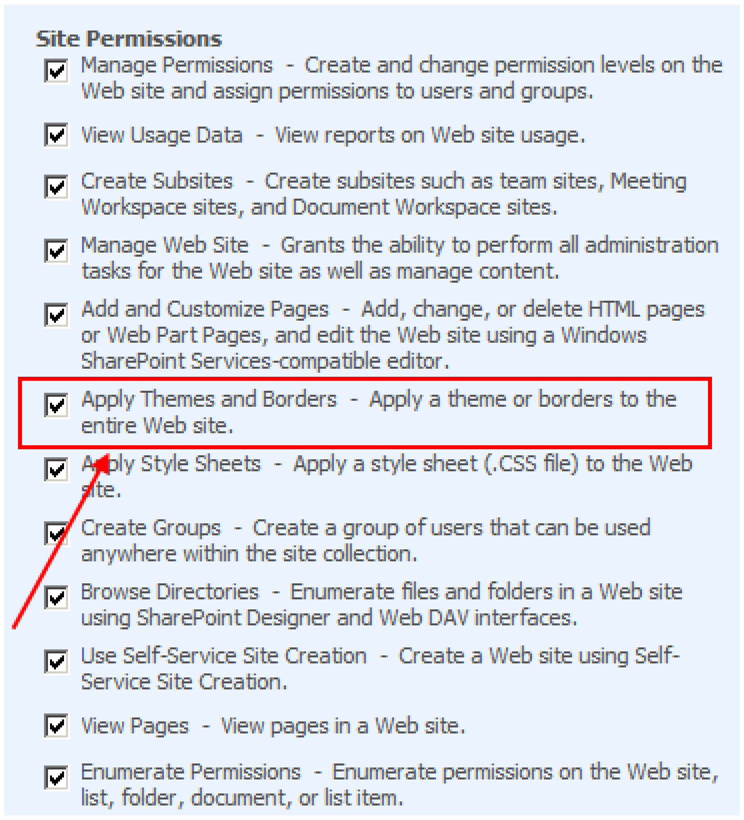
1. Removing the Site Theme menu item from the Site Settings page
2. Changing security on our site so that theming is not available

Removing the Site Theme menu item from the Site Settings page requires another entry in the elements.xml file of our LitwareBranding Feature:

```
<HideCustomAction
      Id="HideTheme"
      HideActionId="Theme"
      GroupId="Customization"
      Location="Microsoft.SharePoint.SiteSettings" />
```

As you might guess from our previous coverage of the elements.xml file, this entry will hide the *Site Theme* link from the *Look and Feel* section of the Site Settings page. This is part of the battle, but it won't stop users from navigating directly to the page and changing the site theme from there.

To accomplish that, we need to make some permission changes. One of the values of the SPBasePermissions enumeration is ApplyThemeAndBorder, and as you can likely guess, this is the one we are interested in. In Central Administration, turning off this permission is available via the *User Permissions for Web Application* link in the *Application Security* section of Application Management, as shown in Figure 9.



Site Permissions

- [✓] Manage Permissions - Create and change permission levels on the Web site and assign permissions to users and groups.
- [✓] View Usage Data - View reports on Web site usage.
- [✓] Create Subsites - Create subsites such as team sites, Meeting Workspace sites, and Document Workspace sites.
- [✓] Manage Web Site - Grants the ability to perform all administration tasks for the Web site as well as manage content.
- [✓] Add and Customize Pages - Add, change, or delete HTML pages or Web Part Pages, and edit the Web site using a Windows SharePoint Services-compatible editor.
- [✓] Apply Themes and Borders - Apply a theme or borders to the entire Web site.
- [✓] Apply Style Sheets - Apply a style sheet (.CSS file) to the Web site.
- [✓] Create Groups - Create a group of users that can be used anywhere within the site collection.
- [✓] Browse Directories - Enumerate files and folders in a Web site using SharePoint Designer and Web DAV interfaces.
- [✓] Use Self-Service Site Creation - Create a Web site using Self-Service Site Creation.
- [✓] View Pages - View pages in a Web site.
- [✓] Enumerate Permissions - Enumerate permissions on the Web site, list, folder, document, or list item.

Figure 9: Removing the ApplyThemeAndBorder permission through Central administration

Ideally, turning off this permission would remove the link from Site Settings, but alas, SharePoint does not make it that easy for us. We still need the `HideCustomAction` element covered above.

Figure 9 shows how you would disable this permission manually through Central Administration, but we would like to do so programmatically. To disable the ApplyThemeAndBorder permission with code, we need to add the following lines to the FeatureActivated event of our LitwareBrandingWebApplication Feature:

```
//block users from applying themes

SPBasePermissions perm = SPBasePermissions.ApplyThemeAndBorder;

WebApp.RightsMask = WebApp.RightsMask & ~perm;

WebApp.Update();
```

To make things easy, I add it right at the end of the method, after the code that updates the web.config file.

To wrap things up nicely, we ought to re-enable this permission whenever our branding is removed, so add the following to the FeatureDeactivating event, again, at the end after we update the web.config:

```
//allow users to apply themes

SPBasePermissions perm = SPBasePermissions.ApplyThemeAndBorder;

WebApp.RightsMask = WebApp.RightsMask | perm;
WebApp.Update();
```

The net effect of these final changes is to hide the Site Themes link from the Site Settings page and stop people from applying a theme if they navigate directly to the /_layouts/themeweb.aspx page. It is important to note that the user will be able to see the Site Theme page, will be able to select a theme, and will be able to click the Apply button. However, at that point, a permission check will be done and they will get an access denied error. It is perhaps not the best user experience, but it is what we have, and after all, they manually navigated to the page despite the fact that we had removed the link, so what do they expect?

Summary

This article has walked through a complete branding solution for changing the look and feel of sites in SharePoint 2007. The LitwareBranding solution has been designed to apply its branding using custom master pages and a custom CSS file on a site collection-wide basis. Along the way you also saw some advanced techniques for swapping out the master page for application pages and for using Feature stapling to automatically apply branding to child sites as they are created. It is also noteworthy that the entire LitwareBranding solution and all of its internal components are deployed using a single solution package. This ensures the easiest and most reliable path to deploy your development efforts across multiple farms running either WSS 3.0 or MOSS.

Additional Resources

For more information, see the following resources:

- Add link to related content on MSDN
- Another link to more content
- Microsoft Office Developer Center

See Also

Implementing a Brand in a SharePoint Publishing Site

#Introduction

Andrew Connell, Andrew Connell Inc. (Microsoft MVP)

Month 2006

Applies to: Microsoft Office SharePoint Server 2007

Summary: Microsoft Office SharePoint Server 2007 Publishing sites are commonly used as the external face for companies and organizations. As the online presence, they should employ a consistent look and feel to make navigation and finding information as easy as possible for users. The challenge comes in the implementers of the site as there are various options, each with its own advantages and disadvantages, in the implementation of brand. This article provides guidance on selecting the right approach for implementing a brand in a SharePoint Publishing site. (XX printed pages)

Contents

#Introduction to Branding SharePoint Publishing Sites

The most recent version of SharePoint, Microsoft Office SharePoint Server (MOSS) 2007, introduced the capability to host content-centric sites on the SharePoint platform. These sites, commonly referred to as Publishing sites for the SharePoint Publishing Features they employ, leverage a series of concepts and capabilities in MOSS 2007 called Web Content Management (WCM). Publishing sites are frequently used in an Internet facing scenario as the public face for a company or organization. Thus, it is very important for these sites to have a consistent look and feel. However developers and designers can quickly get confused with the various options available for how to implement and deploy the site's brand.

This article will explore things that a Publishing site project team should consider when planning how to implement a brand within SharePoint. While many of the concepts and issues covered in this article cover Windows SharePoint Services (WSS) 3.0 sites, the article is foxed on Publishing sites exclusively.

# ProductVersionKeyword_Introduction

# ProductVersionKeyword_MainSubhead

SharePoint Sites are ASP.NET 2.0 Applications

Before diving into the details of branding a SharePoint site, it helps to level the playing field a bit. SharePoint projects are frequently overcomplicated because many think of SharePoint as much more than it really is. Starting with the third generation of SharePoint (WSS 3.0 & MOSS 2007), it is built natively on top of the .NET Framework and can be considered a pure ASP.NET 2.0 application. The SharePoint product team simply implemented a custom HTTP application, and custom HTTP handlers and modules to implement SharePoint and change how ASP.NET 2.0 works by default.

The implementation of the SharePoint architecture is detailed in Chapter 2: SharePoint Architecture of the **INSIDE MICROSOFT WINDOWS SHAREPOINT SERVICES 3.0** *book by Microsoft Press. The chapter is available in the WSS 3.0 Software Development Kit (SDK) as an excerpt.*

While there are many similarities between SharePoint and ASP.NET 2.0, as expected, there are plenty of differences as well. Some of the more prominent similarities and differences are detailed in the following sections.

ASP.NET 2.0 and SharePoint Site Similarities

Because SharePoint is built on top of ASP.NET 2.0, there are not only some striking similarities between the two technologies, but most aspects of ASP.NET 2.0 sites bleed through directly into SharePoint. SharePoint fully leverages ASP.NET 2.0 concepts like the navigation provider and membership provider models. There may some additional configuration to make SharePoint aware of certain aspects of ASP.NET 2.0 technologies, such as in the case of the membership provider model, but SharePoint still fully offloads that work to ASP.NET 2.0.

Specifically to the theme of this article, the technologies and techniques used to branding SharePoint sites is virtually the same as it is in ASP.NET 2.0. Images and cascading style sheets (CSS) are used to implement a custom brand within a traditional ASP.NET 2.0 site. SharePoint employs the exact same techniques. In addition, ASP.NET 2.0 sites offer master pages which simplify the global branding of a site. SharePoint also leverages master pages however in a somewhat different way than ASP.NET 2.0. This is covered in the next section.

ASP.NET 2.0 and SharePoint Site Differences

While there are many similarities between ASP.NET 2.0 and SharePoint sites, there are also quite a few differences. One of these is the concept of customized and uncustomized files. In an ASP.NET 2.0 site it is relatively easy to determine where a file resides by inspecting the URL (when things such as URL rewriting are not employed). This is because ASP.NET 2.0 is configured out-of-the-box (OOTB) to retrieve files directly off the file system. This approach does not work in SharePoint because one ASPX page on the file system could be used as a template for many pages in one or more SharePoint site. To address this, the product team implemented a virtualized file system that is stored in the SharePoint content databases for each SharePoint site. In addition, files in a SharePoint site can be customized or uncustomized. An uncustomized file is that simply exists in the virtualized file system as a pointer to the template it is based on that resides on the file system. Customized files on the other hand are ones that have been modified and who's source is stored in the SharePoint content database. This enables power users to customize the same file on a site-by-site basis without affecting other sites that leverage the same template on the file system. For more information on this concept refer to the MSDN article Understanding and Creating Customized and Uncustomized Files in Windows SharePoint Services 3.0.

The previous section covered master pages as a similarity between ASP.NET 2.0 and SharePoint site. While the same technology is used in both environments, the implementation is a little different. In an ASP.NET 2.0 site, content pages reference a specific master page and this defined by the site designer or developer at design time. This model does not work for SharePoint site because when developing ASPX pages, one cannot always be certain which SharePoint site the page will be used in. Therefore the SharePoint team elected to allow site owners select a master page for use across all pages in a site. Therefore content pages, ASPX files, in a SharePoint site should not be configured to point to a specific master page file. Instead, dynamic tokens are used to tell SharePoint, at runtime, which master page should be used. SharePoint replaces this token with the URL of one of the two master pages that had previously been selected by the site owner for use in the current SharePoint site. This is detailed in the WSS 3.0 SDK: Customizing Master pages in Windows SharePoint Services.

Another big difference between ASP.NET 2.0 and SharePoint is the code and custom file deployment story. In ASP.NET 2.0 sites, custom code and files including DLL's, ASPX's, ASCX's, GIF's, JPG's, CSS and JS's to name just a few are typically deployed as loose files to production systems. While Visual Studio 2008 does have deployment / publish capability, developers do not usually have write access to a production server. Another deployment option in ASP.NET 2.0 sites is to create an installer (MSI) or package all the files into a ZIP for an administrator to use in the deployment on a production server. Finally, a management application such as Microsoft Systems Management Server or a 3rd Party deployment application can be used for the deployment of an ASP.NET 2.0 application.

On the other hand, SharePoint includes a robust custom code and file deployment mechanism. The Windows SharePoint Services Solution Package framework (aka: solution framework, SharePoint solutions or just WSP's) is SharePoint's internal deployment vehicle for custom code and files. Developers package up all custom code and files into a Microsoft Cabinet (*.CAB) file with the *.WSP file extension, add a special manifest file (`manifest.xml`) to the root of the WSP and then add the solution to the SharePoint farm's solution store. The manifest file lets SharePoint know about all the files in the solution, telling it what each file is used for and where it should be deployed on the server. Then, either on demand or at a scheduled time, SharePoint deploys the files to the other SharePoint servers based on what is defined in the manifest and selected in the deployment settings such as which Web application the solution is deployed for. If SharePoint is running in a load balanced environment, the solution framework automatically determines which servers to deploy the solution to and does deploys to all servers are the same time. Finally, the solution framework also supports retracting, or "undo-ing", the deployment. For more information on the solution framework, refer to the official documentation in the WSS 3.0 SDK on MSDN: http://msdn.microsoft.com/en-us/library/aa543214.aspx.

Site Content is not Branding

Before covering branding SharePoint Publishing sites, it's important to understand the difference between content and branding assets. Quite simply, branding is anything that is used to implement the user interface of the site. This could be images or CSS files… thinks that implement the corporate brand or theme. The brand of a SharePoint Publishing site is typically not very dynamic and doesn't change that often. Rather organizations typically undergo a rebranding campaign, either slightly customizing or completely revamping the look and feel of the company. At any rate, the files associated with the brand do not change on a frequent basis (every few days or weeks). In addition, the branding files are usually owned and controlled by the site developers and designers… the team responsible for creating and maintaining the site.

Content, on the other hand, is much more dynamic. Content consists of the text and media, such as images and collateral, that makeup the reason for the site. These are the press releases, product information pages and images of the products. Content is owned and controlled by the content owners and authors rather than the production team responsible for the site. Content also usually follows a publishing process complete with workflow, permissions limiting who can create, approve or publish the content and versioned. Another difference between content and branding is that content is usually updated more frequently than branding files. Some pages might be updated daily or every few days, other pages may be updated much less frequently. Regardless, the time between content updates is usually much shorter than the time between branding updates.

The reason for this distinction is that the two are stored in very different locations and should be kept separate from each other. There are multiple reasons for this separation. First, the individuals who own the content and branding are typically different people or groups. By putting everything in one location, the permissions can get mixed. Second, as outlined above, content typically follows a publishing process with robust workflow and business rules within production whereas branding is validated in a development or test environment and ultimately rolled out into production.

With an understanding of the differences between content and branding assets, it is now time to look at the various deployment options.

Deployment Options for Branding Files

As covered in the previous section, branding files are different, and should be treated differently, from content. First, consider where content should go within a Publishing site. Every Publishing site within a Publishing site collection contains three special libraries: **Pages**, **Images** and **Documents**. The Pages library is where the content pages, also known as the Web pages, should be stored for each site. The Images and Documents library is where content collateral should be stored that is referenced from pages in that same site. Each site has these libraries to facilitate a certain level of control over what sections of the site certain users can manage. For instance, maybe contributors to the Press Releases section of the site should not have the same rights or access to the materials in the Products section. While SharePoint allows them to have rights in both, it does support keeping things isolated if the business requirements demand that.

There are two special libraries in the top level site of the site collection: **Site Collection Images** and **Site Collection Documents**. The reason for these two libraries is that regardless of what site a content owner or author has access to, there are some assets that everyone should have access to. For instance, the company logo or privacy policy should be available to all content authors. Instead of having multiple copies floating around the various Images libraries throughout the sites, they should only reside in one place as well implemented content sites are all about content reuse and not content duplication.

When it comes to branding files, they should be kept separate from the content files for reasons outlined in the previous section. Therefore branding files should not go in the Images, Documents, Site Collection Images or Site Collection Documents libraries throughout the Publishing site collection. So where should branding files be deployed?

As with most things in SharePoint, there are multiple options and each has advantages and disadvantages. The following sections cover three options on where to deploy branding files as well as the advantages and disadvantages of each.

Deploying Branding Files to the Site's Web Root Folder

One obvious choice, especially to those with an ASP.NET 2.0 development background, is to simply deploy the branding files to the root folder (or a subfolder within it) of the Web application hosting the SharePoint Publishing site collection. This works just fine and is familiar to traditional ASP.NET 2.0 developers and designers as it follows a similar model to traditional ASP.NET 2.0 Web sites. However this approach brings with it many disadvantages.

First and foremost, there is no OOTB automated deployment option when deploying files to the Web root. SharePoint solutions, as previously covered in this article, cannot deploy files to the Web root of a Web application. They can only copy files to a location within the SharePoint "12" folder (usually `c:\Program Files\Common Files\Microsoft Shared\web server extentions\12`), modify code access security policies, deploy assemblies to the server's global assembly cache or `\bin` folder in the Web root, and make minor changes to the Web application's `web.config` file. This means that not only can the deployment of the brand not be scheduled using the same OOTB solution framework vehicle, but that if a SharePoint farm employs multiple load balanced Web front end (WFE) servers, the deployment must be coordinated for all servers. This challenge can be mitigated using a deployment management package as previously covered.

Building off this point, when a new SharePoint site is added to the farm, if the SharePoint Web services are started, SharePoint will automatically deploy all previously deployed solutions to the new server. However because the branding files were not deployed using SharePoint solutions, they will not be automatically deployed. Thus, farm administrators must be aware of what manual deployment steps must be implemented on the new server.

One aspect to this approach is the files are stored within the SharePoint site. This means that the files cannot be versioned as they could be within a publishing site, nor could they be scheduled. For instance, maybe a few files that makeup branding changes, like CSS or a new company logo, should not be used before a specific date and time. If the assets were in a Publishing site, they could be scheduled for when they would go live. This also means that the files are not tightly coupled to the site and thus have to be backed up separately from the site.

Another concern with this approach is backup/restore and disaster recovery. When the Publishing site is backed up, either using SQL Server database backups or using SharePoint's backup capabilities, only the content within the site is backed up; none of the files on the file system are backed up.

In addition, if multiple Publishing site collections were hosted in the same Web application, they would all have access to the same branding files. This is because all files in the web applications root folder are accessible by all site collections in the Web application. Granted this may or may not be an issue in the business requirements, but if so it is something to be aware of. However, this may in fact be considered an advantage of this approach!

Deploying Branding Files to the _layouts Directory

Another option is to deploy the files into SharePoint's `_layouts` virtual directory. This folder maps to the `[..]\12\TEMPLATE\LAYOUTS` folder. This approach mitigates a big disadvantage of the previous technique as administrators can deploy files using the SharePoint solution framework. This means the deployment of branding files can not only scheduled but they are also automatically deployed to all SharePoint WFE's in the farm when the solution is deployed.

One of the downsides to this approach is that the files still reside on the file system and thus most of the same concerns mentioned in the previous approach carry over to this one as well. This includes the lack of versioning or scheduling branding files.

In addition, another potential concern here is that files deployed to the `_layouts` virtual directory are available to all SharePoint sites within the farm… not just Publishing sites. This is due to the fact that all SharePoint WFE's in a farm should be virtual mirrors of each other each has a single `[..]\12\TEMPLATE\LAYOUTS` folder. While each SharePoint Web application has a separate `_layouts` virtual directory, they all point to the same folder on each server. This means that every site will have access to the same files. Again, maybe this is not a concern in a specific implementation, but if it is, administrators should be aware of it. However, just like the previous approach, this may be an advantage rather than a disadvantage of this approach. It all depends on the business culture, deployment concerns and business requirements.

Deploying Branding Files to the Site Collection's Content Database

This approach is very different from the previous two approaches. Files do not have to live on the file system. Instead, they can be deployed to the Publishing site collection and thus, be included in the site's virtual file system. This is how the OOTB Publishing Portal site template implements its brand.

The biggest advantage here is that all the branding is included in the same SharePoint content database that contains the content of the site. This means that either SQL Server database backups or SharePoint backups will also contain the branding files. In addition, because the files live in the content database they could be versioned or even scheduled.

Simply saying “they go in the site collection” or “they go in the content database” does not say much. Where exactly should they go? Recall the discussion previously in this article about the differences between content and branding and how they should be kept separate. Therefore images related to the brand should not be stored in any of the site’s Images libraries or the top level site’s Site Collection Images library. Instead, all branding files should be put in the Style Library SharePoint library found in the top level site of the site collection. All authenticated users in the Publishing site are automatically granted access to the Style Library which is a good thing considering the brand will be persisted across virtually all pages in the Publishing site! The same cannot be said for the Images or Site Collection Images libraries. While it may not be terribly common, there are plenty of Publishing sites where users are not granted access to the root of the Publishing site, rather they only have rights within a subsection of the site. Even though the Style Library is in the top level site of the site collection, all authenticated users in a site, even those who only have access to a subsection of the site collection, have access to the Style Library.

This approach may be a bit more comfortable to developers and designers as they can add content directly to the site’s virtual file system using tools such as Office SharePoint Designer 2007. However recall the previous discussion about customized and uncustomized files, especially the details of it. There can be challenges to maintaining the files that live exclusively in the virtual file system (ala content database). However these can be mitigated using the technique of provisioning files using Features. Thus, the brand can be packaged up in a SharePoint solution for a clean and repeatable deployment story and relatively easy updates. The technique of provisioning files using Features is covered in the Understanding and Creating Customized and Uncustomized Files in Windows SharePoint Services 3.0 MSDN article.

Summary

This article has explored the various options available those implementing a custom brand in a Publishing site. As explained for the start, the goal here is not to define the right or wrong approach to implementing a brand in a Publishing site. Rather, the goal is to provide guidance and education around the various options for implementing a brand. Consider all the options, the advantages and disadvantages of each and the specifics of each Publishing site project to adopt the right approach. It likely does not make sense for all projects to adopt the same process and technique.

Acknowledgements

I would like to acknowledge Randy Drisgill and Heather Solomon (SharePoint MVP) for their feedback in discussions that contributed to this article.

Additional Resources

For more information, see the following resources:

- Microsoft Windows SharePoint Services Developer Center
- Microsoft Office Developer Center
- Understanding and Creating Customized and Uncustomized Files in Windows SharePoint Services 3.0

- MSDN Book Excerpts: Inside Windows SharePoint Services 3.0 - Chapter 2: SharePoint Architecture
- MSDN Book Excerpts: Inside Windows SharePoint Services 3.0 - Chapter 3: Pages and Design