The Great SharePoint Adventure 2007

Student Hands On Lab Manual

This course includes the following labs:

- Customizing a WSS Site
- Developing Features
- Developing Custom Application Pages
- Developing Custom Site Pages
- Walkthrough of Custom Branding
- Creating Custom Web Parts
- Site Columns & Content Types
- Integration with OpenXML File Formats
- InfoPath 2007 and Forms Services
- SharePoint Workflows
- Portals
- Web Content Management
- Business Data Catalog
- Excel Services
- Building a Word 2007 Add-In

Lab 01: Customizing a WSS Site

Lab Time: 45 Minutes

Lab Directory: C:/Student/Labs/01_Customization

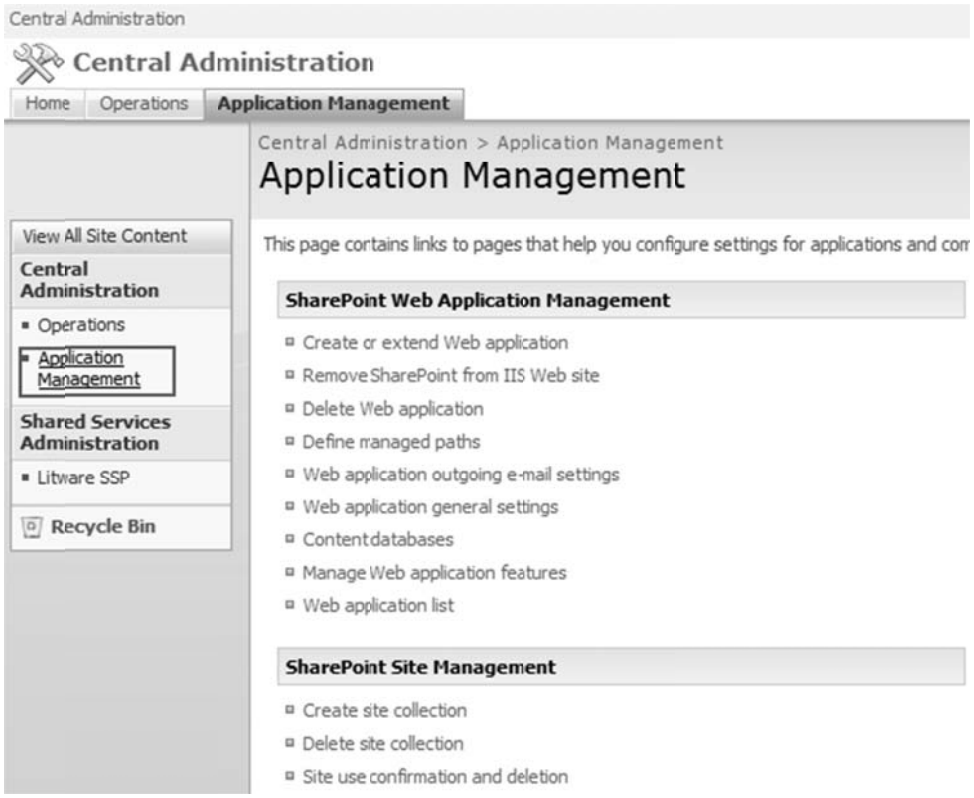
Lab Overview: In this lab you will be familiarizing yourself with basic SharePoint tasks such as Creating a Site Collection/Top-level Site, Creating a Child Site, Creating a List/Document Library and Making some basic Customizations to a Site.

Back Story: The management team at Litware has decided to use Windows SharePoint Services 3.0 to assist with storing and managing the content associated with their consulting projects. You will be instructed to create a top-level team site with a list that tracks a profile for each consulting project. You will also create a document library for storing various project-related documents such as customer presentations and proposals. You will also create child sites to track information about consultants in various divisions of the Litware corporation.

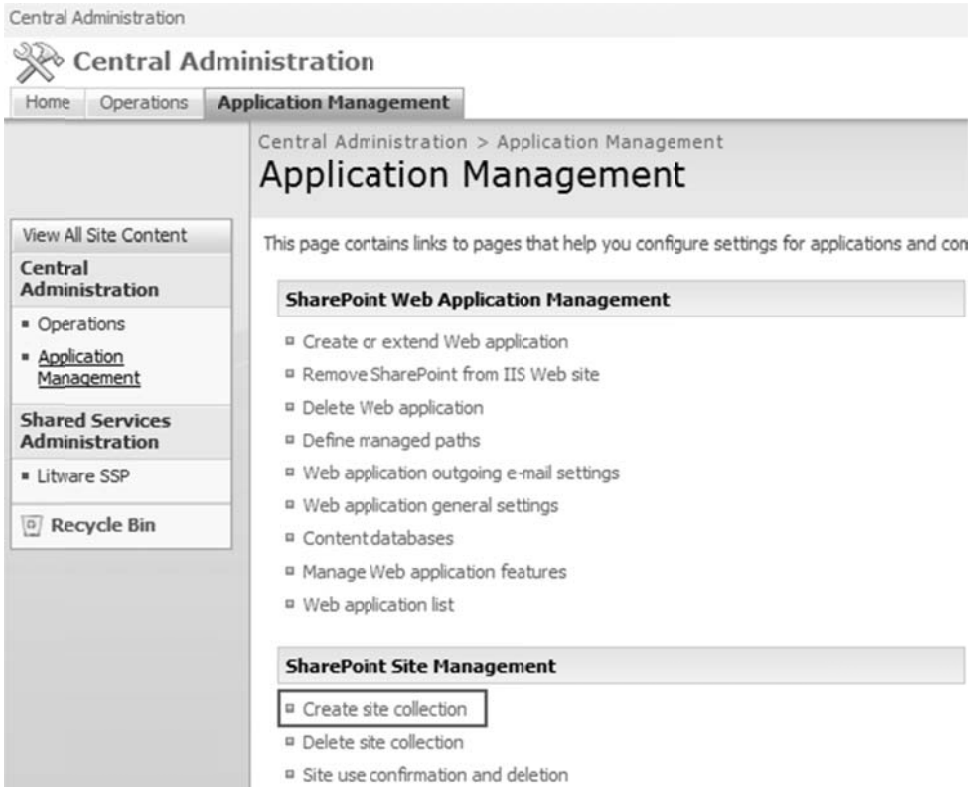
Important: You must finish this lab as written for a later lab to work correctly!

Exercise 1: Create a new site collection and a top-level site

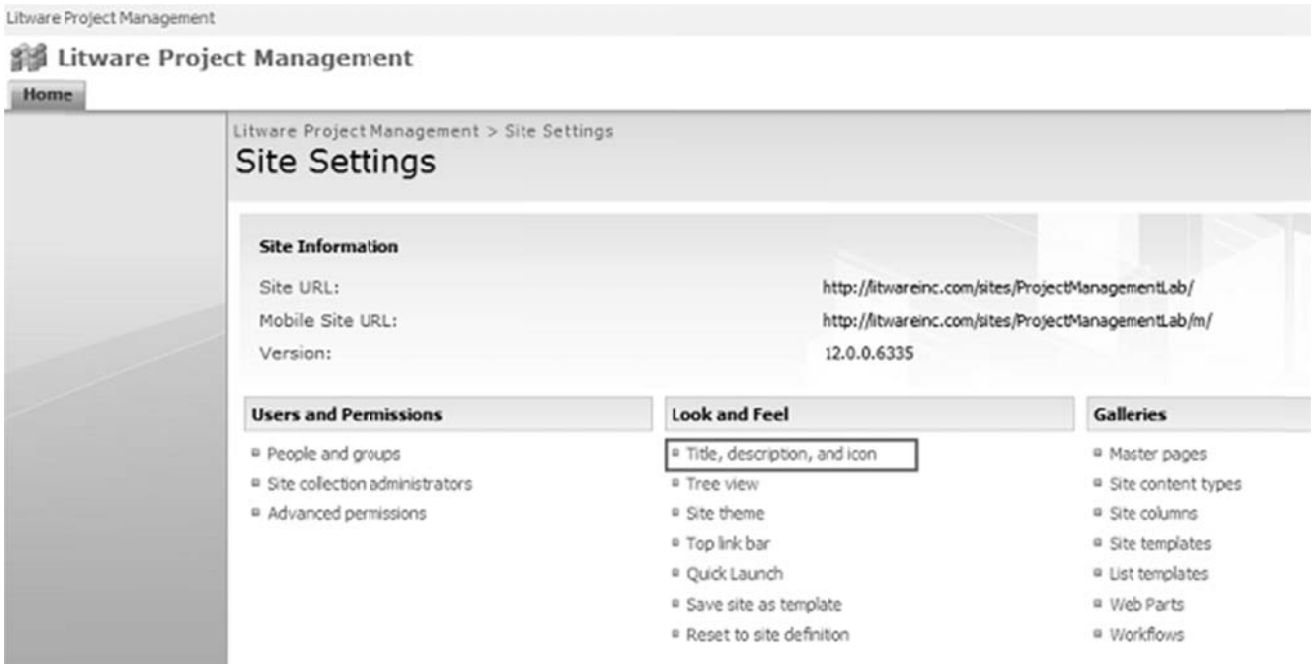
1. Make sure you are logged on as **LITWAREINC\Administrator**. Launch the WSS Central Administration Web site by using the command in the Windows Start menu. You can find the command at **Start >> Administrative Tools >> SharePoint 3.0 Central Administration**.
2. Once the home page of the WSS Central Administration Web site appears, click on the **Application Management** link at the top of the page to get to the Application Management page.



3. Under the **SharePoint Site Management** section, click the link with the caption **Create site collection**.



4. This will take you to the page where you can create a new site collection and a new top-level site. On the **Create Site Collection** page, fill in the required information (see below for instructions) to create a new site collection.
    a. Make sure the Web Application used for site creation is the one named **Litware Public Site** that is accessible through the URL **http://litwareinc.com**.
    b. For the Title use **Litware Project Management**.
    c. Create the new site collection so that it's URL is **http://litwareinc.com/sites/ProjectManagementLab**.
    d. Under the **Template Selection** section, look at all the sites templates that are available. Choose **Blank Site** from the **Collaboration** tab as the site template for the new top-level site that will be automatically created.
    e. Assign the primary site collection owner as **LITWAREINC\Administrator**. Be sure to verify this by using Ctrl+ k or by clicking the check person icon (your entry should become LitwareInc Administrator)
    f. Leave the **Quota Template** with the default setting of **No Quota**
    g. Click **OK** to create the new site collection and top-level site. Once you see the page that confirms everything has been created, navigate to the top-level site using the browser (click the link http://litwareinc.com/sites/ProjectManagementLab).
5. In this step, you will change the title of your site. The Title you entered of **Litware Project Management** is just a little too boring. Choose the **Site Settings** command under the **Site Actions** menu. Once you are at the **Site Settings** page, click the link with the caption **Title, description and icon** under the **Look and Feel** section.



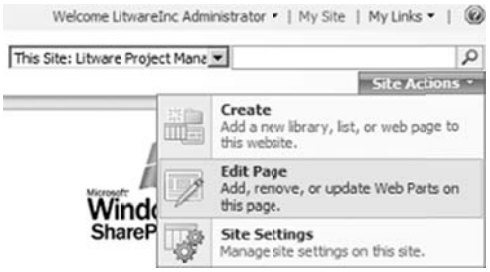
6. When you reach the **General Setting** Page, enter a different title that is a variation of **Litware Project Management**. Click **OK** and confirm that the home page now reflects the new site title.
7. Over the next several steps you will add custom Litware graphics to your new site to replace the standard Microsoft graphics.
    a. First, using the **Windows Explorer**, look inside the lab directory (**.../student/labs/01_Customization/**) and locate the subdirectory named **LitwareGraphics**. It contains several graphics files you will use to customize your site. Begin by copying the **LitwareGraphics folder** (not just the images inside but rather the folder itself along with everything it contains) to a location inside the WSS layouts directory so that you can make your graphics files accessible to all WSS sites within the farm. In particular, copy the **LitwareGraphics** directory into the directory at the following location:

        ```
        C:/Program Files/Common Files/Microsoft Shared/web server extensions/12/TEMPLATE/IMAGES
        ```

        Note that WSS configures the **IMAGES** directory with IIS so that it is a child virtual directory of the **_layouts** virtual directory. This means your graphics files should be accessible using a relative URL within any WSS site that looks like this:

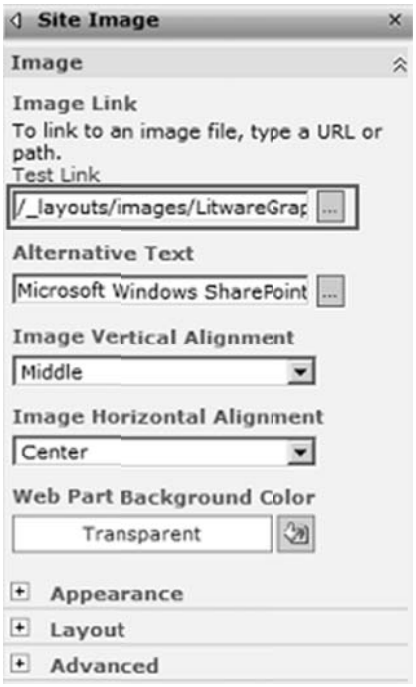
        ```
        .../_layouts/images/LitwareGraphics/
        ```

8. In this step, you will modify the Site Image Web Part on the site's home page so that it displays a custom Litware graphic instead of the default Microsoft graphic.
    a. Navigate to the home page of the site (http://litwareinc.com/sites/ProjectManagementLab/) and place the page into edit mode by choosing the **Edit Page** command under the **Site Actions** menu.



b. Once the page is in edit mode, find the **Site Image** web part (Right Zone, Top Choice) and on that web part's **edit** drop down **menu** select the **Modify Shared Web Part** command from the Web Part's action menu.

c. After you have run this command, you should see a task pane appear in the browser that allows you to modify the **Image Link** property.
d. Assign the **Image Link** property a new value of **/_layouts/images/LitwareGraphics/LitwareSlogan.png** and click **OK**.



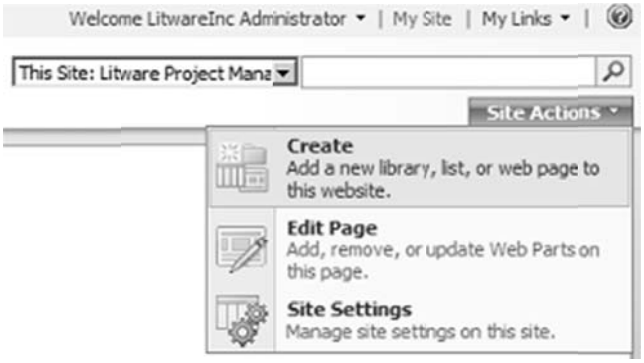
e. In the Right hand corner near the top of the page click **Exit Edit Mode.**

f. Confirm that the site's home page now shows the new Litware slogan graphic.

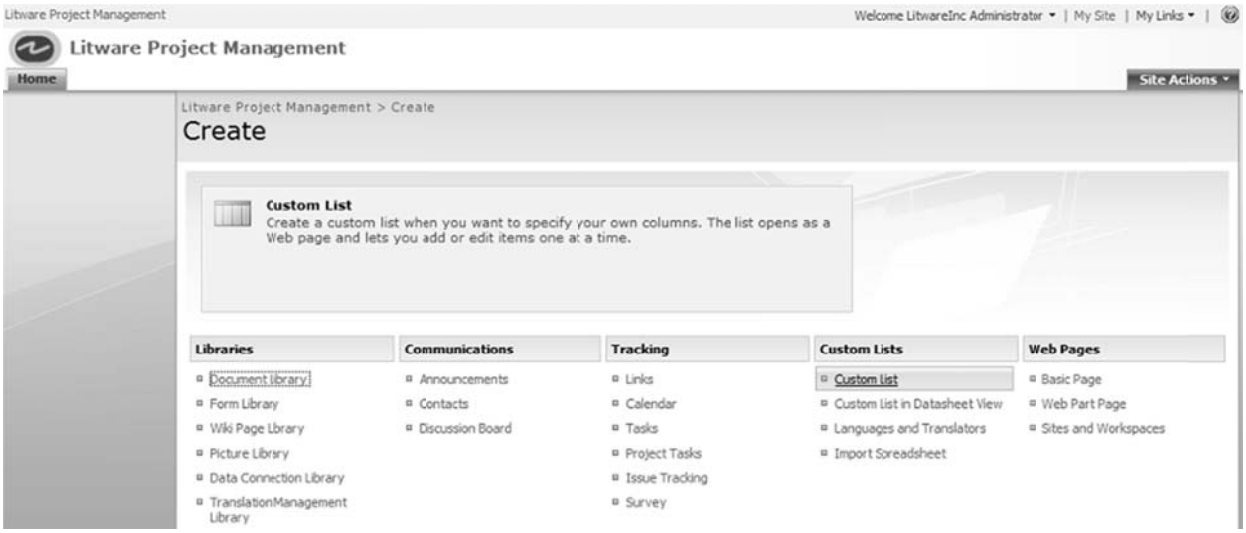
9. In this step we will add a Logo to the banner of our home page.
   a. Go to the Site Settings page (**Site Actions** drop down menu > **Site Settings**)and click on the link with the caption **Title, description and icon** under the **Look and Feel** section.
   b. Now enter a path to assign an URL to the graphics file named LitwareLogo.png. You path should be **/_layouts/images/LitwareGraphics/LitwareLogo.png**.
   c. Click the **Click here to test** and make sure that you can see the small logo, if not check your Url and try again.
   d. Click the **OK** button.
   e. Click on the **Home tab** on your page to navigate back to the main page (upper left side first and only tab on the page)
   f. Verify that the Litware logo appears on the banner of the home page (Upper left corner of web page), if not go back to step 6 and try again.

Exercise 2: Create new list for tracking profiles for consulting projects

1. Over the next few steps, you will create and modify a new SharePoint custom list for managing Litware consulting projects.
    a. Start by clicking the **Create** command from the **Site Actions** menu to navigate to the **Create** Page.

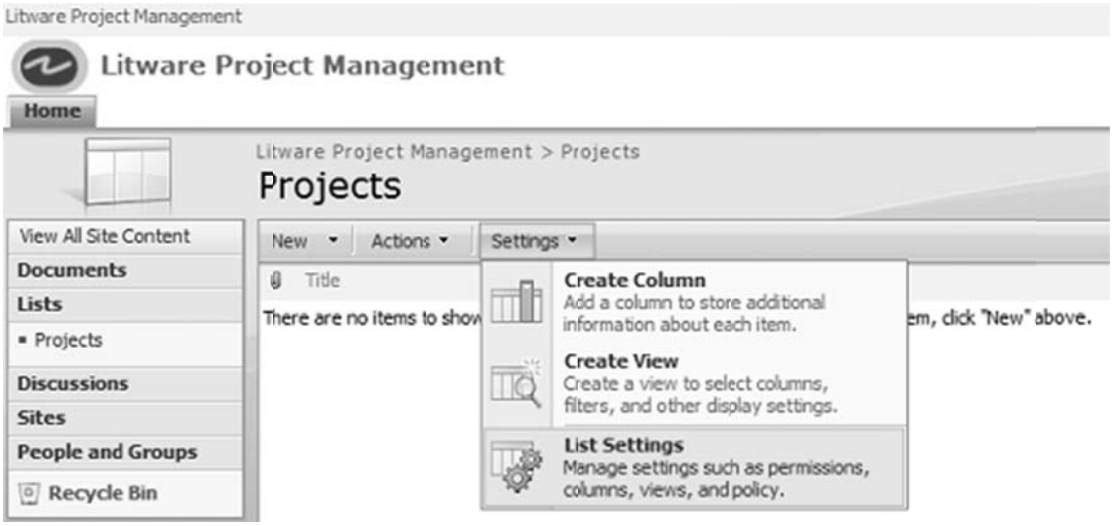


    b. Create a new list by clicking the **Custom List l**ink in the **Custom Lists** section. This will take you to a page that allows you to name the new list.

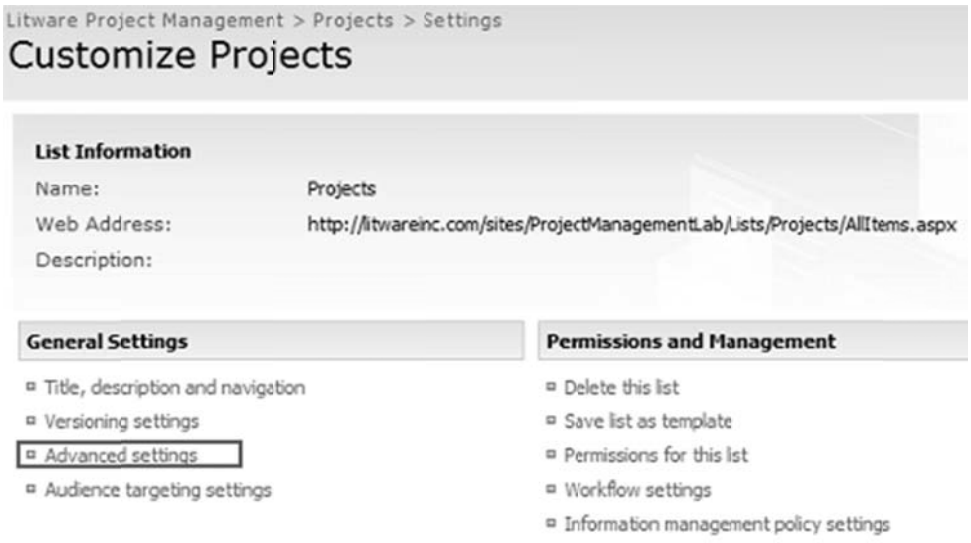


c. Name the new list **Projects** and click **Create**. This brings you to the **All Items** view of the new list.

2. Next you are going to modify the settings for our list to disable the uploading of attachments to the **Projects** list.
    a. Once the **Projects** list has been created, locate and drop down the **Settings** menu on the list's **AllItems.aspx page** (note: **NOT** the one on the Site Actions menu).



    b. Click the **List Settings** command. This will take you to the list settings page that has a title of **Customize Projects** list.
    c. From this page, click on the **Advanced Settings** link in the **General Settings** section. Take note of the different modifications you can make to the list from the **Advanced Settings** page.

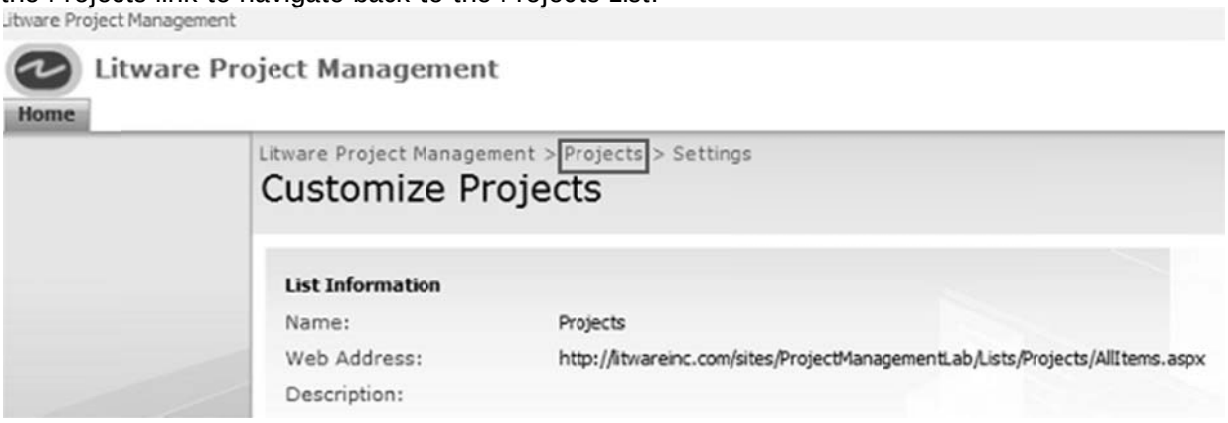


    d. In the **Attachments** section, change the default option to **Disabled** and click **OK** to return to the list setting page. In this case, there is no need for the list to support attaching documents to items within the **Projects** list. If you get a warning message about erasing attachments, note what it says and then click **OK.**

3. From the **List Settings** page, you can add and modify columns for the **Project** list. In this step you will modify the **Projects** list so its set of columns matches the set of columns defined below. (For detailed directions see steps a - d below)

Column Name	Type	Notes
Project	Single Line of Text	**Do not create this column as a new column.** Instead, rename the **Title** column to **Project**.
Client	Single Line of Text	Make this a required column
Contract Signed	Yes/No	Set default value to No
Contract Amount	Currency	Set currency formatting to whatever seems most appropriate
Begin Date	Date and Time	Format this column to show date only
End Date	Date and Time	Format this column to show date only
Created By	Person or Group	**You do not have to create this column**. It is automatically created when you create a new list
Modified By	Person or Group	**You do not have to create this column.** It is automatically created when you create a new list

   a. From the **Projects List** click on the **Settings** drop down and click on **List Settings** (Note: we are already on this screen from step 2).
   b. Scroll down to the **Columns section** and click on the **Title** Column to edit it.
   c. Change the **Column name** to **Project** and click **OK**
   d. From the **Columns section** click **Create Column** and using the chart above configure the following settings:
      i. Type the **Column Name**
      ii. Select the **data type** for the column
      iii. Examine the **Notes field from the above table** for extra configuration options for each column
      iv. Configure extra options as needed and then click **OK**
      v. Repeat until all columns are created and then from the **Customize Projects Menu** underneath the **Columns section** click on **Column ordering**
      vi. Organize the order of the columns so that they match the order in the list above. Note: there may be nothing to do here other than verify the order depending on how you created your columns.
   e. Using the breadcrumbs navigational element (i.e. the one that is near the top of the screen that looks like **Litware Project Management > Projects > Settings**) click on the Projects link to navigate back to the Projects List.



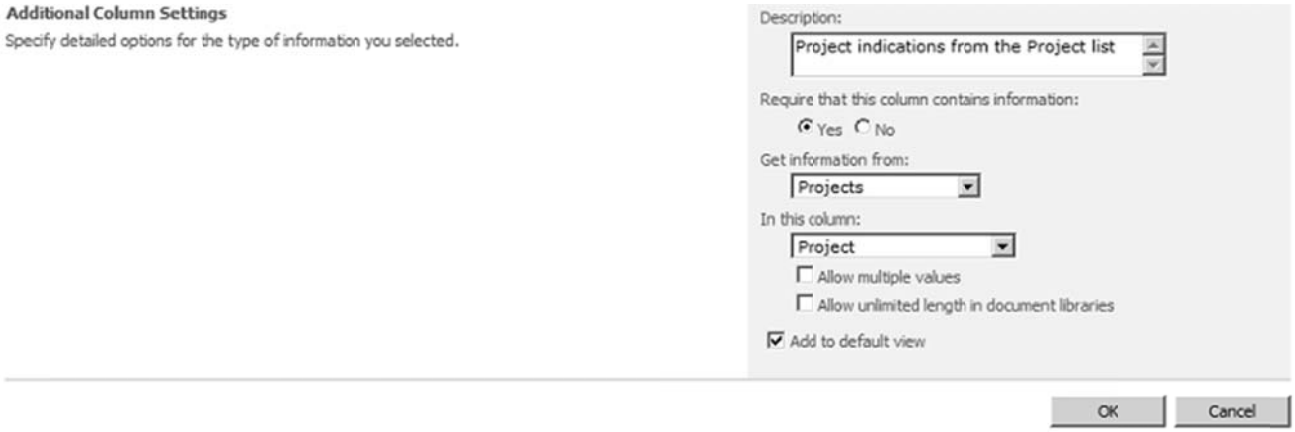
4. When you are done creating the **Projects** list structure, add the following four items so that you have some test data to work with.
    a. From the **Projects** List click on the **New** drop down menu and using the table below create 4 records.

Project	Client	Contract Signed	Contract Amount	Begin Date	End Date
Wing001	Wingtip Toys, Inc	Yes	$250,000	1/1/2005	4/15/2006
AdWks001	Adventure Works	No	$120,000	1/15/2006	6/1/2006
NW001	Northwind Traders	Yes	$1,200,000	4/1/2006	6/1/2007
Cont001	Contoso	No	$75,000	6/1/2005	9/1/2006

5. When finished navigate back to the **Litware Project Management** Home page (i.e. click on the **Home tab** on the left side near the top)

Exercise 3: Creating a new document library for project-related documents

1. Over the next few steps, you will create and modify a new SharePoint document library for storing and managing documents associated with Litware consulting projects.
    a. Start by clicking the **Create** command from the **Site Actions** menu to navigate to the **Create** Page.
    b. Create a new document library by clicking the **Document Library** link in the **Libraries** section. This will take you to a page that allows you to name the new document library and click **Create** to create this document library.
2. In this step, you will add a new column to associate custom metadata with each document that is added to the document library.
    a. Locate and drop down the **Settings** menu on the document library's **AllItems.aspx** page and click the **Document Library Settings** command. This will take you to the document library settings page that has a title of **Customize Project Documents**.
    b. Add a new lookup column to the document library named **Project**.
        i. Underneath the columns section of this page click **Create column**.
        ii. We will use the **Project** column of the **Projects** list you created in the previous exercise as the source for this new lookup Column.
            1. Type Project for the **Column name:**
            2. Choose **Lookup (information already on this site)** for the **type** selection (i.e. the option button list)
            3. Type **Project identifications taken from the Project List** for the **Description.**
            4. Make sure to select the option so that the **Project** column is required.
            5. Be sure to choose **Projects** for the **Get information from:** drop down list choice.
            6. Choose **Project** for the **In this column:** drop down list choice.
            7. Click **OK** to add the new column.



8. Using the breadcrumbs navigation bar on this page (i.e. **Litware Project Management > Project Documents > Settings**) navigate back to the **Project Documents** library (i.e. click on the **Project Documents** link)

3. Now that you have created the **Project Documents** document library and added the lookup column Project, it's time to upload a few documents to test it out.
    a. Using the Windows Explorer, look inside the lab directory and locate the subdirectory named **LitwareDocuments**. It contains several .DOCX files and PPTX files that you can upload to your document library.
    b. Upload each document inside this directory. You should observe that whenever you upload a document into the **Project Documents** document library, WSS brings up a page to prompt you to assign a value to the **Project** column. You should observe that the **Projects** column provides extra metadata for each document and that WSS forces users to associate each and every document inside this document library with a specific project. **Important: You MUST upload each document SEPARATELY if you use the Upload Multiple Documents choice you will bypass the metadata screen altogether for each document.**
        i. On the **Project Documents Library** Page select the **Upload** Drop Down arrow and then click **Upload Document** or just click on **Upload**
        ii. Use the **Browse** button to navigate to **...\Student\Labs\01_Customization\LitwareDocuments**
        iii. You are going to repeat the next set of steps one time for each of the 4 items in this directory (3 PowerPoint and 1 Word document(s))
            1. **Select** the next **item** (From the top 3 PowerPoint and 1 Word document(s)) on the **Choose file** screen and click **Open**
            2. On the **Upload Document: Project Documents** screen click **OK.**
            3. The next screen informs you that the document is currently checked out to you; you need to fill in the requisite metadata and check the document into the library to finish this process.
                a. In the **Project** drop down box pick the appropriate project for this document matching up the Document Name field to the Project. (use the table below for reference as necessary.

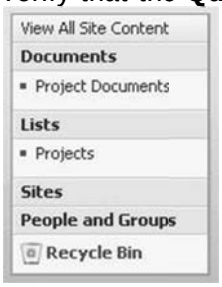
Project	Client
Wing001	Wingtip Toys, Inc
AdWks001	Adventure Works

NW001	**Northwind Traders**
Cont001	**Contoso**

4. Click the **Check In** button to finish the upload process.
5. Repeat step 3.b. once for each of the 4 documents.
6. When finished uploading the four documents navigate back to the Litware Project Management Home page (i.e. click on the **Home tab** near the top left corner of the screen).

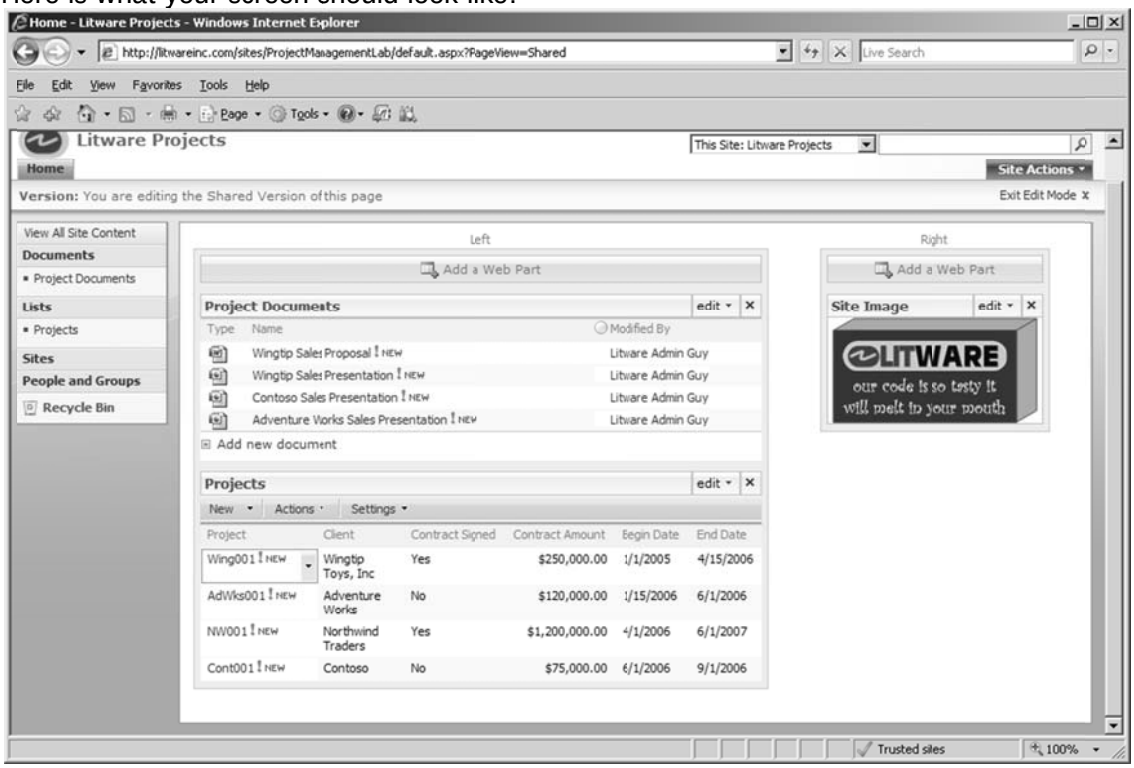
Exercise 4: Customizing the home page

1. In this exercise you will customize the appearance of the **Quick Launch** navigation control on the left-hand side of the home page.
   a. Start by clicking the **Site Settings** command from the **Site Actions** menu to navigate to the **Site Setting Page**.
   b. Next, click on the **Quick Launch** link under the **Look and Feel** section. Once you get to the **Quick Launch** page, you will be able to add, edit and delete links and headings from the **Quick Launch** control. In this case, you want to remove the heading for **Discussions** .
      i. On the **Quick Launch page** click on the **Edit Discussions** icon  (i.e. the one to the left of the **Discussions** section)
      ii. On the **Edit Heading page** click the **Delete** button
      iii. Click **OK**
      iv. Navigate back to the Litware Project Management home page (i.e. click on the **Home tab** near the upper left corner of the page)
   c. Verify that the **Quick Launch** control now looks like this:



2. Next, you will add two Web Parts to the home page to display the **Projects** list and the **Project Documents** document library.
   a. If necessary, Navigate to the home page of the site
   b. From the site Home page, place the page into edit mode by choosing the **Edit Page** command under the **Site Actions** menu.
   c. Once the page is in edit mode, you should click the **Add a web part** link at the top of the left **Web Part Zone**.
   d. Select to add a Web Part for both the **Projects** list and the **Project Documents** document library.
   e. Click **Add**.

f. Here is what your screen should look like:

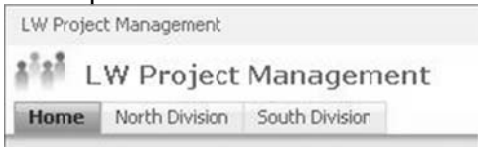


g. When finished be sure to click the **Exit Edit Mode** hyperlink (near the upper right corner of the page).

3. Finally we will create a Portal Navigation Point to link our new Site Collection back to the main portal page.
   a. Click on **Site Actions > Site Settings**
   b. In the **Site Collection Administration** column select the **Portal site connection** choice.
   c. On the Portal Site Connection page:
      i. Select **Connect to portal Site**
      ii. enter http://litwareinc.com as your Portal Web Address:
      iii. enter **Litware Inc.** for the Portal Name:
      iv. click **OK**

Exercise 5: Creating child sites

1. Now you are going to create two new child sites to track Litware consultants by corporate division. One site will be for the **Litware North Division** and the other will be for the **Litware South Division**. However, keep in mind when you create these child sites you want to make it easy for users to navigate between them. Therefore, you want to create a resulting top link bar on the parent site and both child sites to look like this.



2. In this step, you will create a child site named **North Division**.
   a. Start by clicking the **Create** command from the **Site Actions** menu to navigate to the **Create** page.

   b. Create a child site by clicking the **Sites and Workspaces** link in the **Web Pages** section. This will take you to a page that allows you to create a new child site.
   c. Name the site **North Division**
   d. Make it's URL **NorthDivision** (no space in the name)
   e. choose **Blank Site** as the site template
   f. In the **Navigation** section of the New SharePoint Site page, choose **Yes** as the option to **Display this site on the Quick Launch of the parent site** and **Yes** for the option to **Display this site on the top link bar of the parent site**. (i.e. just accept the defaults)
   g. In the **Navigation Inheritance** section of the New SharePoint Site page, choose **Yes** as the option to **Use the top link bar from the parent site.** (i.e. just accept the defaults)
   h. Finally, click **Create** to create the new child site.
3. In this step, you will modify the Site Image Web Part in the **North Division** site so that it displays a Litware North graphic instead of the default Microsoft graphic.
   a. Navigate to the home page of the **North Division** site and place the page into edit mode by choosing the **Edit Page** command under the **Site Actions** menu.
   b. Once the page is in edit mode, find the **Site Image Web Part** (Right Zone) and select the **Modify Shared Web Part** command from the Web Part's **edit** menu.
   c. After you have run this command, you should see a task pane appear in the browser that allows you to modify the **Image Link** property. Assign the **Image Link** property a new value of **/_layouts/images/LitwareGraphics/LitwareNorth.png**
   d. Click **OK**.
   e. Confirm that the child site's home page now shows the Litware North graphic.
   f. Click **Exit Edit Mode** just underneath the **Site Actions** menu.
4. Now you should be able to navigate back and forth between the top-level **Project Management** site and the child **North Division** child site using the top link bar.
5. Now click the **home tab** to return to the top-level site and create a second child site named **South Division**.
   a. Create this new site using the same set of instructions (Exercise 5, Step 2) as you used to create the **North Division** site (note: replace North with South as appropriate).
   b. Also, modify the Site Image Web Part in the South Division site so that it displays the graphic file at **/_layouts/images/LitwareGraphics/LitwareSouth.png** (i.e. follow the instructions in Exercise 5, Step 3 if you need help)
6. At this point, you should be able to quickly navigate between the top-level site and either child site with a single click on the top link bar. This is all the work you will do within the child sites at this time. **Note: You will return to these child sites in a later lab and add some custom lists to track information about Litware consultants.**

Lab 02: Developing Your First Feature

Lab Time: 45 Minutes

Lab Directory: C:/Student/Labs/02_Features

Lab Overview: You are going to start this lab by creating a new Site Collection using the Windows SharePoint Services (WSS) Central Administration Web site. If you are new to using WSS, you should feel free to spend a little time getting familiar with how users create new lists and document libraries. However, the primary goal of this lab is for you to create and test your first WSS Feature.

1. You will create a new Site Collection for Lab02 and optionally add some task lists / document libraries to familiarize ourselves with basic processes in WSS administration.
2. Next you will create a Visual Studio project from scratch and add all the key ingredients to create the classic 'Hello World' Feature (i.e. you will add a menu item on the Site Actions Menu called Hello World that will hyperlink to our Litware Corporate Portal site).
3. Then you will go through the required steps to deploy and test it.
4. Next you will add an event handler that fires and executes code against the WSS object model each time the Feature is activated within a site (i.e. we will add an event handler to additionally alter the site title to Hello World when activated and then return the site title back to the original setting when deactivated).
5. Along the way, you will go through the required steps to make our assembly strongly named and also to deploy it into the Global Assembly Cache (GAC).

Setup: enabling intellisense support for SharePoint CAML code in Visual Studio

1. In order to enable Intellisense support for SharePoint CAML documents in Visual Studio you need to add the sharepoint_catalog.xml file to the Visual Studio Directory "C:\Program Files\Microsoft Visual Studio 9.0\Xml\Schemas\".
    a. Navigate to your **Student\Labs\02_Features\CAML_IntelliSense**\ folder
    b. Double click on the **install.bat** file that you find there.
    c. Run or Close and re-Open Visual Studio.
    d. You should now have IntelliSense when you program SharePoint CAML files in Visual Studio.

For Example: After you add an xml file named “elements.xml” to your application in Visual Studio:

- **Open** the file by double clicking on it in the Solution Explorer
- In the **XML Document Properties** window set the **Schema** (by clicking on the ellipsis “…”) and select the wss.xsd schema and click **OK**.
- You should now see IntelliSense when you click back into the open document and type a “<” character. You should see **Elements, Feature,** and **List**, among other choices.

Exercise 1: Provisioning a new Site Collection

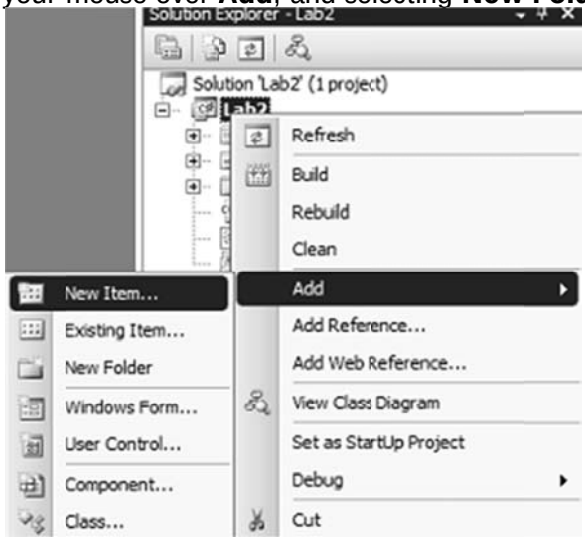
1. Make sure you are logged on as **LITWAREINC\Administrator**. Launch the WSS Central Administration Web site by using the command in the Windows Start menu. You can find the command at **Start >> Administrative Tools >> SharePoint 3.0 Central Administration**.
2. Once the home page of the WSS Central Administration Web site appears, click on the **Application Management** tab at the top of the page to get to the Application Management page. Under the **SharePoint Site Management** section, click the link with the caption **Create site collection**. This will take you to the page where you can create a new site collection and a new top-level site. On the **Create Site Collection** page, fill in the required information (see below for instructions) to create a new site collection.
    a. Make sure the Web Application used for site creation is the one named Public Internet Site that is accessible through the URL **http://litwareinc.com**.
    b. Enter **Lab 2** as the title.
    c. Create the new site collection so that its URL is **http://litwareinc.com/sites/Lab2**.
    d. Under the **Template Selection** section, look at all the sites templates that are available. Choose **Blank Site** from the Collaboration tab as the site template for the new top-level site that is automatically created.
    e. Assign the primary site collection owner as **LITWAREINC\Administrator**. Be sure to verify this by using **Ctrl+k** or by clicking the check person icon  (your entry should become LitwareInc Administrator)
    f. Leave the **Quota Template** with the default setting of **No Quota**
    g. Click **OK** to create the new site collection and top-level site. Once you see the page that confirms everything has been created, navigate to the top-level site using the browser by clicking the link http://litwareinc.com/sites/Lab2 .
3. Now take the time to familiarize yourself with the site you just created and make a few administrative changes through links off the **Site Settings** page. The title you entered of Lab 2 is just a little too boring.
    a. Choose the **Site Settings** command under the **Site Actions** menu.
    b. Once you are at the **Site Settings** page, click the link with the caption **Title, description and icon** under the **Look and Feel** section.
    c. When you reach the Title, Description, and Icon page, enter a different title that is a variation such as **Lab 2 Test Site**. Click **OK** and confirm that the home page now reflects the new site title.
4. If you are new to SharePoint and haven't used the product much as a user, take a few minutes and create a few new lists. (If you are already comfortable with creating and working with lists, you should move on to the next exercise).
    a. In particular, navigate to the **Create** page (available from the **Site Actions** menu) and create a new list from the **Contacts** list type and then add one or two contacts using whatever test data you'd like.
    b. After creating the **Contacts** list, create a new document library. Click on the **New** button and create and save a new Word document back to the document library.

Exercise 2: Creating the 'Hello World' Feature

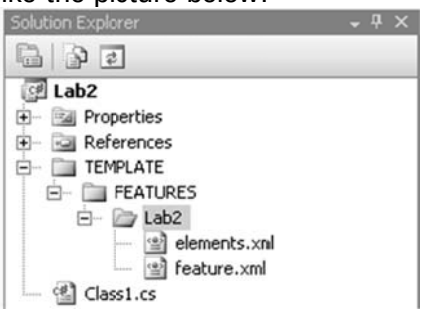
1. In this exercise you will create a new Visual Studio project to develop a very simple Feature. Start by opening Visual Studio 2008 and creating a new Class Library DLL project named Lab2 by going to **File > New > Project**, then choosing your preferred language in the left column of

the box that opens up, and then selecting “Class Library” in the adjacent window. The screen shots for this lab were designed using C#, You may use either C# or Visual Basic .Net as there are solution files for the lab in both languages. While you can create this new project any place you would like, you might prefer to create it at the following path so the project files will reside inside the same directory structure as all the other lab exercises.
C:\Student\Labs\02_Features\Lab

2. Within your new Visual Studio project, add a folder to the root directory of the current project named **TEMPLATE** by right clicking on “Lab 2” in the **Solution Explorer** window, hovering your mouse over **Add**, and selecting **New Folder**.



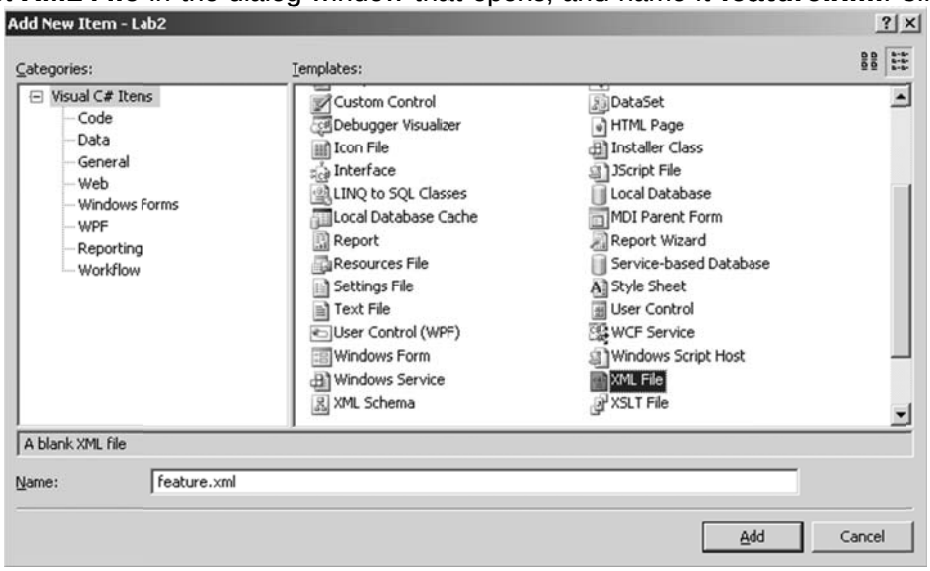
   a. Once you have created the **TEMPLATE** directory, create another directory inside that, named **FEATURES**.
   b. Finally, create another directory inside the **Features** directory using the same name as the name of the Feature project which in this case is **Lab2.**
   c. When you are done creating these folders, your project should have a folder structure like the picture below:



   Note this picture shows what your project should look like after finishing steps 3 and 4 and if you chose the C# language.

3. Create an XML file inside the **/TEMPLATES/FEATURES/Lab2** directory by right clicking on the **Lab2** folder under the **Features** folder, hovering your mouse over **Add**, and then selecting

New Item. Select **XML File** in the dialog window that opens, and name it **feature.xml**. Click the **Add** button.



Inside the new file is where you will add the XML-based information that defines the high-level attributes of the Feature itself.

a. **Open** the file by double clicking on it in the **Solution Explorer**.

b. In the **XML Document Properties** window set the **Schema** (by clicking on the ellipsis "...") and select the wss.xsd schema and click **OK**.

c. You should now see IntelliSense when you click back into the open document and type a "<" character.

d. Add the following XML content to the feature.xml file to define a top-level Feature element along with attributes that define the Feature itself.

```
<Feature Id="B2CB42E2-4F0A-4380-AABA-1EF9CD526F20"
  Title="Lab 2 - Getting Started with WSS Development"
  Description="This is my very first custom feature"
  Scope="Web"
  Hidden="FALSE"
  ImageUrl="menuprofile.gif"
  xmlns="http://schemas.microsoft.com/sharepoint/">
  <ElementManifests>
    <ElementManifest Location="elements.xml" />
  </ElementManifests>

</Feature>
```

4. Next, create the **elements.xml** file in the same directory.

a. **Open** the file by double clicking on it in the Solution Explorer

b. In the **XML Document Properties** window set the **Schema** (by clicking on the ellipsis "...") and select the wss.xsd schema and click **OK**.

c. You should now see IntelliSense when you click back into the open document and type a "<" character.

d. Add the following XML which defines a **CustomAction** element which will allow the user to navigate to another site that has already been defined within the current VPC image:


```
<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 to navigate to another site"
    ImageUrl="_layouts/images/crtsite.gif" >

      <UrlAction Url="http://litwareinc.com/sites/IntranetPortal"/>

  </CustomAction>

</Elements>
```

5. Now that you have implemented the Lab2 Feature by creating the **feature.xml** file and the **elements.xml**, it's time to deploy it. You will do this by creating a new batch file and configuring the current Visual Studio project to run this batch file each time you rebuild the current project.
    a. Start by creating a new batch file in the root directory of your project named **Install.bat**.
        i. You can create a batch file by navigating to your **Solution Explorer** in Visual Studio and right click on your project **Lab2** (i.e. the top item in the solution explorer picture from step 2)
        ii. Click on **Add** > **New Item...**
        iii. Choose **Text File,** name the file **Install.bat** and then click **Add.**
    b. Add the following instructions to **Install.bat**.

```
@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 SDKs\Windows\v6.0A\bin\gacutil.exe"

Echo Installing Lab2.dll in GAC
REM %GACUTIL% -if bin\debug\Lab2.dll

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

Echo Installing Feature
%STSADM% -o installfeature -filename Lab2\feature.xml -force

IISRESET
```

Note that the call to %GACUTIL% is commented out with an REM statement. You will uncomment this instruction in the next exercise after you have configured the Lab2 project to build its output assembly with a strong name.

6. Now that you have created the **Install.bat** file, it's time to configure the Visual Studio project to run it each time you rebuild.
    a. Go to the **Project Properties** for the Lab2 project by right clicking the project in **Solution Explorer** and choosing **Properties** from the menu. If using VB, follow the instructions in step B. If using C# go to step C.
    b. **Visual Basic .Net directions**:
        i. Navigate to the **Compile** tab
        ii. Click on the **Build Events...** button (scroll down if necessary to see this button)

iii. Add the following Post-build event command line instructions.

```
cd $(ProjectDir)
Install.bat
```

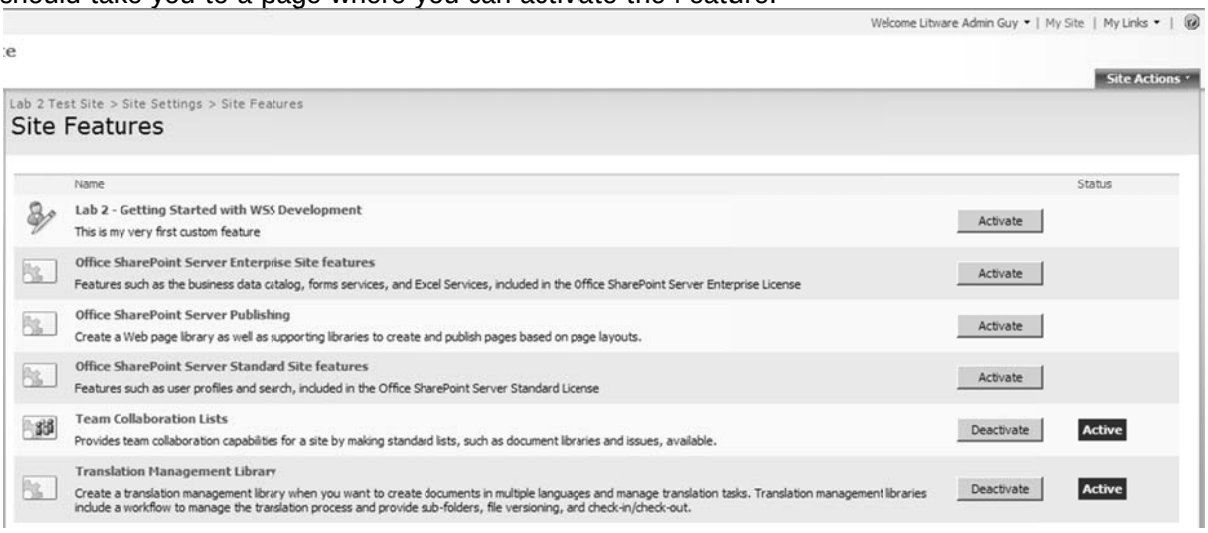
Note that the first line with **cd $(ProjectDir)** is required to change the current directory to that of the project directory. The second line runs the batch file **Install.bat** to copy the Feature files to the correct location and install the Feature with the **InstallFeature** operation of the command-line **stsadm.exe** utility.

c. **C# directions:**
   i. Navigate to the **Build Events** tab.
   ii. Add the following Post-build event command line instructions.

```
cd $(ProjectDir)
Install.bat
```

Note that the first line with **cd $(ProjectDir)** is required to change the current directory to that of the project directory. The second line runs the batch file **Install.bat** to copy the Feature files to the correct location and install the Feature with the **InstallFeature** operation of the command-line stsadm.exe utility.

7. Build your project (i.e. **Build** drop down menu > **Build Lab2**) (this will build and install your feature for use)
8. Once the Feature has been properly installed, you should now be able to activate it within context of a site.
   a. Go to the top-level site of the site collection you created in the previous exercise (http://litwareinc.com/sites/Lab2)
   b. Navigate to the **Site Settings** page through the **Site Actions** menu.
   c. Under the **Site Administration** section, click the link with the title **Site Features**. This should take you to a page where you can activate the Feature.



   d. Activate the **Feature** named **Lab 2 - Getting Started with WSS Development** by clicking the **Activate** button next to the Lab 2 feature.
   e. Make sure your new menu item has been added to the **Site Actions** menu.

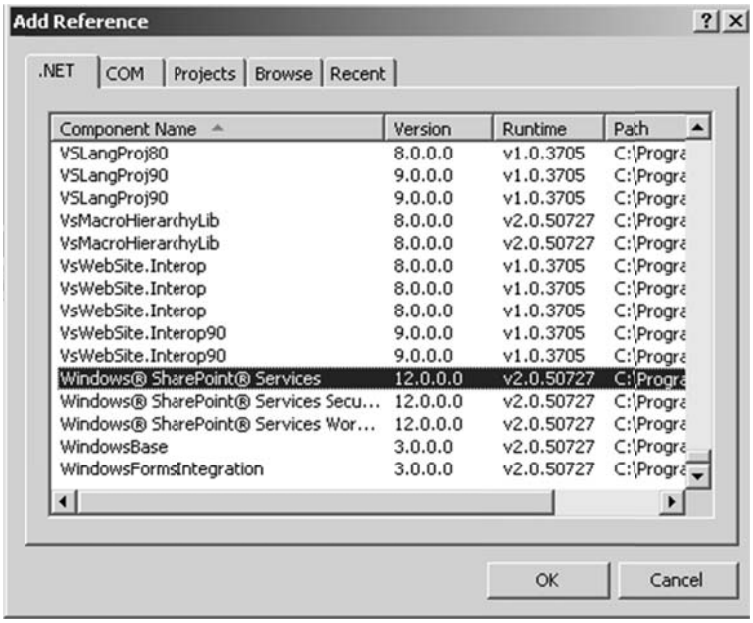
9. Now that you have successfully activated the Feature and tested the custom **Site Actions** menu command, you will learn how to deactivate a feature.
    a. Return to the Site Features page (**Site Actions > Site Settings > Site Features** (underneath the **Site Administration column**)
    b. Deactivate the Feature by clicking the **Deactivate** button next to the Lab2 feature.
    c. Now that you have deactivated the Lab2 Feature, you should be able to verify that the custom Site Actions menu command has been removed from the **Site Actions** drop down.

You have now witnessed the fundamental principle behind Features. Developers create various types of site elements that can be added or removed from a site through the process of activation and deactivation.

Exercise 3: Adding an event handler to your Feature

1. Now it's time to take the example of the Lab2 Feature a little further by adding an event handler with code that programs against the WSS object model.
    a. First, start by adding a project reference to **Microsoft.SharePoint.dll** which is listed as **Windows SharePoint® Services** in Visual Studio's **Add Reference** dialog.
        i. In **Visual Studio** go to the **Solution Explorer** and right click on your project **Lab2** and choose **Add Reference...**
        ii. From the **Add Reference** dialog box go to the **.Net** tab, scroll down the list of assemblies and select the **Windows SharePoint® Services** component.

iii. Click the **OK** button.



2. Now we need to set our **FeatureReceiver** class up to inject a new site title via the object model on Feature Activation and Reset this title to the original on Feature Deactivation. A feature receiver class always inherits from **SPFeatureReceiver**.
   a. **If Using VB.NET**
      i. Next, locate the source file named **Class1.vb** and rename it to **FeatureReceiver.vb**. Next, replace the content of **FeatureReceiver.vb** with the following code.

```
Imports System
Imports Microsoft.SharePoint
Public Class FeatureReceiver
       Inherits SPFeatureReceiver

    Public Overrides Sub FeatureActivated(ByVal properties As
SPFeatureReceiverProperties)
        Dim site As SPWeb = properties.Feature.Parent
        ' track original site Title using SPWeb property bag
        site.Properties("OriginalTitle") = site.Title
        site.Properties.Update()
        ' update site title
        site.Title = "Hello World"
        site.Update()
    End Sub

    Public Overrides Sub FeatureDeactivating(ByVal properties As
SPFeatureReceiverProperties)
        ' reset site Title back to its original value
        Dim site As SPWeb = properties.Feature.Parent
        site.Title = site.Properties("OriginalTitle")
        site.Update()
    End Sub

    Public Overrides Sub FeatureInstalled(ByVal properties As
SPFeatureReceiverProperties)
    End Sub
```



```
    Public Overrides Sub FeatureUninstalling(ByVal properties As
SPFeatureReceiverProperties)
    End Sub

End Class
```

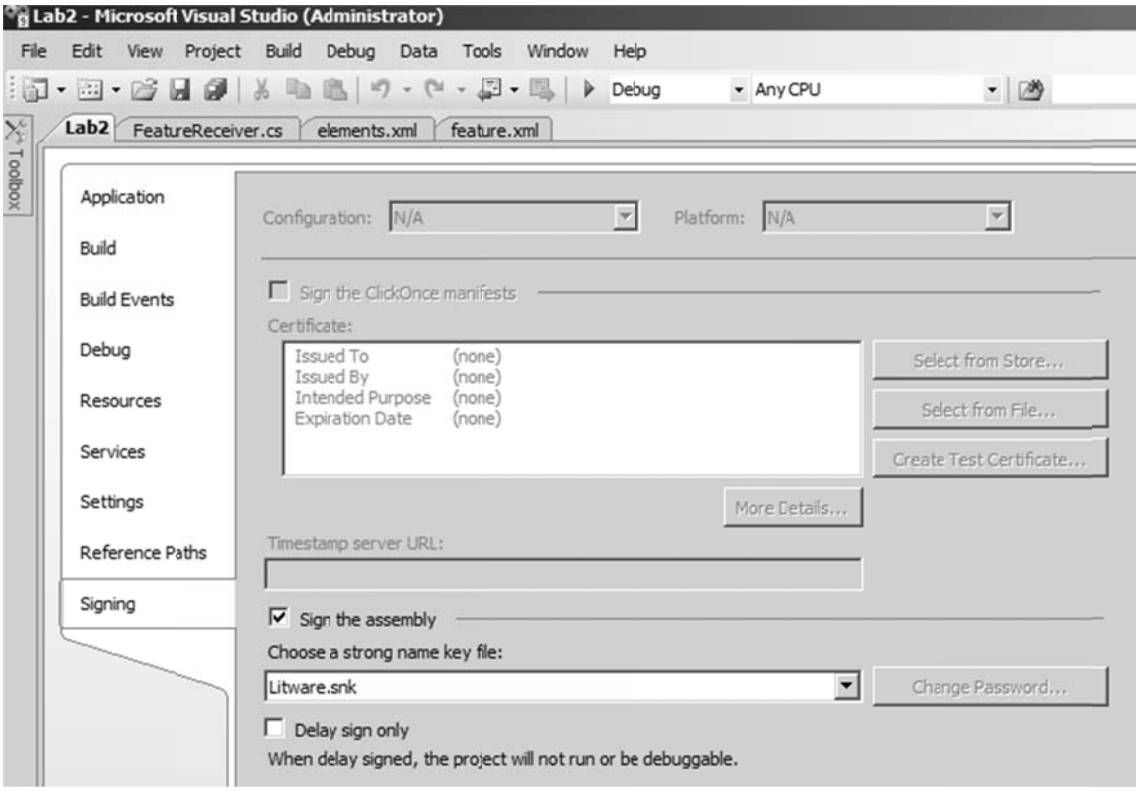
b. **If Using C#**

i. Next, locate the source file named **Class1.cs** and rename it to **FeatureReceiver.cs**. Next, replace the content of **FeatureReceiver.cs** with the following code.

```
using System;
using Microsoft.SharePoint;

namespace Lab2 {
  public class FeatureReceiver : SPFeatureReceiver {

    public override void FeatureActivated(SPFeatureReceiverProperties
properties){
      SPWeb site = properties.Feature.Parent as SPWeb;
      //track original site title using spweb property bag
      site.Properties["OriginalTitle"] = site.Title;
      site.Properties.Update();
      // update site title
      site.Title = "Hello World";
      site.Update();
    }

    public override void FeatureDeactivating(SPFeatureReceiverProperties
properties) {
      // reset site Title back to its original value
      SPWeb site = properties.Feature.Parent as SPWeb;
      site.Title = site.Properties["OriginalTitle"];
      site.Update();
    }

    public override void FeatureInstalled(SPFeatureReceiverProperties
properties){}
    public override void FeatureUninstalling(SPFeatureReceiverProperties
properties) {}

  }
}
```

3. The next step is to update the **Lab2** project so that it's output assembly **Lab2.dll** is built with a strong name. This is a requirement for assemblies that contain Feature receiver classes which must be installed in the GAC in order to work properly.
   a. Copy the **Litware.snk** file from the **Resources** directory into your project directory
      i. In **Visual Studio** navigate to the **Solution Explorer** and rt click on your project **Lab2.** Select **Add** > **Existing Item...**
      ii. In the **Objects of type**: drop down select **All Files (*.*)**
      iii. In the **Look in:** drop down navigate to **\Student\Labs\02_Features\Resources** and select the **Litware.snk** file and click **Add**
   b. Configure the project properties of the **Lab2** project to build its output assembly **Lab2.dll** using this key file.

i. In **Visual Studio** navigate to the **Solution Explorer** and rt click on your project **Lab2** select **Properties**
ii. Navigate to the **Signing** tab
iii. Put a check in the **Sign the assembly** checkbox
iv. From the **Choose a strong name key file:** dropdown list select your **Litware.snk** file.
v. Close the **Project Properties window.**



vi. **re-Build** your project to create a dll with this new strong name.

4. Now that you have implemented the Feature receiver class, the next step is to update the feature.xml file with two new attributes so that WSS knows that there is at least one event handler that should be fired whenever the Feature is activated or deactivated.
   a. In your **Solution Explorer** open the **Feature.xml** file
   b. Add the **ReceiverAssembly** attribute and the **ReceiverClass** attribute to the existing **Feature.xml** file as shown here.
   Note: you can obtain the full 4-part assembly name (for use in the **ReceiverAssembly** Attribute) using the **Reflector** utility that is in your **\Student\Resources** directory.
   Further Note: In order to make this work you must do this **AFTER** you have added the keyfile **Litware.snk** to the **Project Properties** and then re-**build** your project (so that your assembly (dll) will contain the strong name attribute (i.e. **PublicKeyToken**)) but before you add the attributes to the **Elements.xml** file.
   c. In the **...\Student\Resources\Reflector.exe** Utility you will need to examine the **Lab2.dll**.
   File > Open... (VB.NET or C#) Navigate to the **...Lab2\bin\Debug directory** open the **Lab2.dll** file.

d. Select the **Lab2** Assembly in the main Reflector utility window and look at the bottom for the **Name:** entry. You can copy this entry to use for your **ReceiverAssembly** attribute.
e. Your **ReceiverClass** attribute is made up of the **namespace**.**className** **(Note**: these are typically the same as your **ProjectName**.**ClassFileName** if no alterations have been made to the Visual Studio Default choices.)

```
<Feature
  Id="B2CB42E2-4F0A-4380-AABA-1EF9CD526F20"
  Title="Lab 1 - Getting Started with WSS Development"
  Description="This is my very first custom feature"
  Version="1.0.0.0"
  Scope="Web"
  Hidden="FALSE"
  ImageUrl="menuprofile.gif"

ReceiverAssembly="Lab2,Version=1.0.0.0,Culture=neutral,PublicKeyToken=b
38a04419cc857d9"
  ReceiverClass="Lab2.FeatureReceiver"
  xmlns="http://schemas.microsoft.com/sharepoint/" >

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

</Feature>
```

5. Next you need to update your **Install.bat** in order to add the strong named assembly into the GAC
   a. In **Visual Studio** navigate to **Solution Explorer** and open the **Install.bat** file.
   b. Remove just the **REM** keyword in the **REM %GACUTIL%**… line. This will cause the Strongly Named dll to be injected into the GAC.
6. Once you have made these changes to the **feature.xml** and **install.bat** files, you should be able to test your work.
   a. **Rebuild** the Lab2 project so that Visual Studio runs the **Install.bat** file to copy the updated version of **feature.xml** file to the WSS Features directory and to reinstall the updated version of the Feature with WSS. The build process should also compile **Lab2.dll** with a strong name and install it in the GAC.
   Note: you would **normally** be required to run an **IISRESET** command any time you recompile to Lab2.dll to restart the IIS worker process. This is due to the fact that Features and assemblies loaded from the GAC are cached by WSS within the IIS worker process. **HOWEVER** this is done for us automatically as the last line of our **Install.bat** file that we created earlier.
   b. Test your work by activating and deactivating the Feature. You should be able to see the site title changing back and forth each time you activate or deactivate the Lab2 Feature.

Lab 03: Developing Custom Application Pages

Lab Time: 45 Minutes

Lab Directory: C:/Student/Labs/03_Architecture

IMPORTANT NOTE: Because we are building this project piecemeal and testing it as we go, if after you build your project 1 time, the next change and build causes an error when you try to test your addition (i.e. with either SiteInfo.aspx or DocumentInfo.aspx) **YOU WILL NEED TO RE-BUILD YOUR PROJECT AGAIN IN ORDER TO TEST IT OUT**. During the second build your old files were removed but not replaced correctly so an additional re-build of your project will correctly re-deploy these pages. (This tends to occur during Exercise 3 in this lab).

More Info on Important Note: This issue tends to be much more prevalent when working with VB.NET as your source language. As service packs for Visual Studio and .Net are released this issue may become largely mitigated.

Lab Overview: In this lab, you will work with a Visual Studio project to write, deploy and test custom application pages that run out of the virtual **_layouts** directory. This will demonstrate a few ways to use custom application pages and also give you a sense for what it's like to develop them for a WSS solution. At the end of this lab, you will also work with batch files to take several WSS components and compile them into a solution package so that they can be deployed in staging and production environments.

Exercise 1: Provisioning a new Site Collection for testing

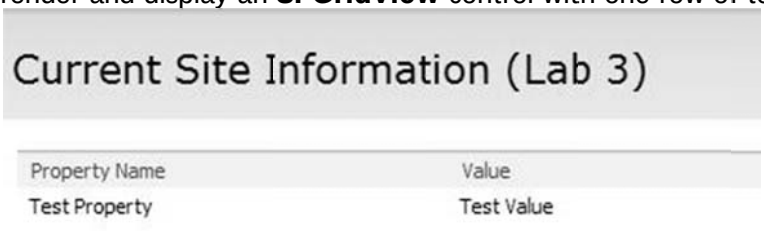
1. Make sure you are logged on as **LITWAREINC\Administrator**. Launch the **WSS Central Administration Web** site by using the command in the Windows Start menu. You can find the command at **Start >> Administrative Tools >> SharePoint 3.0 Central Administration**.
2. Once the home page of the WSS Central Administration Web site appears, click on the **Application Management** tab at the top of the page to get to the Application Management page. Under the **SharePoint Site Management** section, click the link with the caption **Create site collection**. This will take you to the page where you can create a new site collection and a new top-level site. On the **Create Site Collection** page, fill in the required information (see below for instructions) to create a new site collection.
    a. Make sure the Web Application used for site creation is the one named **Litware Public Site** that is accessible through the URL **http://litwareinc.com**.
    b. Enter a title such as **Lab 3 Test Site**.
    c. Create the new site collection so that its URL is **http://litwareinc.com/sites/Lab3**.
    d. Under the **Template Selection** section, look at all the sites templates that are available. Choose **Blank Site** from the **Collaboration** tab as the site template for the new top-level site that is automatically created.
    e. Assign the primary site collection owner as **LITWAREINC\Administrator**. Be sure to verify this by using Ctrl+ k or by clicking the check person icon  (your entry should become LitwareInc Administrator)
    f. Leave the **Quota Template** with the default setting of **No Quota**.

   g. Click **OK** to create the new site collection and top-level site. Once you see that page that confirms that everything has been created, navigate to the top-level site using the browser (click the link http://litwareinc.com/sites/Lab3.)
3. Once you have created and navigated to this new site collection, you can go on to the next exercise.

Exercise 2: Working with a Custom Application Page

1. Launch Visual Studio.
2. Open the project named **Lab3.sln** located inside the **\Student\Labs\03_Architecture\Lab\VB** or **\Student\Labs\03_Architecture\Lab\C#** directory.
   a. This project contains two custom application pages named **SiteInfo.aspx** and **DocumentInfo.aspx** (located in the **Solution Explorer TEMPLATE > LAYOUTS > Lab3** directory), as well as various other source files required to deploy and integrate it into a custom business solution.
   b. Use the **Solution Explorer** to get an idea of the project's directory structure as well as what source files are included as part of this project. Make sure to inspect all subdirectories of the **\TEMPLATE** directory.
3. Open **SiteInfo.aspx** and look inside. You should be able to see that this ASP.NET page is defined to inherit from a class defined inside **SiteInfo.cs** or **SiteInfo.vb** named **Lab3.SiteInfo.** You should notice that this page file contains no code; only HTML mark up and a server control tag to create an instance of the **SPGridView** control named **grdSiteProperties**.
4. Open **SiteInfo.cs** or **SiteInfo.vb** and find the class named **Lab3.SiteInfo.** Note that this code-behind class defines a control field using the same name and type that was used in the server control tag in **SiteInfo.aspx**. This technique is what allows method implementation inside the code-behind class **SiteInfo.cs** to access the **SPGridView** control named **grdSiteProperties**.
5. Examine the Feature that is defined in **feature.xml** and **elements.xml** (located in **Solution Explorer TEMPLATE > FEATURES > Lab3 directory**).
   a. Inside the **elements.xml** file you should be able to see that this Feature defines a single **CustomAction** element to add a custom menu item to the **Site Actions** menu that will allow a user to navigate **SiteInfo.aspx**.
6. Build the project. This will recompile the project's output assembly **Lab3.dll** and then run **Install.bat** which contains commands to install **Lab3.dll** in the GAC, to copy the Feature files into the WSS **FEATURES** directory and to install (or reinstall) the Feature itself (note: just like in Lab 2 the Install.bat file also runs an IISRESET command). At this point, the Feature should be installed and ready for activation in any site within the current farm.
7. Navigate to the site you created in the first exercise located at is **http://litwareinc.com/sites/Lab3**.
   a. Click on **Site Actions > Site Settings** to navigate to the **Site Settings page.**
   b. On the **Site Settings page** click the link to go to the **Site Feature** management page (this link is located under the **Site Administration** column).
   c. You should be able to see the Feature for this lab with a title of **Lab 3 - Working with Application Pages**.
   d. Click on **Activate** to enable this Feature.
8. Open the **Site Actions** menu. You should see a menu item with the title of **Get Site Info**.

a. Select that menu item to navigate to **SiteInfo.aspx**. The page **SiteInfo.aspx** should render and display an **SPGridView** control with one row of test data.



9. Now return to Visual Studio and look at the **OnLoad** event handler method for the page within **SiteInfo.cs** or **SiteInfo.vb**. You should see that this code works with a utility class named **PropertyCollectionBinder** to populate the **SPGridView** control named **grdSiteProperties**.
10. Inspect the **PropertyCollectionBinder** class defined inside **PropertyCollectionBinder.cs** or **PropertyCollectionBinder.vb** in order to get an understanding of how this utility class encapsulates creating a **DataTable** and binding to any **SPGridView** control. Note: you don't have to modify the code inside the **PropertyCollectionBinder** class. You will just leverage the utility class as it already exists.
11. Now, go back to the **OnLoad** method of the **SiteInfo.cs** or **SiteInfo.vb** page class and locate the following line of code.

C#

```
// remove the next line and begin your work here
binder.AddProperty("Test Property", "Test Value");
```

VB.NET

```
' remove the next line and begin your work here
binder.AddProperty("Test Property", "Test Value")
```

12. Remove that line of code and replace it with code which programs against the WSS object model to populate the **SPGirdView** control with name value pairs which display information about the current site collection and the current site. In this step you will be forced to use whatever means available to you (IntelliSense, Visual Studio object browser, etc) to find the correct methods you need within the WSS object model to complete the job.
    a. **Hint**: Your **SiteInfo.vb or SiteInfo.cs** file has an **OnLoad()** method that defines two variables **siteCollection** and **site** that may be **EXTREMELY useful** in solving this problem.
    b. **Further Hint**: copy the **binder.AddProperty** line and **replace** "Test Property" with the Property names in the table below (see table in step 13). **Replace** "Test Value" not with a string value but with a property of either **siteCollection** or **site** to expose the requisite information.
13. You should be able to rebuild the project and then refresh the browser to test your work. When you are done, the resulting **SPGridView** control should look approximately like the one below.

Property Name	Value
Site Collection ID	B928C987-1FD6-414D-A42A-71775E517A69
Site Collection Url	http://litwareinc.com
Count of Sites	1
Site Collection Host Name	litwareinc.com
Site ID	DF602775-8F4B-4E03-84B2-8B346A1F67C4
Site Title	Litware Home
Site Description	
Site URL	http://litwareinc.com
Site Created	3/4/2007 1:57:05 PM
Current User Name	Litware Admin Guy
Is Site Root Web?	True
Site Locale	en-US
Site Time zone	(GMT-05:00) Eastern Time (US and Canada)

Exercise 3: Writing an Application Page to display document information

1. Now it's time to work on a second application page. This one will display information to the user about a specific document. The project already includes the source files for a custom application page **DocumentInfo.aspx** in the same location as **SiteInfo.aspx**. Also there is a code-behind class file named **DocumentInfo.cs or DocumentInfo.vb** in the same location as **SiteInfo.cs** or **SiteInfo.vb**.
2. Open **DocumentInfo.aspx** and **DocumentInfo.cs** or **DocumentInfo.vb** in code view and take a minute to review all the code inside these files.
3. The new page named **DocumentInfo.aspx** will be used to display information about a specific document. Therefore, it makes sense to add a new custom ECB (Edit Control Box) menu item for all documents within all document libraries. The purpose of this new ECB menu is to provide users with a command so that they can redirect to **DocumentInfo.aspx** and get more information about a specific document upon request.
    a. Go to the **elements.xml** file and add the following element.

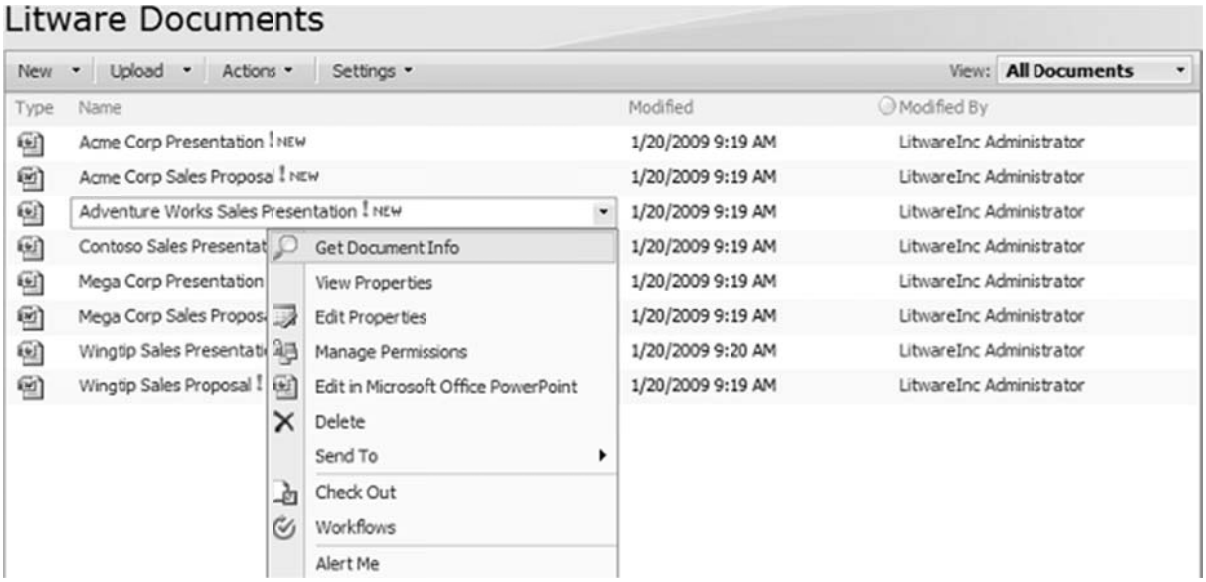
```
<CustomAction
  Id="DocumentInfo"
  RegistrationType="List"
  RegistrationId="101"
  ImageUrl="/_layouts/images/GORTL.GIF"
  Location="EditControlBlock"
  Sequence="10"
  Title="Get Document Info" >

  <UrlAction Url="~site/_layouts/Lab3/DocumentInfo.aspx"/>
</CustomAction>
```

    b. The CAML you have just added to **elements.xml** is not yet complete. When a user redirects from a particular document to **DocumentInfo.aspx**, the code behind this custom application page will require query string parameters to get information about the GUID ID for the document library (i.e. SPList object) and the integer ID for the

specific document (i.e. SPListItem object). Modify the **Url** attribute of the **UrlAction** element with the following value.

```
~site/_layouts/Lab3/DocumentInfo.aspx?ItemId={ItemId}&amp;ListId={ListId}
```

4. Now rebuild the project. This should copy the updated version of the Feature files into the **\FEATURES** directory, reinstall the new updated Feature definition and then run an IISRESET operation.
5. Now it's time to check your work and make sure the ECB menu appears as it should.
    a. Return to the test site and create a new document library. When you create the new document library, you can name it anything you would like.
    b. Once you have created the document library, add a new document or upload an existing document.
    c. Once you have added a document to the document library, drop down the document's Edit menu (i.e. in the document library when you point to your newly created\uploaded document a drop down arrow appears inside of a box that appears around the document.). When you click on the drop down arrow or the Edit Control Box, you should see the ECB menu.



    d. When you drop down the ECB menu for this document, you should see the new custom ECB menu item that was added by our Feature (i.e. the **Get Document Info** choice).
    e. When you select this custom ECB menu item, you should be redirected to **DocumentInfo.aspx**. However, the code behind this application has not yet been updated to show any information about the current document (therefore, we will receive our test data on this page when it tries to load).
6. Update the **OnLoad** event handler method inside **DocumentInfo.cs** to fetch the query string parameters named **ItemId** and **ListId**. Use these values to create an **SPList** object for the document library and an **SPListItem** object for the document. Now, program against the WSS object model to get the information required to make the **SPGridView** control on **DocumentInfo.aspx** look like this.

Property Name	Value
List ID	{9FF39ECF-B518-4E61-8423-BE0FDD9D2C38}
List Title	Proposals
List Root Folder	Proposals
Item ID	1
Item Name	Acme Sales Proposal March 2007.docx
Item URL	Proposals/Acme Sales Proposal March 2007.docx
Document Template Url	Proposals/Forms/template.doc
Document Author	Litware Admin Guy
Document Size	15,534 bits
Last Modified	By Litware Admin Guy on 3/5/2007 6:15:27 AM
File Checkout Status	None

a. Once again, we have left it up to you to use IntelliSense and the Visual Studio object browser to find the correct methods in the WSS object model. If you get stuck and you want some help, you can look at the **DocumentInfo.cs** file in the solution directory for the lab at **\Student\Labs\03_Architecture\Solution**. When you are done, rebuild the project and refresh the browser to test your work.

b. **Hint**: In order to facilitate working with the object model you should create a couple of variables to help you traverse the appropriate properties. Using these variables and the structure outlined below you should be able to add the appropriate properties to your **DocumentInfo** code file:

C#

```
// get list information
string ListId = Request.QueryString["ListId"];
SPList list = site.Lists[new Guid(ListId)];

// get list item information
string ItemId = Request.QueryString["ItemId"];
SPListItem item = list.Items.GetItemById(Convert.ToInt32(ItemId));

// provided the list is a Document Library, get the fine info for the list
item
if (list is SPDocumentLibrary) {
  SPDocumentLibrary documentLibrary = (SPDocumentLibrary)list;
  SPFile file = site.GetFile(item.Url);
}
```

VB.NET

```
' get list information
Dim ListId As String = Request.QueryString("ListId")
Dim list As SPList = site.Lists(New Guid(ListId))

' get list item information
Dim ItemId As String = Request.QueryString("ItemId")
Dim item As SPListItem = list.Items.GetItemById(Convert.ToInt32(ItemId))

' provided the list is a Document Library, get the fine info for the list
item
If TypeOf list Is SPDocumentLibrary Then
  Dim documentLibrary As SPDocumentLibrary = CType(list, SPDocumentLibrary)
  Dim file As SPFile = site.GetFile(item.Url)
End If
```


c. **Further Hint:** Open the **DocumentInfo.cs** or **DocumentInfo.vb** file from the solution to see the relevant properties.

Exercise 4: Working with a Solution Package to deploy this WSS solution

1. Now it is time to use a solution package to deploy this entire solution of WSS components from your development machine into a staging environment or a production environment.
   a. The first thing you need to do is to uninstall and remove all the components from your development machine. Start up a command prompt and run the following batch file named **Uninstall.bat** to reverse everything that was done by **Install.bat**. You can find this batch file at the following path.

      C#
      ```
      \Student\Labs\03_Architecture\Lab\C#\Uninstall.bat
      ```

      VB.NET
      ```
      \Student\Labs\03_Architecture\Lab\VB\Uninstall.bat
      ```
2. Once you have run **Uninstall.bat**, you should be able to verify that the **Lab3** feature is no longer activated or installed. You can do this by going to the **Site Feature** management page (**/_layouts/ManageFeatures.aspx** or **Site Actions > Site Settings > Site Features**) and verifying that the **Lab3** feature is not available for activation.
3. Now within the current project, use the Visual Studio Solution Explorer to look at the files inside the **Solution** folder. Locate the two files named **manifest.xml** and **cab.ddf**. You will not be modifying these files in this exercise, but just be looking at them. Once you have seen what is inside, you will then run batch files to build and deploy the solution package.
4. First, inspect **manifest.xml**. This file represents the top-level metadata file for the solution package and will be read by the WSS installer whenever the solution package is installed or deployed.

```
<Solution
  SolutionId="42262980-404A-4e1c-916A-FD72B031AEDF"
  xmlns="http://schemas.microsoft.com/sharepoint/" >

  <FeatureManifests>
    <FeatureManifest Location="Lab3\feature.xml" />
  </FeatureManifests>

  <TemplateFiles>
    <TemplateFile Location="LAYOUTS\Lab3\DocumentInfo.aspx"/>
    <TemplateFile Location="LAYOUTS\Lab3\SiteInfo.aspx"/>
    <TemplateFile Location="IMAGES\TPG\PithHelmet.gif"/>
  </TemplateFiles>

  <Assemblies>
    <Assembly Location="Lab3.dll" DeploymentTarget="GlobalAssemblyCache" />
  </Assemblies>

</Solution>
```

5. Next, look at what is inside the **cab.ddf** file. These are the instructions used to generate the solution package named **Lab3.wsp**. **Note** that there is an **individual line for each file that needs to be added to the solution package**. As you can imagine, maintaining a **.ddf** file and keeping it in sync with the **manifest.xml** file is a very tedious and error-prone undertaking. **Unfortunately**, **Microsoft provides no developer tools to help create or maintain these files.**

```
.OPTION EXPLICIT ; Generate errors
```



```
.Set CabinetNameTemplate=Lab3.wsp
.set DiskDirectoryTemplate=CDROM ; All cabinets go in a single directory
.Set CompressionType=MSZIP;** All files are compressed in cabinet files
.Set UniqueFiles="ON"
.Set Cabinet=on
.Set DiskDirectory1=Package


Solution\manifest.xml manifest.xml
TEMPLATE\FEATURES\Lab3\feature.xml Lab3\feature.xml
TEMPLATE\FEATURES\Lab3\elements.xml Lab3\elements.xml
TEMPLATE\LAYOUTS\Lab3\SiteInfo.aspx LAYOUTS\Lab3\SiteInfo.aspx
TEMPLATE\LAYOUTS\Lab3\DocumentInfo.aspx LAYOUTS\Lab3\DocumentInfo.aspx
TEMPLATE\IMAGES\TPG\PithHelmet.gif IMAGES\TPG\PithHelmet.gif
bin\Debug\Lab3.dll Lab3.dll

;*** <the end>
```

6. Now you have seen the two main files used to build a solution package. As you may know, the **cab.ddf** file is used as input to the **MAKECAB.EXE** utility to create a solution package named **Lab3.wsp**. The **manifest.xml** file is included inside the solution package and this file acts as the main manifest used by the WSS installer during solution package deployment.
7. Now within the current project, use the Visual Studio Solution Explorer to look at the files inside the **Package** folder. You should see the following four batch files which have been created to assist you with creating and managing the solution package:
    a. **CreateSolutionPackage.cmd**
    b. **DeleteSolutionPackage.cmd**
    c. **DeploySolutionPackage.cmd**
    d. **InstallSolutionPackage.cmd**
8. Inspect (**but do not run**) the batch file named **CreateSolutionPackage.cmd** which has been written to build **Lab3.wsp**. The contents of this batch file look like this:

```
@echo off
if EXIST Lab3.wsp del Lab3.wsp
cd ..
makecab /f Solution\cab.ddf
cd Package
```

9. Inspect (**but do not run**) the batch file named **InstallSolutionPackage.cmd** to install **Lab3.wsp** into the farm-scoped solution package store. The contents of this batch file look like this:

```
@SET SPDIR="c:\program files\common files\microsoft shared\web server
extensions\12"
%SPDIR%\bin\stsadm -o addsolution -filename Lab3.wsp
```

10. Finally, inspect (**but do not run**) the batch file named **DeploySolutionPackage.cmd**. This batch file accomplishes the same steps as you have seen with **CreateSolutionPackage.cmd** and **InstallSolutionPackage.cmd** and then it deploys the **Lab3.wsp** package in the current farm. You can also see that this batch **file begins by retracting and removing any previous version of this solution package before reinstalling and redeploying it**. Now, take a detailed look through the batch file named **DeploySolutionPackage.cmd**.

```
@SET STSADM="c:\program files\common files\microsoft shared\web server
extensions\12\bin\stsadm.exe"

Echo Retracting Solution Package Lab3.wsp
%STSADM% -o retractsolution -name Lab3.wsp -immediate
%STSADM% -o execadmsvcjobs
```



```
Echo Deleting Solution Package Lab3.wsp
%STSADM% -o deletesolution -name Lab3.wsp -override
%STSADM% -o execadmsvcjobs

Echo Generating Solution Package Lab3.wsp
if EXIST Lab3.wsp del Lab3.wsp
cd ..
makecab /f Solution\cab.ddf
cd package

Echo Installing Lab3.wsp in WSS Solution Package Store
%STSADM% -o addsolution -filename Lab3.wsp
%STSADM% -o execadmsvcjobs

Echo Deploying Solution Package Lab3.wsp
%STSADM% -o deploysolution -name Lab3.wsp -immediate -allowGacDeployment -force
%STSADM% -o execadmsvcjobs
```

11. Now it's time to update the post-build event of the Visual Studio project named Lab3. Currently, the post-build event instructions for the project are configured to run **Install.bat**.

```
cd $(ProjectDir)
Install.bat
```

a. Modify the post-build event instructions to run **DeploySolutionPackage.cmd** instead of **Install.bat**. Note that you must add a line to change the current directory to the **Package** directory before you run **DeploySolutionPackage.cmd**.

```
cd $(ProjectDir)Package
DeploySolutionPackage.cmd
```

b. In **Solution Explorer** right-click the project and choose **Properties** to view the **Project Properties** for the **Lab3** project. If using VB go to step i. If using C# go to step ii.

Visual Basic .Net directions:

1. Navigate to the **Compile** tab
2. Click on the **Build Events...** button (scroll down if necessary to see this button)
3. Edit the Post-build event command line instructions so that they match the following.

```
cd $(ProjectDir)Package
DeploySolutionPackage.cmd
```

a. Note that the first line with **cd $(ProjectDir)Package** is required to change the current directory to that of the project directory. The second line runs the command file **DeploySolutionPackage.cmd** to deploy this package.

C# directions:

1. Navigate to the **Build Events** tab.
2. Add the following Post-build event command line instructions.

```
cd $(ProjectDir)Package
DeploySolutionPackage.cmd
```

   b. Note that the first line with **cd $(ProjectDir)Package** is required to change the current directory to that of the project directory. The second line runs the command file **DeploySolutionPackage.cmd** to deploy this package.

12. Now, rebuild the project. Monitor the progress of the run **DeploySolutionPackage.cmd** by inspecting the **Output** window for Visual Studio.
    a. From the Visual Studio **Build** menu select **Build Lab 3.**
13. Now, it's time to make sure everything has been installed correctly and that your solution works after the solution package has been deployed.
    a. Navigate to **Site Feature** management page (**/_layouts/ManageFeatures.aspx** or **Site Actions > Site Settings > Site Features**).
    b. Ensure the **Lab3** feature is there and can be activated.
    c. Once you have activated it, go to the Document Library you created and test out the ECB menu item and custom application pages to make sure they work.
14. If you are interested, go through the steps of retracting the solution package as well.
    a. Start by deactivating the **Lab3** feature in the current site.
    b. Once you have deactivated the **Lab3** feature, bring up a command prompt and run the batch file named **DeleteSolutionPackage.cmd** which contains the follow command-line instructions.

```
@SET STSADM="c:\program files\common files\microsoft shared\web server
extensions\12\bin\stsadm.exe"
%STSADM% -o retractsolution -name Lab3.wsp -immediate
%STSADM% -o execadmsvcjobs
%STSADM% -o deletesolution -name Lab3.wsp
```


Lab 04: Developing Custom Site Pages

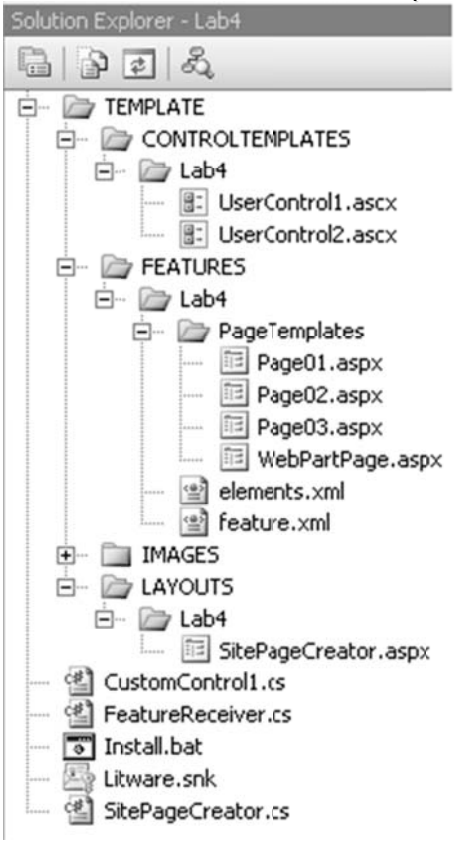
Lab Time: 45 Minutes

Lab Directory: C:/Student/Labs/04_SitePages

Lab Overview: In this lab, you will work with a Visual Studio project to write, deploy and test custom site pages that exist within the virtual file system of a WSS site. This will teach you how to create and instantiate page templates using a Feature. It will also show you how to create site pages on-the-fly using custom code.

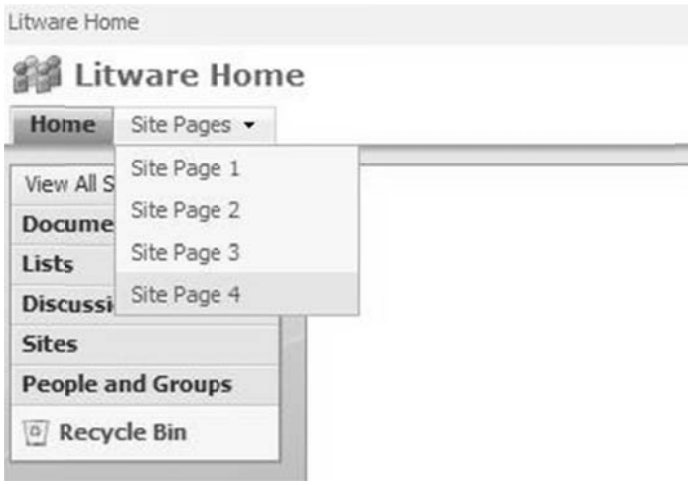
Exercise 1: Build and Activate a Feature which provisions site pages

1. Launch Visual Studio. Open the project named **Lab4.sln** located inside the **\04_SitePages\Lab** directory.
    a. This is a project that contains a Feature named **Lab4** that defines several page templates that are used to provision site page instances upon activation.
    b. Use the **Solution Explorer** to get an idea of the project's directory structure as well as what source files are included as part of this project.

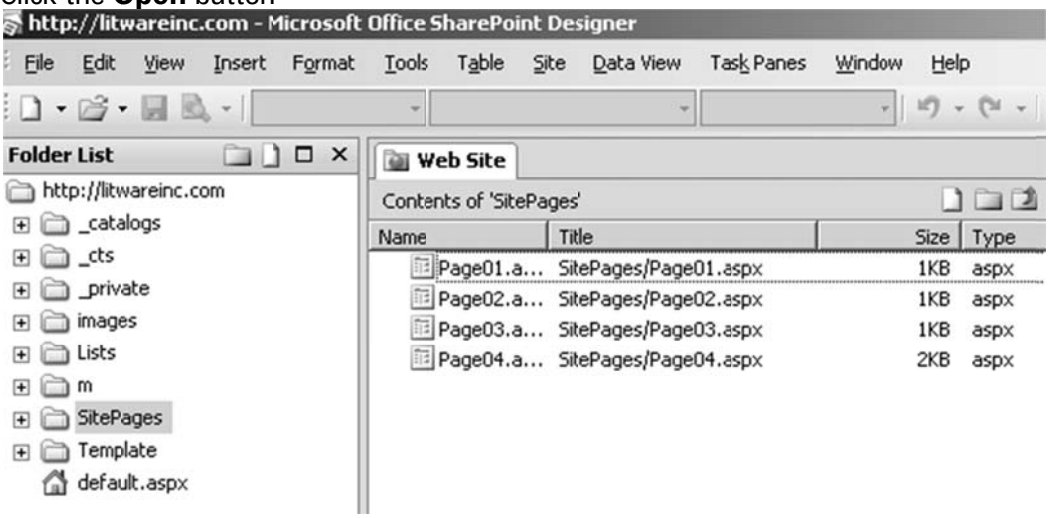


2. Look at the code inside the page template definitions for **Page01.aspx**, **Page02.aspx**, **Page03.aspx** and **WebPartPage.aspx** that are located in the **PageTemplates** directory within the **Features\Lab4** directory. This should give you a sense of what goes inside a simple page template.

   a. Note that all three Page0x.aspx files (note: x = the page #) utilize the **default.master** master page that most all site pages utilize which gives all of your custom pages a similar look and feel to the built in site pages (i.e. same navigational elements and images across the top and down the left side of the screen)
   b. Note that the location of the **MasterPageFile** "~masterurl/default.master" maps to the "C:\Program Files\Common Files\Microsoft Shared\web server extensions\12\TEMPLATE\GLOBAL" directory. This master page is used widely across many different Site Pages in WSS.
   c. Note that all three Page0x files have a **Content** area defined in them (asp:Content...>). This area contains the unique items in these particular pages (outside of those defined in the master page)
   d. Note that **Page03.aspx** utilizes two Custom Controls (**UserControl1.ascx** and **UserControl2.ascx**) that are defined in your /TEMPLATE?CONTROLTEMPLATE/Lab4/ directory.
3. Open **elements.xml** and see how **File** elements are nested inside the **Module** element to automatically provision instances from these page templates upon Feature activation.
4. Open the **feature.xml** file and note that the **Lab4** Feature has been configured with a Feature receiver class named **Lab4.FeatureReceiver** that has event handlers that fire during activation and deactivation.
   a. Look at this event handler code inside **FeatureReceiver.cs** or **FeatureReceiver.vb**.
   b. You can see there is code in the **FeatureActivated** event handler that creates a new drop down menu to the current site's top-link bar that includes menu items to navigate to the site page instances that have been provisioned in **elements.xml**.
   c. There is also code inside the **FeatureDeactivated** event handler to remove these menu items and also to delete the directory named **SitePages** in which all the site pages have been provisioned.
5. Build the project. This will recompile the project's output assembly **Lab4.dll** and then it will run **Install.bat** which contains commands to install **Lab4.dll** in the GAC, to copy the Feature files into the WSS **FEATURES** directory and to install (or reinstall) the Feature itself. At this point, the Feature should be installed and ready for activation in any site within the current farm.
6. To test this feature you should either create a new top-level site or go to an existing site such as the one at **http://litwareinc.com** and activate the **Lab4** Feature.
   a. **Site Actions > Site Settings > Site Features** (Under Site Administration column)
   b. Locate the **Lab4 - Working with Site Pages** feature and click the **Activate** button
7. After the Feature has been activated, you should see a new drop down menu appear in the top-link bar of the current site:



8. Use this menu to navigate to each of the pages.
    a. When you get to **Page04.aspx**, you should be able to see it is a Web Part Page. (Note: do not wait for the spinning green SharePoint loading symbol to finish loading the web part… go directly to step B instead).
    b. If you go into **Edit Mode**, you should be able to move the Web Part instance that is there between zones. (notice that this web part is a Watch My Gears Run web part that is custom built to perpetually display the green spinning SharePoint Symbol...).
    c. You should also be able to add new Web Part instances just like with any other Web Part Page.
9. Launch the **SharePoint Designer** and open the site in which you just activated the **Lab4** Feature.
    a. **Start > SharePoint Designer 2007** (pinned to the start menu near the top)
    b. **File menu > Open Site...**
    c. In the **Site name:** text box type **http://litwareinc.com/**
    d. Click the **Open** button



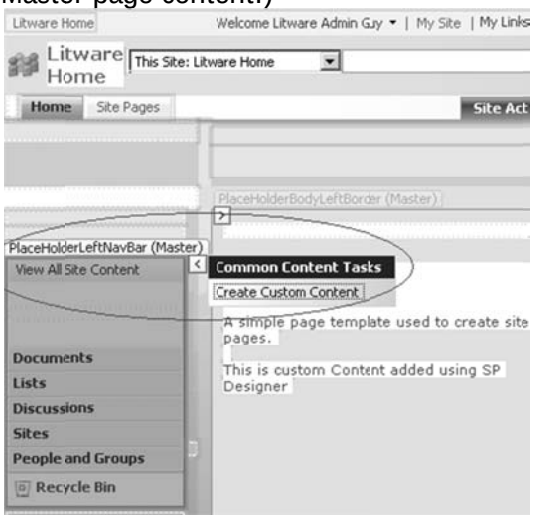
10. You should be able to verify that **SharePoint Designer** can recognize and open each of the site pages that have been provisioned by the **Lab4** Feature.
    a. In **SharePoint Designer** under the **Folder List** click on **SitePages** to see the 4 pages that our feature provided.
    b. Open **Page01.aspx** in **SharePoint Designer** and make a simple custom change.

i. In the **Contents of** '**SitePages**' area double click on **Page01.aspx** this will open Page01 for editing with SP Designer
ii. Notice how the Master Page is loaded (Loading Master Page...) along with our unique content provided by Page01. SharePoint Designer is about the only tool that can pull all of the disparate information together to allow us to edit the page as if these files were all stored in one central location (i.e. when you open a page in SP Designer it will not only traverse the physical directory structure on the Web server to find the default.master page but also pull Page01.aspx out of the SQL Content Database where it is stored and put these two back together so that we may edit them using a graphical view or code view. Compare this to our experience in Visual Studio where at best we only see the code view of our page fragment (i.e. just the code for the Page01.aspx file).

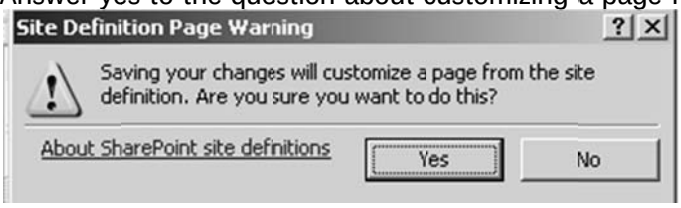
11. Edit your **PlaceholderMain (Custom)** by just clicking into this area and put your insertion point after the default text. Add a line by pressing enter and type the text you see in the Oval.

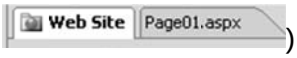
12. You may even edit bits of the master page area (left side and top of screen) by clicking in a master page area and selecting the arrow icon and selecting **Create Custom Content** . However doing this will Un-ghost the Master Page area for this page and new changes to the master page will no longer be picked up on this page so the recommendation is to avoid making changes to the Master page area of a page using SharePoint Designer (**Note**: if you selected **Create Custom Content** then you should select the arrow icon **>** again and click on **Default to Master's content** and click **yes** to the next question about reverting the region to the

Master page content.)

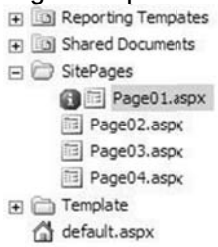


13. Save your change.
    a. In SharePoint designer Click on the **File menu** and click **Save**
    b. Answer yes to the question about customizing a page from the site definition



    c. Return to the browser and refresh **Page01.aspx**.
    d. Now return to **SharePoint Designer.**
    e. Under the **Folder List** (i.e. Click on the **Web Site** tab 

) expand the **SitePages** Folder if necessary). You should be able to see the customization changes made by SharePoint Designer (you may have to expand the **View** menu and click **Refresh** or press the **F5** key on your keyboard to refresh this screen).
        i. Notice the blue icon next to the file in the **Folder List** tool window in SharePoint Designer indicating the page has been customized (e.g. unghosted). This means that this version of Page01.aspx on this site is now independent of any other site collections feature activated Page01.aspx. Therefore any change to the base Page01.aspx will NOT be seen on this Site Collections Page01 in the future...



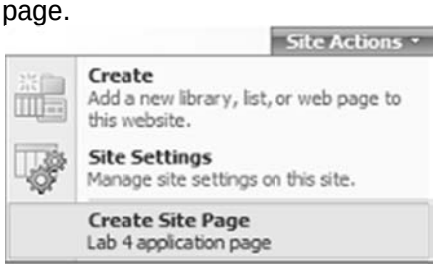
14. Now we will re-ghost this page.
    a. Right-click **Page01.aspx** and select **Reset to Site Definition**

b. Click Y**es** to the Site Definition Page Warning (**n**ote that SharePoint will return the page to its uncustomized (e.g. ghosted) state.

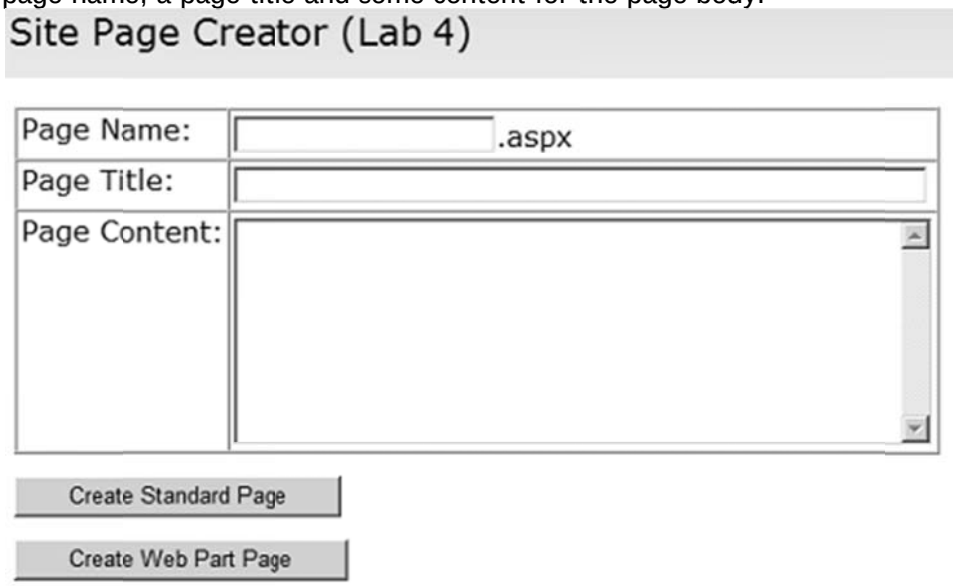
c. Note that when we reset a site using SharePoint designer a backup copy of the changed page is created as a safety net.
d. Note: Should you not want to keep this version you should delete it as it will be accessible via the newly created site link otherwise. In our case we will leave this alone as we may need to utilize it later.

Exercise 2: Create a Site Page using Code

1. In **Internet Explorer** let's examine the Custom application page added by our feature: **SitePageCreator.aspx**.
   a. From the site you activated the feature on, drop down the **Site Actions** menu for the site, you should notice that there is a menu item to navigate to a custom application page.



   b. Select the **Create Site Page** menu item from the **Site Actions** menu to navigate to the page. You can see it supplies three textbox controls which allow the user to enter a new page name, a page title and some content for the page body.



2. Your job is now to write code inside this page's event handlers to take the user input from **SitePageCreator.aspx** to create a new site page on the fly. When you are done your code should add the new site page to the **\SitePages** directory within the virtual file system of the current site. It should also update the drop down menu on the top link bar to add a new menu item which allows users to navigate to this new page.
    a. In Visual Studio open **C:\Student\Labs\04_SitePages\Lab\C#\Lab4.sln**, and then open the file named **SitePageCreator.cs** or **SitePageCreator.vb**.
    b. Next locate the event handler method named **btnCreateStandardPage_Click**.
    c. You need to accomplish several things to create a page on the fly here (note: you may use these detailed directions below or look at an existing page (like Page01.aspx to get a sense of what we need to write out here).
        i. In order to write things on the fly you need to utilize a stream writer and a memoryStream to write your html into

C#
```
MemoryStream stream = new MemoryStream();
StreamWriter writer = new StreamWriter(stream);
```

VB.Net
```
Dim stream As New MemoryStream()
Dim writer As New StreamWriter(stream)
```

        ii. Next we need to utilize the writer to write out our @Page directive (**Note**: although this code is broken up across multiple lines it **MUST BE ENTERED** on one physical line in order to run as written)

C#
```
writer.WriteLine(
  "<%@ Page MasterPageFile='~masterurl/default.master'
       meta:progid='SharePoint.WebPartPage.Document' %>");
```

VB.Net
```
writer.WriteLine(
  "<%@ Page MasterPageFile='~masterurl/default.master'
       meta:progid='SharePoint.WebPartPage.Document' %>")
```

        iii. Next we need to utilize the writer to write out our asp: Content tag and then populate this tag with information taken from the SitePageCreator.aspx page (namely the Title field and the Page Content field)
(**Note**: this Content tag matches up to a **ContentPlaceHolder** tag defined in the default.master page file)

C#
```
writer.WriteLine("<asp:Content runat='server'
ContentPlaceHolderID='PlaceHolderMain'>");
writer.WriteLine("<h3>" + txtPageTitle.Text + "</h3>");
writer.WriteLine(txtPageContent.Text);
writer.WriteLine("</asp:Content>");
```

VB.Net
```
writer.WriteLine("<asp:Content runat='server'
ContentPlaceHolderID='PlaceHolderMain'>")
writer.WriteLine(("<h3>" + txtPageTitle.Text + "</h3>"))
writer.WriteLine(txtPageContent.Text)
writer.WriteLine("</asp:Content>")
```

        iv. Now we need to flush out the writer to persist this information to the stream object

C#
```
writer.Flush();
```

VB.Net
```
writer.Flush()
```

v. Next we need to take the site name provided by the user on the **SitePageCreator.aspx** page and create our physical page name adding this new page to our current Site's web pages.

C#
```
string NewPageUrl = @"SitePages/" + txtPageName.Text + ".aspx";
this.Web.Files.Add(NewPageUrl, stream);
```

VB.Net
```
Dim NewPageUrl As String = "SitePages/" + txtPageName.Text + ".aspx"
Me.Web.Files.Add(NewPageUrl, stream)
```

vi. Now we should add navigational elements to our main page to allow easy access to this newly created page.

C#
```
// grab the existing site TopNavigationBar for our use
SPNavigationNodeCollection topNav =
this.Web.Navigation.TopNavigationBar;

// create dropdown menu for custom site pages
foreach (SPNavigationNode node in topNav[0].Children) {
  if(node.Title.Equals("Site Pages")) {
    SPNavigationNode newNode = new
SPNavigationNode(txtPageName.Text,NewPageUrl);
    node.Children.AddAsLast(newNode);
  }
}
```

VB.Net
```
' grab the existing site topNavigationBar for our use
Dim topNav As SPNavigationNodeCollection =
Me.Web.Navigation.TopNavigationBar

' create dropdown menu for custom site pages
Dim node As SPNavigationNode
For Each node In topNav(0).Children
  If node.Title.Equals("Site Pages") Then
    Dim newNode As New SPNavigationNode(txtPageName.Text, NewPageUrl)
    node.Children.AddAsLast(newNode)
  End If
Next node
```

vii. Lastly we will utilize the **SPUtility** class to navigate directly to our newly created page

C#
```
SPUtility.Redirect(Web.Url + @"/" + NewPageUrl,
SPRedirectFlags.Default, this.Context);
```

VB.Net
```
SPUtility.Redirect((Web.Url + "/" + NewPageUrl,
SPRedirectFlags.Default, _ Me.Context)
```

3. When you are done, rebuild the project. That will also run Install.bat to redeploy our changes and test your work. You should be able to use the application page named

SitePageCreator.aspx to dynamically add new pages to the current site along with an associated navigation menu item.

a. Navigate to your drop down menu **Site Actions > Create Site Page**
b. Using this page make up some information: Name, Title, Content and enter it where appropriate
c. Click the **Create Standard Page** Button
d. You should now be on the newly created page.

Exercise 3: Create a Web Part Page using Code

1. Within the source file named **SitePageCreator.cs** or **SitePageCreator.vb**, locate the event handler method named **btnCreateWebPartPage_Click**. Your job in this exercise is to write code inside this event handler to take the user input from **SitePageCreator.aspx** and to create a new Web Part Page on the fly just as you did in the previous exercise. However, instead of creating a new site page, you should clone the Web Part Page instance named **WebPartPage.aspx**

C#

```
// clone Web Part Page template to create new Web Part Page
string NewPageUrl = @"SitePages/" + txtPageName.Text + ".aspx";
SPFile template = Web.GetFile(@"Template\WebPartPage.aspx");
template.CopyTo(NewPageUrl);
```

VB.Net

```
' clone Web Part Page template to create new Web Part Page
Dim NewPageUrl as string = "SitePages/" + txtPageName.Text + ".aspx"
Dim template as SPFile = Web.GetFile("Template\WebPartPage.aspx")
template.CopyTo(NewPageUrl)
```

2. After creating the page, use the **SPLimitedWebPartManager** class to add a new **ContentEditor** Web Part to the page that has the title and body content that the user entered on the page.

C#

```
// add Content Editor Web Part
SPFile NewPage = Web.GetFile(NewPageUrl);
SPLimitedWebPartManager mgr;
mgr = NewPage.GetLimitedWebPartManager(PersonalizationScope.Shared);
ContentEditorWebPart wp1 = new ContentEditorWebPart();
wp1.Title = txtPageTitle.Text;
wp1.ChromeType = PartChromeType.TitleOnly;
wp1.AllowClose = false;
XmlDocument doc = new XmlDocument();
string ns1 = "http://schemas.microsoft.com/WebPart/v2/ContentEditor";
XmlElement elm = doc.CreateElement("Content", ns1);
elm.InnerText = txtPageContent.Text;
wp1.Content = elm;
```

VB.Net

```
' add Content Editor Web Part
Dim NewPage As SPFile = Web.GetFile(NewPageUrl)
Dim mgr As SPLimitedWebPartManager
mgr = NewPage.GetLimitedWebPartManager(PersonalizationScope.Shared)
Dim wp1 As New ContentEditorWebPart()
wp1.Title = txtPageTitle.Text
wp1.ChromeType = PartChromeType.TitleOnly
```



```
wp1.AllowClose = False
Dim doc As New XmlDocument()
Dim ns1 As String = "http://schemas.microsoft.com/WebPart/v2/ContentEditor"
Dim elm As XmlElement = doc.CreateElement("Content", ns1)
elm.InnerText = txtPageContent.Text
wp1.Content = elm
```

3. Now you will add this newly created web part to the Left Zone of the existing page using the **LimitedWebPartManager** (mgr) that we created earlier. Also, you will specify that this item should be displayed first in that zone.

C#

```
// add Web Part to Left Zone
mgr.AddWebPart(wp1, "Left", 0);
```

VB.Net

```
' add Web Part to Left Zone
mgr.AddWebPart(wp1, "Left", 0)
```

4. Next you will add this page to the site's **TopNavigationBar** in a node entitled "**Site Pages**". Make certain that this new page is appended to the end of the existing page list.

C#

```
// add navigation node
SPNavigationNodeCollection topNav = this.Web.Navigation.TopNavigationBar;
foreach (SPNavigationNode node in topNav[0].Children) {
  if (node.Title.Equals("Site Pages")) {
    SPNavigationNode newNode = new SPNavigationNode(txtPageName.Text,
NewPageUrl);
    node.Children.AddAsLast(newNode);
  }
}
```

VB.Net

```
' add navigation node
Dim topNav As SPNavigationNodeCollection =
Me.Web.Navigation.TopNavigationBar
Dim node As SPNavigationNode
For Each node In topNav(0).Children
  If node.Title.Equals("Site Pages") Then
    Dim newNode As New SPNavigationNode(txtPageName.Text, NewPageUrl)
    node.Children.AddAsLast(newNode)
  End If
Next node
```

5. Now, just like earlier, we will utilize the **SPUtility** class to navigate directly to our newly created page.

C#

```
// navigate directly to the newly created Web Part Page
SPUtility.Redirect(Web.Url + @"/" + NewPageUrl, SPRedirectFlags.Default,
this.Context);
```

VB.Net

```
' navigate directly to the newly created Web Part Page
SPUtility.Redirect(Web.Url + "/" + NewPageUrl, SPRedirectFlags.Default,
Me.Context)
```

6. When you are done, rebuild the project, which will also run Install.bat to re-deploy our changes and test your work. You should be able to use the application page named **SitePageCreator.aspx** to dynamically add new Web Part pages to the current site along with an associated navigation menu item.
    a. Navigate to your drop down menu **Site Actions > Create Site Page**

b. Using this page make up some information: Name, Title, Content and enter it where appropriate
c. Click the **Create Web Part Page**
d. You should now be on the newly created Web Part page.

Lab 05: Walkthrough of Custom Branding

Lab Time: 20 Minutes

Lab Directory: C:/Student/Labs/05_Branding

Lab Overview: You will not be required to write any code in this walkthrough. You will just be required to deploy a WSS solution for a custom branding feature and to use this solution to apply branding to a site collection.

Exercise 1: Open and Build the project

1. Launch Visual Studio. Open the project named **CustomBranding.sln** located inside the **\Student\Demos\05_Branding\CustomBranding** directory.
2. Take a moment to familiarize yourself with all the source files inside this project.
3. Open and inspect the batch file named **Install.bat** so you can see what it does.
4. Rebuild the project. This will rebuild **CustomBranding.dll** and then run **Install.bat**. After this, you can go to a site and test it out.

Exercise 2: Activating and using the Custom Branding feature on a site

1. Go to any top-level site such as the one located at **http://litwareinc.com**. Go to the **Site collection features** management page (**_layouts/ManageFeatures.aspx?Scope=Site**). Remember that this is different from the Site features management page (**_layouts/ManageFeatures.aspx**).
2. Find the feature with the title of **A Sample Feature: Custom Branding**. Activate this feature.
3. Now, when you drop down the Site Actions menu you should see a new command with a title of **Custom Branding**. Click on the **Custom Branding** menu to navigate to the custom application page named **CustomBrand.aspx**.
4. Once you are at the application page named **CustomBrand.aspx**, click on the command button named **Apply Custom Litware Brand**. Now go to various pages within the current site to see the branding changes.
5. Navigate back to the page named **CustomBrand.aspx** and click the command button named **Remove Custom Litware Brand**. This should remove the branding changes and put the site's look and feel back to its original state.
6. Go back to Visual Studio and open the file named **CustomBrand.aspx** in code view. Note that this file contains inline code which programs against the WSS object model to apply and remove the branding changes. Review all this code so you can understand how it is able to apply and remove branding for the entire site collection.
7. If you have time, inspect these other source files within the project to see how they add to this branding solution.
    a. **\TEMPLATE\FEATURES\CustomBranding\feature.xml**

b. **\TEMPLATE\FEATURES\CustomBranding\elements.xml**
c. **\TEMPLATE\FEATURES\CustomBranding\Litware.master**
d. **\TEMPLATE\LAYOUTS\1033\styles\Litware\LitwareBrand.css**

Lab 06: Creating Custom Web Parts

Lab Time: 60 Minutes

Lab Directory: C:/Student/Labs/06_WebParts

Lab Overview: In this lab, you will create a simple "Hello World" Web Part and then proceed to modify it so that it will be able to calculate business information from field values within the lists of your sites.

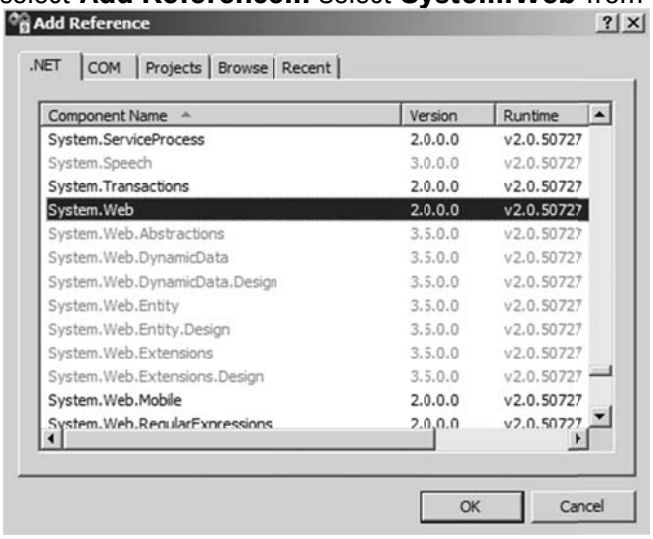
- In exercise 1 you will create a simple Web Part based class library and see how to import it into SharePoint.
- In exercise 2 you will build on this by adding a conditional property to change the display of the text.
- In exercise 3 you will delve into the SharePoint object model to pull data out of a SharePoint list and aggregate it for display in our custom Web Part.

Exercise 1: Create and test the traditional "Hello World" Web Part

In this exercise you will write a custom Web Part using Visual Studio 2008. You will be provided with a class library DLL project as a starting point.

1. Begin this lab by opening the **..\06_WebParts\Lab\VB\LitwareWebPartsLab.sln** or **..\06_WebParts\Lab\C#\LitwareWebPartsLab.sln** file in Visual Studio 2008.
    a. Take a moment and familiarize yourself with the project inside this solution named **LitwareWebPartsLab**. You should notice that the **LitwareWebPartsLab** project is a class library project that has **NOT** been configured to build an Assembly DLL with a strong name. (Hint: in your **Solution Explorer** right-click on your **LitwareWebPartsLab** project and select **Properties.** On the **Signing** tab you will find the **Sign the assembly** check box. Verify that it is **NOT SELECTED**).
    b. **Note**: see http://support.microsoft.com/kb/839300 for information about the issues with strongly named assemblies and \bin directory deployment in .Net.
2. Note there is already an existing class file named **RevenueWebPart.cs** or **RevenueWebPart.vb** with a class named **RevenueWebPart**. It is your mission to transform this standard boring class into a Web Part for use with the **Project Management** site you created earlier in this Course.
3. Add a reference to **System.Web.DLL**. This system assembly has types that you will need to write ASP.NET-style Web Parts.

a. In your **Solution Explorer**, right-click on your **LitwareWebPartsLab** project and select **Add Reference...** Select **System.Web** from the list and click the **OK** button.



4. Modify the **RevenueWebPart** class so that you have an Imports (**VB**) or using (**C#)** statement for this (**System.Web.UI.WebControls.WebParts**) namespace and then make your class inherit from the **WebPart** class defined inside the **WebParts** namespace.

C#

```
using System;
using System.Web.UI.WebControls.WebParts;

namespace LitwareWebPartsLab {
  public class RevenueWebPart : WebPart {
    // Web Part code goes here
  }
}
```

VB.Net

```
Imports System
Imports System.Web.UI.WebControls.WebParts

Public Class RevenueWebPart Inherits WebPart
  ' Web Part code goes here
End Class
```

5. Inside the **RevenueWebPart** class, override the **RenderContents** method with a minimal "Hello, world" implementation. (Note: you will need to add either an **Imports** (VB) or **using** (C#) statement for **System.Web.UI** as the **HtmlTextWriter** class exists in this namespace)

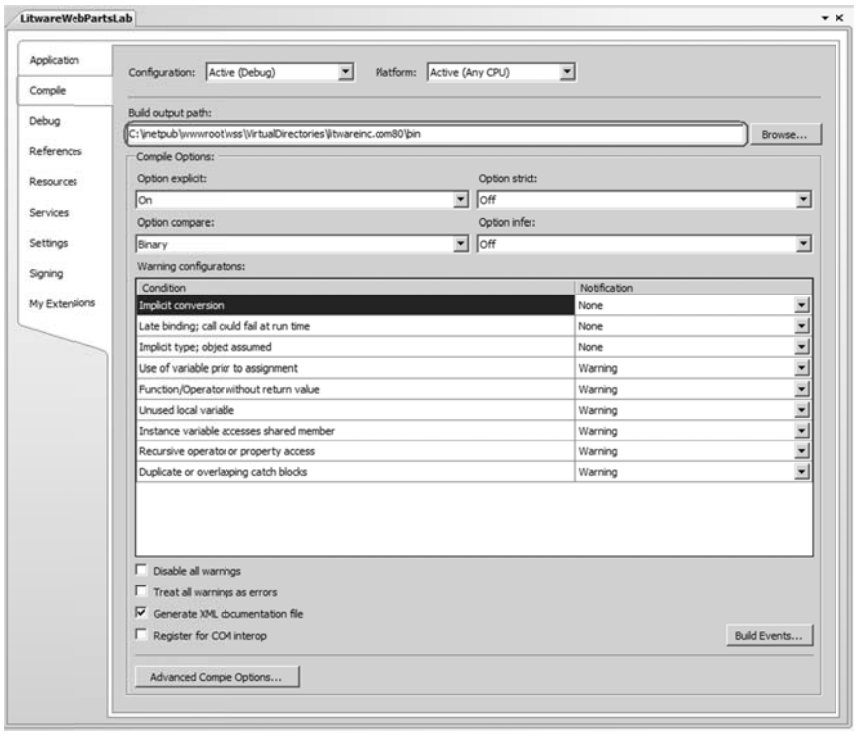
C#

```
using System;
using System.Web.UI;
using System.Web.UI.WebControls.WebParts;

namespace LitwareWebPartsLab {
  public class RevenueWebPart: WebPart {
   protected override void RenderContents(HtmlTextWriter writer) {
     writer.Write("Hello, world");
   }
  }
}
```

VB.Net

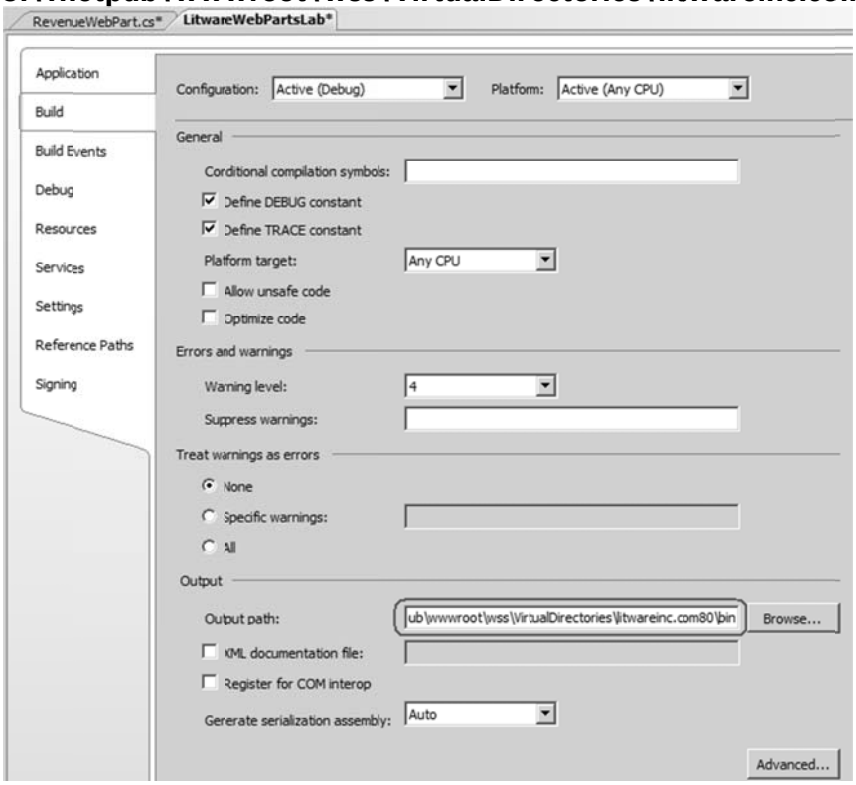
```
Imports System
Imports System.Web.UI
Imports System.Web.UI.WebControls.WebParts

Public Class RevenueWebPart Inherits WebPart
  Protected Overrides Sub RenderContents(ByVal writer As
  System.Web.UI.HtmlTextWriter)
    MyBase.RenderContents(writer)
    writer.Write("Hello, world")
  End Sub
End Class
```

6. Remember that a Web Part DLL must be located in a place where the WSS runtime can find it. When you deploy a Web Part DLL, you have the option of putting it in either the local **\bin** directory or the **Global Assembly Cache (GAC)**. In this case, you are going to compile the assembly DLL output for the **LitwareWebPartsLab** project directly into the **\bin** directory of the default Web site.
    a. Modify the **LitwareWebPartsLab** project **Output path** to the following path by right clicking on your **LitwareWebPartsLab** project in **Solution Explorer** and select **Properties**
    b. In VB:
    In your **Project Properties** window select **Compile*** and then **Browse** and set the **Build output path** to
    **C:\Inetpub\wwwroot\wss\VirtualDirectories\litwareinc.com80\bin **.



In C#:
In your **Project Properties** window select **Build** and then **Browse** and set the

Output path to
C:\Inetpub\wwwroot\wss\VirtualDirectories\litwareinc.com80\bin.



7. Build the **LitwareWebPartsLab** project.
    a. After you have compiled the project, use the Windows Explorer to verify that the assembly DLL has been created in the bin subdirectory inside root directory of the Web Application running on port 80. Remember that the root directory is at the following path

```
C:\Inetpub\wwwroot\wss\VirtualDirectories\litwareinc.com80\
```

8. In order to be able to use Web Parts deployed into the local ...\bin directory you must modify the **<trust level=...>** tag in the **Web.config**
    a. By default, code access security permissions for the bin directory are low; only pure execution is allowed. Although our code in this exercise can currently run with the minimal trust level, in most cases you need to elevate these permissions to make your assembly run correctly, for example, if your Web Part requires access to the SharePoint object model. In fact, we MUST modify this now as it will be required for this lab to run correctly before the end of exercise 3.
    b. **Note**: There are two ways to elevate permissions **(MAKE SURE YOU USE OPTION ii (EASY WAY))**:
        i. (**Recommended way** for deployment) Create a new trust policy file, and point your web.config file at the new file. This option is more complicated but it gives you a precise attribution of permissions for your Web Parts. For more information about trust policy files, see Microsoft Windows SharePoint Services and Code Access Security.

ii. (**Easy Way** for Development/Debugging) Raise the .Net trust level of the bin directory.
    1. Open the **Web.config** file in **C:\Inetpub\wwwroot\wss\VirtualDirectories\litwareinc.com80** in Visual Studio.
    2. In the **Web.config** file in the Web application root, there is a tag called **<trust>** with a default attribute of **level="WSS_Minimal"**.
       Note: this the easiest way to locate this tag is to do a find/replace function (**ctrl+h**) and search for **trust level**.
    3. Change this level to **WSS_Medium**.
       Note: While this option is simpler and preferred for testing and debugging, it grants arbitrary new permissions you might not need and is less secure than creating a new trust policy file.
    4. **Save** your changes to this file

9. In order to be able to use this newly created web part we must mark it as a "**SafeControl**" in the **Web.config** file for the Web Application.
    a. Edit the **Web.config** file from the **C:\Inetpub\wwwroot\wss\VirtualDirectories\litwareinc.com80** directory.
    b. Add a **SafeControl** setting to the **Web.config** file, in the preexisting **<SafeControls></SafeControls>** element, for your new Web Part that follows this format.

```
<SafeControl Assembly="[add assembly name]" Namespace="[add
namespace]" TypeName="*" />
```

i. When you add the **SafeControl** element for your Web Part, the namespace should be written as **LitwareWebPartsLab** and the 4-part assembly format string should look like this.

```
LitwareWebPartsLab, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=null
```

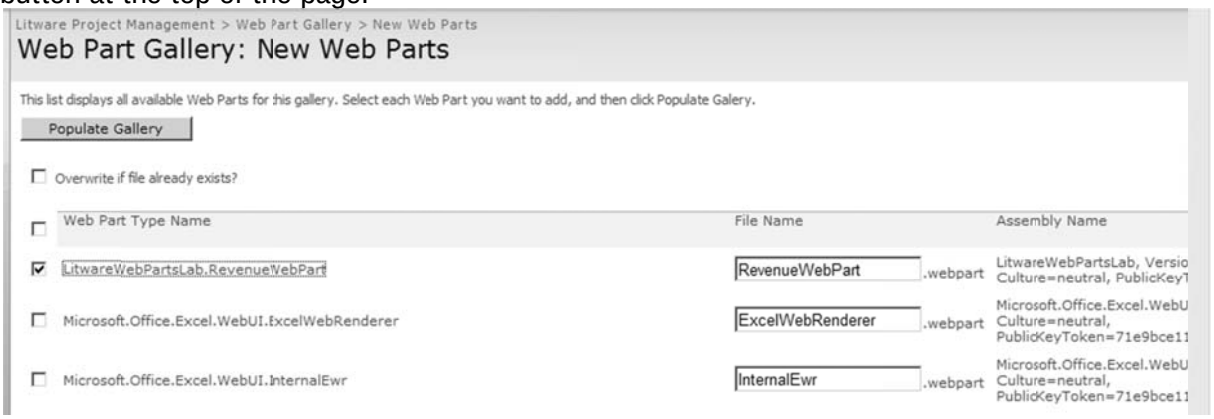
ii. **Note:** you can obtain the full 4-part assembly name using the **Reflector** utility that is in your \Student\Resources directory.
    1. In your **..\Student\Resources** directory open the **Reflector** utility (double click on **Reflector.exe**).
    2. In the Reflector Utility you will need to examine the **LitwareWebPartsLab.dll**. **File > Open...** (VB.NET or C#) Navigate to the **C:\Inetpub\wwwroot\wss\VirtualDirectories\litwareinc.com80** directory open the **LitwareWebPartsLab.dll** file.
    3. Select the **LitwareWebPartsLab** Assembly in the main Reflector utility window and look at the bottom for the **Name:** entry. You can copy this entry to use for your **SafeControl** setting.
    4. When finished your entry should look like this

```
<SafeControl Assembly="LitwareWebPartsLab, Version=1.0.0.0,
Culture=neutral,
PublicKeyToken=null" Namespace="LitwareWebPartsLab" TypeName="*"
Safe="True" />
```

Note: this needs to be on **one line** even though it is spread across several for display here
Extra Information: The **Assembly** attribute comes from the dll name while the **Namespace** is the exact namespace where the class file exists. **TypeName** is the class name (* means all classes in the namespace). **Safe** is whether you may use this on a SharePoint implementation.

iii. **Note**: As the **Web.config** is **case sensitive**, ensure that you type **LitwareWebPartsLab** in EXACTLY or it will not work.
iv. **Save** your changes.

10. Now that you have compiled the Web Part DLL into the **\bin** directory and configured it in the **SafeControl** list of the default Web Site, you can now add the **RevenueWebPart** Web Part to the Web Parts gallery of a top-level site.
    a. Go to the **Project Management** site (**http://litwareinc.com/sites/ProjectManagementLab/**) and click the **Site Settings** command from the **Site Actions** menu.
11. Next we need to add the web part to our site collection so that we may utilize it on our Web Part Pages.
    a. On the **Site Settings** page, click on the **Web Parts** link in the **Galleries** section to navigate to the **Web Part Gallery** page.
    b. Click the **New** button on the toolbar to navigate to the page that allows you to add new Web Parts to the gallery.
    c. You should see **LitwareWebPartsLab.RevenueWebPart** as a selection. Click the check box to the left of the Web Part to select it, and then click the **Populate Gallery** button at the top of the page.



    d. On the **Web Part Gallery** page, verify that the **RevenueWebPart.webpart** has been added to the list. It should be listed alphabetically, and also it should have the green **!NEW** marker next to it.
12. Now that you have added the **RevenueWebPart** Web Part to the gallery, you can add it to any page in the current site collection.
    a. Navigate to the home page of the **Project Management** site.
    b. On the **Litware Project Management** home page select **Edit Page** from the **Site Actions** menu.
    c. Click **Add a Web Part** in the **Web Part zone** on the right-hand side. Note: the **RevenueWebPart** is located in the **Miscellaneous Web Part** Category; you will need to scroll down to locate it. Check the box next to it, and click **ADD**.
    d. Drag the newly added web part below the Litware Logo web part.

e. When you are done, the Web Part should look like this:

f. Be sure to click on the **Exit Edit Mode** hyperlink (near the top right corner of the page).

Exercise 2: Add a persistent property to a Web Part

In this exercise you will add a property to your Web Part class and expose it to SharePoint.

1. Modify the **RevenueWebPart** so that it contains a persistent Boolean personalizable property named **ShowTextAsBold**. The property should give the user the ability to toggle it on and off within the task pane of the browser. When the user has the property disabled, the **RevenueWebPart** should render its output using a standard font as you have already done. When the property is enabled, the **RevenueWebPart** should render its output using a bold font with a larger size. If you're in the mood, you can also change the color of the font.
   a. Open the **RevenueWebPart** in Visual Studio 2008.
   b. First you need to create a protected Boolean variable called **_ShowTextAsBold**. Next we will create a public property called **ShowTextAsBold** utilizing the **_ShowTextAsBold** for internal storage and add attributes to the property to expose this property in SharePoint.

C#

```
public class RevenueWebPart : WebPart {

  protected bool _ShowTextAsBold = true;

  [ Personalizable(),
    WebBrowsable(true),
    WebDisplayName("Show Text As Bold"),
    WebDescription("Enable to turn on bold font output") ]
  public bool ShowTextAsBold {
    get { return _ShowTextAsBold; }
    set { _ShowTextAsBold = value; }
  }

  protected override void RenderContents(System.Web.UI.HtmlTextWriter
writer)...
```

VB.Net

```
Public Class RevenueWebPart Inherits WebPart
```



```
Protected _ShowTextAsBold As Boolean = True

 < Personalizable(), _
   WebBrowsable(True), _
   WebDisplayName("Show Text As Bold"), _
   WebDescription("Enable to turn on bold font output")> _
Public Property ShowTextAsBold() As Boolean
  Get
    Return _ShowTextAsBold
  End Get
  Set
    _ShowTextAsBold = value
  End Set
End Property

Protected Overrides Sub RenderContents(writer As
System.Web.UI.HtmlTextWriter)...
```

c. Now we need to utilize this setting in our **RenderContents** method. If the **ShowTextAsBold** property is set to "**true**" then we will Write out "**Hello, world**" in a **12pt, Bold, Red Font.** If "**false**" we will just write out "**Hello, world**" in a **plain** default font, as before.

C#

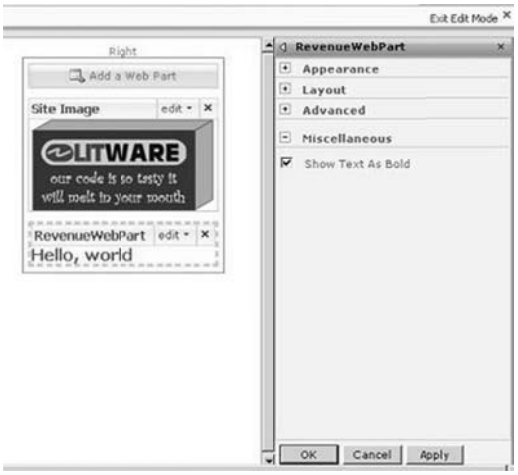
```
protected override void RenderContents(System.Web.UI.HtmlTextWriter writer)
{

 if (_ShowTextAsBold) {
   writer.AddStyleAttribute(HtmlTextWriterStyle.FontWeight,
"Bold");
   writer.AddStyleAttribute(HtmlTextWriterStyle.FontSize,
"12pt");
   writer.AddStyleAttribute(HtmlTextWriterStyle.Color, "Red");
   writer.RenderBeginTag(HtmlTextWriterTag.Span);
   writer.Write("Hello, world");
   writer.RenderEndTag(); // </Span>
 }
 else {
   writer.Write("Hello, world");
 }

}
```

VB.Net

```
Protected Overrides Sub RenderContents(writer As
System.Web.UI.HtmlTextWriter)

 If _ShowTextAsBold Then
   writer.AddStyleAttribute(HtmlTextWriterStyle.FontWeight,
"Bold")
   writer.AddStyleAttribute(HtmlTextWriterStyle.FontSize, "12pt")
   writer.AddStyleAttribute(HtmlTextWriterStyle.Color, "Red")
   writer.RenderBeginTag(HtmlTextWriterTag.Span)
   writer.Write("Hello, world")
   writer.RenderEndTag() ' </Span>
```

```
 Else
   writer.Write("Hello, world")
 End If

End Sub
```

d. **Build** your project.
e. Navigate to your **Litware Project Management** home page. (http://litwareinc.com/sites/ProjectManagementLab/)
Note: If you still have this page open, refresh it (**F5**) or the following steps will not work correctly.
f. You should immediately notice that "**Hello, world**" is bigger, bolder, and redder than before.
g. Click on the small drop down arrow in the upper right corner of your **RevenueWebPart** and select **Modify Shared Web Part**.
h. Expand out the **Miscellaneous** section (i.e. click on the **+** next to the word **Miscellaneous**). The Web Part should appear as below in the browser.



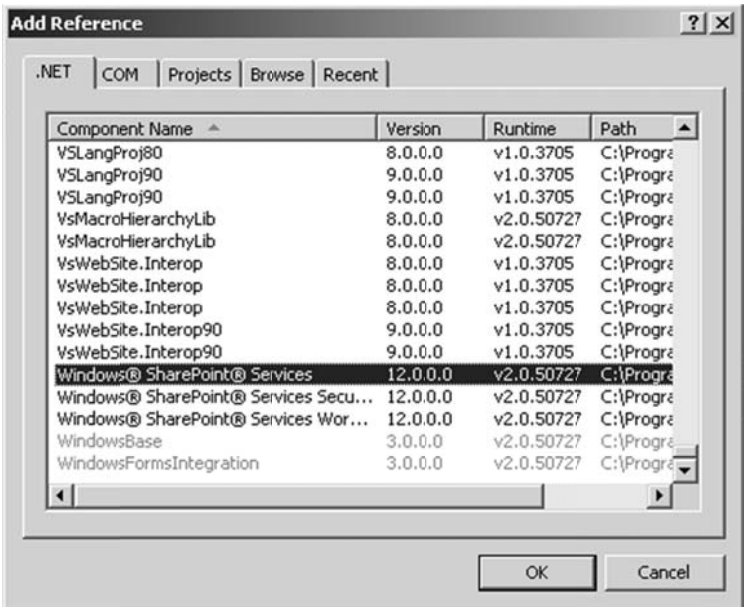
i. Remove the check from the **Show Text As Bold** checkbox and click the **OK** button
j. Your **RevenueWebPart** should now show "**Hello, world**" in regular black text.

Exercise 3: Use the WSS object model to read data from a SharePoint list

In this exercise, you will modify the **RevenueWebPart** to show data from the **Projects** list. Up to this point, the Web Part has been written as a standard ASP.NET Web Part that can run in any ASP.NET site. Now, it's time to add WSS-specific code to the DLL so it can access data within a WSS list.

1. Begin by adding a reference to **Microsoft.SharePoint.dll**.
    a. In your **Solution Explorer** right click on your **LitwareWebPartsLab** project and select **Add Reference...**

b. Scroll down and select **Windows SharePoint Services** and click **OK.**



2. Now you should bring the **Microsoft.SharePoint** namespace into scope in your **RevenueWebPart** class.

C#
```
using Microsoft.SharePoint;
```

VB.Net
```
Imports Microsoft.SharePoint
```

3. Next, inside the **RenderContents** method we will modify this code to access data within our **Projects** list.
   a. You will add this new code **before** the existing **IF ELSE** statement.
   b. Write the code to acquire a reference to the current site (i.e. **SPWeb** object) and then obtain a reference to the **Projects** list (i.e. **SPList** object).

C#
```
SPWeb ProjectManagementSite = SPContext.Current.Web;
SPList Projects = ProjectManagementSite.Lists["Projects"];
```

VB.Net
```
Dim ProjectManagementSite As SPWeb = SPContext.Current.Web
Dim Projects As SPList = ProjectManagementSite.Lists("Projects")
```

   c. Next, enumerate through the list items one by one and add the contract amounts together to calculate a total.

C#
```
decimal TotalRevenue = 0;
foreach (SPListItem Project in Projects.Items) {
  TotalRevenue += Convert.ToDecimal(Project["Contract Amount"]);
}
```

VB.Net
```
Dim TotalRevenue As Decimal = 0
Dim Project As SPListItem
For Each Project In Projects.Items
  TotalRevenue += Convert.ToDecimal(Project("Contract Amount"))
Next Project
```

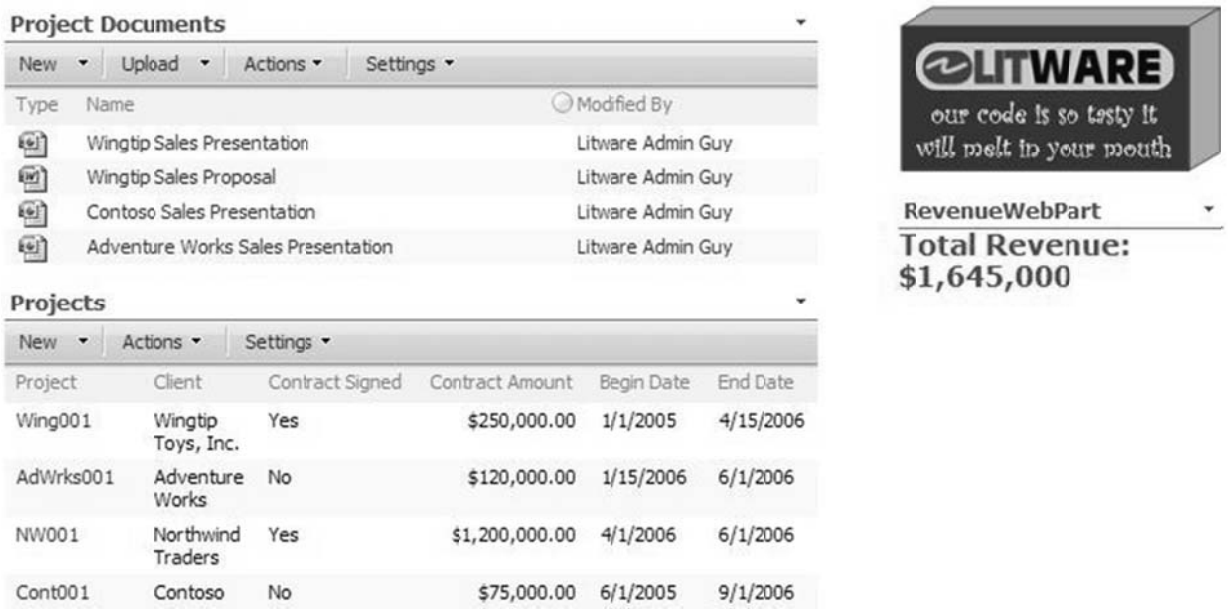
   d. Finally, modify the existing **IF ELSE** statement to display the total summation of the contract amounts as the **total revenue**. Be sure to format this total so that it has a US currency display.

C#
```
if (_ShowTextAsBold) {
  writer.AddStyleAttribute(HtmlTextWriterStyle.FontWeight, "Bold");
  writer.AddStyleAttribute(HtmlTextWriterStyle.FontSize, "12pt");
  writer.AddStyleAttribute(HtmlTextWriterStyle.Color, "Red");
  writer.RenderBeginTag(HtmlTextWriterTag.Span);
  writer.Write("Total Revenue: " + TotalRevenue.ToString("$#,###"));
  writer.RenderEndTag(); // </Span>
}
else {
  writer.Write("Total Revenue: " + TotalRevenue.ToString("$#,###"));
}
```

VB.Net
```
If _ShowTextAsBold Then
```

```
  writer.AddStyleAttribute(HtmlTextWriterStyle.FontWeight, "Bold")
  writer.AddStyleAttribute(HtmlTextWriterStyle.FontSize, "12pt")
  writer.AddStyleAttribute(HtmlTextWriterStyle.Color, "Red")
  writer.RenderBeginTag(HtmlTextWriterTag.Span)
  writer.Write(("Total Revenue: " +
TotalRevenue.ToString("$#,###")))
  writer.RenderEndTag() ' </Span>
Else
  writer.Write(("Total Revenue: " +
TotalRevenue.ToString("$#,###")))
End If
```

e. **Build** your project.
f. Before you test your code, make sure you have updated the trust level in the **web.config** file to a setting value of **WSS_Medium** as discussed in Exercise 1 Step 8. Since you have just begun to program against the WSS object model, this is the point in the lab where your code will begin to fail if you have not changed the trust level from the default setting of **WSS_Minimal**. Note that you should always run an **IISRESET** operation when you have changed the trust level in the **web.config** file. Do this now by going to **START > RUN**, and then entering **iisreset** into the textbox.
g. Navigate to your **Litware Project Management** home page (http://litwareinc.com/sites/ProjectManagementLab/)
Note: If you still have this page open refresh it (**F5**) or the following steps will not work correctly.
h. You should immediately notice that "**Hello, world**" has been changed to "**Total Revenue: $1,645,000**".
i. Click on the s**mall drop down arrow** in the upper right corner of your **RevenueWebPart** and select Modify Shared Web Part.
j. Expand out the **Miscellaneous** section (i.e. click on the **+** next to the word **Miscellaneous**).
k. Add a check in the **Show Text As Bold** checkbox and click the **OK** button
l. Your **RevenueWebPart** should now display "**Total Revenue: $1,645,000**" in a bigger, bolder, and redder font than before.
m. Your Web Part output should look like the screenshot below. When you test your web part, verify that you can see your Web Part dynamically update its output when you add and/or modify projects in the Projects list.



Lab 09: Site Columns and Content Types

Lab Time: 45 Minutes

Lab Directory: C:/ Student/Labs/09_ContentTypes

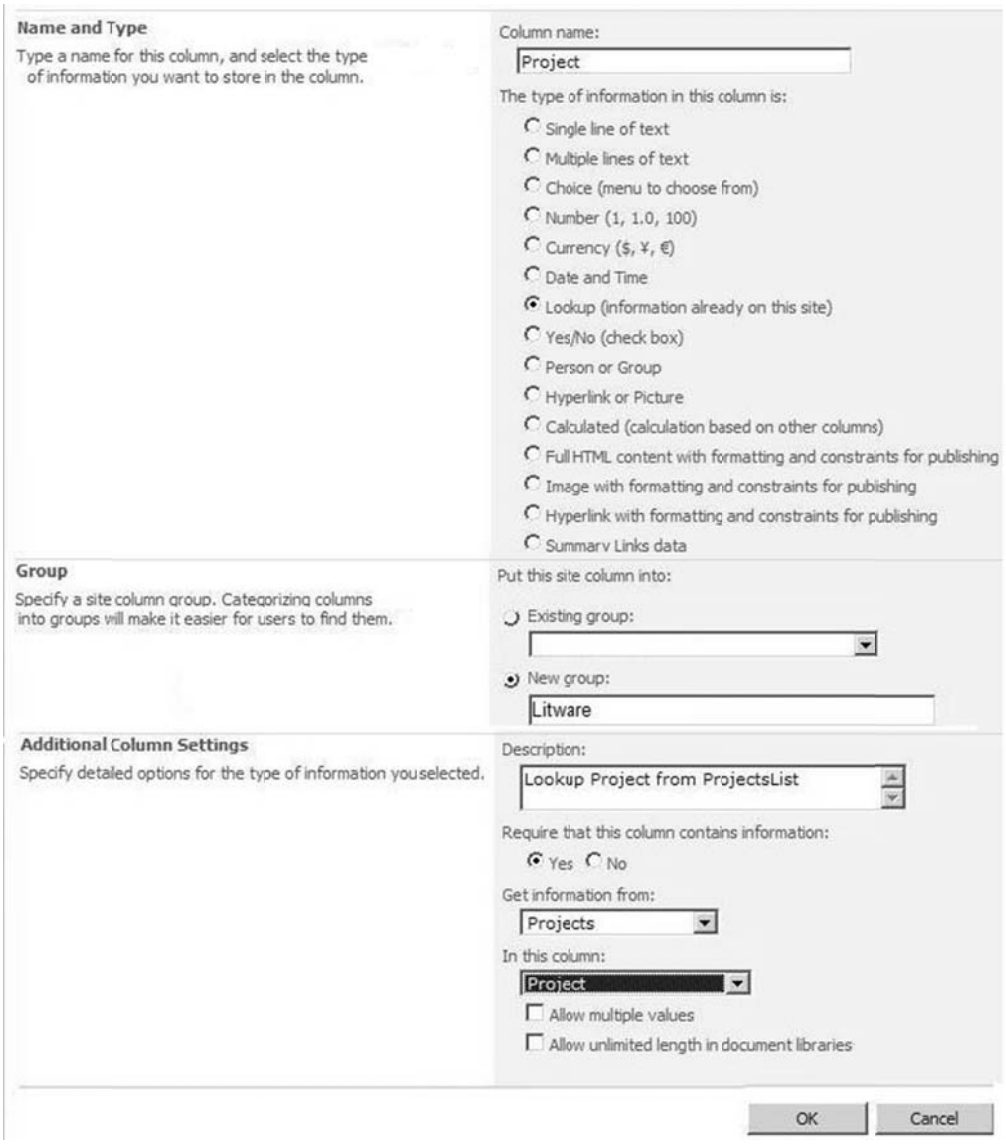
Lab Overview: In this lab you will add tracking functionality—managing timesheets for Litware consultants. You will also improve the way that content is stored in the Project Management site and its child sites by using two new features introduced with WSS 3.0: site columns and content types.

1. In the first Exercise, you will create a site column definition in the top-level **Project Management** site that performs a lookup on the **Projects** list you created in the earlier lab.
2. In Exercise two, you will use this Lookup site column to define a column in the list of a child site (the **North Division** sub-site) within that same site collection.
3. In the Third Exercise, you will create several new content types that allow users to store their documents in the **Project Documents** document library in a more structured way.

Exercise 1: Create a site column in the top-level site for doing Project lookups

In this first exercise, you will create a site column in the top-level **Project Management** site that performs look ups on the **Project** column of the **Projects** list.

1. Start by navigating to the **Project Management** site you created in the first lab on Customizing a WSS site. (http://litwareinc.com/sites/ProjectManagementLab/default.aspx).
2. Click the **Site Settings** command in the **Site Actions** menu to navigate to the **Site Setting** page.
3. Then click the **Site columns** link under the **Galleries** section to navigate to the **Site Column Gallery** page.
4. Click the **Create** link on the toolbar to navigate to the **New Column** page where you will be able to create a new site column:
    a. Name the new site column **Project**.
    b. Make the new site column a **Lookup** column.
    c. Add the new column to a new group named **Litware**.
    d. In the **Additional Column Settings** section add a brief description and modify the site column so it requires information.
    e. Next, assign the **Projects** list as the source the lookup should get its information from.
    f. Then assign the **Project** column as the column to be used in the lookup.
    g. When you are done filling out the **New Column** page and it looks like the one below, click **OK** to create the new site column.



Exercise 2: Create a custom list for tracking consultant timesheet information in the North Division sub-site using the Project lookup Column

Now that you have created the **Project** site column, it's time to use it in a new list. You will create a list that uses this site column in the **North Division** site (you created this in Lab 1) that is a child site of the **Project Management** site. This scenario will demonstrate how a site column can easily enable lookups across sites within a site collection.

1. Begin by navigating to the **North Division** site (located on the tabs near the top of the **Litware Project Management** site).

2. Over the next few steps, you will create a new SharePoint custom list which we will then modify for tracking consultant timesheet information.
    a. Start by clicking the **Create** command from the **Site Actions** menu to navigate to the **Create** Page.
    b. Create a new list by clicking the **Custom List** link in the **Custom Lists** section. This will take you to a page that allows you to name the new list.
    c. Name the new list **Timesheets** and click **Create**.
3. Once the **Timesheets** list has been created, locate and drop down the **Settings** menu on the list's AllItems.aspx page and click the **List Settings** command. This will take you to the list settings page that has a title of **Customize Timesheets**.
    a. From this page, click on the **Versioning Settings** link in the **General Settings** section.
    b. Modify the list from the **Versioning Settings** page to enable the option to **Create a version each time you edit an item in this list** and click **OK** to return to the list settings page. In this case, you will be able to track whenever and whoever makes changes to timesheet items in the list.
4. Now it's time to define the columns needed for the **Timesheet** list using the **List Settings** page. In this step, you will modify the **Timesheets** list so its set of columns matches the set of columns defined below.

Column Name	Type	Notes
TimesheetID	Single Line of Text	Do not create this column as a new column. Instead, rename the **Title** column to **TimesheetID**.
Consultant	Choice	To simplify this lab, make this a choice column with three choices for the consultant. The choices should be "**Britta Simon**", "**John Rodman**" and "**Mike Fitzmaurice**"
Project	Lookup	This column should be based on **Project** site column created earlier in this exercise. Create this by clicking the **Add from existing site columns** link, choosing **Litware** from the dropdown list, clicking on **Project** in the listbox, clicking **Add**, and then clicking **OK**.
Submitted On	Date and Time	Format this column to show date only. Make this a required column and make the default value the current date (i.e. "**Today's Date**".
Hours	Number	Make this a required column and set the minimum value to 1 hour and maximum to 24 hours. The maximum is due to the fact that each timesheet item will be for only one specific day.
Hourly Rate	Currency	Make this a required column and set currency formatting to whatever seems most appropriate
Created By	Person or Group	You do not have to create this column. It is automatically created when you create a new list
Modified By	Person or Group	You do not have to create this column. It is automatically created when you create a new list

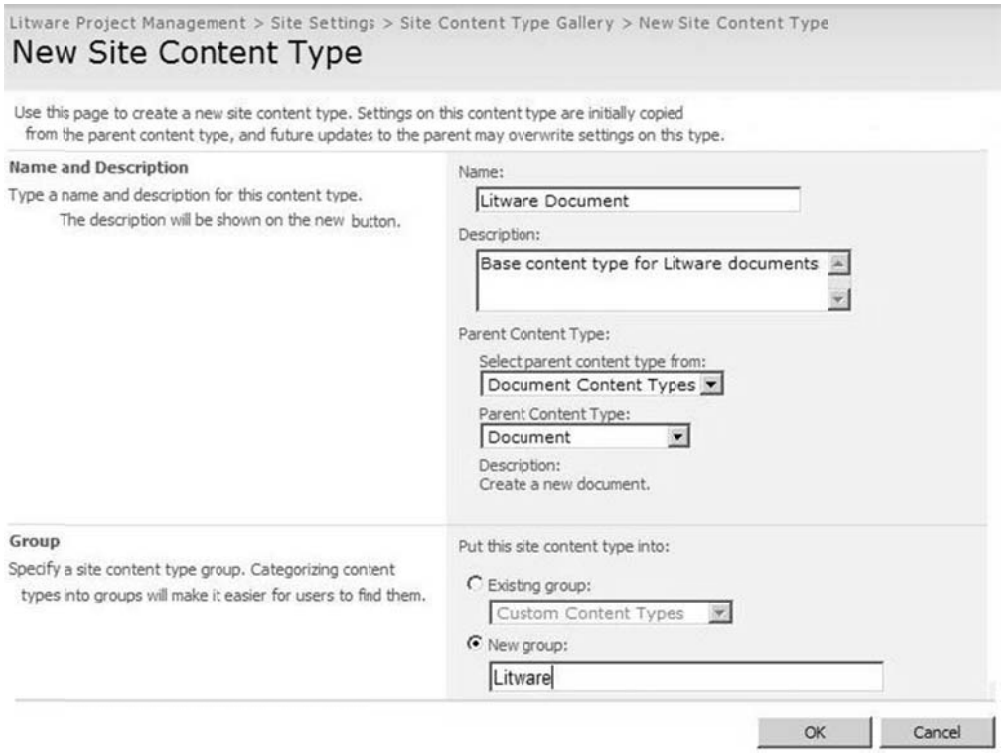
5. When you are done creating the **Timesheets** list structure, add five timesheet items so that you have some test data to work with. Add the data from the items below. You should see that the Project column is performing lookup from the Projects list in the parent **Project Management** site.

TimesheetID	Consultant	Project	Submitted On	Hours	Hourly Rate
Timesheet01	Britta Simon	AdsWks001	January 2, 2007	6	50
Timesheet02	Britta Simon	AdsWks001	January 3, 2007	5	50
Timesheet03	Britta Simon	AdsWks001	January 4, 2007	8	50
Timesheet04	Britta Simon	AdsWks001	January 5, 2007	4	50
Timesheet05	Britta Simon	AdsWks001	January 6, 2007	3	50

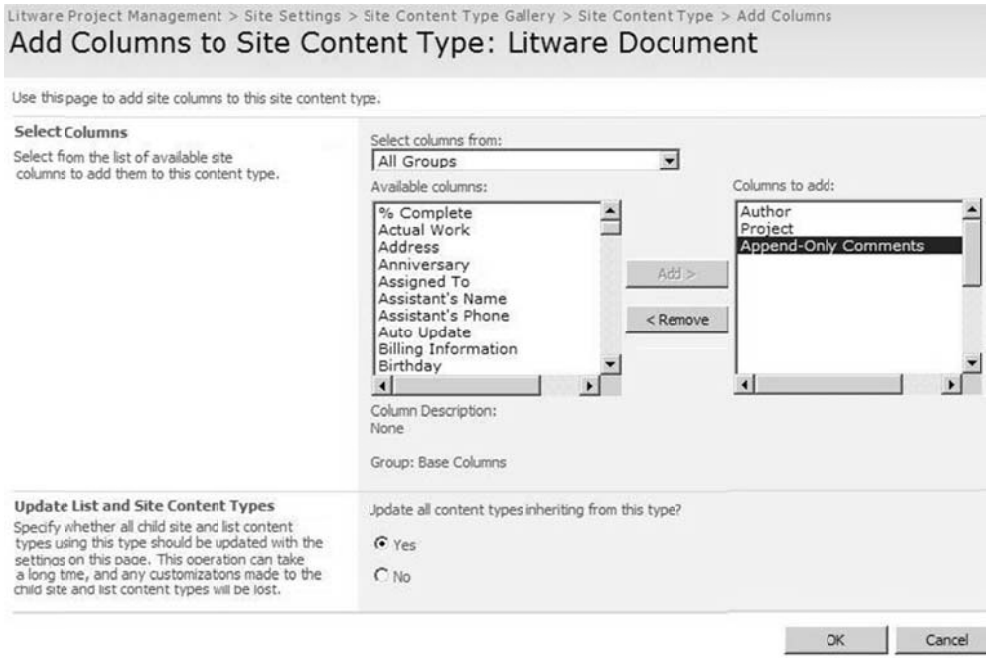
Exercise 3: Create a document library that uses custom content types

In this exercise, you will create a few new custom content types and use them within the **Project Documents** document library. First, you will create a content type named **Litware Document** that will serve as a base content type. Next you will create two derived content types named **Litware Proposal** and **Litware Presentation** that derive from this base content type.

1. Start by navigating back to the **Litware Project Management** site (i.e. click on the **Home** tab near the top left of your screen).
2. Click the **Site Settings** command in the **Site Actions** menu to navigate to the **Site Setting** page.
3. Then click the **Site content types** link under the **Galleries** section to navigate to the **Site Content Type Gallery** page.
4. Click the **Create** link in the toolbar to navigate to the page that allows you to create a new content type. Fill in the page with the information that is supplied below. When you are finished be sure to click **OK**.



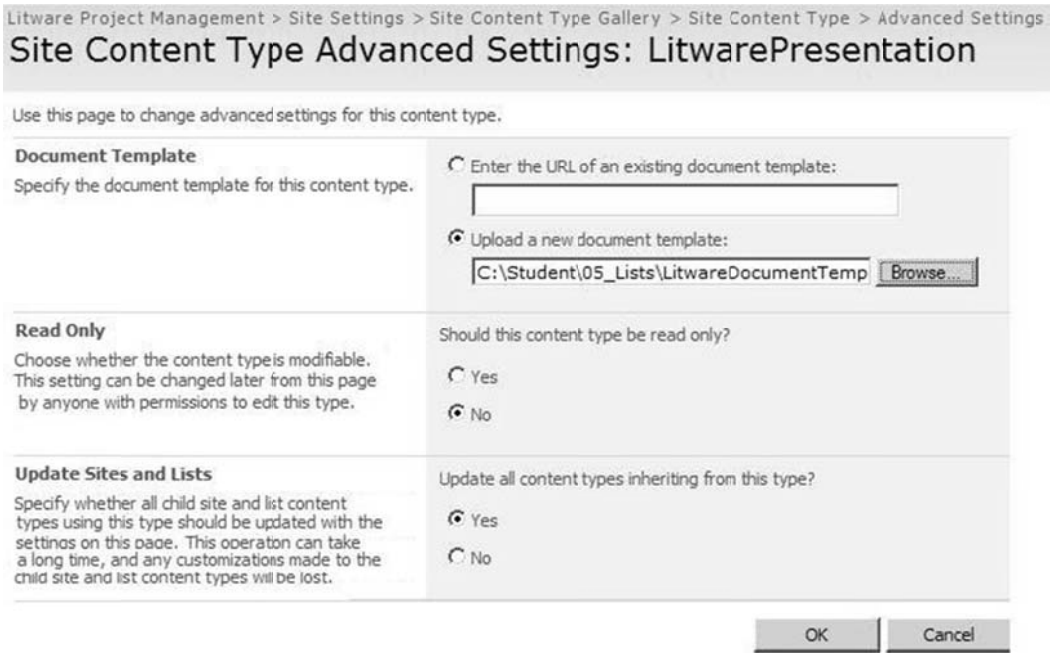
5. Next, add three columns to the new **Litware Document** content type from the existing set of site columns (click on **Add from existing site columns** link). In particular:
    a. Add the **Author** column
    b. Add the **Project** column
    c. Add the **Append-Only Comments** column.
    d. Note: the **Author** column and the **Append-Only Comments** column are provided as standard WSS site columns while the **Project** column is the custom one you created in the previous lab exercise.



6. Click **OK** to finish adding columns to your **Litware Document** content type and then navigate back to the **Site Content Type Gallery**.
7. Over the next few steps you will create two derived content types: **Litware Proposal** and **Litware Presentation**. These two content types will inherit **Litware Document** and, therefore, automatically contain the columns you created in the previous steps. You will extend these two derived content types by giving them different document templates.
    a. Create a new content type named **Litware Presentation** that inherits from **Litware Document**. To do this, fill in the new content type page using the information shown below and click **OK**.

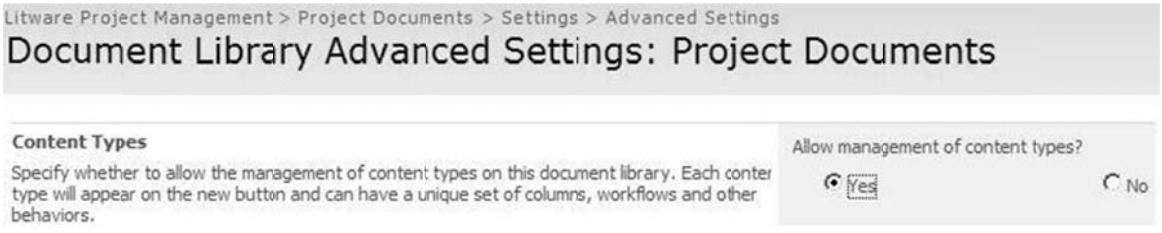
8. After you click **OK** to create the **Litware Presentation** content type, you will be taken to a page that shows the settings and columns for this new content type.
    a. You should observe that the **Litware Presentation** content type contains the same columns that you defined in the base content type **Litware Document**.
    b. Now click on the **Advanced Settings** link to get to the page that allows you to upload a document template for this content type.
    c. Click on **Upload a new document template:**, and the click **Browse**. Use the template file named **LitwarePresentationTemplate.potx** that is located in the **..Labs\09_ContentTypes\LitwareDocumentTemplates** directory. Once you locate this file be sure to click **Open**.

d. Click **OK** when you are done to upload the document template.



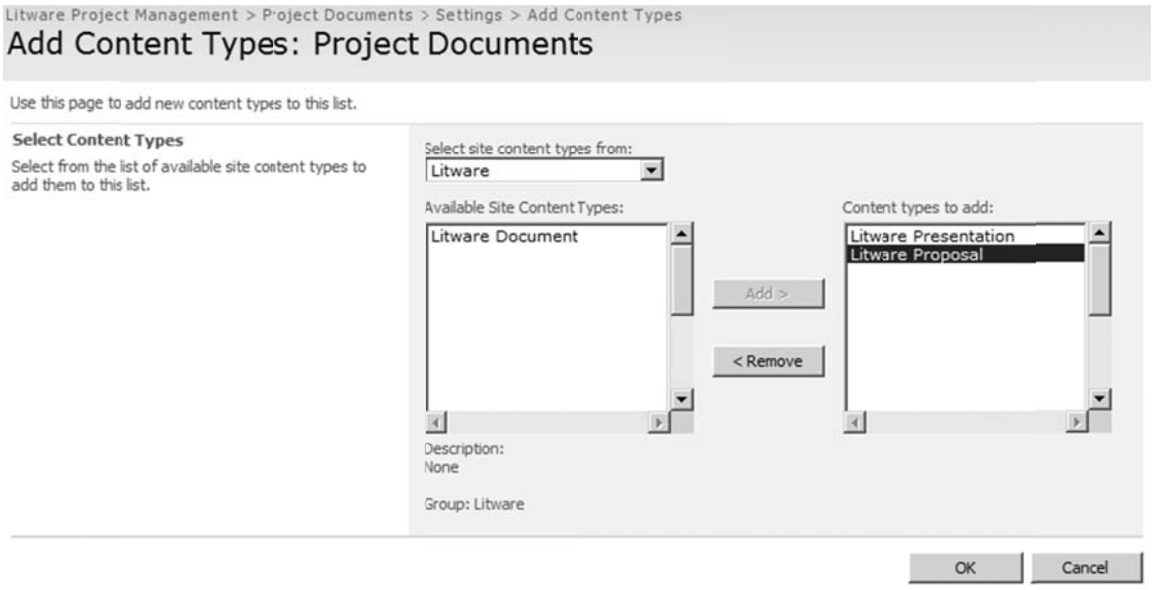
e. Click on **Site Content Type Gallery** in the breadcrumb list at the top of the page to return back to the **Site Content Type Gallery**.

9. Create a second derived content type named **Litware Proposal** using the exact same steps you used to create the **Litware Presentation** content type. The only thing you should do differently other than giving this content type a different name is to use **LitwareProposalTemplate.dotx** as the file for the document template instead of **LitwarePresentationTemplate.potx**. When you are done with this step you will have two content types that you can now use within the document library named **Project Documents**.
10. Now you will modify an existing document library to utilize your new content types
    a. Navigate back to your **Litware Project Management** site (i.e. Click on the **Home** tab)
    b. Navigate to your document library named **Project Documents.**
    c. Go to the **Document Library Settings** page for the **Project Documents** library (i.e. **Settings** Drop Down > **Document Library Settings**).
    d. Click the **Advanced Settings** link to get to the page where you can enable the setting that allows for multiple content types as shown below. Click **OK** after you have enabled this setting.



11. Add the content types **Litware Proposal** and **Litware Presentation** to the document library.
    a. In the **Customize Project Documents** page locate the **Content Types** section and click on the **Add from existing site content types** link.
    b. Select **Litware** in the **Select site content types from** dropdown.

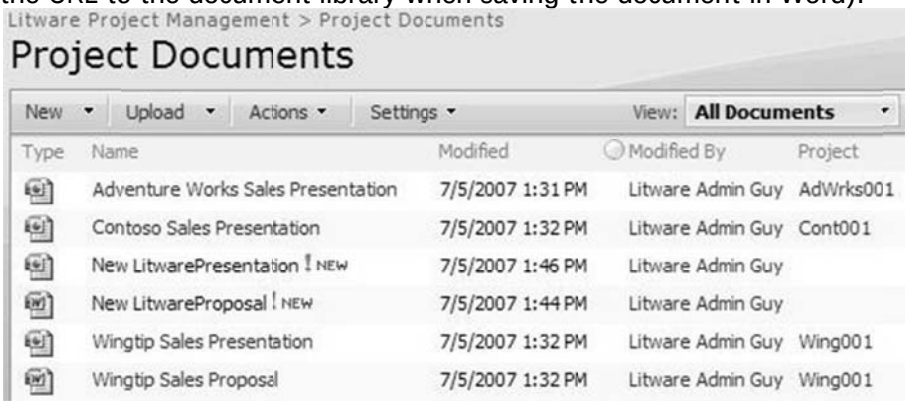
c. select **Litware Presentation** and **Litware Proposal** in the **Available Site Content Types** list and click **Add**. Be sure to click **OK** when done



12. Now, navigate back to your **Project Documents** page and examine the files that are already in the **Project Documents** document library by choosing the **Edit Properties** command in the drop-down menu for each individual document. This will bring you to a page that allows you to change the content type for each document.
    a. Change the content type for each document in the document library to either **Litware Proposal** or **Litware Presentation** (**Hint**: match up the names of the Documents with their associated document type—Proposal to Proposal, and Presentation to Presentation).
    b. When you are done, be certain that there are no documents still assigned to the standard content type named **Document**.
13. Go back to the **Document Library Settings** page for the document library named **Project Documents** (**Settings** > **Document Library Settings**) and remove the original content type named **Document**.
    a. On the **Customize Project Documents** page in the **Content Types** section click on the **Document** Content type which will take you the **List Content Type: Document** page
    b. On the **List Content Type: Document** page in the **Settings** section click on **Delete this content type** and click on **OK** on the popup window that appears.
    c. **Note**: you will not be able to remove the **Document** content type if there are still one or more documents in the document library that have this content type. If you have trouble removing the **Document** content type, go through each document in the **Project Documents** library and make sure they are configured with a content type of either **Litware Proposal** or **Litware Presentation** and not **Document**.
14. **IMPORTANT:** In a single machine virtual world (like our lab setup) you may encounter problems saving to the document library (weird error about how there is no connectivity) or using the document information panel tie in with SharePoint (appears in each Office app near the top of the document) . To fix this we must **stop** the **System Event Notification Service**.
    a. Open a command prompt (**start menu - run -** type **cmd -** press **enter**).

b. From the cmd prompt type **net stop sens** and press **enter** this will stop the System Event Notification Service and you should now be able to use the Office-SharePoint integration features.

15. Use the **New** drop down menu of the **Project Documents** library to create and save a new Litware proposal and a new Litware presentation.
    a. Give these new documents titles like **Presentation Test** and **Proposal Test**.
    b. Allocate both of these documents to the **NW001 Project**.
    c. Save these documents back to the **Project Documents** library with the names: **New Litware Presentation** and **New Litware Proposal**
    d. When you have successfully completed the lab steps up through this point, the **New** menu should look like the one shown below. (It is possible that you first have to enter the URL to the document library when saving the document in Word).



16. **Important:** You will notice that in this view Project information is only showing up for the existing documents and not for the two new ones you just added... let's investigate this behavior.
    a. Navigate to the Project Document Settings page **(Settings** > **Document Library Settings)**
    b. Look at the Columns section, note that there are **TWO** Project items: One from the Document Library itself (created in Lab 01A) and the other from our Content Types. (i.e. Even though these two columns are identical in information because they come from two separate sources they are stored as separate information.)
    c. **As time permits:** To fix this you would need to **remove the Project Column that is not Used in anything** (i.e. you remove the Project setting marked remove in the figure below) and then you would need to re-enter the project information in the old documents (that now utilize the Content Types we created). As well you would need to

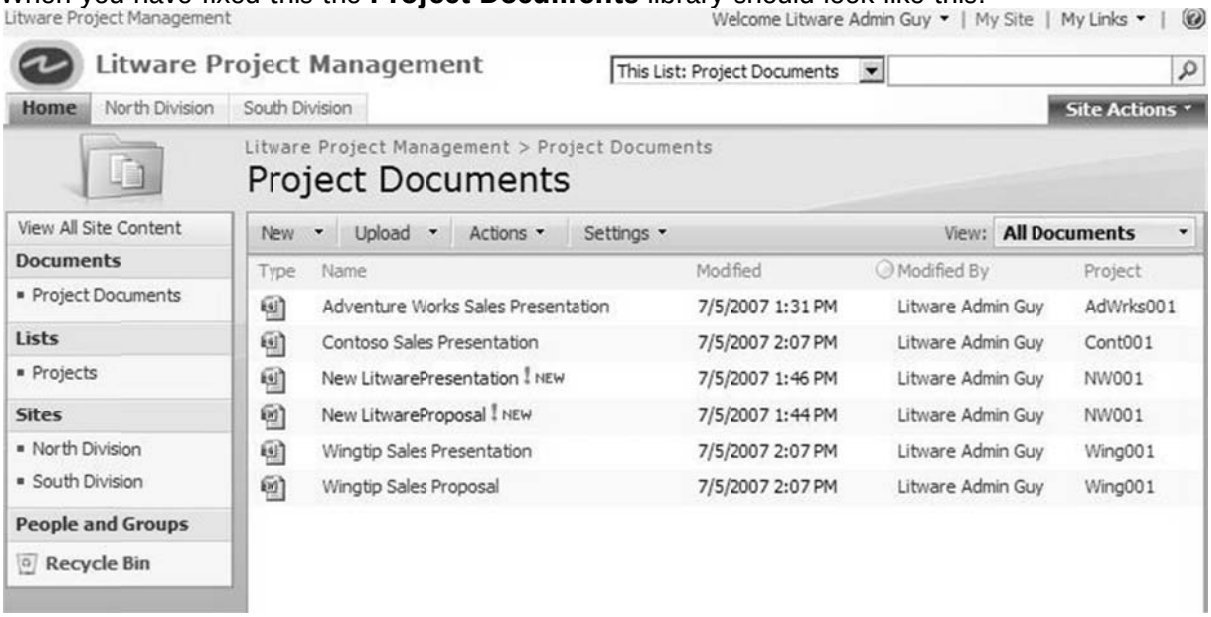
add this Project Column to the default view.

Columns

A column stores information about each document in the document library. Because this document library allows multiple content types, some column settings, such as whether information is required or optional for a column, are now specified by the content type of the document. The following columns are currently available in this document library:

Column (click to edit)	Type	Used in
Append-Only Comments	Multiple lines of text	LitwarePresentation, LitwareProposal
Author	Single line of text	LitwarePresentation, LitwareProposal
Project	Lookup	LitwarePresentation, LitwareProposal
Project	Lookup	This is the one we need to remove
Title	Single line of text	LitwarePresentation, LitwareProposal
Created By	Person or Group	
Modified By	Person or Group	
Checked Out To	Person or Group	

d. When you have fixed this the **Project Documents** library should look like this:



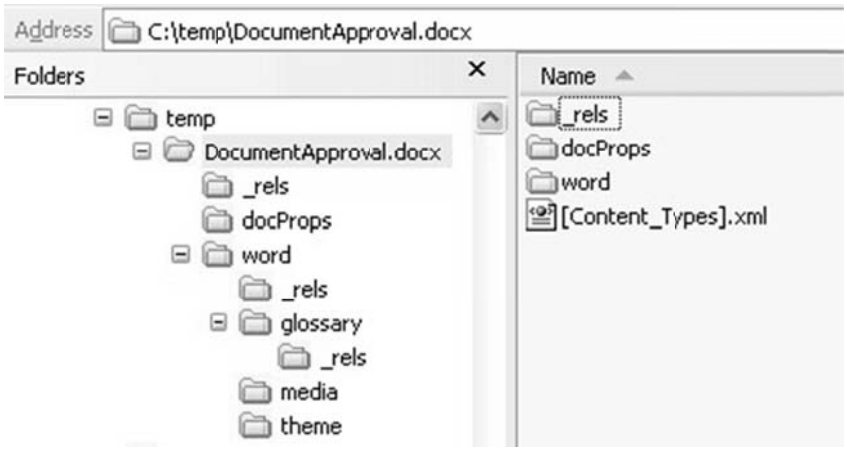
Lab 10: Integration with Office Open XML File Formats

Lab Time: 30 Minutes

Lab Directory: C:/Student/Labs/10_OpenXml

Lab Overview: Litware has 100s of Case Studies gathered through the years. Our goal is to move all those documents into SharePoint. But before that, your job is to build a tool that helps administrators discover what we have in our inventory of Case Studies.

Exercise 1: Examine a DocX file through Windows Explorer

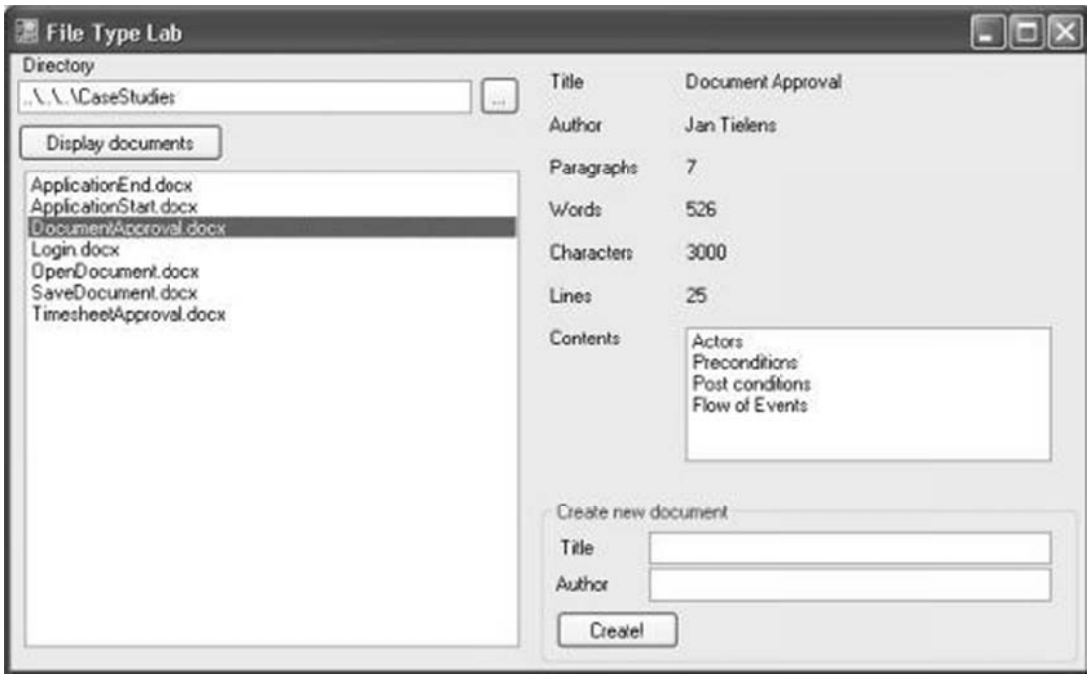


1. Copy the **CaseStudies** folder from the **StarterFiles** folder to the **Lab** folder and open the file **Lab\CaseStudies\DocumentApproval.docx** in Word (you can find it in the **StarterFiles\CaseStudies** directory), by double clicking on it. Take a look at the contents of this file. Notice that there is an image and some headings.
2. Close Word and rename the **DocumentApproval.docx** file to **DocumentApproval.docx.zip** and open it.
3. Navigate to the word directory in the zip file and open the **document.xml** file. Review the content of this **WordML** file and search for the different headings in the XML.
4. Close the **document.xml** file and navigate to the media directory, then open the **image1.jpeg** file. Close the file.
5. Navigate back to the **word** directory and then to the **_rels** directory and open the **document.xml.rels** file. In this file you'll find the **rId6** relationship that points to the image you've just opened. Close the file.
6. Next open the **document.xml** file in the word directory to find where the image is referenced in the xml. Close the file.
7. Finally go back to the root of the zip file, navigate to the **docProps** directory, and open the **app.xml** where you'll find all the application specific properties for this document. Close the file.
8. Let's make a small change to the Word document without actually using the Word application. Extract the contents of the **DocumentApproval.docx.zip** to a folder by right

clicking on the zip folder and choosing **Extract All**. Now go to the newly extracted folder (if you accepted the defaults in the extract process, then it will be in the same directory as the **zip** folder) and navigate to the **word** directory.

9. Open the file footer2.xml in Microsoft Visual Studio 2008 and look for the word Confidential, change it and any surrounding text to Litware Confidential 2008 - Don't distribute this document! Save the changes.
10. Now we need to copy **the footer2.xml** file back to the **zip** folder and compress it back into a **docx** file by doing the following:
    a. After you have saved the **footer2.xml** file, go to the directory you saved it into (the folder that you extracted from the zip file).
    b. Right click on **footer2.xml** and click **Copy**.
    c. Navigate to **DocumentApproval.docx.zip > word** and paste the **footer2.xml** file there—you will have to approve that you want to replace the existing file.
    d. Navigate back to the **Case Studies** folder.
    e. Delete the extracted folder called **DocumentApproval.docx**.
    f. Change the name of DocumentApproval.docx.zip back to DocumentApproval.docx.
    g. Verify the changes by opening the document in Word.

Exercise 2: Build the Case Study explorer tool



1. Copy the **FileTypeLab** folder from the **StarterFiles** into the **Lab** Folder.
2. Open **Visual Studio 2008**, and go to **File>Open>Project/Solution**, navigate to the folder you just copied, and open the **FileTypeLab.sln** file.
3. Start the application (press **F5**) to check out what's already there. When you click the ellipsis button [...] to the right of the text box where the directory name is displayed, you can select a directory that will be processed.
4. Read the following sub-point for understanding. Note: you are not being asked to complete a task. This is just for explanation:

5. Once you've selected a directory you can click the **Display documents** folder that will load all files with the **.docx** extension in the list. Selecting a document does not do anything yet. We expect you to write this code. Also the **Create** button at the bottom right of the form isn't implemented yet. If you have time, this can be done as an optional exercise.
6. Now close the application so we can start writing some code.
7. First, add a reference to **WindowsBase.dll** which contains the new .NET 3.0 packaging API definitions within the **System.IO.Packaging** namespace. Note that you might not see this assembly as one of the component names inside the .NET tab of the **Add References** dialog. If this is the case, you will have to use the Browse tab to navigate to the following directory to add **WindowsBase.dll** as a reference within the current project:

```
C:\Program Files\Reference Assemblies\Microsoft\Framework\v3.0
```

8. Next open the code for Form1 and add the following using statements at the top:

```
using System.IO;
using System.Xml;
using System.IO.Packaging;
```

9. Open the region named Exercise 2, and add the following code to the **ProcessDocument** method. This code opens the package and receives the file name as a parameter.

```
Package package = Package.Open(fullFileName, FileMode.Open);
```

10. The following code loads the core properties out of the package. We need these properties to retrieve the title and the author of the document. Add this code below the code you just added in the **Exercise 2** region:

```
// Get core properties
XmlDataDocument doc = new XmlDataDocument();
Uri uri = new Uri("/docProps/core.xml", UriKind.Relative);
PackagePart propsPart = package.GetPart(uri);
doc.Load(propsPart.GetStream(FileMode.Open));

XmlNamespaceManager nsmgr = new XmlNamespaceManager(doc.NameTable);
nsmgr.AddNamespace("cp",
       "http://schemas.openxmlformats.org/package/2006/metadata/core-
properties");
nsmgr.AddNamespace("dc", "http://purl.org/dc/elements/1.1/");
titleTextBox.Text = doc.SelectSingleNode("cp:coreProperties/dc:title",
nsmgr).InnerText;
authorTextBox.Text = doc.SelectSingleNode("cp:coreProperties/dc:creator",

       nsmgr).InnerText;
```

11. Retrieving the application specific properties is done in the same fashion. But before we do that, we're going to check if that part is available in the package. Place this code under the last code.

```
// Get application specific properties
doc = new XmlDataDocument();
uri = new Uri("/docProps/app.xml", UriKind.Relative);
if (package.PartExists(uri))
{
    PackagePart appPart = package.GetPart(uri);
    doc.Load(appPart.GetStream(FileMode.Open));

    nsmgr = new XmlNamespaceManager(doc.NameTable);
    nsmgr.AddNamespace("ap",
              "http://schemas.openxmlformats.org/officeDocument/2006/extended-
properties");
```



```
    paragraphsTextBox.Text =
              doc.SelectSingleNode("ap:Properties/ap:Paragraphs",
nsmgr).InnerText;
    wordsTextBox.Text =
              doc.SelectSingleNode("ap:Properties/ap:Words", nsmgr).InnerText;
    characterTextBox.Text =
              doc.SelectSingleNode("ap:Properties/ap:Characters",
nsmgr).InnerText;
    linesTextBox.Text =
              doc.SelectSingleNode("ap:Properties/ap:Lines", nsmgr).InnerText;
}
else
{
    paragraphsTextBox.Text = "na";
    wordsTextBox.Text = "na";
    characterTextBox.Text = "na";
    linesTextBox.Text = "na";
}
```

12. Finally we need to write some code to retrieve the contents of the document. We are going to list all the items that have the Heading2 style applied in the list box. To do so, you can use the WordML available in the document.xml file. Place this code under the last code you added.

```
// Get contents
contentsListBox.Items.Clear();
doc = new XmlDataDocument();
uri = new Uri("/word/document.xml", UriKind.Relative);
PackagePart docPart = package.GetPart(uri);
doc.Load(docPart.GetStream(FileMode.Open));

nsmgr = new XmlNamespaceManager(doc.NameTable);
nsmgr.AddNamespace("w","http://schemas.openxmlformats.org/wordprocessingml/2006/
3/main");
foreach (XmlNode node in doc.SelectNodes("//*[name()='w:pStyle']", nsmgr))
{
    if (node.Attributes["w:val"].Value == "Heading2")
    {
        System.Xml.XPath.XPathNavigator nav = node.CreateNavigator();
        nav.MoveToParent();
        nav.MoveToNext();

        nav.MoveToChild("t",
              “http://schemas.openxmlformats.org/wordprocessingml/2006/3/main");
        contentsListBox.Items.Add(nav.Value);
    }
}
```

13. To conclude, we need to close to package opened.

```
// Close the package
package.Close();
```

14. Press **F5** to start your application and verify your work. You can find some sample Case Studies in the folder **Lab\CaseStudies**. Close the application before moving on to **Exercise 3**.

Optional Exercise 3: Extend to explorer to create new documents using The Open XML Format SDK

1. Add a reference to the **DocumentFormat.OpenXml** assembly.
2. Add the necessary namespace to work with the **Open Xml Format SDK**:

```
using DocumentFormat.OpenXml;
using DocumentFormat.OpenXml.Packaging;
```

3. Open the region **Exercise 3** and add the following code to the **CreateNewDocument** method. Use the **Create** method of the **WordprocessingDocument** class. This code creates a new object of type **WordprocessingDocument**.

```
// create the package
string fullFileName = directoryTextBox.Text + "\\" + newTitleTextBox.Text +
".docx";
using (WordprocessingDocument wordDoc =
WordprocessingDocument.Create(fullFileName,
       WordprocessingDocumentType.Document))
{
   // add code here
}
```

4. In the folder **Labs\FileTypeLab** you find a file with the name **part_main.xml**. It contains the contents for the main part of the word document. Inspect the content.
5. Remove the comment within the using statement and add code to create the main part of the document. Load the **part_main.xml** into the main document part and save the changes.

```
   // Set the content of the document so that Word can open it.
   MainDocumentPart mainPart = wordDoc.AddMainDocumentPart();
   StreamWriter partWriter =
        new StreamWriter(wordDoc.MainDocumentPart.GetStream(FileMode.Create,
        FileAccess.Write));
   XmlDocument doc = new XmlDocument();
   doc.Load(@"..\..\part_main.xml");
   doc.Save(partWriter);
   partWriter.Close();
```

6. Now you are going to add a document part for the core properties. In the folder **Labs\FileTypeLab** you find a file with the name **part_core.xml**. It contains the contents for the word document properties. Inspect the content.
7. Then add a document part for the document core properties. Load the contents of **part_core.xml** into the new part:

```
   // Add properties part
   CoreFilePropertiesPart propertiesPart = wordDoc.AddCoreFilePropertiesPart();
   StreamWriter propWriter =
        new StreamWriter(wordDoc.CoreFilePropertiesPart.GetStream(
        FileMode.Create, FileAccess.Write));
   // First load the part_core.xml file
   XmlDocument propdoc = new XmlDocument();
   propdoc.Load(@"..\..\part_core.xml");
```

8. You are going to fill out some of the document properties like the title and the author of the document. You can achieve this by using **XPath**. First create an instance of type **XmlNamespaceManager** and add the necessary namespaces for accessing the package and the core properties:

```
   // Manage namespaces to perform XML XPath queries.
```



```
   NameTable nt = new NameTable();
   XmlNamespaceManager nsmgr = new XmlNamespaceManager(nt);
   nsmgr.AddNamespace("cp",
       "http://schemas.openxmlformats.org/package/2006/metadata/core-
properties");
   nsmgr.AddNamespace("dc", "http://purl.org/dc/elements/1.1/");
```

9. Fill out the properties like **title** and **creator** by using **XPath**:

```
   // Then set our own values
   propdoc.SelectSingleNode("cp:coreProperties/dc:title", nsmgr).InnerText =
         newTitleTextBox.Text;
   propdoc.SelectSingleNode("cp:coreProperties/dc:creator", nsmgr).InnerText =
         newAuthorTextBox.Text;
   propdoc.Save(propWriter);
   propWriter.Close();
```

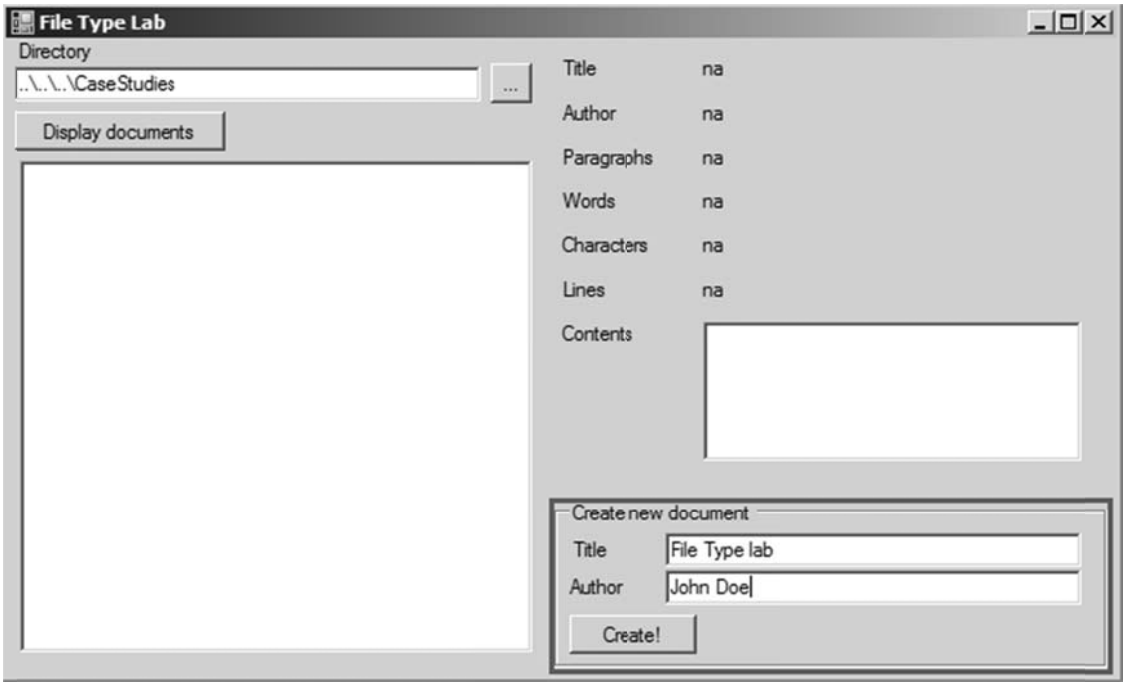
10. Notice that you don't have to add relationships anymore.
11. Application properties like word count, line count, paragraph count are part of the extended document properties. Therefore you have to create a document part of type **ExtendedFileProperties** part:

```
   // Set application specific properties
   ExtendedFilePropertiesPart extpropsPart =
wordDoc.AddExtendedFilePropertiesPart();
   StreamWriter extpropsWriter =
         new StreamWriter(extpropsPart.GetStream(FileMode.Create,
FileAccess.Write));
   XmlDocument extdoc = new XmlDocument();
   extdoc.Load(@"..\..\part_ext.xml");
   extdoc.Save(extpropsWriter);
   extpropsWriter.Close();
```

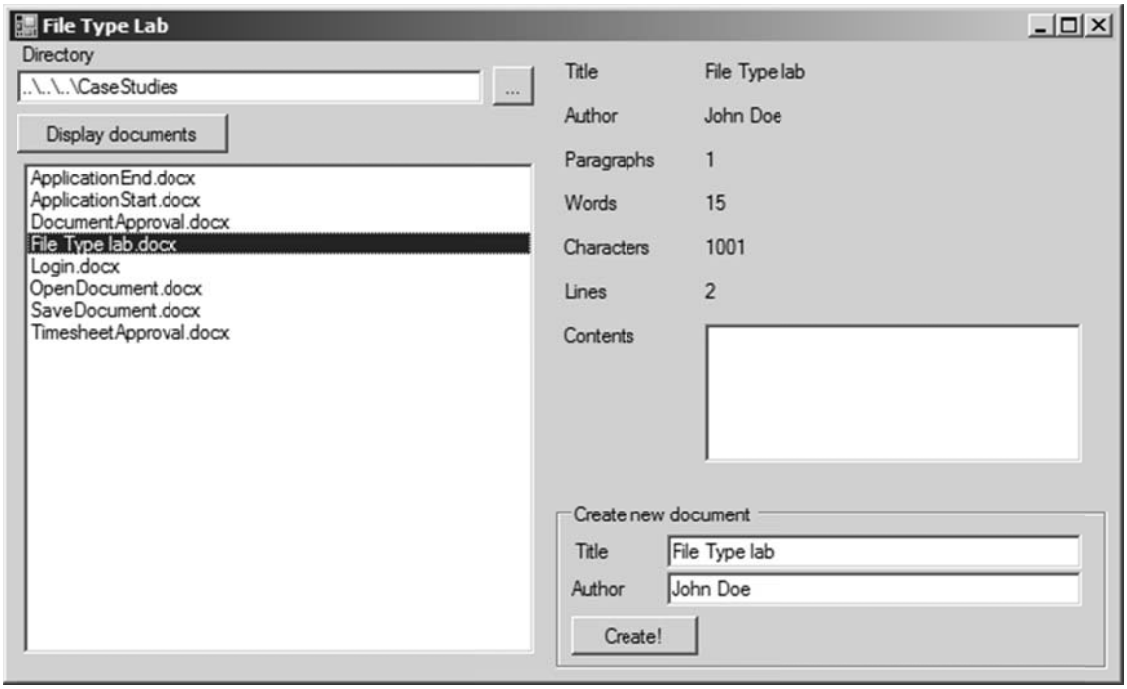
12. Word documents also contain styles. You can add a **StyleDefinitionPart** to the document but this is a child of the **MainDocumentPart** (and not a child of the document like the **CoreFilePropertiesPart**)

```
   // add styles to it
   if (wordDoc.MainDocumentPart.StyleDefinitionsPart != null)
   {
       wordDoc.MainDocumentPart.DeletePart(
            wordDoc.MainDocumentPart.StyleDefinitionsPart);
       StyleDefinitionsPart stylePart =
            wordDoc.MainDocumentPart.AddNewPart<StyleDefinitionsPart>();
       StreamWriter styleWriter =
            new StreamWriter(stylePart.GetStream(FileMode.Create,
FileAccess.Write));
       XmlDocument styledoc = new XmlDocument();
       styledoc.Load(@"..\..\part_styles.xml");
       styledoc.Save(styleWriter);
       styleWriter.Close();
   }
```

13. This finishes off the exercise. Save and build your work. Pres **F5** to run the application. Fill out a title and your name in the **Create new document** group box. Click the **Create** button to create the document.



14. Click the **Display documents** button again and select your newly created document. The document properties should get displayed.



Student Challenge: Write file format explorer with Windows tree view

As a challenge, think about creating a small Windows application an administrator can use to view the internals of an Office 2007 document. Use a tree view control to display the container structure

in a panel. When the user selects a node representing an XML-based document part or a relationship, display the XML content in a second panel using an embedded Web browser control.

Lab 12: Working with InfoPath Forms

Lab Time: 45 Minutes

Lab Directory: C:/Student/Labs/12_InfoPath

Lab Overview: In this lab, you will learn how to create and deploy an InfoPath Form. In addition, you will gain experience with the new managed InfoPath object model—embedding InfoPath within your own applications. The first two exercises are based on WSS technologies and the last two are based on what MOSS brings to the table.

- In Exercise 1, you are responsible for creating an InfoPath template that will be used by the Litware consultants to fill in their daily timesheets.
    - NOTE: it is our recommendation that you skip Exercise 1 for now and start with Exercise 2, as you have limited time to complete this lab in class and this is not an InfoPath design class. We have included the instructions for Exercise 1 so that you may explore this over lunch or at a later point in time.
- In Exercise 2, you will configure data connections back to SharePoint to facilitate efficiently filling in these forms.
- In Exercise 3, you will deploy the template so that the consultants will use the InfoPath smart client application to fill in the timesheet data.
- In Exercise 4, you will transform your template so that it becomes a content type deployable via MOSS Forms Services.

Important Note: This lab MUST BE COMPLETED as written, through Exercise 4, for Lab 12 (on workflow) to function correctly.

Lab Setup—2 pieces:

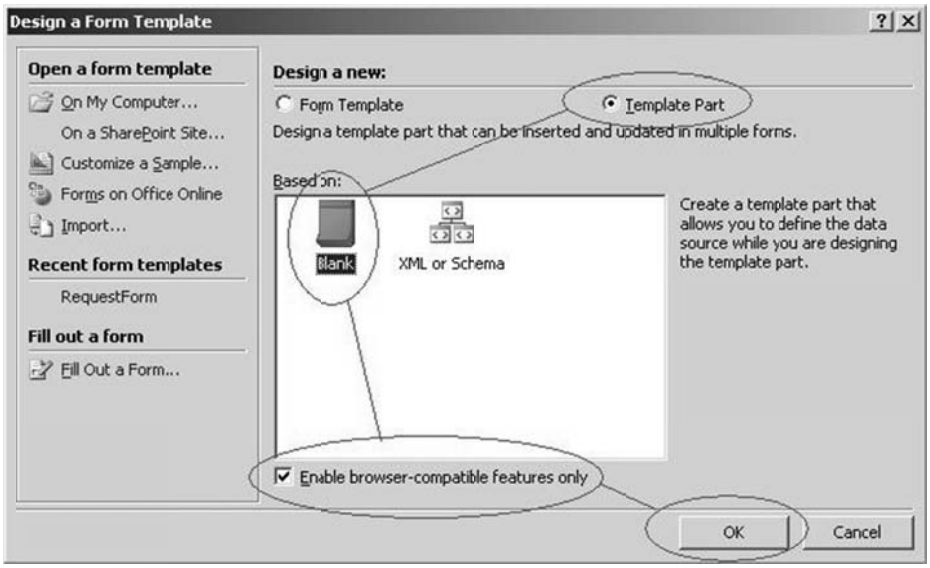
1. Lab 08 on Custom Content types must be completed before you can start this lab.
2. This will alter Active Directory to set the Department to Development and also set the manager field for certain users. This will also extend the profile database in SharePoint to include a setting for **HourlyRate** and it will pre-populate it for those in the Development Department.
    a. Note: if you want to review the source code that injects information into the user profiles via the object model this is located in the **…\12_Infopath\Starter Files\Data\ProfileMod** folder.
    b. Run the .bat file located in the InfoPath lab directory **\Starter Files\Mod11Starter.bat**. When finished type exit and press enter to close the cmd window
    c. Navigate to **SharePoint 3.0 Central Administration** (Start - All Programs - Microsoft Office Server - SharePoint 3.0 Central Administration)
    d. In Central Administration on your **Quick Launch** bar (left side of screen) click on **Litware SSP**
    e. In the **Litware SSP User Profiles and My Sites** section click on **User profiles and properties**

f. Click on **Start full import** (this will pull the new settings from Active Directory into SharePoint so that we might utilize them in this lab).

Exercise 1: Create an InfoPath form to capture timesheet info

NOTE: it is our recommendation that you skip Exercise One for now and start with Exercise 2, as you have limited time to complete this lab in class and this is not an InfoPath design class. We have included the instructions for Exercise 1 so that you may explore this over lunch or at a later point in time.

1. As a starter, you are going to create two InfoPath template parts - one for the Litware banner and one for a footer containing the text Litware uses on all of its internal documents.
   a. Open InfoPath 2007 and in the Getting Started dialog, choose the Design a Form Template link.
   b. In the Design a Form dialog, select the option to create a Template Part based on the Blank template and also check the Web browser enabled check box since you will later examine issues with Forms Server deployment. Click OK to close the dialog.



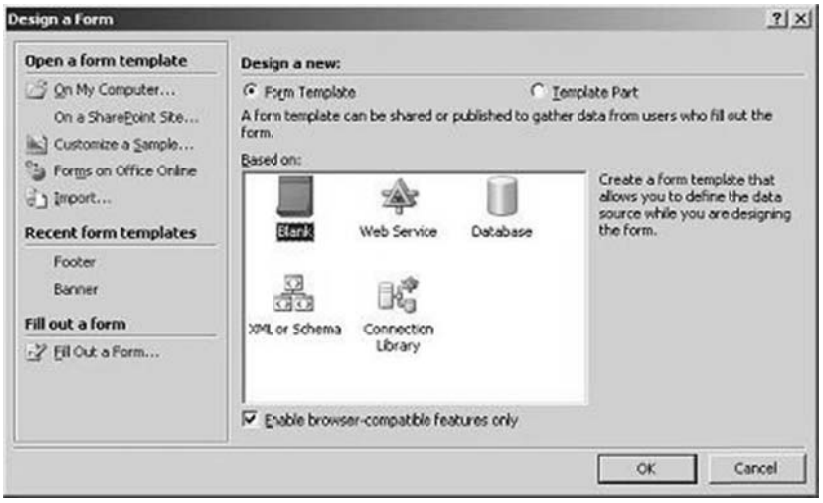
   c. In the Design Tasks pane, select Layout and double-click the Two-Column Table option.
   d. In the first cell, add Litware Inc as text and in the second one Our code is so tasty, it will melt in your mouth. Center the Text in both cells. Apply some border and shading to the table.
   e. Add a second row to the table and merge the two cells. Type **Add here the title of the template** in the merged cell.

   f. Save the template part in your **/Lab/Template Parts** folder as Banner.xtp and close InfoPath.
   g. Open InfoPath and follow the steps A and B again to create the second template part.
   h. In the Design Tasks pane, select Layout and double-click the One-Column Table option.

i. Add the text **Litware Inc. Internal Document - All Rights Reserved**. Use the font and italic option to change the look and feel.

j. Save the template part in your **/Lab/Template Parts** folder as Footer.xtp and close InfoPath.

2. Now that you have your two re-usable InfoPath blocks, start to create the main InfoPath template to capture the timesheet information. In the first instance, you just have to take care of the layout and the controls.
   a. Open InfoPath 2007 and in the Getting Started dialog, choose for Design a Form Template.
   b. In the **Design a Form** dialog, select the option to create a **Form Template** based on the **Blank** template and also check the Web browser enabled check box since you will later deploy for a Forms Server site. Click OK to close the dialog.



   c. In the **Design Tasks** pane, click on Controls. Review the controls you can use.
      i. Q: Why are there so few controls?
      ii. A: Because we are mandating that this form be usable both in InfoPath 2007 and Forms Services via the web, our control choices are limited to those that are usable both places.
   d. Click on the Some controls are not compatible with the current form and have been hidden box at the bottom of the pane. Review the message box.
   e. Click on the Add or Remove Custom Controls link at the bottom of the pane.

f. Add the two template parts you have created in step 1 and close this dialog.

g. Drag and drop the banner on the blank view

h. Press Enter 2 times to add a two blank lines

i. Drag and drop the footer.

j. Add **Daily Timesheet Report Form** as the text in the banner for the second row.

k. Place your insertion point (the flashing vertical line on the screen) in the first blank line (i.e. using the mouse click on the first blank line.)

l. Go back to the **Design Tasks** pane and select Layout.
   i. Hint: on the right hand side of your InfoPath design window there is the Controls list. At the top of this list is a link that says "Design Tasks" click on this to return to the main Design Tasks pane and then select the Layout choice from this main screen.

m. Add a **Custom Table** with 4 columns and 3 rows between the banner and footer.
   i. Add **Consultant**: in the first cell of row 1
   ii. Add **Email**: in the first cell of row 2
   iii. Add **Date**: in the third cell of row 1
   iv. Add **Phone**: in the third cell of row 2
   v. Add **Hourly Rate**: in the first cell of row 3

n. **Merge**  the other three cells of row 3 into 1 cell.

o. Go back to the **Design Tasks** pane and select **Controls**.
   i. Hint: on the right hand side of your InfoPath design window there is the **Controls** list. At the top of this list is a link that says "**Design Tasks**" click on this to return to

the main **Design Tasks** pane and then select the **Controls** choice from this main screen.

p. Add the following controls to the view:
   i. A **Text Box** in the second cell of row 1
   ii. A **Date Picker** in the fourth cell of row 1
   iii. We will leave the second cell and fourth cell of row 2, and the second cell of row 3 empty for the moment.

q. Select the table and use the border and shading  to create a nice looking table.

r. Place your insertion point (the flashing vertical line on the screen) in the second blank line (i.e. using the mouse click on the blank line immediately beneath the table.)

s. Add a **Repeating Table** with **4 columns** after the table you just created.
   i. Hint: from your **Insert Menu** select **Repeating Table**...

t. Add **Project**, **Task**, **Description** and **Hours** as headers.

u. Right-click the **Text Box** under **Project** and **Change To: Drop-Down List Box**.

v. Click the **Preview** button in the toolbar to check the result of your work.

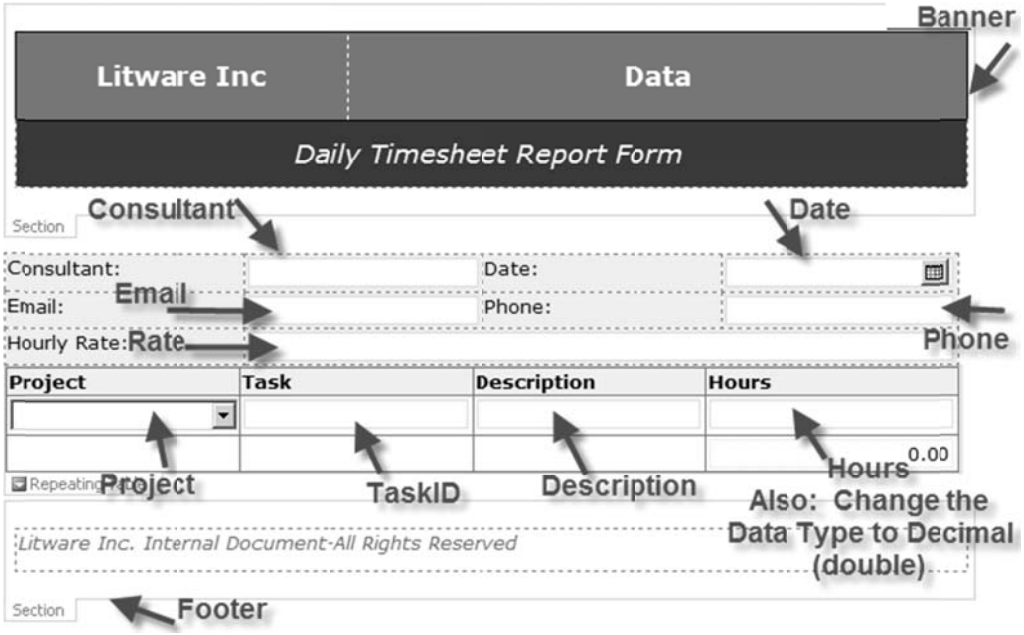
w. Close the preview.

x. Save the template as **TimeSheet.xsn** under your **/Lab** folder.

y. You have started really from scratch. While you were busy in these previous steps adding the controls to the view, InfoPath generated an XSD schema for you. The only thing to do now is make changes to the default names and types.

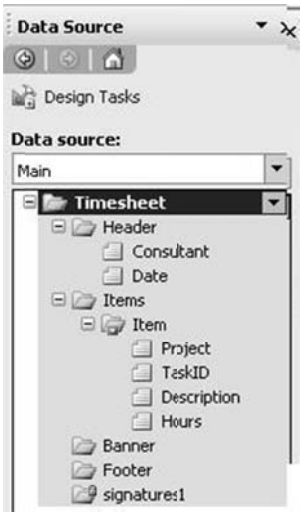
3. Go back to the Design Tasks pane and select Data Source.
   a. Double-click each the elements in the data source and make the following changes:
      i. rename **myFields** to **timesheet**
      ii. rename **group1** to **items**
      iii. rename **group2** to **item**

iv. Using the image below rename the remaining items as follows:



b. Next add a group under the **Timesheet** element called Header (use the **Move Up** to position the new group)
    i. Hint: on the right side of the screen using the Data Source panel, expand the drop down off of timesheet and click Add...
c. Move the **Consultant and Date** field to the Header group.
d. Move the Banner and the Footer all the way to the bottom. Hint:
    i. In the **Data Source Panel** select **Banner** and then select **Move Down** off the drop down menu.
    ii. Repeat for the **Footer**.
    iii. Select **Header** and then select **Move Up** off the drop down menu.
    iv. select the Date field and select Move... off the drop down menu select the Header folder and
    v. Then select **OK**.

vi. Your **Data Source Main** should look as follows:

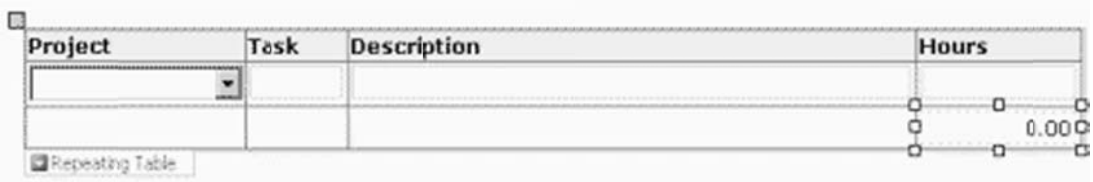


e. Right-click the table and open the **Repeating Table Properties**. On the **Display** tab check the **Include Footer** checkbox. Click OK.

f. Add an **Expression Box** in the Hours column in the footer row. Use the formula editor to insert a Sum function for the Hours field. Change the type of the expression box content to Decimal. Hint:

i. Select **Design Tasks** in your **Data Source** panel and then select **Controls**
ii. Place your insertion point in the last cell in the last row of your repeating table (i.e. 4th column, row 3)
iii. Select **Expression Box** from the Controls Panel
iv. Select the Function button 
v. Select Insert Function...
vi. Select sum and click **OK**
vii. double click to insert field and pick the Items - Item - Hours field
viii. Click **OK**
ix. In the Result section Format as: drop down box select Decimal
x. Click **OK**

g. Be sure to format both the **Hours** Repeating field and the **Sum of Hours** field so that they have a Right Alignment and always display 2 decimal places.



h. Save the template.

Exercise 2: Pre-populate the InfoPath form with SharePoint Data

In this exercise we will be setting up security on our InfoPath form to allow data connections with the SharePoint Server and we will set this form up to pull data from the SharePoint Profile Database and the Projects list that exists on our Litware Project Management Site.

Exercise Setup:

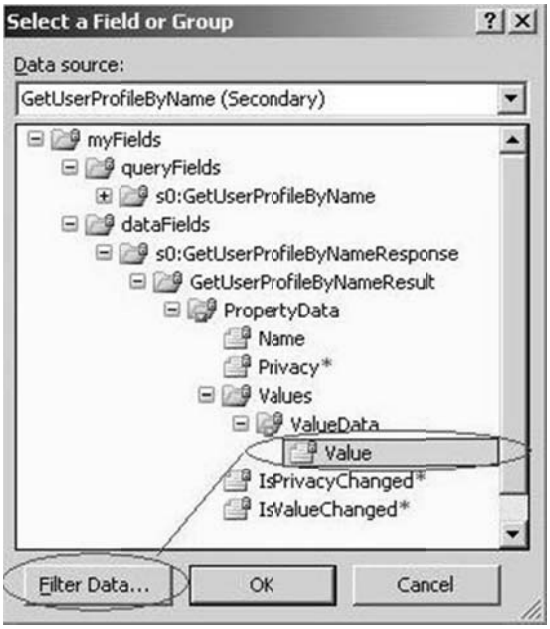
- If you did not complete Exercise 1 then you must first copy the starter files located at ...**12_InfoPath\Starter Files\exercise2** to your **...\12_InfoPath\Lab** directory.
- You will then need to open the **TimeSheet.xsn** in **Design mode** by right clicking on it and selecting **Design**.

Exercise:

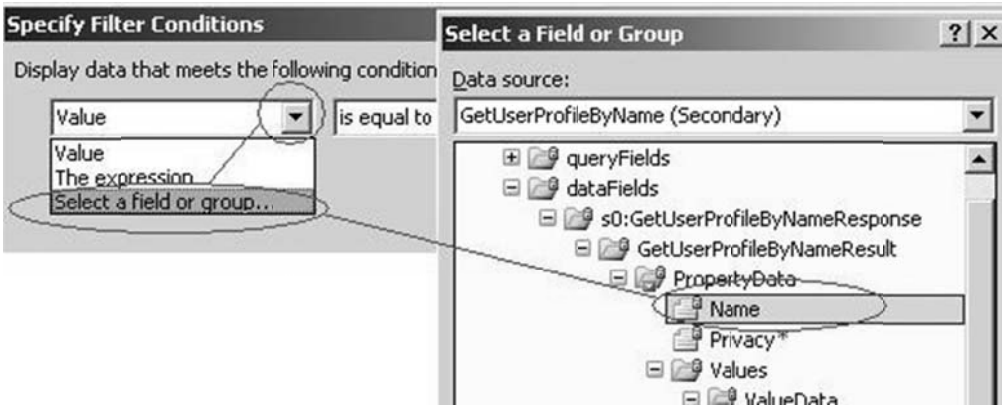
1. First you'll need to create a Data connection to pull user profile information into our form.
    a. Add a connection to receive data by using the **Tools > Data Connections...** menu item.
    b. Click the **Add...** button.
    c. Select **Create a new connection to:** and then **Receive data** as the source type.
    d. Click the **Next >** button.
    e. Select **Web Service** on the data source screen.
    f. Click the **Next >** button.
    g. When asked for the location of the web serice type **http://litwareinc.com/_vti_bin/UserProfileService.asmx**
    h. Click the **Next >** button.
    i. In the **Select an operation:** list select **GetUserProfileByName**
        i. *Note: this method will retrieve a **PropertyData** array (i.e. like a repeating table of key/value pairs)*
    j. Click the **Next >** button.
    k. Click **Next >** again
        i. *Note: we could pass a parameter here to select a certain user account. However, we are using the default functionality of this method, which is to find the current users account if no parameters are passed.*
    l. Click **Next >** a third time
        i. *Note: as we do not wish to store an offline copy of this data up front there is nothing to do here.*
    m. Click the **Finish** button then click **Close** to return back to your InfoPath Form.
        i. *Note: Automatically retrieve data when form is opened is selected by default.*
2. Now that we have the **PropertyData** array we will populate several fields on our form with this data
    a. Double click on the **TextBox** in the cell next to **Consultant**: (located in row 1 column 2 of the first table)
        i. *Note: when you repeat this step later you will be using an expression field instead of a text box and using the table from step H. to identify the field location next to which*

you will add the expression field. DO NOT FOLLOW THIS NOTE THE FIRST TIME YOU RUN THIS... (i.e. for the Consultant choice)

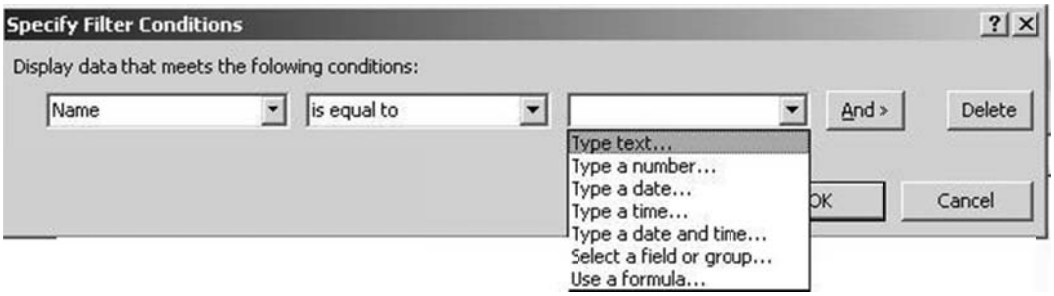
b. click the **Fx** button 
c. On the **Insert Formula** dialog box click **Insert Field or Group**...
d. In the **Data Source:** dropdown box select the secondary data source **GetUserProfileByName**
e. Drill down through the fields until you select the **Value** Field (see Graphic below) and then click Filter Data...



f. On the **Filter Data** dialog box click the **Add...** button.
g. Then fill this form in using the graphics below. **Note**: be sure to change the first drop down box from **Value** to **Name** by selecting the "**Select a field or group...**" and then using the graphics below to complete the task.



h. Next set the third drop down to **Type text...**



i. After Selecting **Type text...** type "**AccountName**" and Click **OK** four to five times to close all dialogs.

    1. *Note: when you repeat this step later you will be replacing AccountName with the value found in the **Type Text** column from the table found in step j. but DO NOT DO THIS THE FIRST TIME THROUGH…*

j. Now you will add an expression field next to **Email**: (i.e. Row 2 column 2)

    i. Put your insertion point into row 2 column 2 of the first table

    ii. using your **Insert** menu select **More controls...** and then select **Expression** box from the **Advanced** section.

k. Repeat the steps (a—h), for the three Expression fields in the table below.

    i. **Note**: After completing these steps for **Email** you will once again need to insert an expression field next to **Phone**, and then do steps a—h again. Finally you will need to insert an expression field next to **Hourly Rate**: and run steps a—h again.

    ii. **Important**: As InfoPath 2007 is very temperamental when it comes to Forms Services (i.e. browser enabled forms) these steps MUST BE COMPLETED in the exact order they are written (i.e. add 1 expression field and then immediately configure it before moving to the next field)

    iii. **Further Note**: even the act of editing an existing Expression field will cause validation errors in browser compatibility. The easiest way to resolve these is to delete and re-create the Expression field and the expression it contains.

Field	Type Text
Email	WorkEmail
Phone	WorkPhone
Hourly Rate	HourlyRate

l. Double click on the **Consultant TextBox** and from the **Display** tab select **Read-only** to ensure that users cannot alter this important information.

m. Save the changes to the form template, and click **Overwrite** if the warning message comes up.

    i. *Note: You may receive a warning regarding the fact that when you change data sources on an existing form template existing forms will also be converted to this*

new template starting point possibly causing a permanent loss of data. This warning does not apply to us as we have yet to publish this form.

n. Finally you'll need to create a data connection for the list of projects as well. This time you'll be using a SharePoint list as the starting point (this list should already exist on our **Project Management Site http://litwareinc.com/sites/ProjectManagementLab/Lists/Projects**).

3. First add a connection to receive data of type **SharePoint library or list**.
   a. Add a connection to receive data by using the **Tools > Data Connections...** menu item.
   b. Click the **Add...** button.
   c. Select **Create a new connection to:** and then **Receive data** as the source type.
   d. Click the **Next >** button.
   e. Select **SharePoint Library or List** and click **Next>**
   f. Use a path of **http://litwareinc.com/sites/ProjectManagementLab/** and click **Next**.
   g. Select the **Projects** list and then click **Next** to advance to the column selection page.
   h. Uncheck all of the columns except **Project** and click **Next** twice then **Finish** to create the new connection.
   i. Click **Close** to close the **Data Connections** window.
4. Now you will connect the **Project** drop down on the page to the data connection created in the previous step.
   a. Right click the **Project** drop down and select **Drop-Down List Box Properties**.
   b. In the **List box entries** section, select **Look up values from an external data source** and select the **Projects** data source.
   c. Next click the button to the right of the **Entries** text box  and select the **:Project** node.
   d. Click **OK (two times)** to assign the connection.
5. Now that the data is populated, you'll need to perform some validation checks on the data entered into the form.
   a. Set today's date as the default date
   b. Double click on the **Date** field (to the right of the **Date:** title field).
      i. Click the **Fx** button. 
      ii. Click on **Insert Function**... button
      iii. select the **Today** function from the **Most Recently Used** Category
      iv. Click **OK** three times
   c. Create a data validation rule for the date field. Restrict users from entering a date in the future.
      i. Double click on the **Date** field (to the right of the Date: title field).
      ii. Click on **Data Validation**... then click on **Add**...
      iii. Set the condition **Date is greater than** and then from the third drop down box select **Use a formula**...
      iv. Type **today()** for the formula and click **OK**
      v. Set the **Screen Tip:** to **You cannot select a date in the future**

vi. Set the **Message:** to **You cannot record work for a future date... All work recorded must be for Today or earlier dates**.
vii. Click **OK (three times)**

d. Change the background color of the expression box that calculates the total of hours to red if the amount is bigger than 8
   i. Right Click on the **Total Hours Expression Box** (i.e. the one with **0.00** in it) and select **Expression Box Properties**...
   ii. On the **General** tab select the expression in the **XPath**: box and copy it (i.e. use **Ctrl+C**)
   iii. Select the **Display** tab then click on **Conditional Formatting**... then click **Add**...
   iv. Set the condition to **The expression**
   v. Set the condition Text box to **sum(my:Items/my:Item/my:Hours) > 8**
      1. Hint: Put your insertion point into this text box and press **Ctrl+v** to paste your formula in and then add **> 8**
   vi. if this condition is true set the **Font color**: to a shade of **Red**
   vii. Click **OK (Three times)**

6. **Save** the template as **TimeSheet.xsn** in the Lab folder. Test your form by clicking the **Preview** button.
   a. Note: You will be Informed about a potential security concern, click **Yes**.
   b. If you are also questioned about working offline select the **Try to connect** button in response to this.
   c. Take your time to inspect the form. Close the **Preview**.
7. The last part of this exercise is to prepare to save the data in the InfoPath form into a SharePoint Form Library. To do this, you'll need to create a new Form Library named **Daily Timesheets** in the **http://Litwareinc.com/sites/ProjectManagementLab/SouthDivision** site.
   a. Navigate to the **South Division** site using the **South Division** tab at the top of the **Litware Project Management** site
   b. Select **Site Actions - Create - Form Library** (from the **Libraries** section)
   c. Name the new Library **Daily Timesheets**
   d. Click **Create**
8. As a final step you add a button control to the view that will be used to submit all of the data to SharePoint document library.
   a. Return to your InfoPath form in Design mode.
   b. Add a new **Data Connection** using the **Tools** and then **Data Connections** in the InfoPath menu. Click **Add**.
   c. This one will be a connection to **Submit data** (Create a new connection to **Submit data**).
   d. Click **Next.**
   e. Select **To a document library on a SharePoint site.**
   f. Click **Next.**
   g. Enter the URL to the document library **http://litwareinc.com/sites/ProjectManagementLab/SouthDivision/Daily%20Timesheets/**

   h. Click the **fx** button to the right of the filename to reset the name of the saved form.
   i. Set the name of the form to **concat(consultant, "_", date)**. You'll need to build this string, don't just paste into the text box. Do this by double clicking each section of the formula and choosing "**Consultant**" and "**Date**" from the **Header** group. For the center section you must type **"_"**.
   j. Click **OK** to close the **Insert Formula** dialog.
   k. Check the **Allow overwrite if file exists.**
   l. Click **Next** and **Finish** to close the **Data Connection Wizard**.
   m. Click **Close** to close the **Data Connections** dialog.
9. Drop a **Button** control on the Form between the **Repeating Table** and the **Footer**.
10. Double-click the control and change the **Label** and **ID** to **Submit**.
11. Use the **Rules** to add a new rule with no condition but with the following three actions:
   a. **Submit using a Data Connection** - the SharePoint one you just created.
   b. **Show a Dialog Box message** with a notice that "**Your timesheet was received successfully.**"
      i. Note: this action is not supported by InfoPath Forms Services...
   c. **Close** this form: No Prompt
12. Click **OK (three times)** to return back to your form design window.
13. Time to test out your work.
   a. **Save** the template and close InfoPath.
   b. Double-click the **TimeSheet.xsn** file in your **/Lab** folder.
   c. Click **Yes** in response to the security concern.
   d. Fill in some data and then click the **Submit** button.
   e. If you get the message box that you entered information for, then your information was processed with success by the Web service.
   f. You can verify if the information has been added to the database by using the **TimeSheetReport** application you worked with in the lab on template-based solutions.

Exercise 3: Publish form to WSS forms library

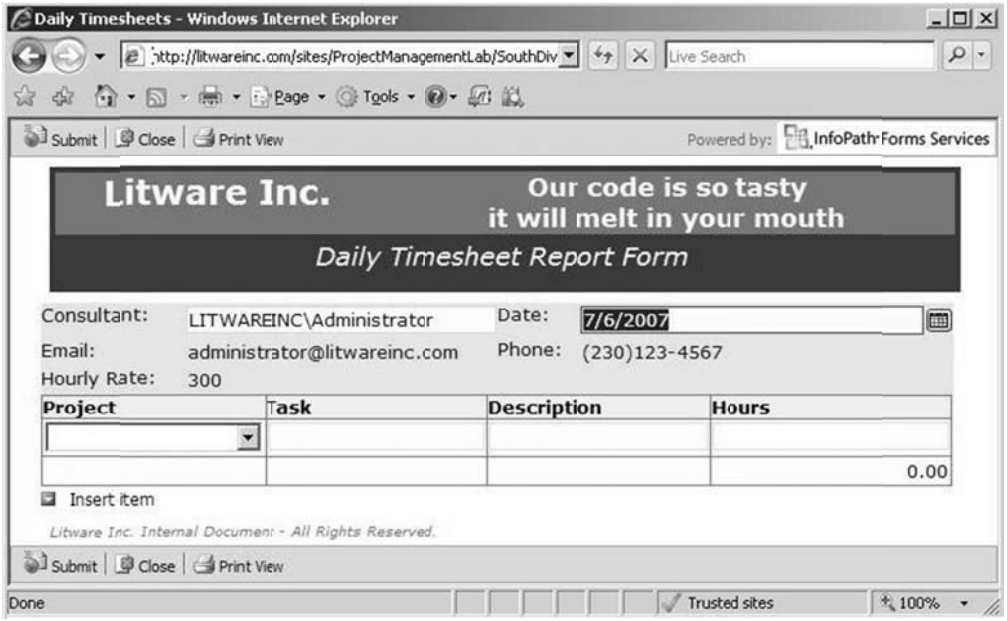
In this exercise, you are playing the role of the administrator who is responsible for the deployment of the InfoPath template. You will be deploying the template to a SharePoint forms library and make it available via the Forms Server for those consultants that do not have InfoPath installed on their laptops (i.e. you will begin utilizing MOSS Forms Services)

1. Open the **TimeSheet.xsn** file you created in the previous exercise in design mode.
2. Since you'll be using **Forms Services** to render this form, you'll need to make some changes to the submission process.
    a. First remove the button from the form.
    b. Configure the submit options using **Tools -> Submit Options**.
    c. Check **Allow users to submit this form**
    d. Select **Send form data to a single destination** with a destination of **SharePoint document library**.
    e. Select **SharePoint Library Submit** as the data connection to use.
    f. Click **Advanced >>** and select **Close** the form in the **After submit** drop down list.
    g. Click **OK** to apply the changes.
3. Next you'll need to disable the save option on the form to force the user to submit, not save.
    a. Open **Tools -> Form Options.**
    b. Select the **Open and Save** pane.
    c. In this pane uncheck **Save and Save As** to disable saving.
    d. Click **OK.**
4. In the **Design Tasks** pane (right side of InfoPath screen) click on the **Design Checker**.
    a. You should not see any red icons indicating a compatibility error.
    b. You might see one or more blue icons indicating a warning.
5. Go back to the **Design Tasks** pane and click the **Publish Form Template** link. This wizard will guide you through all the needed steps to make your template available via SharePoint.
    a. Click **OK** to the modification message.
    b. Select **To a SharePoint server with or without InfoPath Forms Services.**
    c. Click **Next.**
    d. Give the URL to the SharePoint server (**http://litwareinc.com/sites/ProjectManagementLab/SouthDivision/**)
    e. Click **Next**
        i. **Note**: if you have a problem publishing your site you may have to run "**net stop sens**" from a cmd prompt. This is only an issue on a single machine development environment and only affects Office 2007 items interacting with SharePoint.
    f. You will note that you cannot currently enable this form to be filled out using a browser this is controlled via a MOSS site collection feature.
        i. Keep your InfoPath Design window open and in a browser navigate to **http://litwareinc.com/sites/ProjectManagementLab/**

ii. Select **Site Actions > Site Settings > Site collection Features** (from the **Site Collection Administration** column)
iii. Locate and Activate the **Office SharePoint Server Enterprise Site Collection** features choice (among other things this controls forms services)
iv. Return to your InfoPath Design window and click **Back** and then click **Next** you should see that the browser based option is now available.
    1. **Note**: If you now see an additional warning that our form requires administrator approval before it can be used via a browser it is likely for one of two reasons:
        a. you have browser incompatible items on your form (in your Design Tasks Pane switch to Design Checker and look for Errors (i.e. items with a red X)
        b. you have somehow enabled a VSTO project for your InfoPath form (i.e. created a code behind page) To check this, in InfoPath select Tools - Form Options - select Programming (from Category:) - look for the Remove Code button (if this is enabled you've got code behind problems) to remove them click Remove Code and then click OK
        c. After fixing the issues try Steps 5 a through f again, then move on to step g.

g. Check **Enable this form to be filled out by using a browser** and select **Document Library** click **Next**.
h. Select **Update the form template in an existing document library** and choose the **Daily Timesheets** library. Click **Next >.**
i. Add **Consultant** and **Date** as new columns to be created.
j. Click **Next** and **Publish.**
k. In the complete dialog, click **Open this form template in the browser** to view the form in the browser.

l. Once the publish process is completed, close InfoPath.



6. Test the template by clicking **New** in the **Daily Timesheets** forms library. Notice that your form opens in InfoPath and not the browser. By default forms will use the browser only if InfoPath isn't installed. Fill in the form and submit your changes using the **Submit** button.
7. As an administrator of the forms library, you can specify that the InfoPath form always has to be filled in using the browser, even if the consultant has InfoPath installed on his machine.
    a. In the forms library, select **Settings** and then **Form Library Settings**.
    b. Click on **Advanced Settings.**
    c. Activate the **Display as a Web page** option in the **Browser-enabled Documents** section
    d. Click **OK** and using the breadcrumbs navigate back to the **Daily Timesheets** Forms Library
8. Test the template by clicking **New** in the forms library. The form will be made available in the browser now.

Exercise 4: Creating an InfoPath Form Content Type for use with Forms Services

In Exercises 1-3 you have created an InfoPath Form Template that works well with the Browser and the InfoPath Client Application. Your next task is to set this Form up so that it might be deployed as a content type to enable re-use with multiple Forms libraries. In order to accomplish this you will need to use managed code to dynamically determine the server and library name where the form was launched and then modify the **FolderUrl** property of our submit data connection to ensure submission to the correct library. You will also need to modify the security settings to allow this content type to be deployed on the server and used with different Form Libraries.

Important Note: Up until now we have been able to switch between using InfoPath or the browser to submit our forms. However, once we give our Form the ability to automatically detect the location it was launched from (browser or InfoPath) things begin to change. One limitation is that once we decide

to deploy our Form via Forms Services (i.e. Farm Administrator deployed forms), which gives us the ability to re-use these forms in many places and use many more features in the browser-based scenario, certain scenarios are not supported for InfoPath direct usage. One of these scenarios is setting up your form to dynamically determine the location from which it was launched. InfoPath has no way to determine where the form is currently coming from if the template is stored centrally in the SharePoint Forms Services location, therefore we are forced to utilize only the browser based form by the end of this Exercise.

1. If necessary open your **TimeSheet.xsn** in design mode in InfoPath 2007
2. Add a "hidden" string to your Form to hold submission location information (to allow future edits to existing documents).
    a. In your **Design Tasks** pane (right side of InfoPath window) select **Data Source.**
    b. In the **Data Source: Main** select the **Timesheet** drop down and then click **Add**...
    c. Set the Name: to **strPath** and click **OK.**
    d. Repeat step 2 this time setting the Name: to **strExists** also set the **Default value**: to **N**
3. Modify your **Submit Options** to allow for dynamically picking up the Forms starting location.
    a. Navigate to **Tools - Form Options - Select Programming** from the **Category**: list
    b. Set your Form template code language to either **C#** or **Visual Basic.**
    c. Set your Project Location to **...\Labs\12_InfoPath\Lab\MOSS.**
    d. Click **OK.**
    e. Navigate to **Tools - Submit Options.**
    f. Select the option for **Perform custom action using Code** and click the **Edit Code** button. This will open a VSTA programming window.
        i. In the VSTA programming window we need to add code to discern the current location the form is being launched from.
4. Inside the **FormCode** Class file read over the comments, notice that you cannot use Member variables in browser-enabled forms. You will need to use the **FormState** property bag to store location information for our content type. Add the following code inside of your **FormCode Class**.

C#

```
private object _libraryUri
{
  get { return FormState["_libraryUri"];}
  set { FormState["_libraryUri"] = value; }
}
```


VB.NET

```
Private Property _libraryUri() As Object
  Get
    Return FormState("_libraryUri")
  End Get
  Set
    FormState("_libraryUri") = value
  End Set
End Property
```

5. Next you need to use the Forms Loading event handler to populate the **_libraryUri** property. **Note**: The **EventManager.FormEvents.Submit** is already present in the code. You must add the Event wireup to the **InternalStartup()** method first:

C#

```
public void InternalStartup() {
  EventManager.FormEvents.Submit += new
SubmitEventHandler(FormEvents_Submit);
  EventManager.FormEvents.Loading += new
LoadingEventHandler(FormEvents_Loading);
}

public void FormEvents_Loading(object sender, LoadingEventArgs e) {
  // We need to Get the Uri (or SaveLocation in a browser form)
  // of the library the form was opened from.
  // First determine if the form was opened in the browser or not
  if (Application.Environment.IsBrowser) {
    // If true, test for and use the "SaveLocation" from the
InputParameters
    if
(!String.IsNullOrEmpty(e.InputParameters["SaveLocation"].ToString())) {
      _libraryUri = e.InputParameters["SaveLocation"].ToString();
    }
  }
  else {
    // NOTE: Even though we will not be able to utilize this code path in
    //       our current scenario, you may have a situation where you
want
    //       to just use InfoPath to fill in the form (and not Forms
Services)
    // So If the form was opened in the client, we will get the Uri
    if (!String.IsNullOrEmpty(this.Template.Uri.ToString())) {
      _libraryUri = this.Template.Uri.ToString();
    }
  }
}
```

VB.NET

```
Public Sub InternalStartup()
  AddHandler EventManager.FormEvents.Submit, AddressOf FormEvents_Submit
  AddHandler EventManager.FormEvents.Loading, AddressOf
FormEvents_Loading
```



```
End Sub 'InternalStartup

Public Sub FormEvents_Loading(sender As Object, e As LoadingEventArgs)
  ' We need to Get the Uri (or SaveLocation in a browser form)
  ' of the library the form was opened from.
  ' First determine if the form was opened in the browser or not
  If Application.Environment.IsBrowser Then
    ' If true, test for and use the "SaveLocation" from the
InputParameters
    If Not
[String].IsNullOrEmpty(e.InputParameters("SaveLocation").ToString()) Then
      _libraryUri = e.InputParameters("SaveLocation").ToString()
    End If
  Else
    'NOTE: Even though we will not be able to utilize this code path in
    '      our current scenario, you may have a situation where you want
    '      to just use InfoPath to fill in the form (and not Forms
Services)
    ' So If the form was opened in the client, we will get the Uri
    If Not [String].IsNullOrEmpty(Me.Template.Uri.ToString()) Then
      _libraryUri = Me.Template.Uri.ToString()
    End If
  End If
End Sub 'FormEvents_Loading
```

6. Now you will modify the **FormEvents_Submit** method to utilize the **_libraryUri** to define the location for forms submission.

C#

```
public void FormEvents_Submit(object sender, SubmitEventArgs e) {
  //Create a temporary storage location for the final path
  string strPath = "";

  //Create a Navigator object for the main DOM
  XPathNavigator xnDoc = this.MainDataSource.CreateNavigator();

  //Create Navigator objects for each field
  XPathNavigator xnPath =
    xnDoc.SelectSingleNode("my:Timesheet/my:strPath",
this.NamespaceManager);
  XPathNavigator xnExists =
    xnDoc.SelectSingleNode("my:Timesheet/my:strExists",
this.NamespaceManager);

  // First we must determine if the form is being edited or if it is new
  if (xnExists.Value.ToString()=="Y") {
    strPath = xnPath.Value.ToString();
  }
  else {
    // New form
    // Create a temporary storage location for Uri
    string strUri = _libraryUri.ToString();
    if (Application.Environment.IsBrowser) {
      // because the form is browser based, the libraryUri value is just
      // the server name and library - so we just need to get
```



```
      // the URL without the last "/"
      strPath = strUri.Substring(0, strUri.LastIndexOf("/"));
    }
    else {
      // because we opened the form in InfoPath
      // Parse just the path to the document library
      // The URL up to /Forms...
      strPath = strUri.Substring(0, strUri.IndexOf("Forms") - 1);
    }
    xnExists.SetValue("Y");
    xnPath.SetValue(strPath);
  }
  // If the submit operation is successful, set
  // e.CancelableArgs.Cancel = false;
  // Write your code here.
  // Get a reference to the submit data connection
  FileSubmitConnection fc =
    (FileSubmitConnection)this.DataConnections["SharePoint Library
Submit"];
  // Modify the URL we want to submit to using strPath
  fc.FolderUrl = strPath;
  // Execute the submit connection
  try {
    fc.Execute();
    e.CancelableArgs.Cancel = false;
  }
  catch {
    e.CancelableArgs.Cancel = true;
  }
}
```

VB.NET

```
Public Sub FormEvents_Submit(sender As Object, e As SubmitEventArgs)
  ' Create a temporary storage location for the final path
  Dim strPath As String = ""

  ' Create a Navigator object for the main DOM
  Dim xnDoc As XPathNavigator = Me.MainDataSource.CreateNavigator()

  ' Create Navigator objects for each field
  Dim xnPath As XPathNavigator = _
    xnDoc.SelectSingleNode("my:Timesheet/my:strPath",
Me.NamespaceManager)
  Dim xnExists As XPathNavigator = _
    xnDoc.SelectSingleNode("my:Timesheet/my:strExists",
Me.NamespaceManager)

  ' First we must determine if the form is being edited or if it is new
  If xnExists.Value.ToString() = "Y" Then
    strPath = xnPath.Value.ToString()
  Else 'new form
    'Create a temporary storage location for Uri
    Dim strUri As String = _libraryUri.ToString()
    If Application.Environment.IsBrowser Then
      ' Because the form is browser based, the libraryUri value is just
```



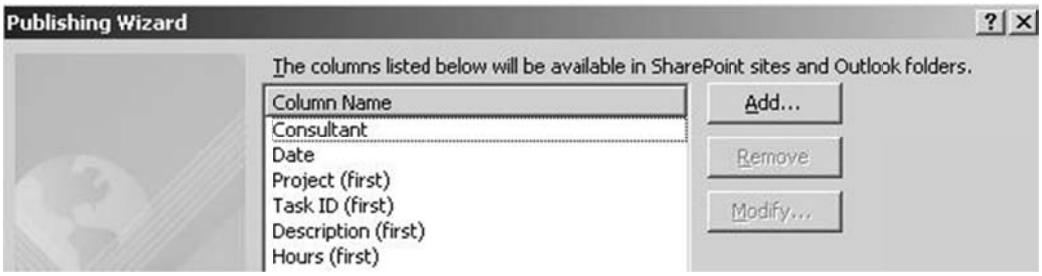
```
      ' the server name and library - so we just need to get
      ' the URL without the last "/"
      strPath = strUri.Substring(0, strUri.LastIndexOf("/"))
    Else
      ' Because we opened the form in InfoPath
      ' Parse just the path to the document library
      ' The URL up to /Forms...
      strPath = strUri.Substring(0, strUri.IndexOf("Forms") - 1)
    End If
    xnExists.SetValue("Y")
    xnPath.SetValue(strPath)
  End If

  ' If the submit operation is successful, set
  ' e.CancelableArgs.Cancel = false;
  ' Write your code here.
  ' Get a reference to the submit data connection
  Dim fc As FileSubmitConnection = _
    CType(Me.DataConnections("SharePoint Library Submit"),
FileSubmitConnection)

  ' Modify the URL we want to submit to using strPath
  fc.FolderUrl = strPath
  ' Execute the submit connection
  Try
    fc.Execute()
    e.CancelableArgs.Cancel = False
  Catch
    e.CancelableArgs.Cancel = True
  End Try

End Sub 'FormEvents_Submit
```

7. Build your Timesheet VSTA application
8. Close Visual Studio if the build was successful.
9. Click the **OK** button to close the **Submit Options** dialog.
10. Now you need to prepare the content type for deployment. Because this contains back end code this type must be administrator approved and signed with a certificate. **Note**: We will use a self-signed certificate for testing purposes. For the real world you would need to use a certificate granted from a trusted authority like Verisign.
    a. First you need to set the security level for this form and specify a certificate for the form to use.
        i. On the InfoPath Menu Bar select **Tools > Form Options**
        ii. In the **Category:** list Select **Security and Trust**
        iii. Remove the check from **Automatically determine security level (recommended)**
        iv. Select **Full Trust**
            1. **Note**: this will ensure that we are not questioned about security. We are in essence stating that we have verified this form will do no harm once deployed. Forms with Full Trust MUST BE approved by an administrator prior to use.
        v. Select the **Sign this form template** check box (under Form Template Signatures)

vi. Click the **Create Certificate...** button and click **OK** to the message about Self-Signed certificates.
vii. Click **OK**.

b. In InfoPath save the form

11. Now you will attempt to publish this form template as before.
    a. Go back to the **Design Tasks** pane and click the **Publish Form Template** link. This wizard will guide you through all the needed steps to make your template available via SharePoint.
    b. If you see a message warning you about modifying an existing Form, Click OK to the modification message.
    c. Select **To a SharePoint server with or without InfoPath Forms Services.**
    d. Click **Next.**
    e. Give the URL to the SharePoint server (**http://litwareinc.com/sites/ProjectManagementLab/SouthDivision**/)
    f. Click **Next**.
    g. Check **Enable this form to be filled out by using a browser** and notice that you now only have one choice available.
    h. Select **Administrator-approved form template (advanced).**
    i. Click **Next**.
    j. Create/Browse to the C:\ location and create a folder named **Forms** (you must create this folder before attempting to use it).
    k. Specify a filename of **TimeSheet.xsn**.
    l. Click **Save** (verify that your path is **C:\Forms\TimeSheet.xsn**).
    m. Click **Next**.
    n. Add the following columns (see figure) to those already there (i.e. **Consultant** and **Date**) as new columns to be created.



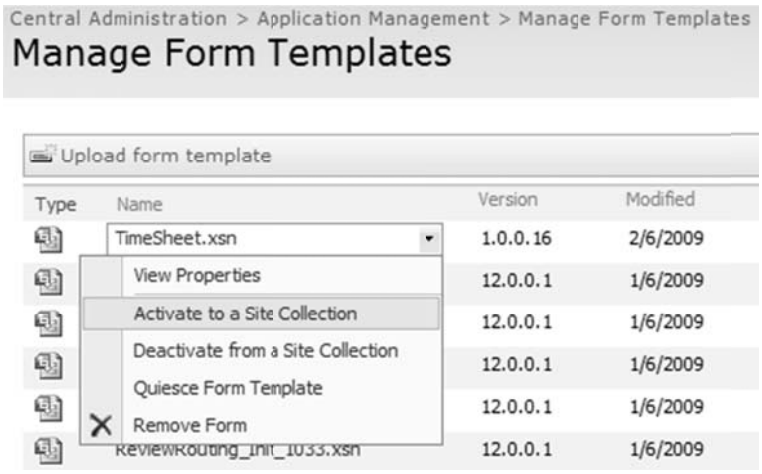
    o. Click **Next** and **Publish**. **Note**: Take note of the published location. This is the location we will need later to upload the form. In a production environment this location is likely to be a

network file share that a SharePoint Farm Administrator can easily navigate to.

p. click **Close**.
q. Once the publish process is completed, close InfoPath.

12. Next we will verify that the Form template is ready for uploading by using **SharePoint Central Administration.**
    a. Click **Start > All Programs > Microsoft Office Server > SharePoint 3.0 Central Administration**.
    b. In the top navigation bar, click the **Application Management** tab.
    c. On the **Application Management** page, under **InfoPath Forms Services**, click **Upload** form template.
    d. On the **Upload Form Template** page, click **Browse....**
    e. In the **Choose file** window, browse to **C:\Forms\TimeSheet.xsn**, click the template, and then click **Open**.
    f. Click **Verify**.
    g. In the **Report Details** section, look for any errors and warnings for the form template.
        i. **Note**: If the system warns you that the template already exists, click **Application Management**, click **Manage form templates**, click the arrow that appears next to the form template, and then click **Remove Form**. On the **Remove Form Template** page, click **Remove**. Try step 11 again.
    h. If you did not receive a warning, click **OK**.
13. After you have verified that there are no problems with your Form, you will upload the form template by using **SharePoint Central Administration.**
    a. You should still be on the **Upload Form Template** page, click **Browse....**
    b. In the **Choose file** window, browse to **C:\Forms\TimeSheet.xsn**, click the template, and then click **Open**.
    c. Click **Upload**.
        i. **Note**: The form template is uploaded to the server. Although uploaded, it is not yet available to users. It must be activated by a farm administrator who has site collection administration permissions. You will do this next.
    d. Click **OK**.
14. To make the form template available to users, the form must be activated to a site collection.

a. You should be on the **Manage Form Templates** page.
b. Point to the **TimeSheet.xsn** form template that you want to activate, click the arrow that appears, and then click **Activate to a Site Collection**.



c. On the **Activate Form Template** page, click the **Site Collection URL**, then click **Change Site Collection**.
d. Using the **Select Site Collection** page, click the **/sites/ProjectManagementLab** site, and then click **OK**. If this worked, skip ahead to Step e.
    i. **Note**: If your site collection is not listed ensure that the web application is set to http://litwareinc.com.
        1. To fix this, click the Web Application URL, and then click Change Web Application.
        2. Click the http://litwareinc.com Web application on the Select Web Application page.
    ii. Try step 8 again.
e. Verify that the information looks correct on the **Activate Form Template** page, and then click **OK**.

15. Now we need to verify that the form template is available on the site collection.
    a. In **Internet Explorer**, browse to **http://litwareinc.com/sites/ProjectManagementLab.**
    b. Click **View All Site Content**.
    c. On the **All Site Content** page, in the **Document Libraries** section, click **Form Templates**. The **TimeSheet** template should be in the **Form Templates** list.
16. Finally we can now utilize the form template as a template for a library
    a. From the **Litware Project Management** site browse to your **South Division** - **Daily Timesheets Library**
    b. Select and delete each existing test form in this library to make way for the new template.
    c. On the top navigation bar, click **Settings**, and then click **Form Library Settings**.
    d. On the **Customize Daily Timesheets** page, click **Advanced settings**.
    e. On the **Form Library Advanced Settings: Daily Timesheets** page, under **Allow management of content types**, select **Yes**, and then click **OK**.

f. On the **Customize Daily Timesheets** page, in the **Content Types** section, click **Add from existing site content types**.
g. On the **Add Content Types: Daily Timesheets** page, in the **Available Site Content Types** list, select the **TimeSheet** form template to use as the template for this library, click **Add**, and then click **OK**.
h. On the **Customize Daily Timesheets** page, in the **Content Types** section, click on the **Form** content type.
i. On the **List Content Type: Form** page, select **Delete this content type** and select **OK** to the confirmation message.

17. Now you can test out this new form by entering some data. Note: If you edit an existing form you will receive a warning message stating that "**There has been an error while processing the form**." If you Show error details your are informed that "The given key was not present in the dictionary." This error is related to internal data on your InfoPath form and can be ignored with no ill effects, Click Continue.

Lab 13: Creating Custom Workflows for MOSS

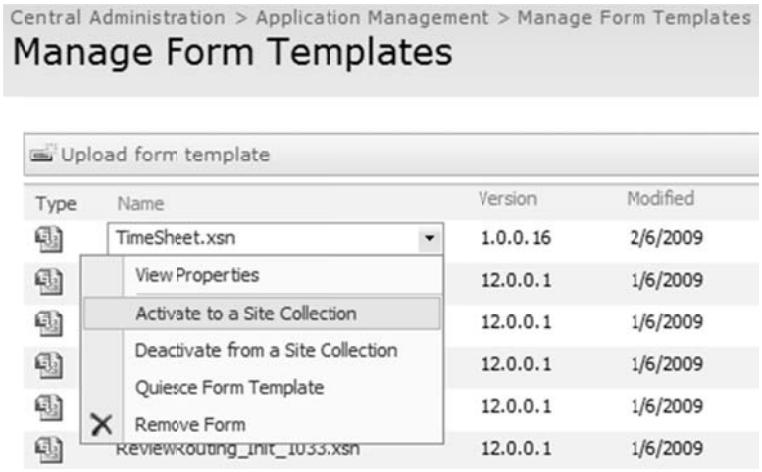
Lab Time: 60 minutes

Lab Overview: Litware is a consultancy company requiring each of their consultants to fill-in a daily timesheet summarizing all of their activities done during that day. The timesheet data is stored in a custom SharePoint list for later use. At the end of every week managers must review the timesheets to approve or disapprove the hours submitted.

- In Exercise 1 you will create a workflow to create a task assigned to the consultant's manager and provide them with a form that allows easy acceptance or rejection of the timesheet.
- In Exercise 2, you will use Visual Studio to create a custom workflow.

Lab Setup—5 pieces:

- **Lab 09** on Custom Content types **MUST** be completed before you can start this lab.
- Perform this step **ONLY** if you didn't make **Lab 12** on **InfoPath**:
    - Open **Windows Explorer**.
    - Navigate to the starter files of this lab: **C:\Student\Labs\13_Workflow\Starter Files\Setup**.
    - Double-click the **Mod11Starter.bat**. This file will alter Active Directory to set the Department to Development and also set the manager field for certain users. It will also extend the profile database in SharePoint to include a setting for **HourlyRate** and it will pre-populate it for those in the Development Department.
        - Navigate to **SharePoint 3.0 Central Administration** (Start - All Programs - Microsoft Office Server - SharePoint 3.0 Central Administration)
        - In Central Administration on your **Quick Launch** bar (left side of screen) click on **Litware SSP**
        - In the **Litware SSP User Profiles and My Sites** section click on **User profiles and properties**
        - Click on **Start full import** (this will pull the new settings from Active Directory into SharePoint so that we might utilize them in this lab).
    - Double-click the **deploytimesheetsolution.bat** file to add and deploy the InfoPath timesheet solution.
    - To make the form template available to users, the form must be activated to a site collection.
        - Navigate to **Central Administration** > **Application Management** > **Manage Form Templates** page.
        - Point to the **TimeSheet.xsn** form template that you want to activate, click the arrow that appears, and then click **Activate to a Site Collection**.



  - On the **Activate Form Template** page, click the **Site Collection URL**, then click **Change Site Collection**.
  - Using the **Select Site Collection** page, click the **/sites/ProjectManagementLab** site, and then click **OK**.

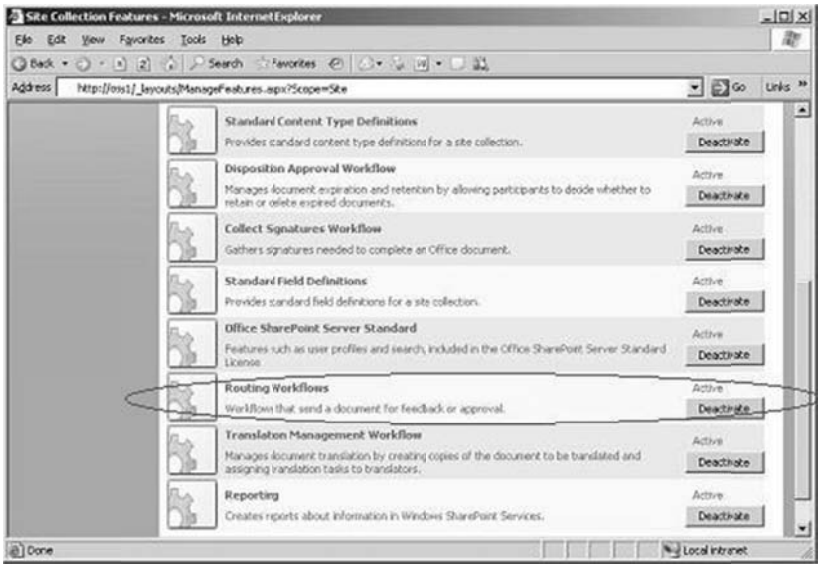
- Perform the following steps **ONLY** if you attempted but did not complete **Lab 12** on **InfoPath**:
  - Open **Windows Explorer**.
  - Navigate to the starter files of this lab: **C:\Student\Labs\13_Workflow\Starter Files\Setup**.
  - Double-click the **retracttimesheetsolution.bat** file to retract and delete the InfoPath timesheet solution.
  - Double-click the **deploytimesheetsolution.bat** file to add and deploy the InfoPath timesheet solution.
  - Finally we can now utilize the form template as a template for a library
- Perform this step **ONLY** if you didn't make **Lab 12** on **InfoPath** or if you attempted but did not complete **Lab 12** on **InfoPath**:
  - Open Internet Explorer and navigate to **http://litwareinc.com/sites/ProjectManagementLab/SouthDivision**.
  - If you have a **Daily Timesheets** forms library:
    - Open the **Daily Timesheets** forms library.
    - Select and delete each existing test form in this library to make way for the new template.
  - If you have no **Daily Timesheets** forms library:
    - Create a new library based on the **Forms Library** template and give it the name **Daily Timesheets**.
  - On the top navigation bar, click **Settings**, and then click **Form Library Settings**.
  - On the **Customize Daily Timesheets** page, click **Advanced settings**.
  - On the **Form Library Advanced Settings: Daily Timesheets** page, under **Allow management of content types**, select **Yes**, and then click **OK**.
  - On the **Customize Daily Timesheets** page, in the **Content Types** section, click **Add from existing site content types**.

    - On the **Add Content Types: Daily Timesheets** page, in the **Available Site Content Types** list, select the **TimeSheet** content type to use as the template for this library, click **Add**, and then click **OK**.
    - On the **Customize Daily Timesheets** page, in the **Content Types** section, click on the **Form** content type.
    - On the **List Content Type: Form** page, select **Delete this content type** and select **OK** to the confirmation message.
- **EVERYONE** must set the necessary permissions on both the **Litwareinc** site and the **ProjectManagementLab** site**:**
    - Open Internet Explorer and navigate to **http://litwareinc.com**.
    - Select **People and Groups** from the **Quick Launch**.
    - Select the **Litware Inc Members** group.
    - Click the **New** button and choose **Add User**.
    - Add the following string in the **Add Users** section: **BrianC;AngelaB;JayH**
    - Click the **Check** button to validate the users.
    - Click the **OK** button.
    - Navigate to the **http://litwareinc.com/sites/ProjectManagementLab** site.
    - Select **People and Groups** from the **Quick Launch**.
    - Select the **Litware Project Management Members** group.
    - Click the **New** button and choose **Add Us**er.
    - Add the following string in the Add Users section: **BrianC;AngelaB;JayH**
    - Click the **Check** button to validate the users.
    - Click the **OK** button.
    - Check if you have a **Daily Timesheets** forms library in the **ProjectManagementLab** site. A custom workflow feature is a site collection scoped feature. You will need this library to be able to properly create a custom workflow project in **Visual Studio 2008**. If you have no **Daily Timesheets** forms library, create a new library based on the **Forms Library** template and give it the name **Daily Timesheets**.
    - On the top navigation bar, click **Settings**, and then click **Form Library Settings**.
    - On the **Customize Daily Timesheets** page, click **Advanced settings**.
    - On the **Form Library Advanced Settings: Daily Timesheets** page, under **Allow management of content types**, select **Yes**, and then click **OK**.
    - On the **Customize Daily Timesheets** page, in the **Content Types** section, click **Add from existing site content types**.
    - On the **Add Content Types: Daily Timesheets** page, in the **Available Site Content Types** list, select the **TimeSheet** content type to use as the template for this library, click **Add**, and then click **OK**.
    - On the **Customize Daily Timesheets** page, in the **Content Types** section, click on the **Form** content type.

Exercise 1: Associating and Initiating SharePoint Workflows

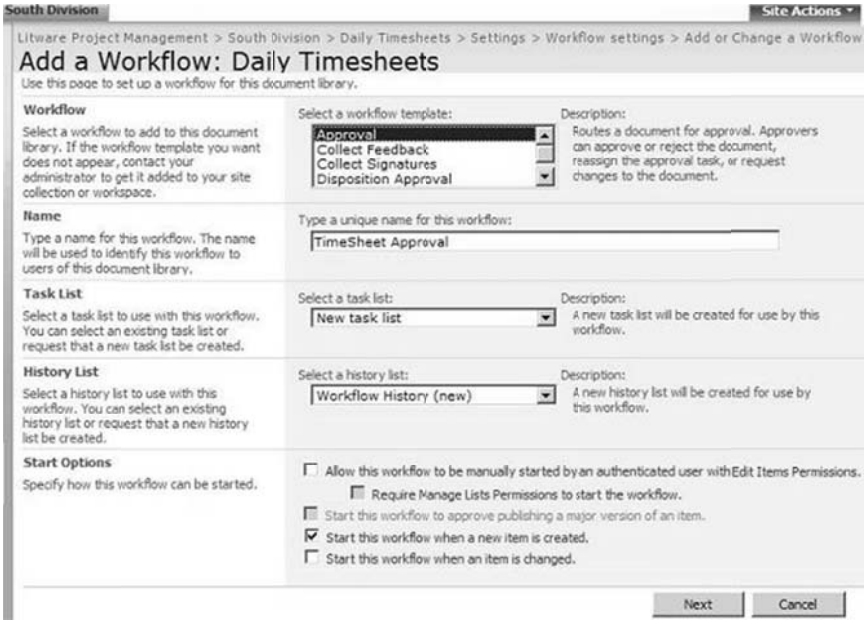
SharePoint provides several workflow types out of the box. One of those workflow types is an approval workflow. Whenever a user needs to interact with the workflow, a task is added to a **Tasks** list. To better facilitate user interaction, the approval workflow provides InfoPath forms to simplify interaction with the tasks. In this exercise you'll attach an approval workflow to the **Daily Timesheets** document library to facilitate manager approval of the timesheets.

1. Before you can use the **Approval** workflow which is included with SharePoint, you'll need to enable the **Routing Workflows** feature on the site collection.
    a. Open the site at **http://Litwareinc.com/sites/ProjectManagementLab** and navigate to the site settings using the **Site Actions -> Site Settings** menu.
    b. In the **Site Collection Administration** page find and click the **Site Collection Features** link.
    c. In the features list, find the **Routing Workflows** feature and make sure that it is activated.



2. Now that all the preliminary work is done, it's time to associate the **Approval** workflow with the **Daily Timesheet** Form library.
    a. Open the site at **http://Litwareinc.com/sites/ProjectManagementLab/SouthDivision** and navigate to the **Daily Timesheets** Form library.
    b. Open the Form library settings by using the **Settings -> Form Library Settings** menu item.
    c. On the Form library settings page, navigate to **Workflow Settings** in the **Permissions and Management** section.
    d. Select the **Approval** workflow template and name the workflow **TimeSheet Approval**.
    e. Choose **New task list** from the Task List dropdown.
    f. Leave the default setting **Workflow History (new)** so a new workflow history list will be created for you.
    g. Be sure to uncheck **Allow this workflow to be manually started by an authenticated user with Edit Items Permissions**.

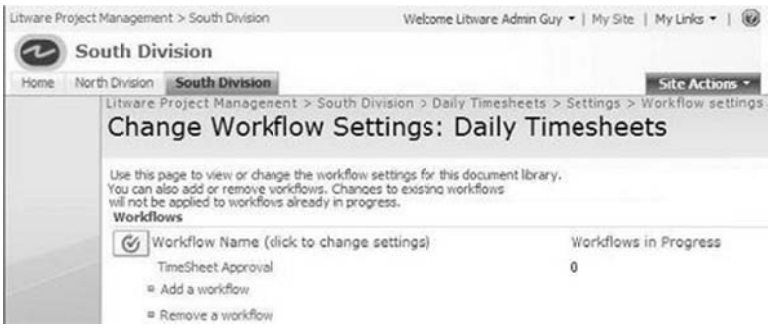
h. Check the box next to **Start this workflow when a new item is created**.



i. Click the **Next** button to move to the next step.

j. This page allows you to configure many settings related to the approval process. You're only interested in the **Approvers** text box. Enter **LITWAREINC\Administrator** so the administrator must approve the document. Press **Ctrl + k** to Check the name.

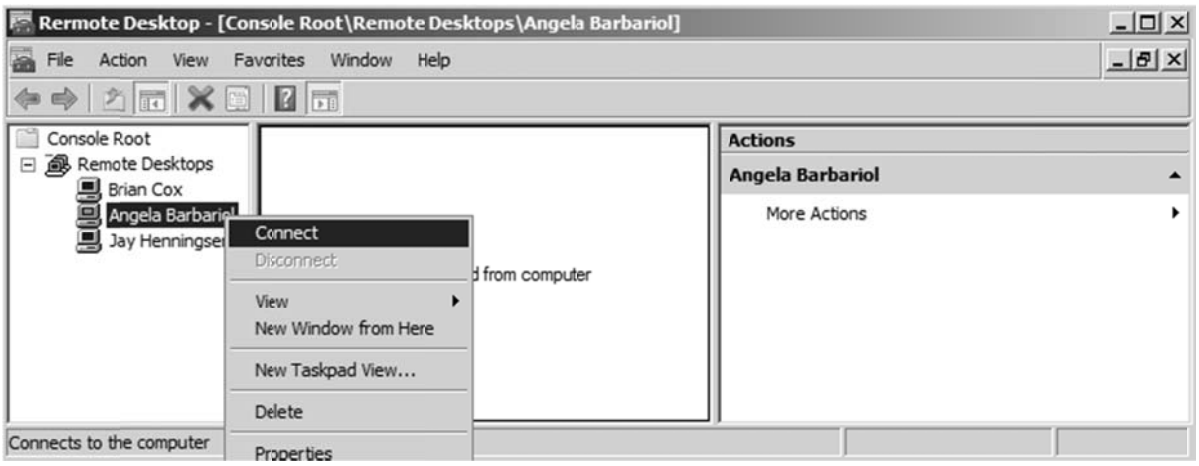
k. Finally click **OK** to complete the workflow creation.



3. You've successfully created the **Approval** workflow and associated it with the form library. All that's left to do is test the approval process.

a. Minimize all Windows so you can see the desktop. You should be able to see four shortcuts that allow you to start Remote Desktop sessions under the identities of different users. Click the shortcut with the caption of RemoteDesktop.msc to launch the Remote Desktop management console. Make sure the management console is maximized.

b. Click the node **Angela Barbariol** and choose **Connect**.



c. Enter the password **pass@word1**. This launches a remote desktop session for the user Brian Cox.

d. You should notice that you have opened a remote desktop session on this computer for **AngelaB**.

e. Open **Internet Explorer** and open the site at **http://Litwareinc.com/sites/ProjectManagementLab/SouthDivision** and navigate to the **Daily Timesheets** document library.

f. Create a **New** Timesheet for Angela according to the following picture.

Note: if you ran the setup instead of making the InfoPath lab, you will have a textbox instead of a dropdown in the Project column. In that case fill out the project name you see in the picture.

i. Click on **Submit** for the timesheet.

ii. Then click **OK** to the completion message.

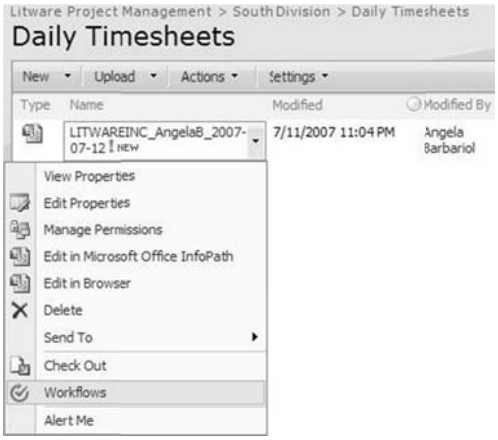
iii. Notice that the **TimeSheet Approval** for Angela's Timesheet is **In Progress** (the workflow automatically starts as a result of the document being created)

g. Click on Angela's **Start** menu and select the **Log Off** choice.

h. Click **OK** to the message that appears. This will allow us to act as the approver (i.e. Administrator).

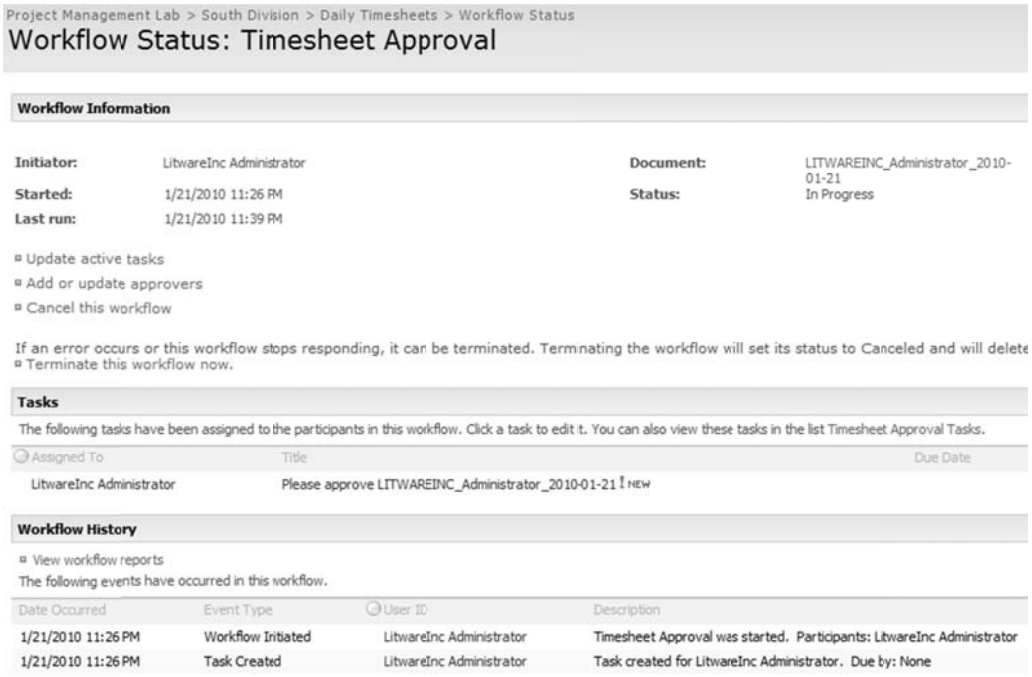
4. As the **Administrator**, navigate to the **Daily Timesheets** Form library in the **Litware Project Management - South Division** site.

a. Select the **Workflow** menu item for the AngelaB timesheet in the document library.

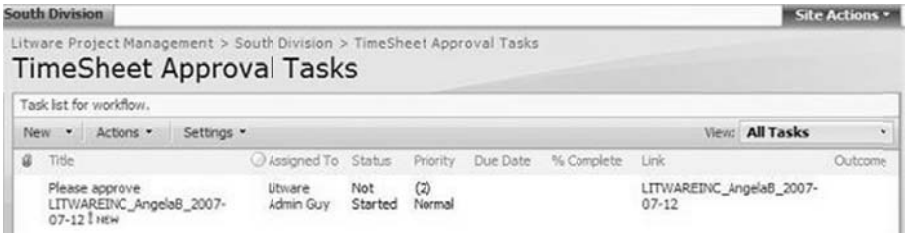


b. Notice that the workflow is running with a status of **In Progress**.

c. Click on the **TimeSheet Approval** hyperlink and take a look at the workflow status. As part of the process the workflow has created a task for the user **LITWAREINC\Administrator** to resolve.



d. Navigate to the **Tasks** list to see the task that has been created by the workflow.



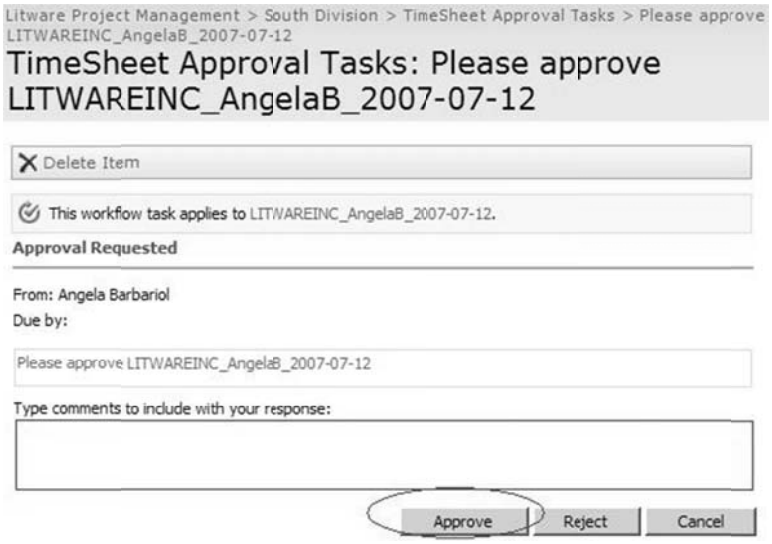
e. From the **South Division** site click on **View All Site Content**

f. Select the **Tasks** list with a description of **Task list for workflow** that was auto-generated for us upon the creation of our workflow.

5. The next step in the workflow is to complete the task so the workflow can continue.

   a. On the **TimeSheet Approval Tasks** list edit the properties of the task in the list either by selecting **Edit Item** in the menu or clicking the item link.

b. On the approval form, you can enter comments and either approve or reject the document. In this case **approve** the document by clicking the **Approve** button.



6. The last step is to verify that the workflow has completed successfully and that the timesheet is approved. Navigate to the **Daily Timesheet** list on the South Division Site. You'll notice that the **Timesheet Approval** column for the document you approved is set to **Approved**.

Exercise 2: Create a sequential workflow with Visual Studio 2008

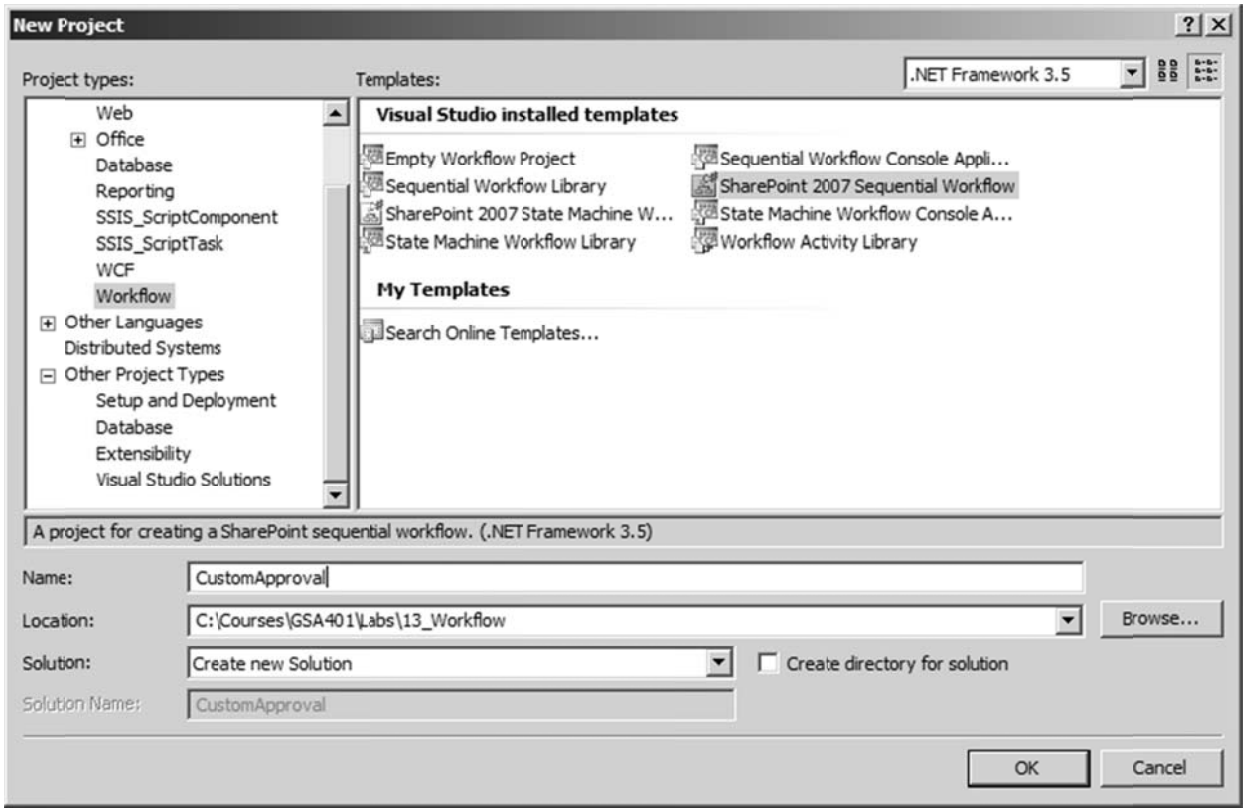
Even though SharePoint provides several workflow solutions, sometimes you need more functionality than is provided out of the box. In this case developers can create custom workflow solutions as well as custom InfoPath 2007 forms for managing workflow initialization and task completion. This exercise will help you to create a custom approval workflow that utilizes Visual Studio 2008 to create the workflow and InfoPath 2007 to create the user interface components.

There are several Main Tasks in this lab:

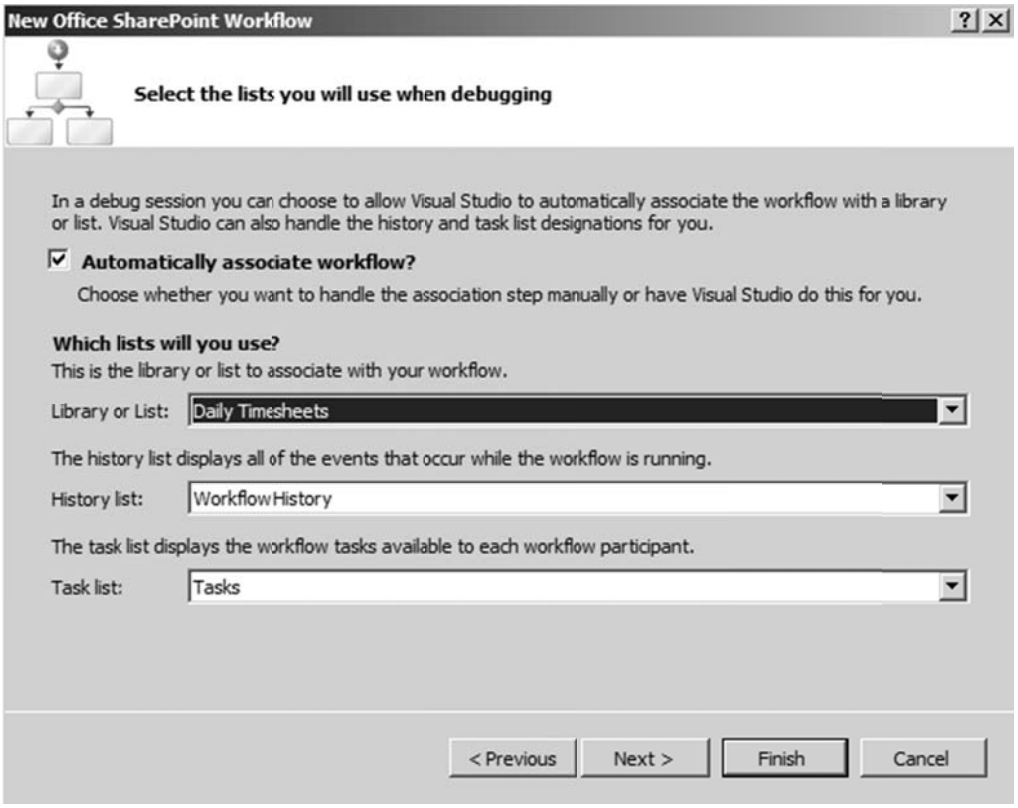
- Task A: Create the Basic SharePoint Workflow using the STSDEV utility
- Task B: Create and Publish the InfoPath Support Forms needed by the Workflow
- Task C: Modify the Basic workflow to make use of the InfoPath Forms
- Task D: Configure the workflow Feature for Deployment

Task A: Create the Basic SharePoint Workflow

1. Start by creating a new project in **Visual Studio 2008** using the **SharePoint 2007 Sequential Workflow** template. Make sure the **.NET Framework 3.5** is selected. Give it the name **CustomApproval**. Save it in the **C:\Student\Labs\13_Workflow\Lab** directory.



2. This starts the **New Office SharePoint Workflow** wizard.
    a. Fill out **Custom Approval** as name for the workflow.
    b. Fill out the URL to the SharePoint site **http://litwareinc.com/sites/ProjectManagementLab**.
    c. Click the **Next** button.
    d. Select the **Daily Timesheets** library as the library you want to associate with your workflow.
    e. Leave the two other selections (**Workflow History** and **Tasks**) as is.



f. Click the **Next** button.

g. In the last step of the wizard ensure that the checkboxes for **Manually by users** and **When an item is created** are checked. Click the **Finish** button.

3. The first step in implementing the workflow is to retrieve the creation data information such as the manager's user name that needs to approve the document and any special comments associated with the document approval process.

   a. Start by adding three member fields to the **Workflow1.cs** or the **Workflow1.vb** class.

      i. View the code for **Workflow1.cs** by right clicking **Workflow1.cs** in **Solution Explorer** and selecting **View Code**.

      ii. Add the following three member fields to the class as **strings: user, comments,** and **taskStatus**. Also, add a member field called **workflowId** as **Guid**.

C#

```
namespace CustomApproval
{
   public sealed partial class SequentialWorkflow01:
      SharePointSequentialWorkflowActivity
   {
      public SequentialWorkflow01()
      {
         InitializeComponent();
      }

      public Guid workflowId = default(System.Guid);
      public SPWorkflowActivationProperties workflowProperties =
         new SPWorkflowActivationProperties();
```



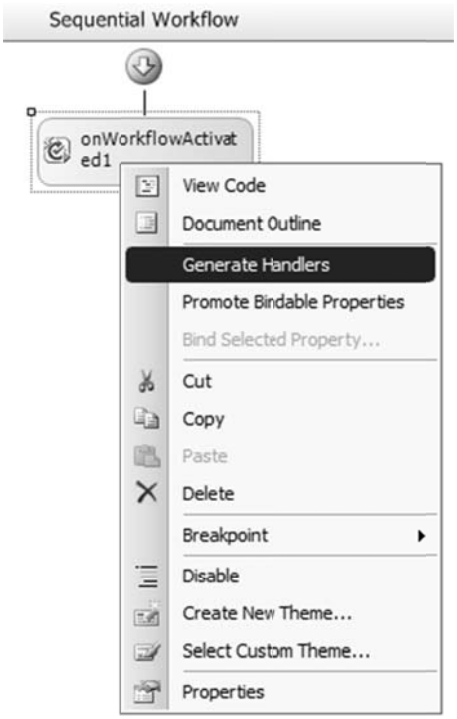
```
    public string user = default(string);
    public string comments = default(string);
    public string taskStatus = default(string);
```

VB.NET

```
Public Class Workflow1
    Inherits SharePointSequentialWorkflowActivity

    Public Sub New()
        MyBase.New()
        InitializeComponent()
    End Sub

    Public workflowProperties As SPWorkflowActivationProperties = _
       New Microsoft.SharePoint.Workflow.SPWorkflowActivationProperties
    Public workflowId As Guid

    Public user As String
    Public comments As String
    Public taskStatus As String
```

b. Next using your **Solution Explorer** open **Workflow1.cs** in designer mode.

c. Right-click on **onWorkflowActivated1** activity and choose **Generate Handlers** to generate a handler for the **Invoked** event.



d. Implement the **onWorkflowActivated1_Invoked** handler by storing the **workflowId** of the new SharePoint workflow in the internal field.

C#

```
private void onWorkflowActivated1_Invoked(
 object sender, ExternalDataEventArgs e)
```



```
{
   // store the new workflow's id
   workflowId = workflowProperties.WorkflowId;
}
```

VB.NET

```
Private Sub onWorkflowActivated1_Invoked(ByVal sender As Object, _
   ByVal e As ExternalDataEventArgs)
   ' store the new workflow's id
   workflowId = workflowProperties.WorkflowId
End Sub
```

4. Next you'll need to create a new task using the **CreateTask** activity. Start by creating several member fields in the **Workflow01.cs** or the **Workflow1.vb** class to store information about the tasks properties before and after editing. These will be used to retrieve the results of the task's completion. Use the code sample below to help you add the code to the class. Add these after your other Class member variables.

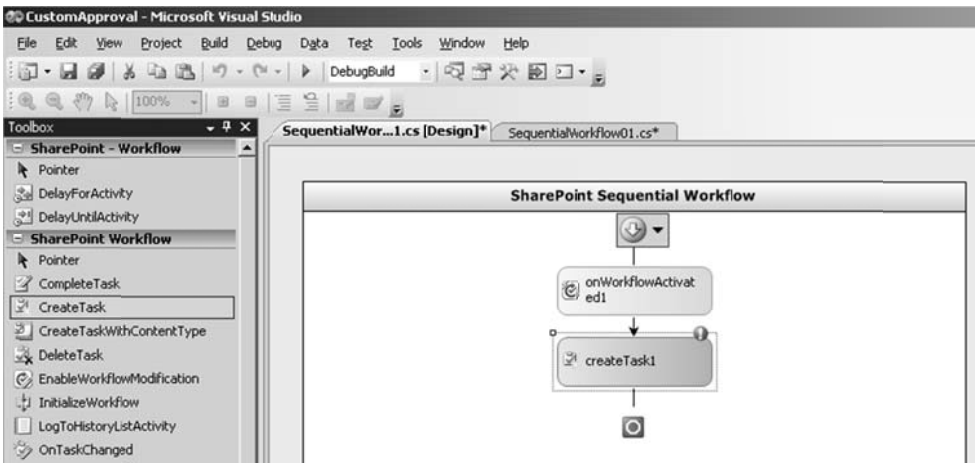
C#

```
public Guid taskId = default(System.Guid);
public SPWorkflowTaskProperties taskProperties =
   new SPWorkflowTaskProperties();
public SPWorkflowTaskProperties beforeProperties =
   new SPWorkflowTaskProperties();
public SPWorkflowTaskProperties afterProperties =
   new SPWorkflowTaskProperties();
```

VB.NET

```
Public taskID As Guid
Public taskProperties As New SPWorkflowTaskProperties()
Public beforeProperties As New SPWorkflowTaskProperties()
Public afterProperties As New SPWorkflowTaskProperties()
```

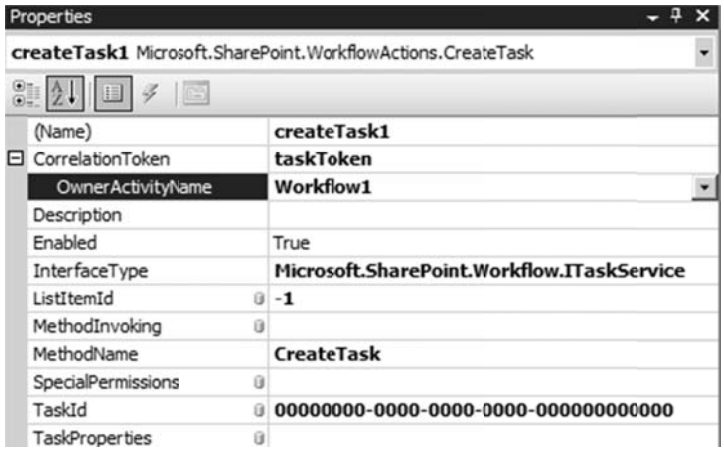
   a. Open **Workflow.cs** or **Workflow1.vb** in designer mode and drag a **CreateTask** activity onto the canvas after the activation activity. **Hint**: this item should be located in the **Toolbox** in the **SharePoint Workflow** section.



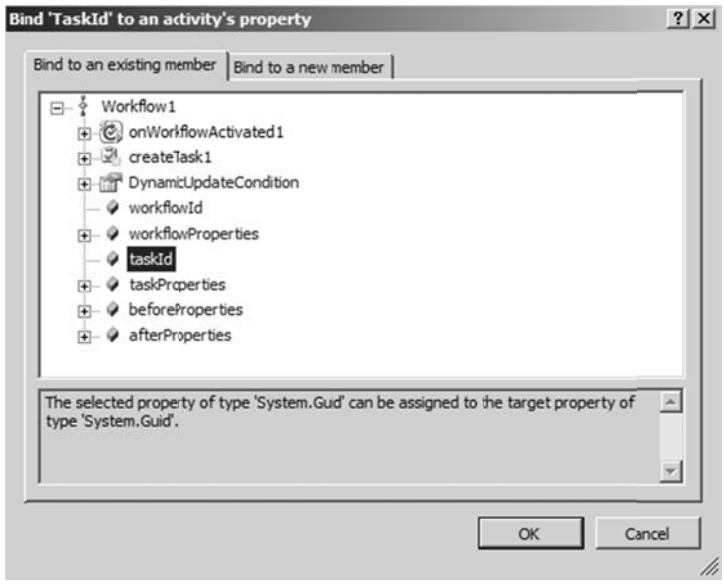
   b. Now that the activity is created, you'll need to configure its **CorrelationToken** property:
      i. First, in the **Workflow1** Design Window select the **CreateTask1** item. Examine your properties window to ensure that it is now showing properties for **createTask1**.

ii. In the properties set the **CorrelationToken** to **taskToken** (i.e. type **taskToken** in the textbox)

iii. Next expand out the **CorrelationToken** (click the + to the left of **CorrelationToken**), and then set the **CorrelationToken's OwnerActivityName** to **Workflow1** (using the dropdown choices).



iv. Set its **TaskId** property to **taskId**—click on the ellipsis (…) and choose **taskId** from the dialog, then click **OK**.



v. Follow those same instructions to set its **TaskProperties** property to **taskProperties**. These values are used by the activity to manage the lifetime of the workflow and to provide parameters for the task creation.

c. Finally you'll need to populate the properties of the task using the **taskProperties** variable. To access the code, right click **createTask1** (in the **Workflow1** Design window) and click **Generate Handlers**. Using the code below, populate the task properties. The only property that is required is the **taskID** which was set in the designer.

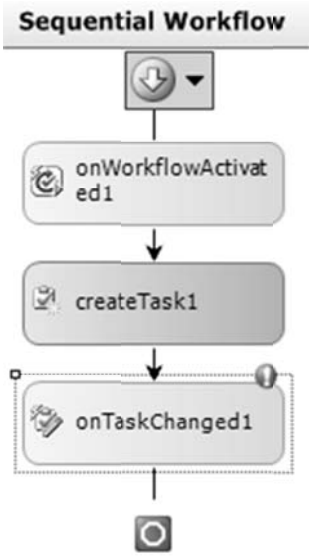
C#


```
private void createTask1_MethodInvoking(object sender, EventArgs e)
{
   // initialize the task ID
   taskId = Guid.NewGuid();

   // populate the properties of the task
   taskProperties.AssignedTo = user;
   taskProperties.Description = "Approve the document.";
   taskProperties.Title = "Timesheet Approval";

   // populate the type of the form and extended properties
   taskProperties.ExtendedProperties["User"] = user;
   taskProperties.ExtendedProperties["Comments"] = comments;
}
```

VB.NET

```
Private Sub createTask1_MethodInvoking(ByVal sender As Object, _
    ByVal e As EventArgs)
    ' initialize the task ID
    taskID = Guid.NewGuid()

    ' populate the properties of the task
    taskProperties.AssignedTo = user
    taskProperties.Description = "Approve the document."
    taskProperties.Title = "Timesheet Approval"

    ' populate the type of the form and extended properties
    taskProperties.ExtendedProperties("User") = user
    taskProperties.ExtendedProperties("Comments") = comments
End Sub
```

5. The previous step created a new task, now it's time to wait for the task to change and store the results of the task.
    a. Open **Workflow1.cs** or **Workflow1.vb** in designer mode and drag a **OnTaskChanged** activity onto the canvas after the **createTask1** activity.



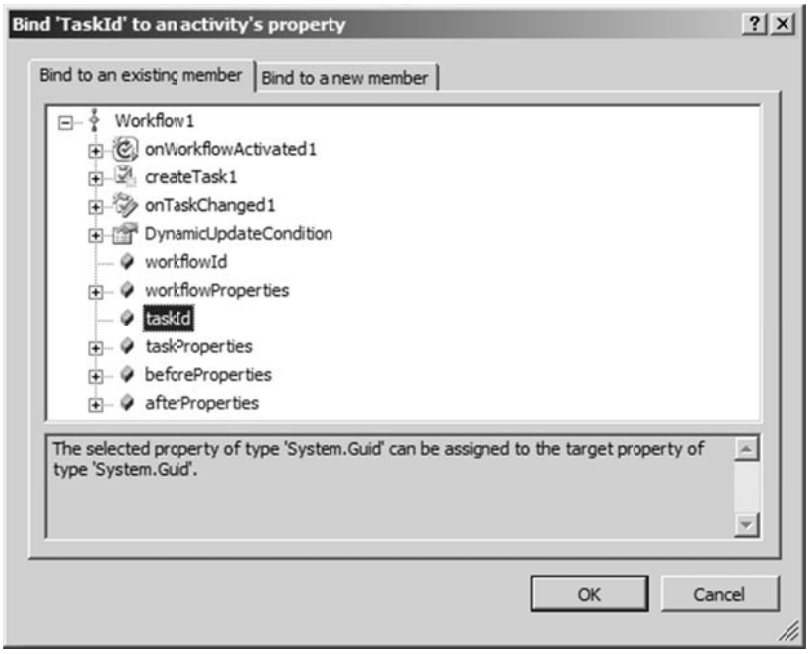
    b. Now that the activity is created, you'll need to configure its **CorrelationToken**.

i. First, in the **Workflow1** Design window select the **OnTaskChanged1** item. Examine your properties window to ensure that it is now showing properties for **OnTaskChanged1**.

ii. In the properties set the **CorrelationToken** to **taskToken** (i.e. type **taskToken** in the textbox).

iii. Expand out the **CorrelationToken** property and set the **CorrelationToken's OwnerActivityName** to **Workflow1**.

iv. Then set its **TaskId** property to **taskId** by clicking the **…** button and choosing tasked from the treeview in the binding dialog.



v. Set the **BeforeProperties** to **beforeProperties**, and its **AfterProperties** to **afterProperties** using the same way.

6. Finally you'll need to write some code associated with the **OnTaskChanged1** handler that will retrieve the status value from the after properties and store it in the member variable **taskStatus**.
   a. To access the code, right click **onTaskChanged1** (in the **Workflow1 Design** window) and choose **Generate Handlers**.

C#

```
private void onTaskChanged1_Invoked(object sender, ExternalDataEventArgs e)
{
   // retrieve the TaskStatus property from the infopath form
   taskStatus =
this.afterProperties.ExtendedProperties["TaskStatus"].ToString();
}
```

VB.NET

```
Private Sub onTaskChanged1_Invoked(ByVal sender As Object, _
    ByVal e As ExternalDataEventArgs)
    ' retrieve the TaskStatus property from the infopath form
    taskStatus =
Me.afterProperties.ExtendedProperties("TaskStatus").ToString()
```



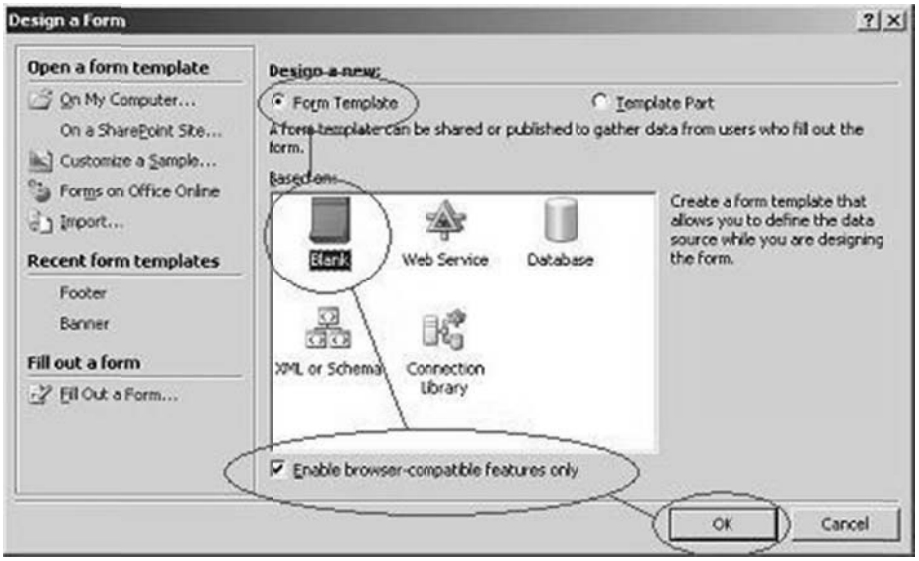
```
End Sub
```

7. The last step in defining the workflow is to complete the task and set its status to the **TaskStatus** value retrieved in the previous step.
    a. Open **Workflow1.cs** or **Workflow1.vb** in designer mode and drag a **CompleteTask** activity onto the canvas after the **onTaskChanged1** activity.
    b. Once again, you'll need to set the **CorrelationToken** property to **taskToken** and then verify/set the **CorrelationToken's OwnerActivityName** to **Workflow1**. Set its **TaskId** property to **taskId**, its **TaskOutcome** to **taskStatus**. The **TaskOutcome** property is the text value that is used when setting the status value of the completed task.
8. Now that the workflow is complete, you'll need to sign the assembly and build the workflow assembly.
    a. To sign the assembly, navigate to the **Signing** tab on the project properties window, by right-clicking on **CustomApproval** in the Solution Explorer and clicking on **Properties**. Check the **Sign the assembly** checkbox, and use the dropdown to browse for the **CustomApproval.snk** file in the directory **C:\Student\Labs\13_Workflow\Starter Files\Lab**.
    b. Save your changes and close the **Properties** page.
9. Finally **build** the assembly and fix any compile errors that may come up.

Task B: Create and Publish the InfoPath Support Forms needed by the Workflow

Now that you've finished the workflow, it's time to create the InfoPath 2007 forms that will be used by the workflow. This task is focused on creating these forms.

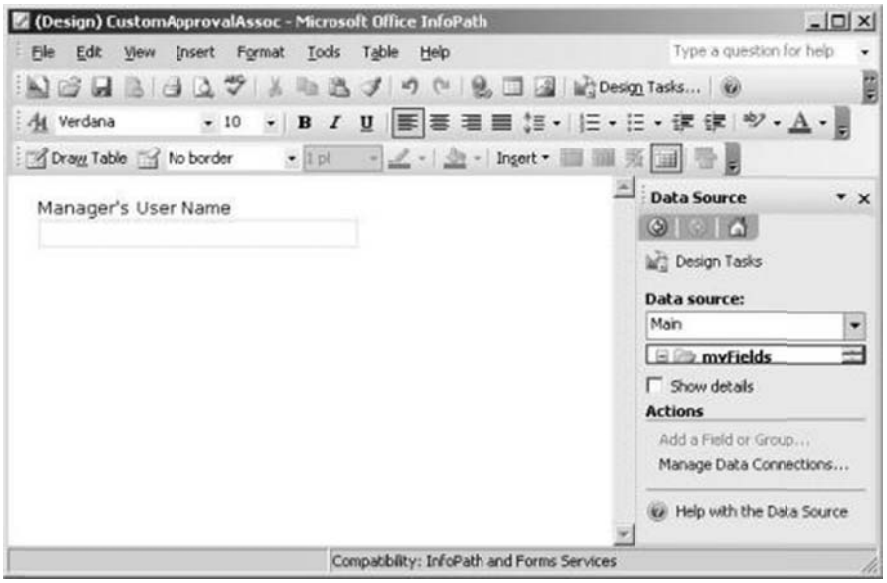
1. The first step in using a workflow is to associate one with a document library or list. In this form, you'll need to retrieve the manager's username to assign the approval tasks to.
    a. Start by opening InfoPath 2007 and designing a new blank form that is web enabled, by choosing **Design a Form Template** from the pop-up starter window and choosing the options in the picture below:



b. Select the **Data Source** view in the **Design tasks** pane and add two **string** fields named **User** and **Comments** nodes to the data source.

c. Next add the text **Manager's User Name** to the top of the form followed by a text box bound to the **User** data source field.

   i. **Hint**: After typing the text in go to the **Data Source** panel and click on the **User** field then click on the **dropdown arrow** and select **Text Box**. This will insert a textbox but you may need to remove the label **User**:



d. The last control to add to the form is the **Create** button that will submit the form back to SharePoint. Switch to the controls task pane and drag a button out onto the form.

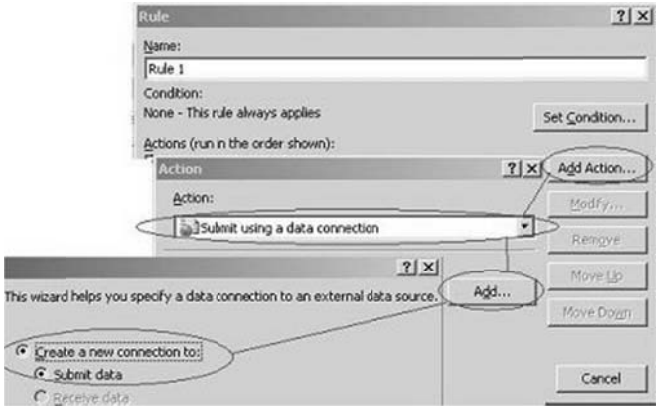
e. Right click on the new button and click **Button Properties**

   i. Change the button's **Label** and **ID** to **Create**.

   ii. Next click **Rules**... (in the following steps you will create two actions, **submit** and **close form**).

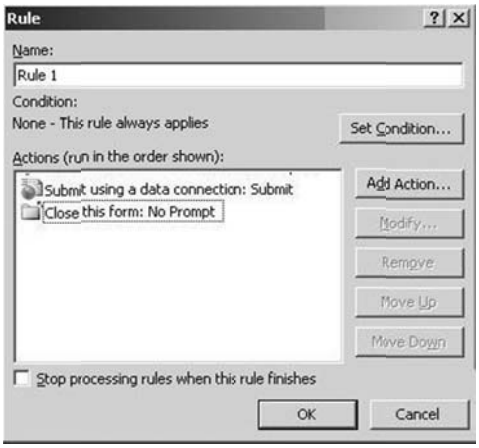
      1. Then click **Add - Add Action**... and select **Submit using a data connection**.

a. Click **Add**... then **Create a new connection to: Submit data** and click **Next>**



b. On the **Destination for submitting your data** page select **To the hosting environment**... and click **Next>** and **Finish**. Then click **OK**.

2. Add another **action** that **closes the form** and click **OK**.



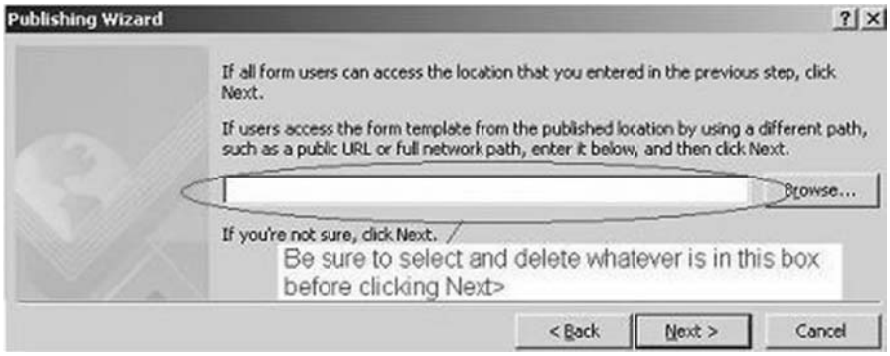
3. Click **OK** (3 more times)

f. Set the Security for the Form
   i. In InfoPath on the **Tools** menu - select **Form Options** - select **Security and Trust** - Remove the check from **Automatically determine security level (recommended)** and select **Domain** ...
   ii. Click **OK**

g. Now that the form is done, save it to **C:\Student\Labs\13_Workflow\Lab\CustomApprovalAssoc.xsn**. (**Note**: you will have to type **CustomApprovalAssoc.xsn** in the **File Name:** textbox.)

h. Finally **Publish** the completed form to the **network location**
   i. From the **File** menu select **Publish**... then specify **To a network location** and click **Next>**
   ii. Click **Browse**... to specify the location to publish your form and use the path **C:\Student\Labs\13_Workflow\Lab\CustomApproval\ CustomApprovalAssoc.xsn** (**Note**: you will have to type in **CustomApprovalAssoc** in the **File Name:** textbox) and click **OK** and then **Next>**

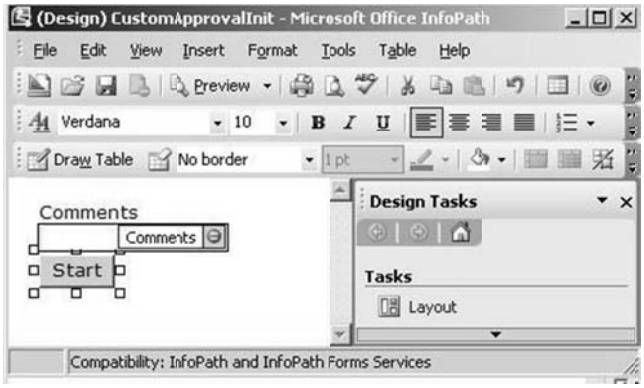
iii. Delete the alternate access path. Then click **Next>**



iv. Click **OK** on the pop-up window.

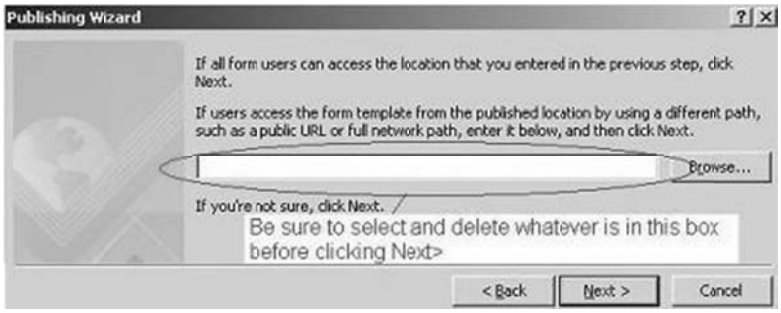
v. Click **Publish** and then click **Close** to complete publication.

2. Next you'll need to define the form that is displayed when a new instance of the workflow is started.
   a. Start by **Opening** the **C:\Student\Labs\13_Workflow\Lab\CustomApprovalAssoc.xsn** template in **Design** mode and doing a **Save As...** to save it as **C:\Student\Labs\13_Workflow\Lab\CustomApprovalInit.xsn**. This is done so the primary data source will have the same namespace in both forms.
   b. Next you will modify this form.
      i. Change the phrase **Manager's User Name** to **Comments** at the top of the form.
      ii. Change the binding of the text box to the **Comments** data source field.
      iii. Right-click on the **User** TextBox and select **Change Binding...**
      iv. Change the field binding from **User** to **Comments** and click **OK**
      v. Then change the **Label** and **ID** of the button to **Start**, by double clicking on the button and setting the two fields.



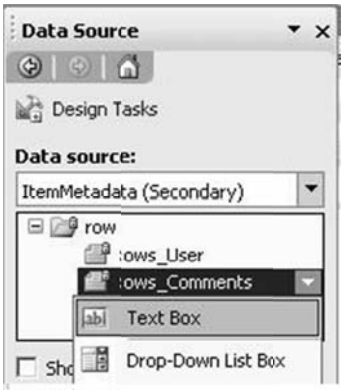
   c. Set the Security for the Form
      i. In InfoPath on the **Tools** menu - select **Form Options** - select **Security and Trust** - Remove the check from **Automatically determine security level (recommended)** and select **Domain** ...
      ii. Click **OK**
   d. Now that the form is done, save it (again to **C:\Student\Labs\13_Workflow\Lab\CustomApprovalInit.xsn**).
   e. Finally **Publish** the completed form to the **network location**

i. From the **File** menu select **Publish**... then specify **To a network location** and click **Next>**
ii. Click **Browse**... to specify the location to publish your form and use the path **C:\Student\Labs\13_Workflow\Lab\CustomApproval\CustomApprovalInit.xsn** and click OK and then Next>
    1. **Important**: As you have already published this form under a different name you **MUST** edit the **Form template path and file name**: and **Form template name**: to be **CustomApprovalInit** or you will overwrite your other form.
iii. Make sure that you remove alternate access path so that it is blank. Then click **Next>**

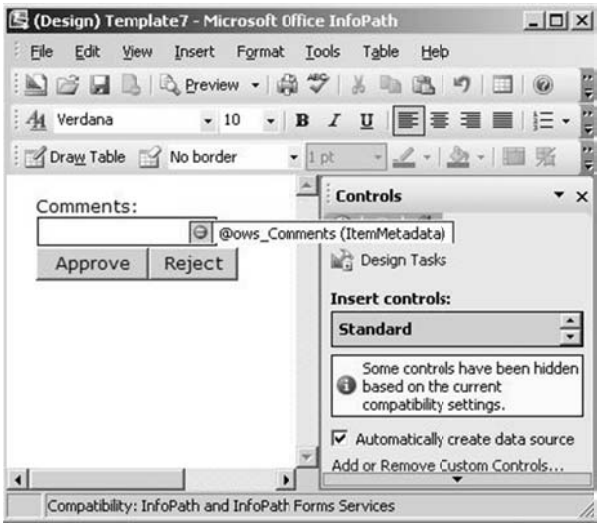


iv. Click **OK** on the Warning message that appears.
v. Click **Publish** and then click **Close** to complete publication.

3. The last form to create is the form that is used when performing a task. In this form the manager will be presented with the comment entered in the Init form and two buttons, one to approve the document and another to reject the document.
    a. Start by opening InfoPath 2007 and design a new blank form that is web enabled.
    b. Select the **Data Source** view in the **Design Tasks** pane and add a **string** field named **TaskStatus** to the data source.
    c. Now you'll need to add a **Receive XML** data connection.
        i. Select the **Tools -> Data Connections** menu item.
        ii. Click **Add** then **Create a new connection to: Receive Data.**
        iii. Click **Next**.
        iv. Then specify **XML document** for your data source and click **Next>**
        v. On the **XML data file details** page click **Browse**... then use the **C:\Student\Labs\13_Workflow\Starter Files\Lab\ItemMetadata.xml** file to populate the data source. This data connection is used to provide the form with the initialization information created in the other forms.
        vi. Leave the **Include the data as a resource file** checked.
        vii. click **Next>** and then **Next>** again and **Finish**. Click **Close** to close your **Data Connections** window.
    d. Next add the text **Comments** to the top of the form followed by a text box bound to the **ows_Comments** field in the **ItemMetadata** source. (Note: Remove the **OWS Comments**: label if necessary)



e. The last controls to add to the form are the **Approve** and **Reject** buttons that will submit the form back to SharePoint. In **Design Tasks**, switch to the **Controls** task pane and add two buttons onto the form. **Rename** the buttons and set their **ID** to **Approve** and **Reject** using the properties window.



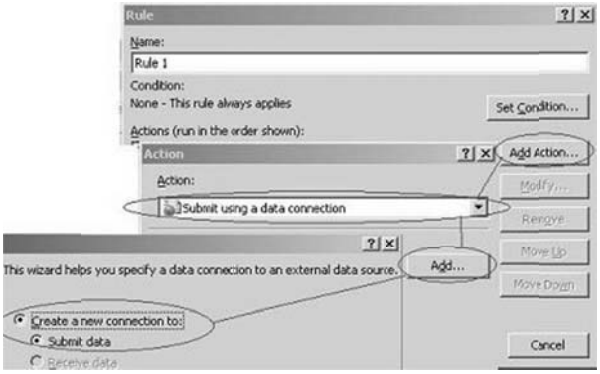
f. Configure the actions on the **Approve** button by right clicking the button and selecting properties. Then add a new rule to the button.

g. In that new rule, add three actions:

   i. First, add an action to **Set a field's value.** Use the **TaskStatus** field and set its value to **Accepted**.

   ii. Second, on the Rule click **Add Action...** again and select **Submit using a data connection**.

1. Click **Add...** then **Create a new connection to: Submit data** and click **Next>**

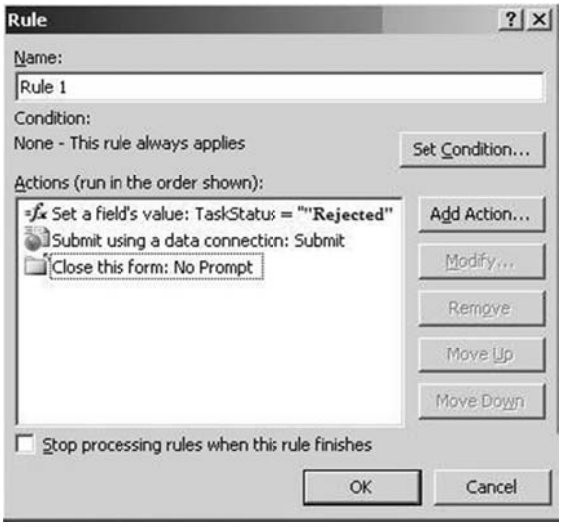


2. On the **Destination for submitting your data** page select **To the hosting environment**... and click **Next>** and **Finish**. Then click **OK**.

iii. Thirdly, Add another **action** that **closes the form** and click **OK**.

iv. Click **OK** (3 more times)

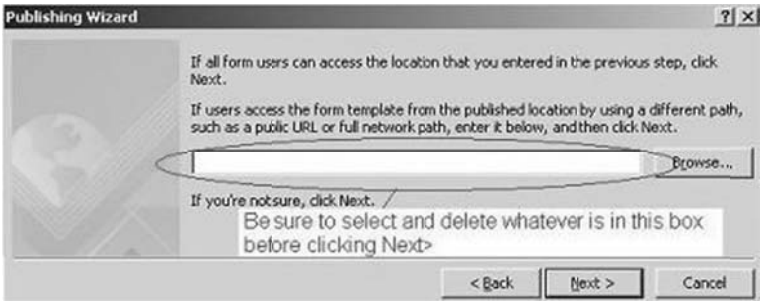
h. Repeat the previous two steps (i.e. **F** and **G**) for the Reject button, but set the **TaskStatus** to **Rejected** instead of **Accepted**. Also, be sure to use the "Submit" connection that you already created for the **Approve** button for this button as well.



i. Set the Security for the Form
    i. In InfoPath on the **Tools** menu - select **Form Options** - select **Security and Trust** - Remove the check from **Automatically determine security level (recommended)** and select **Domain** ...
    ii. Click **OK**

j. Now that the form is done, save it to **C:\Student\Labs\13_Workflow\Lab\CustomApprovalTask.xsn**.

k. Finally **publish** the completed form to the **network location** using the path **C:\Student\Labs\13_Workflow\Lab\CustomApproval\CustomApprovalTask.xsn.** When you publish, make sure that the alternate access path is blank



Task C: Modify the Basic workflow to make use of the InfoPath Forms

The final step in completing the workflow is to read the values entered in the **CustomApprovalAssoc** and **CustomApprovalInit** Forms. To do this you'll need to add a few lines of code to the **onWorkflowActivated1_Invoked** method that will parse the XML in the **workflowProperties.InitiationData** property. This property is set using the XML document returned from the **CustomApprovalInit** form.

1. First you'll need the namespace for the document that was generated by Infopath. To get this namespace open **CustomApprovalInit** in design mode, by right-clicking on the file name and clicking on **Design**, and switch to the **Data Source** task pane.
    a. Next right click the **myFields** node in the **Main** data source and select **Properties**. Switch to the **Details** pane and copy the namespace from the text box.

    b. Next return to your **Visual Studio CustomApproval** Workflow and **open** the **Workflow1.cs** in **Code view**.
    c. Add **using System.Xml;** to the using statements at the top of the code file.

d. Then add the code below to the **onWorkflowActivated1_Invoked** to initialize the internal user and comments fields.

i. ***IMPORTANT****: MAKE SURE YOU* ***UPDATE*** *THE* ***NAMESPACE*** *IN THE CODE BELOW TO MATCH THE NAMESPACE YOU COPIED IN STEP* ***A*** *OR YOUR SOLUTION WILL NOT WORK!*

C#

```
// InitiationData now contains the data that came from the form.
// We can store this data into local variables.
XmlDocument doc = new XmlDocument();
doc.LoadXml(workflowProperties.InitiationData);

//BE SURE TO REPLACE THE FOLLOWING NAMESPACE WITH THE ONE YOU COPIED IN
STEP A OR //THE SOLUTION WILL NOT WORK!!!

XmlNamespaceManager nsmgr = new XmlNamespaceManager(doc.NameTable);
nsmgr.AddNamespace( "my",
 "http://schemas.microsoft.com/office/infopath/2003/myXSD/2007-07-
13T02:58:42");

user = doc.SelectSingleNode("my:myFields/my:User", nsmgr).InnerText;
comments = doc.SelectSingleNode("my:myFields/my:Comments",
nsmgr).InnerText;
```

VB.NET

```
Dim doc As New Xml.XmlDocument()
doc.LoadXml(workflowProperties.InitiationData)

Dim nsmgr As New Xml.XmlNamespaceManager(doc.NameTable)
nsmgr.AddNamespace("my", _
   "http://schemas.microsoft.com/office/infopath/2003/myXSD/2007-07-
13T02:58:42")

user = doc.SelectSingleNode("my:myFields/my:User", nsmgr).InnerText
comments = doc.SelectSingleNode("my:myFields/my:Comments",
nsmgr).InnerText
```

Task D: Configure the workflow Feature for Deployment

Now that the workflow assembly and the InfoPath 2007 forms are completed, you'll need to define the feature that will install the workflow.

1. In **Visual Studio Solution Explorer** open the **feature.xml**.
   a. Change the **Title** into **Litware Timesheet Approval Workflow Feature**.
   b. Inside the **ElementManifests** element, add a 3 **ElementFile** tags:

```
<ElementFile Location="CustomApprovalAssoc.xsn" />
<ElementFile Location="CustomApprovalInit.xsn" />
<ElementFile Location="CustomApprovalTask.xsn" />
```

   c. Notice that the feature references the **Microsoft.Office.Workflow.Feature.dll** as entry point for this workflow.
   d. Notice also the **<Properties>** element that contains the following children:

```
<Properties>
```



```
    <Property Key="GloballyAvailable" Value="true" />
    <Property Key="RegisterForms" Value="*.xsn" />
</Properties>
```

2. Open the **workflow.xml** file and follow these instructions to fill in the template.
    a. Find the **Workflow** element named **Workflow1** and set its attributes to these values:
        i. **Name= " Litware Timesheet Approval Workflow "**
        ii. **Description="This workflow does a whole bunch of great stuff..."**
        iii. **AssociationUrl="_layouts/CstWrkflIP.aspx"**
        iv. **InstantiationUrl="_layouts/IniWrkflIP.aspx"**
        v. **ModificationUrl="_layouts/ModWrkflIP.aspx"**
        vi. **TaskListContentTypeId="0x01080100C9C9515DE4E24001905074F980F93160"**
    b. Inside the **<Workflow>** element, find the **<MetaData>** element with its child elements:

```
<Workflow Name="Litware Custom Timesheet Approval". . .>
   <MetaData>
      <Association_FormURN></Association_FormURN>
      <Instantiation_FormURN></Instantiation_FormURN>
      <Task0_FormURN></Task0_FormURN>
      <AssociateOnActivation>false</AssociateOnActivation>
  </MetaData>
</Workflow>
```

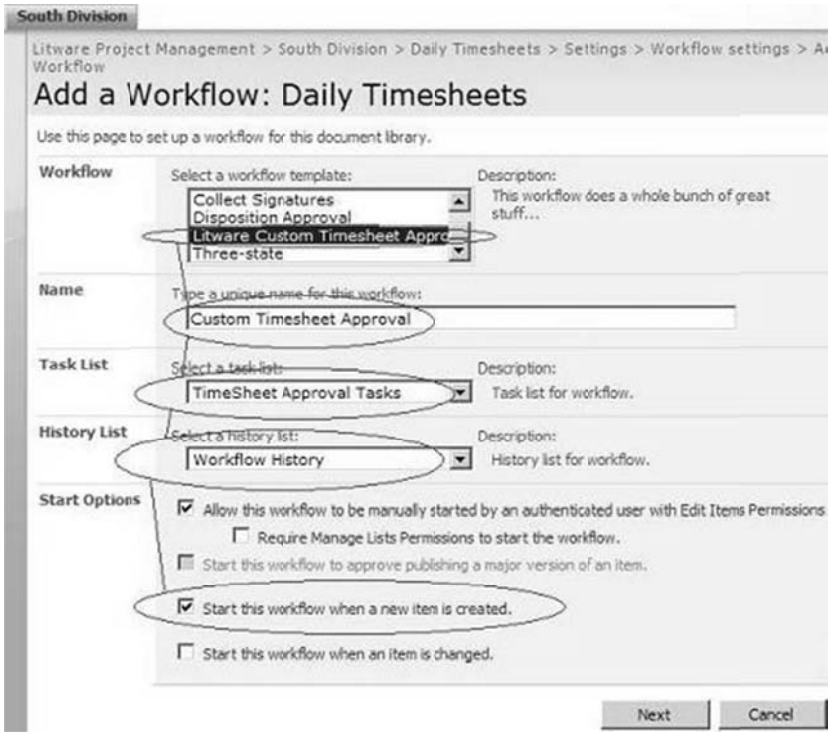
    c. Remove the **<StatusPageUrl>** element.
    d. Populate the **MetaData** section with InfoPath 2007 form URNs for the **Assoc**, **Init**, and **Task** forms we created earlier.
        i. To find the URN, open each InfoPath 2007 form in **Design** mode by doing **Right Click > Design** on the file name, and select **File -> Properties** in the menu. **Copy** the **ID** field.
        ii. When finished your MetaData Element should look similar to the code below, except with different URNs and with the Workflow attributes included. After examining your modifications **Save** and **Close** the **WorkflowTemplates.xml** file. **NOTE:** Ensure that there is **NO** leading or trailing whitespace inside the **Association_FormURN**, **Instantiation_FormURN**, or **Task0_FormURN** tasks or you will get an error when the form is viewed in SharePoint.

```
<Workflow Name="Litware Custom Timesheet Approval". . .>
    <MetaData>
       <Association_FormURN>
urn:schemas-microsoft-com:office:infopath:CustomApprovalAssoc:-myXSD-
2007-07-13T02-58-42
      </Association_FormURN>
      <Instantiation_FormURN>
urn:schemas-microsoft-com:office:infopath:CustomApprovalInit:-myXSD-2007-
07-13T02-58-42
      </Instantiation_FormURN>
      <Task0_FormURN>
urn:schemas-microsoft-com:office:infopath:CustomApprovalTask:-myXSD-2007-
07-13T06-48-40
      </Task0_FormURN>
      <AssociateOnActivation>false</AssociateOnActivation>
   </MetaData>
```



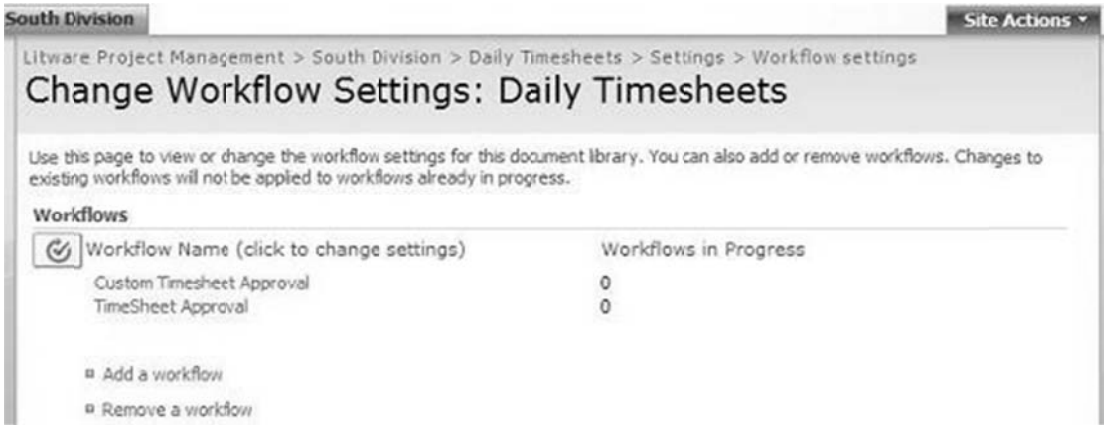
```
</Workflow>
```

3. **Build** your project and correct any errors.
4. There are still 3 files to copy from the **C:\Student\Labs\13_Workflow\Starter Files\Lab** directory to the project directory. Take your time to inspect them:
    a. Manifest.xml
    b. Wsp_structure.ddf
    c. install.bat
5. **Build** your **CustomApproval** project.
6. Open **Windows Explorer** and double-click the **intall.bat** file to deploy your custom approval workflow solution.
7. Now that all the preliminary work is done, it's time to activate the feature and associate the **Custom Timesheet Approval** workflow with the **Daily Timesheet** document library.
    a. Open the site at **http://Litwareinc.com/sites/ProjectManagementLab** and choose **Site Settings** from the **Site Actions** menu.
    b. Click the link for **Site Collection Features** under the **Site Collection Administration** section.
    c. Find the feature named **Litware Timesheet Approval Workflow** and click its **Activate** button.
    d. Open the site at **http://Litwareinc.com/sites/ProjectManagementLab/SouthDivision** and navigate to the **Daily Timesheets** document library.
    e. Open the Form library settings by using the **Settings -> Form Library Settings** menu item.
        i. On the Form library settings page, navigate to **Workflow Settings** in the **Permissions and Management** section.
        ii. If you are routed to the **Change Workflow Settings** screen, click the **Add a Workflow** link to associate the workflow with the document library. If you are not routed to this page, then move on to the next step. [Note: You will only be routed to the **Change Workflow Settings** screen if there is already another workflow associated with this document library.]
        iii. Select the **Litware Custom Timesheet Approval** workflow template and name the workflow **Custom Timesheet Approval**.
        iv. Select the **Timesheet Approval Tasks** list from the **Tasks List** dropdown. This list was automatically created as part of the original Timesheet Approval workflow in the **Select a task list:** drop down (or select to just create a new one), and select the **Workflow History** choice from the **Select a history List:** dropdown.
        v. Place a check in the **Start this workflow when a new item is created** box



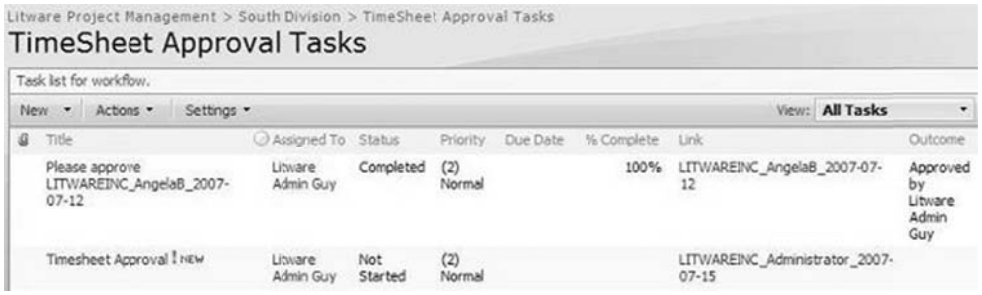
vi. Finally click the **Next** button to move to the next step.

f. This next page utilizes your **CustomApprovalAssoc** InfoPath Form to set the Managers user name. Enter **LITWAREINC\Administrator** so the administrator must approve the document then click **Create** to complete the workflow creation.

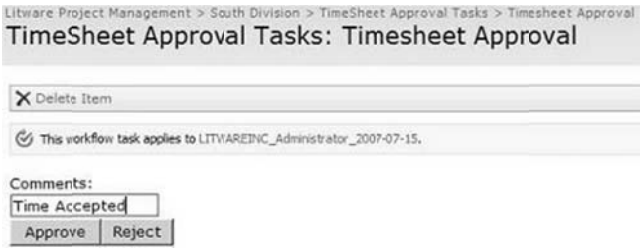


8. You've successfully created the **Custom Timesheet Approval** workflow and associated it with the document library. Now we need to remove the original workflow.
    a. From your **Change Workflow Settings** page select **Remove a workflow**.
    b. On the **Remove Workflows** page specify **No New Instances** for the **TimeSheet Approval** workflow and click **OK**.
9. Now all that's left to do is test the approval process.
    a. In the **Remote Desktop** console right-click **Angela Barbariol** to open a Remote Desktop connection.

b. On Angela's desktop, open the site at **http://Litwareinc.com/sites/ProjectManagementLab/SouthDivision** and navigate to the **Daily Timesheets** document library.
c. Create a **New Daily Timesheet**, and fill it in with sample values and click **Submit**.
d. The workflow automatically starts. You should see **In Progress** underneath the **Custom Timesheet Approval** column on the **Daily Timesheets** form library.
e. This workflow created a task for the user **LITWAREINC\Administrator** to resolve. In order to complete the workflow we must shut down Angela's RDP session and once again become the Administrator.
    i. Go to Angela's **Start** menu select **Log off** and click **OK**
f. As the Administrator navigate to the **Tasks** list to see the task that has been created by the workflow.
    i. Navigate to **http://Litwareinc.com/sites/ProjectManagementLab/SouthDivision**
    ii. Select **View All Site Content**
    iii. Select the **Tasks** List that you selected for this workflow.



10. The next step in the workflow is to complete the task so the workflow can continue.
    a. Click on the **Timesheet Approval** task in the **l**ist.
    b. On the approval form, you can either **Approve** or **Reject** using the appropriate buttons.



11. The last step is to verify that the workflow has completed successfully and that the timesheet is approved. Navigate back to the **Daily Timesheets** form library. You'll notice that the **Custom Timesheet Approval** column for the document you approved is set to **Completed**.

Student Challenge: Conditional branching in a workflow

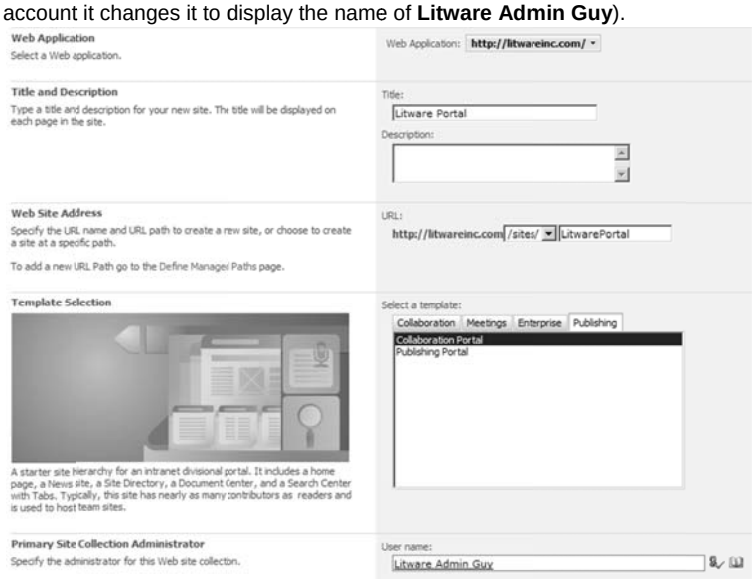
If a timesheet is approved, the Regional manager needs to make sure the consultant is paid. Extend the workflow using the **Windows Workflow If-Else** activity to create a new task for the Regional manager only if the **TaskStatus** is set to **Accepted**. Update the workflow and redeploy the feature then test your changes. You will know you've implemented it correctly, if the manager approves the timesheet and a new task is created for the Regional manager. Use **LITWAREINC\JayH** as the Regional manager.

Lab 14: Creating and Customizing a Corporate Portal Site

Lab Overview: In this lab you will create and grow out a new site collection created from the **Collaboration Portal** site template. This will give you an opportunity to see how working with a MOSS site template adds quite a bit of functionality on top of the site templates included with WSS such as Blank Site and Team site.

Exercise 1: Creating a new Corporate Portal site

1. You will begin by creating a new site collection with a top-level site based on the **Corporate Portal** site template. This **Corporate Portal** site will be used as an intranet site to provide information to Litware employees.
    a. Make sure you are logged onto the VPC as **LITWAREINC\Administrator**.
    b. Open **SharePoint 3.0 Central Administration** and navigate to the **Application Management** page.
    c. On the **Application Management** page, click the **Create Site Collection** link in order to navigate to the **Create Site Collection** page.
    d. Create a new site collection using the following values.
        i. The target Web Application is **http://litwareinc.com**
        ii. Title is **Litware Portal**
        iii. Web Site Address is **http://litwareinc.com/sites/LitwarePortal**
        iv. The Template Selection is **Collaboration Portal** which can be found under the **Publishing** tab
        v. The Primary Site Collection Administrator is **LITWAREINC\Administrator**. (Note that when SharePoint resolves this account it changes it to display the name of **Litware Admin Guy**).



    e. Click **OK** to create the new Portal site.
    f. Once the site has been provisioned, click the link in the **Central Administration** site to navigate to the site at http://litwareinc.com/sites/LitwarePortal.

2. Once you have opened the site, take a moment to explore what is inside.
    a. Click on the **View All Site Content** link on the top of the **Quick Launch** bar to navigate to the standard WSS application page which displays what lists, document libraries and child sites exist. Answer the following questions.
        i. What lists have already been created in this top level site?
        ii. What document libraries have already been created in this top level site?
        iii. What child sites have already been created?

b. Drop down the **Site Actions** menu and inspect the way the menu commands are structured. You should notice that the Corporate Portal site has a **Site Actions** menu structure that is very different from a standard WSS site. This is due to the fact that the Corporate Portal site template automatically activates a feature named **Office SharePoint Server Publishing**. This feature adds quite a bit of additional functionality on top of what is contained in a standard WSS site.

c. Within the **Site Actions** menu, choose the click on the command **Site Actions >> Modify All Site Settings**. This will allow you to navigate to the standard WSS site settings page (_layouts/settings.aspx). However, you should notice that the links are arranged differently than in a standard WSS site.

i. Some standard WSS links have disappeared from the **Site Settings** page. For example, you should notice there is no longer a standard link to modify the Top link bar or the **Quick Launch** menu. That's because SharePoint Server provides a different way to modify navigation links with this site.

ii. Many new links have been added to the **Look and Feel** sections as well as the **Site Administration** section and the **Site Collection Administration** section.

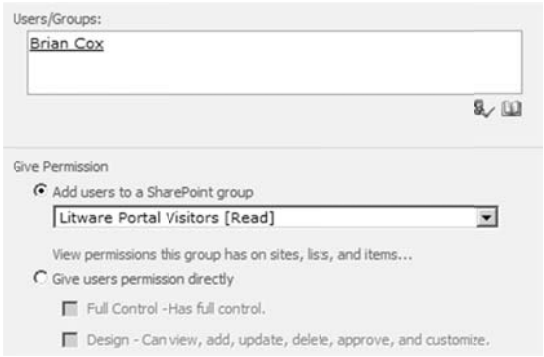
3. Add the user **LITWAREINC\BrianC** as a visitor of the site.

a. From the Site Settings page, click the **People and groups** link under the **Users and Permissions**.

b. Drop down the **New** menu and click **New User**.

c. Add the user account **LITWAREINC\BrianC** as a new member of the **Litware Portal Visitors** group. Note that SharePoint will change the account name to its display name of Brian Cox when it resolved the account against Active Directory).

d. Choose **Litware Portal Visitors [Read]** from the **Give Permission** section

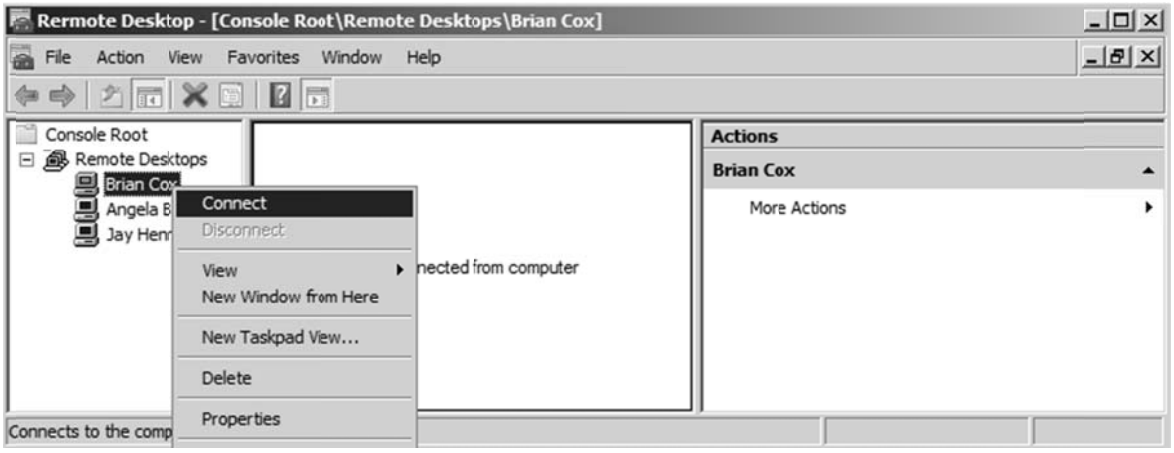


e. Click **OK**

4. Log in as Brian Cox and access the new Corporate Portal site. This will allow you to see what the site looks like to a user that has no permissions to add and modify content. You will use a Remote Desktop connection to log on as Brian Cox.

a. Minimize all Windows so you can see the desktop. You should be able to see four shortcuts that allow you to start Remote Desktop sessions under the identities of different users. Click the shortcut with the caption of **RemoteDesktop.msc** to launch the Remote Desktop management console. Make sure the management console is maximized.

b. Right-click the node **Brian Cox** and choose **Connect**.



c. Enter the password **pass@word1**. This launches a remote desktop session for the user Brian Cox.

d. While running in the remote desktop session for Brian Cox, access the new portal site at **http:///litwareinc.com/sites/LitwarePortal**. You should be able to access the home page of the Litware Portal site and navigate to the various lists, document libraries and child sites below.

e. While running as Brian Cox, you should note that you cannot see the **Site Actions** menu. However, you do have navigation menus that allow you to move around the site and examine its content.

f. Do not close out Brian's session. Leave it open so you can return to it later. Minimize the remote desktop Windows so you can resume running as the **Litware Administrator**.

5. While running as **LITWAREINC\Administrator**, return to the Litware Portal site. Now it is time to create a new child site for Litware Human Resources department.

a. Make sure you are at the top-level site at http://litwareinc.com/sites/LitwarePortal.

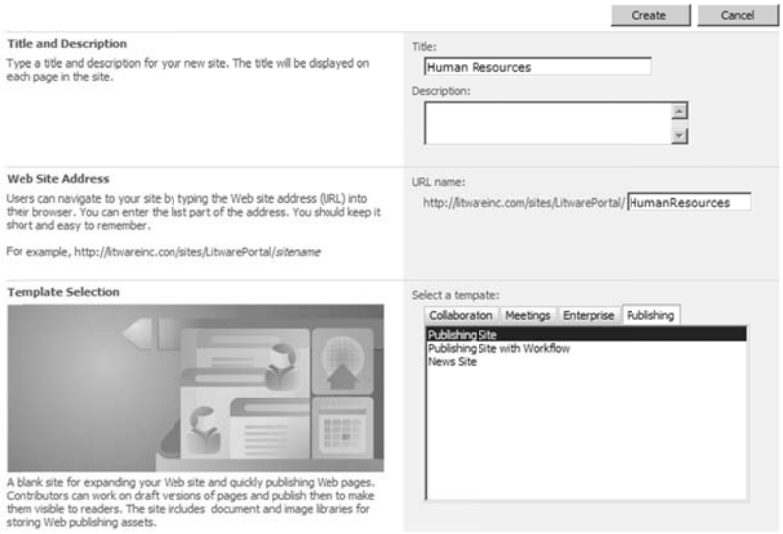
b. Drop down the **Site Actions** menu and click the **Create Site** command.

c. Create the new child site using the following settings.

i. Site Title is **Human Resources**.

ii. URL name is **http://litwareinc.com/sites/LitwarePortal/HumanResources**.

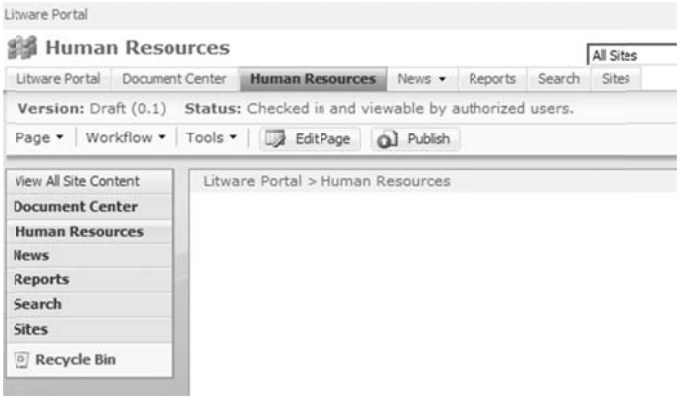
iii. Template Selection is **Publishing Site** available under the **Publishing** tab.



d. Click the **Create** button to create the new child site.

e. Once SharePoint has created the new site, you will see the landing page for the site as shown below. Notice there is a new toolbar know as the **Page Editing Toolbar** that allows an authorized user to approve the site so it can be seen

by site members. Also note that a new link has been added to the top-link bar so users can navigate to the new **Human Resources** site. Do not click on the **Publish** link until you are instructed to do so.



f. Click the **Edit Page** button to take the current page into editing view. Find the Field Control with the caption of **Page Content** and click the **Edit Content** link to go into editing mode. Add some text that would be a typical announcement on a page within a Human Resources site. When you are done editing the text, find and click the button on the **Page Editing Toolbar** with the caption **Check In to Share Draft**.

g. Return to the **Remote Desktop Session** for Brian Cox and refresh the browser window showing the portal site. Note that you should not yet see the new child site titled **Human Resources** because it has not yet been published. An authorized user must publish this site before it can be seen by visitors such as Brian Cox.

h. Leave the **Remote Desktop Session** for Brian Cox and start running as the **Litware Administrator** again.

i. Now, it is time to make the new page and the entire **Human Resource** site visible to all site users. Click the **Publish** button on the **Page Editing Toolbar** to make the **Human Resource** site visible to everyone.

j. Return to the **Remote Desktop Session** for Brian Cox and refresh the browser. Now this user should be able to see and navigate the **Human Resources** site and see the content you added to the home page.

k. Leave the **Remote Desktop Session** for Brian Cox and start running as the **Litware Administrator** again.

6. Now it's time to create a new page within the **Human Resources** site.

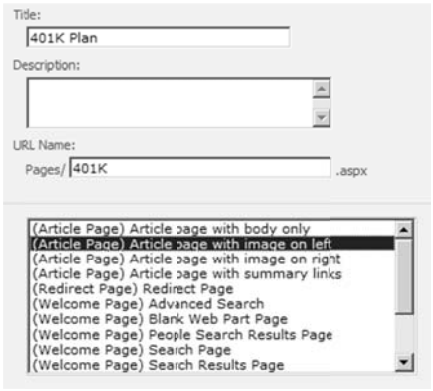
a. From the **Site Actions**, click on the **Create Page** command.

b. On the **Create Page** page, fill in the following information.

i. Title is **401K Plan**.

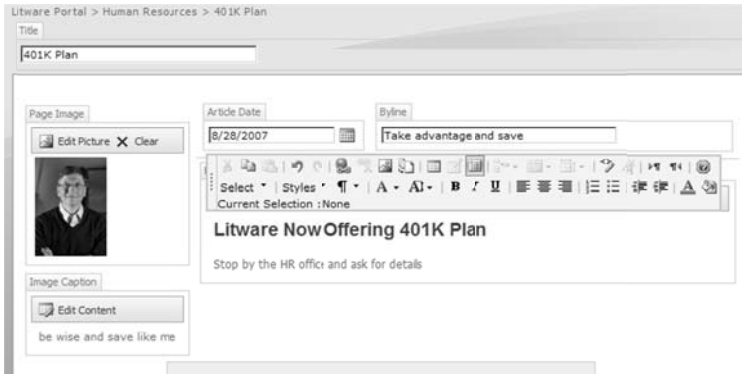
ii. URL Name is **401K**.

iii. Page Layout is **Article page with image on left**.



c. Click the **Create** button to create the page.

d. Once the page is created, you will find yourself in edit mode. Fill in content within the field controls with something similar to the sample content shown below. Note that you can upload pictures into the portal site from **C:\Student\Resources\Pictures** or **C:\Student\Resources\LitwareGraphics**.

e. 

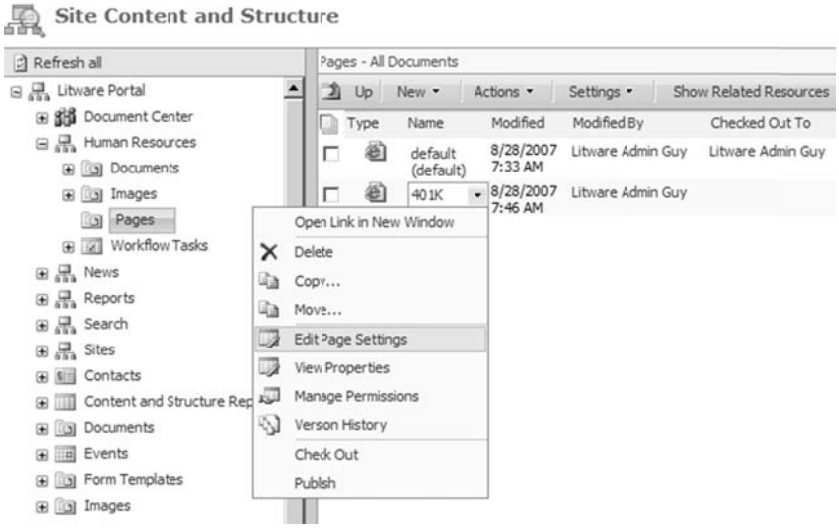
f. Click the **Publish** button on the **Page Editing Toolbar** so that the page can be seen by all users.

7. On the top-left portion of the browser window, click on the **Litware Portal** link to return to the portal's top-level site. From the **Site Actions** menu, click the **Manage Content and Structure** command.

   a. On the **Site Content and Structure** page, Use the Tree View control on the left-hand side to drill down into the **Human Resources** site.

   b. Look inside the **Pages** document library and locate the page you just created named **401k.aspx**.

   c. Drop down the **ECB** menu for the page **401K.aspx** and see what commands are available.



8. Congratulations! You have finished this Lab.

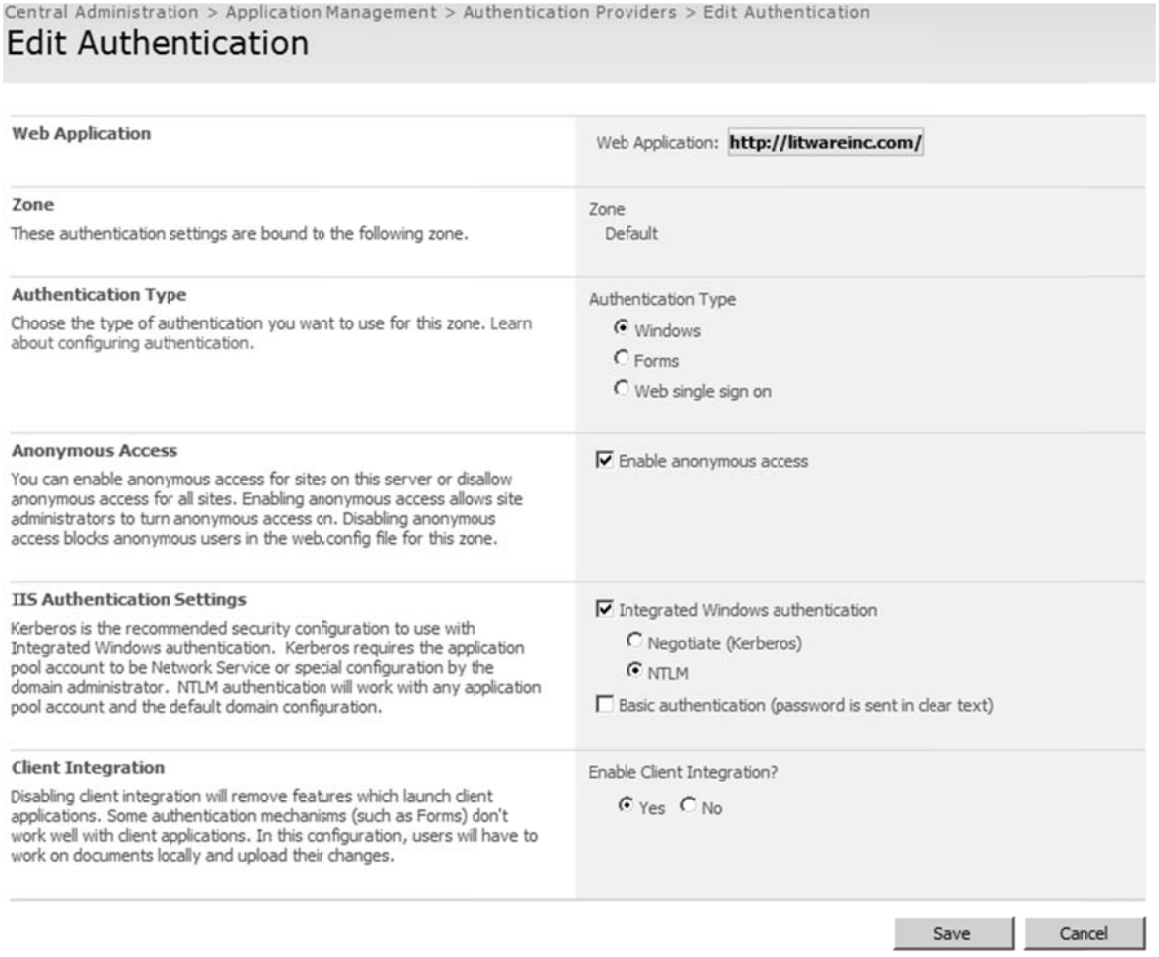
Lab 15: Web Content Management with MOSS 2007

Lab Overview: Litware management has decided to use **Microsoft Office SharePoint Server 2007** to create an Internet site that will be used to deliver information about the company and its services. The goal is to enable a number of persons in the company who are not IT/Dev-skilled to create and manage the content. Your job in this lab is to prepare the infrastructure to support this scenario.

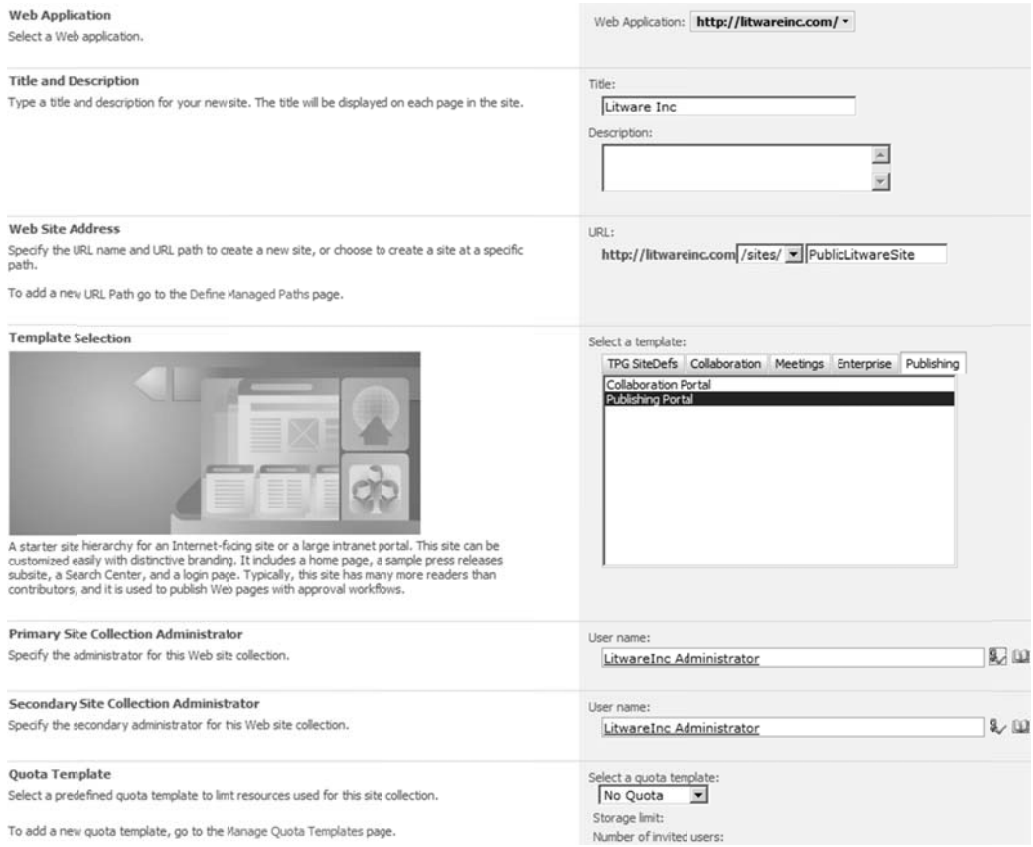
Exercise 1: Creating a new Publishing Portal site

To start, you will be asked to create the site based on the site definition that is made available with **Microsoft Office SharePoint Server 2007**.

1. Open **SharePoint 3.0 Central Administration** and navigate to the **Application Management** page.
2. Locate the **Application Security** group and click on the **Authentication Providers** link. Once you are at the Authentication Providers page, click on the link to configure the **Default Zone** for the Web Application at the URL of **http://litwareinc.com**. Enable **Anonymous Access** for the Web Application at the URL **http://litwareinc.com**.

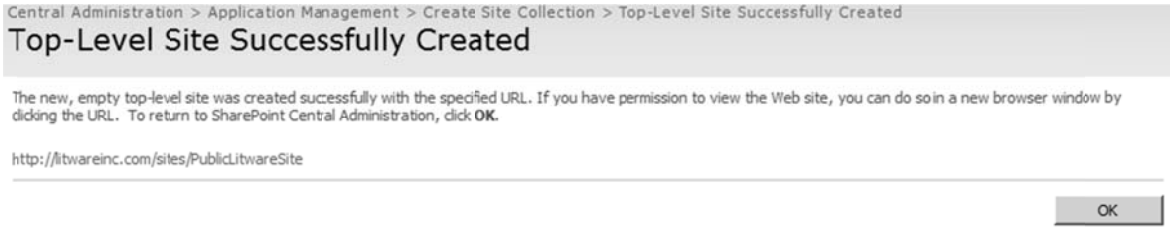


3. Click the **Save** button.
4. Locate the **SharePoint Site Management** group on the **Application Management** page and click the **Create Site Collection** link in order to navigate to the **Create Site Collection** page. Create a new site collection using the following values:
   a. Title: **Litware Inc**
   b. URL: **http://litwareinc.com/sites/PublicLitwareSite**
   c. Site template: **Publishing Portal**



   d. Primary site collection administrator: **LITWAREINC\Administrator**
   e. Secondary site collection administrator: **LITWAREINC\Administrator**

5. Press the **OK** button to start the process of creating the new site collection. When this process has completed successfully, you will see this administration page that links to the new site.



6. Click on the link **http://litwareinc.com/sites/PublicLitwareSite** to open a new browser session with the newly created WCM Publishing site.

Exercise 2: Configuring the site for anonymous access

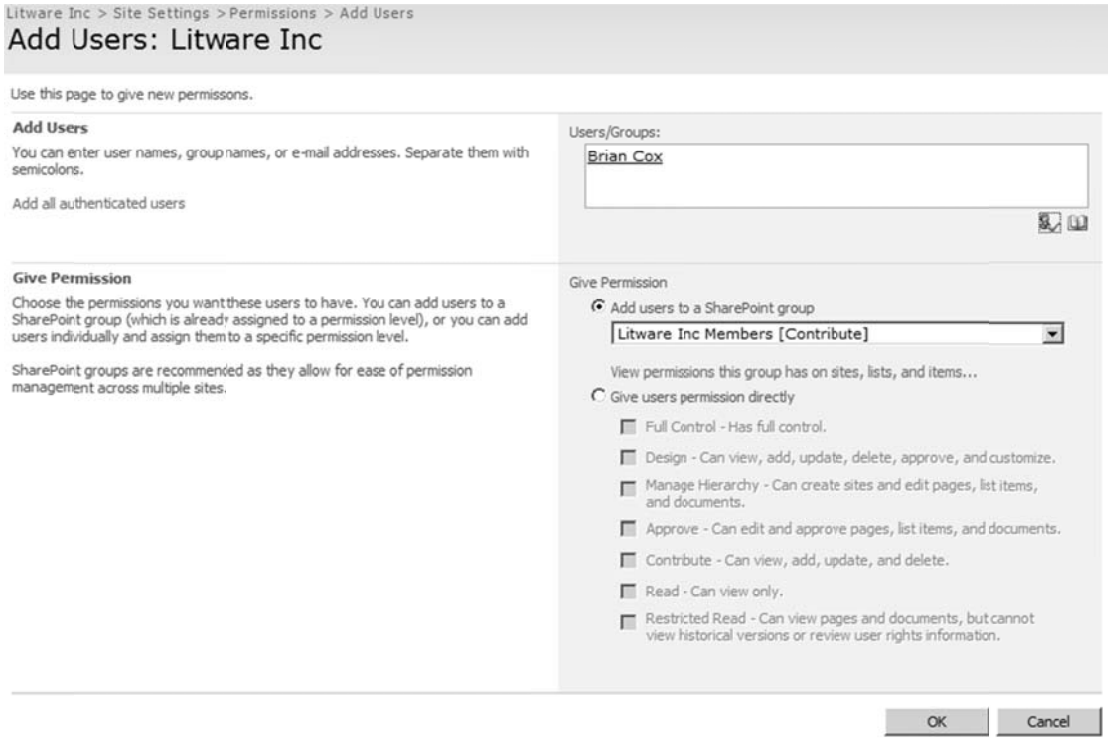
1. Since this will be an Internet-facing Web site, you should enable anonymous access for this site. You have already done it at the level of IIS but now you need to also enable it for the site itself.
2. On the home page of the new site, click on **Enable anonymous access**.
3. Select **Entire Web Site** as the part accessible by anonymous users.

4. Click **OK** to navigate to the **Permissions** page. Here you have to add the anonymous user to the list

    a. Click **New** and the **Add Users.**

    b. Add **LITWAREINC\BrianC** in the users box and verify the account.

    c. Set the group to **Litware Inc. Members [Contribute]**.



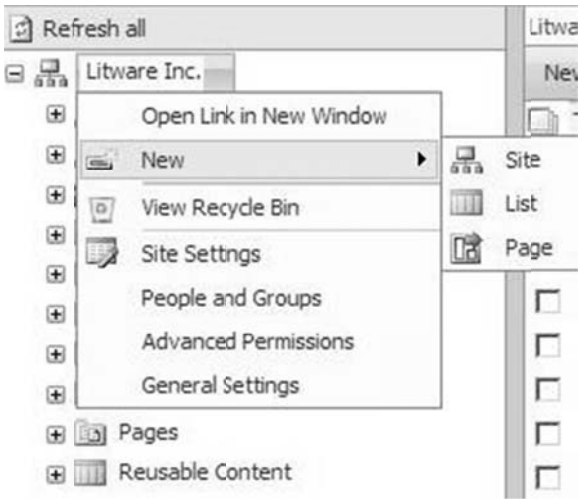
    d. Press **OK**.

5. To test out your work, in the right hand corner click on **Welcome Litware Admin Guy**, and select sign out. At the pop up, select **No** and click the link to **Go back to site**.

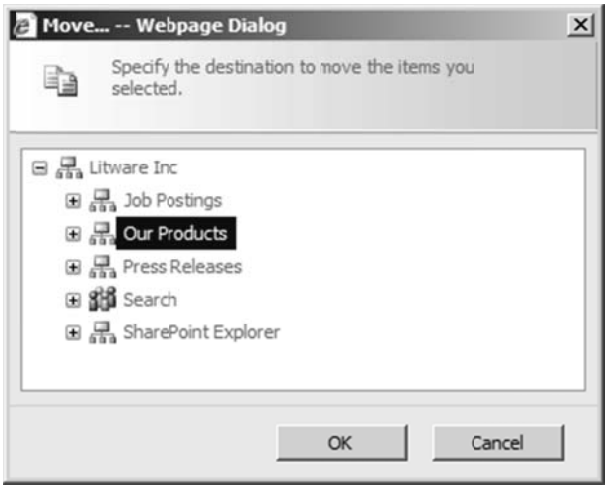
Exercise 3: Creating Sub Sites

At this moment, you have one top-level site and one sub site called **Press Releases**. In the exercise, you will create a couple more.

1. Sign back in, by clicking the **Sign In** hyperlink in the right hand corner.
2. Use the **Site Actions** to click on the **Create Site**.
    a. Title of the site is **Job Postings**.
    b. Last part of the URL is **JobPostings**.
    c. Click **Create** to start the process.
3. Navigate again to the home page of the top-level site.
4. Now use **Site Actions** to click on **Manage Content and Structure**
    a. Use the ECB on the root node and select **New | Site**.



    b. Title of the site is **Our Products.**
    c. Last part of the URL is **Products.**
    d. Click **Create** to start the process.
5. Create one more sub site under the top-level site with the title **SharePoint Explorer** and a URL **SharePointExplorer**.
6. The **SharePoint Explorer** sub site actually should have created under the **Products** site. Use the ECB for **the SharePoint Explorer Site** and select **Move**. On the **Move…**Web Page Dialog, select **Our Products** and Click **OK**.

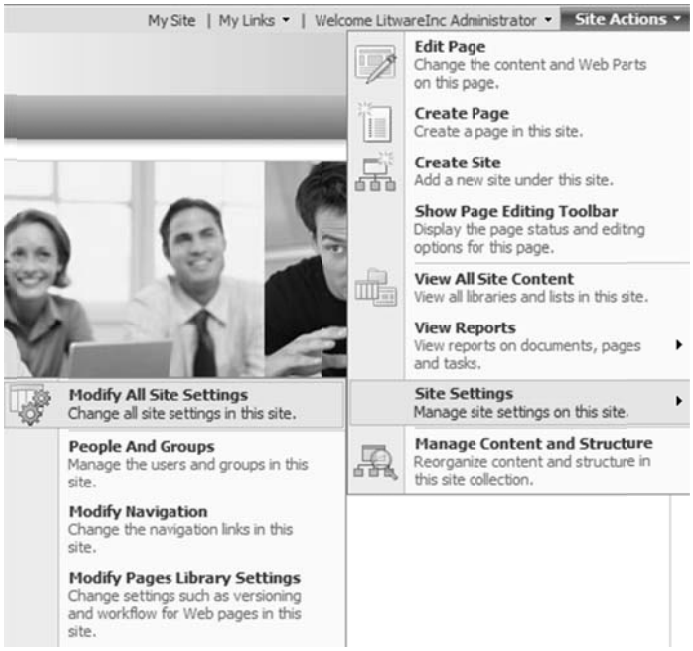


7. Navigate back to **http://litwareinc.com/sites/publiclitwaresite** and take a look at the navigation bar.

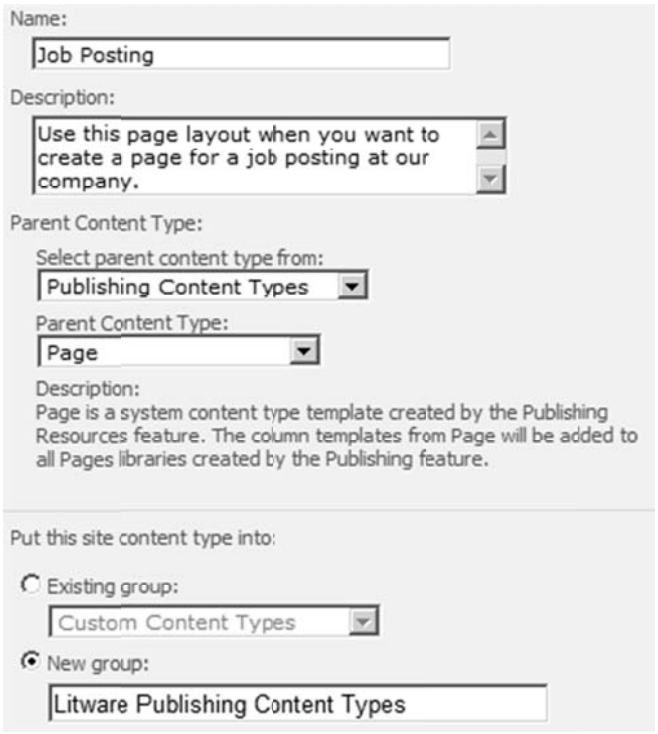
Exercise 4: Creating a Page Layout

In this exercise you are asked to create templates or page layouts for a job posting and for a product description page.

1. Navigate to the home page of the top-level site.
2. Open the **Site Settings** page.
3. Locate the **Galleries** group and click on **Site Columns** link to open the **Site Column Gallery** page.
4. Create now a new **Site Column** with the following values:
    a. Set the name of the column to **Job Description** of type **Full HTML.**
    b. Create a new group called **Litware Columns.**
    c. Enter a small description like **'Type here the description of the job we are looking for'.**
    d. Press **OK.**
5. Create a second **Site Column** with the following values:
    a. Set the name of the column to **Required Skills** of type **Choice.**
    b. Use the **Litware Columns** group.
    c. Enter a small description like **'Select one or more skills that are required for the job.**'
    d. Enter a couple of choices, like **'SharePoint Administration'**, '.**NET Development'**, ...
    e. Display choices using: **Checkboxes.**
    f. Set the **Allow Fill-In Choices** to **Yes.**
    g. Press **OK.**
6. Create a third **Site Column** with the following values:
    a. Name: **JobTitle**
    b. Type: **Single Line of Text**
    c. Use the **Litware Columns** group.
    d. Press **OK.**
7. Open the **Site Settings** page again by clicking the **Site Actions** button choosing **Site Settings > Modify All Site Settings**.



8. Locate the **Galleries** group and click on **Site Content Types** to open the **Content Types Gallery** of the site collection.
9. Create a new **Site Content** with the following values:
    a. Name of the content type is **Job Posting.**
    b. Enter a small description like **'Use this page layout when you want to create a page for a job posting at our company**.'.
    c. Select **Page** from the **Publishing Content Types** group as the parent.
    d. Create a new group called **Litware Publishing Content Types.**

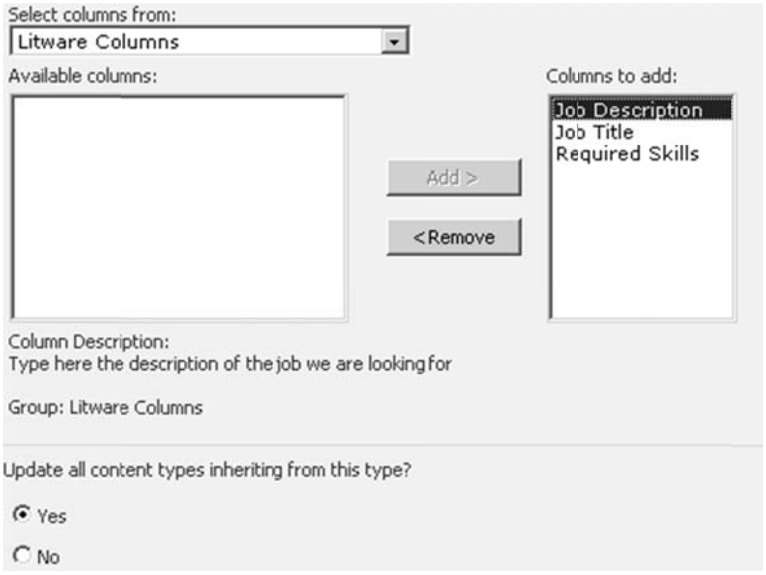


e. Press **OK**.

10. Now add 3 columns to the **Job Posting** content type. Click **Add from Existing Columns**.

    a. Add **Job Title** from the **Core Contact** and **Calendar Columns.**

    b. Add **Job Description** and **Required Skills** from the **Litware Columns.**

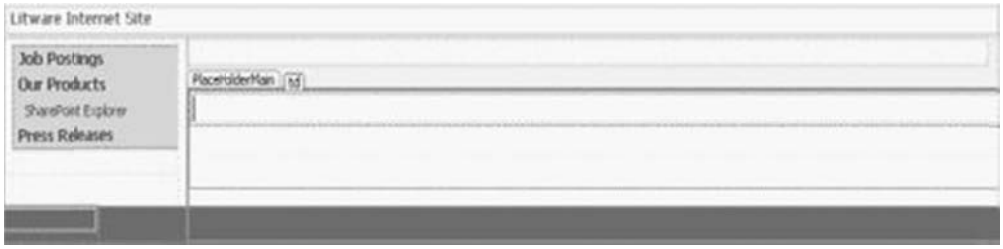


c. Press **OK**.

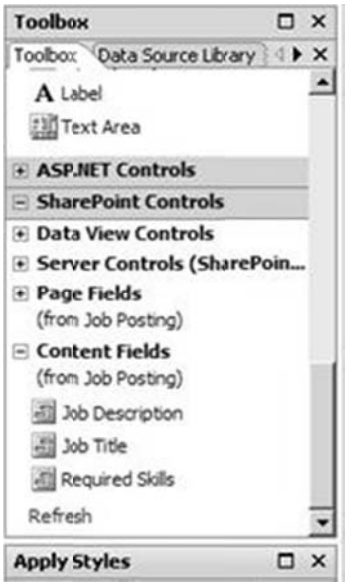
11. Click on **Site Settings** in the breadcrumb to return to the **Site Settings** page.

12. Under the **Galleries** group select **Master Pages and page layouts**. Click **New** and select **Page Layout**.

13. In the **New Page Layout** page fill out the following values:
    a. Content Type Group: **Litware Publishing Content Types**
    b. Content Type Name: **Job Posting**
    c. URL name: **JobPosting**
    d. Title: **Job Posting**
    e. Description: **Job Posting Template**
14. Click **OK** to create the page layout.
15. Your next task is to use the **SharePoint Designer** and populate the page layout file with the field controls. Open **SharePoint Designer** and open the **http://litwareinc.com/sites/PublicLitwareSite** site.
    a. In the **Folder List** under **_catalogs** click on **masterpage.**
    b. Double-click the **JobPosting.aspx** item.
    c. Position yourself in the **PlaceHolderMain** and insert a table of 6 rows and one column.



    d. In the first row add **Are your looking for the following job?**. Apply the **.HeaderTitle_Large** as style.
    e. Drag and drop **Job Title** from the toolbox into the second row.



f. Set the **CssClass** property also to **.HeaderTitle_Large**.

g. Drag and drop **Job Description** into the 3th row.

h. In the **Properties** pane apply the following values:

   i. **AllowExternalUrls**: **false**

   ii. **AllowTextMarkup**: **false**

i. Type '**You must have the following skills:**' in the 5th row and apply the **.HeaderTitle_Large** as style.

j. Drag and drop **Required Skills** in the 6th row.

k. Save the page.

16. Go back to the **Master Page and Layout Gallery** in the **Site Settings** page.

17. As you notice in the **Master Page Gallery** in your browser the **JobPosting.aspx** is in draft mode. Check in the page as a major version and next approve the page.

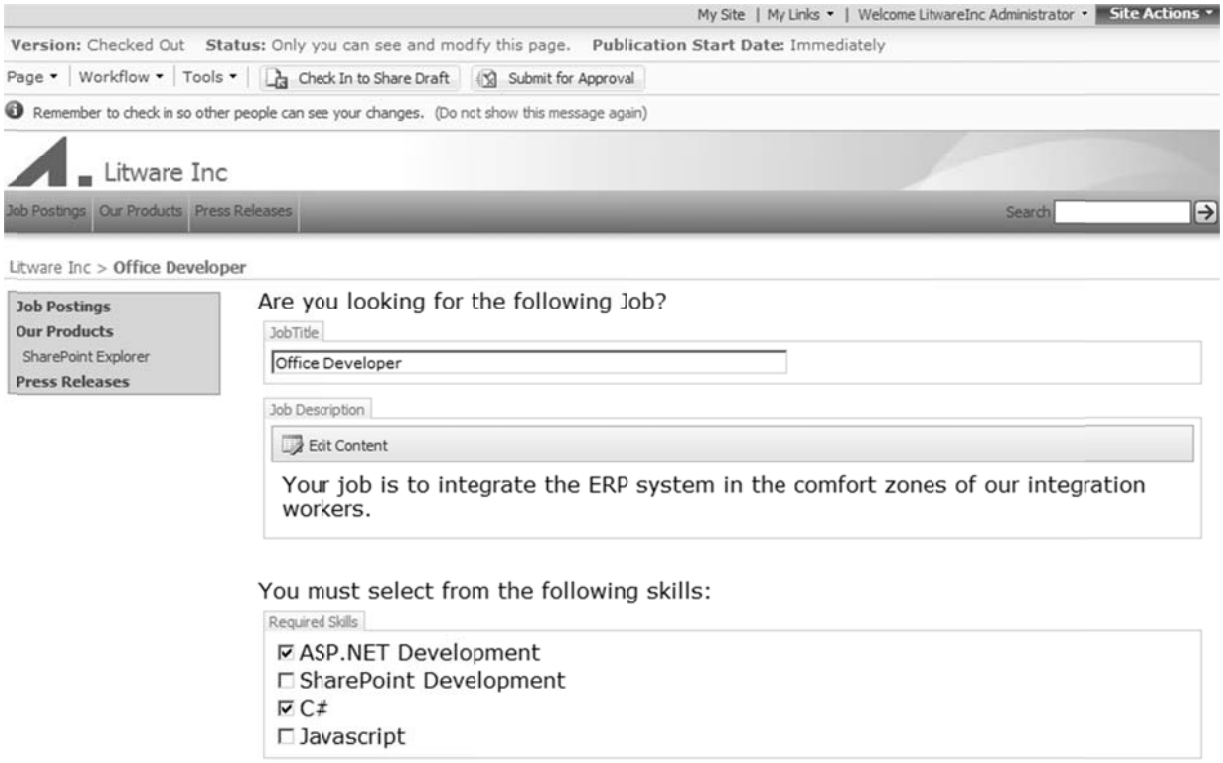
Exercise 5: The Publishing Cycle

Now change your role and play the content author who is going to use the newly created job posting layout page.

1. Go to the **Job Postings** site and use the **Site Actions** to **Create a Page**.

2. Provide **Office Developer** as the name of the page and select the new **Job Postings** layout page. Press the **Create** button.

3. Type in some data in the field controls.



4. When finished, save your page in the database. Click the **Check in to Shared Draft** button in the **Page Editing** toolbar.
5. Now hit the **Submit for Approval** button.

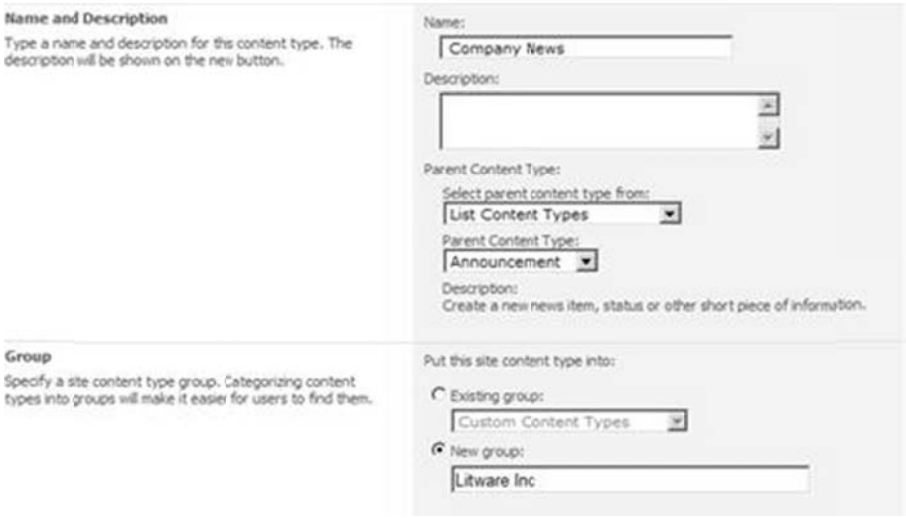
6. Start the approval workflow.

7. Navigate to the **Pages** library of the **Job Postings** site. You should see the page in pending mode.
8. Use the ECB to approve the page.
9. When approved, check out the page by going to the **Job Postings** site again.

10. This finishes the exercise.

Exercise 6: Utilizing and Customizing the Content Query Web Part

1. The first step in this exercise is to create the content type to roll up. The goal of this lab is to create a company news content type. This content type will be added to the announcements list in the different departments and then displayed for the public to see on the homepage. Out of the box, the content query web part only displays the title of the item; we will be customizing it to show both the title and the body using **SharePoint Designer**.
    a. Navigate back to the top level site: **http://litwareinc.com/sites/PublicLitwareSite**
    b. On the **Site Actions** menu, select Site **Settings > Modify** all site settings
    c. Under the **Galleries** section, click **Site content types**
    d. Click create and fill in the **New Site Content Type** page as seen below.



    e. Click on **LitwarePortal** in the breadcrumb to return to the top level site.

2. In this step we will add an custom list to the **Job Postings** site and configure to use both the item and company news content types. This allows members of this site to go to one spot to create both news for just this division and news for the entire company.
    a. Navigate to the **Job Postings** site.
    b. On the **Site Actions** menu, select **View all Site Content**.
    c. Click **Create > Custom List.**
        i. Name: Announcements
        ii. Click **Create**.
    d. On the **Announcements** page, click **Settings > List Settings.**
    e. On the **Customize Announcements** page, click **Advanced Settings.**
        i. The first option is **Content Types**, select **Yes > OK**
    f. Now you need to add the **Company News** content type, in the **Content Types** section, click **Add from existing site content types.**
    g. Select **Company News**, and click **Add**, then **OK.**

h. Navigate back to the **Announcements** list and add one **Company News** list item and one **Announcement** list item.

i. *Hint: If you select the drop down arrow next to **New**, you will see both **Company News** and **Announcement**, select **Announcement** the first time to create and item and company news the second time*.

3. On the **Litware Inc Portal** home page, you will be adding the content query web part and then customizing it to show the **Company News Content Type.**

a. Make sure you are the **Litware Inc Portal** home page, **http://litwarinc.com/sites/publiclitwaresite.**

b. On the **Site Actions** menu, select **Edit Page.**

c. Delete the content out of the **Summary Links** control on the left, by clicking the icon to the left of each link, and selecting **Delete**. Delete the **Press Releases** web part in the top zone, by clicking **Edit** and **Delete**. At the warning, select **OK**.

d. Click **Add a Web Part** in the **Top Zone**.

e. Select **Content Query Web Part** and click **Add.**

f. On the content query web part you have just added to the page, click **edit** > **modify** shared web part.

g. In the task pane on the right hand side, expand **query.**

h. Under **List Type**, select **Custom List.**

i. Under **Content Type** select the following:

i. Show items of the this content type group: **Litware Inc.**

ii. Show items of this content type: **Company News.**

j. Expand the **Appearance** section and rename the web part, **"Important News".**

k. Click **OK**.

4. The content query web part should now be displaying on your home page only the announcement you created using the **Company News Content Type**. Currently it only shows the title, it would be helpful if it gave a bit more information for your users.

a. On the content query web part select **Edit > Export.**

b. Select **Save > Save**. This should save the file to your **Desktop** folder.

c. Open the file with **SharePoint Designer**, **File > Open > Browse to the Desktop > Select Important_News.webpart > Open.**

d. Locate the following line of code approximately line 68:

```
<Property name="CommonViewFields" type="String"/>
```


e. Change this line of code to: (Note: you are adding the field using the internal name of field and the type of field, separated by a comma. The internal name of the field can be found by locating the column, right clicking and selecting properties.

```
<property name=”CommonViewFields” type=”String”>Body, Rich
HTML</property>
```

f. Save the file.

g. Navigate back to the **Litware Portal** home page.

h. Choose **Site Actions > Edit Page.**

i. Select **Add a Web Part in the Top Zone**, in the bottom right corner select **Advanced Web Part Gallery** and options, this will allow us to import our newly configured web part.

j. On the **Browse** bar in the right hand corner, click the drop down arrow, select **Import**.

k. Click **Browse**, and navigate to the **Important_News.webpart** file you just edited (it should be saved on the desktop). Click **Upload > Import**.

l. Navigate back to **SharePoint Designer** and open the site **http://litwareinc.com/sites/publiclitwaresite/**.

m. In the **Folder List** on the left side, expand **Style Library** and then **XSL Style Sheets**

n. Double click on **ItemStyle.xsl** to open it.

o. Replace the following line of code, approximately line 40:

```
<div class="description">
  <xsl:value-of select="@Description" />
</div>
```

With:

```
<xsl:variable name="body">
    <xsl:call-template name="removeMarkup">
      <xsl:with-param name="string" select="@Body"/>
  </xsl:call-template>
</xsl:variable>

<div class="description">
    <xsl:value-of select="$body"/>
</div>
```

p. Place your cursor at the beginning of the last line of code:

```
</Xsl:stylesheet>
```

And hit return.

q. Place your cursor in the line you just added and add the following piece of code:

```
<xsl:template name="removeMarkup">
    <xsl:param name="string" />
  <xsl:choose>
```



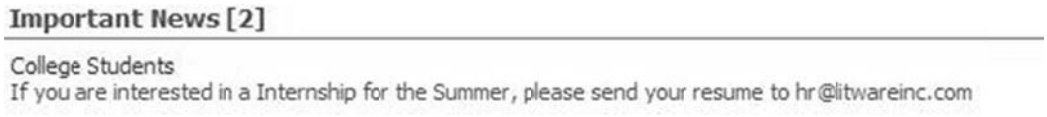
```
      <xsl:when test="contains($string, '&lt;')">
          <xsl:variable name="nextString">
        <xsl:call-template name="removeMarkup">
                  <xsl:with-param name="string" select="substring-
after($string, '&gt;')" />
              </xsl:call-template>
          </xsl:variable>
          <xsl:value-of select="concat(substring-before($string, '&lt;'),
$nextString)" />
        </xsl:when>
    <xsl:otherwise>
        <xsl:value-of select="$string" />
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

r. Save the file.

s. Right-click the file and select **Check in**. On the Check in menu, select **Publish a Major Version**.

t. Go back to the homepage, select **Submit for Approval > Start**.

u. On the page editing toolbar, select **Approve**. On the **Workflow Tasks** page, select **Approve**.



v. If you would like, delete the first content query web part you added to the page. Remember that since we exported the web part and customized it, they are not the same.

5. You finished this lab successfully.

Lab 16: Business Data Catalog

Lab Time: 60 Minutes

Lab Directory: C:/Student/Labs/16_BDC

Lab Overview: The Business Data Catalog, or BDC in short, is a new feature introduced with Microsoft Office SharePoint Server 2007, which allows an easy integration between SharePoint and business data present in back-end systems such as SAP, Siebel and Microsoft SQL Server. The BDC is activated as a service within the Shared Services model of SharePoint 2007. The shared service can be utilized from various places inside a site such as Web Parts, lists, the search environment and from user profiles. One of the major design goals of the BDC is to enable minimal-code data retrieval from a disparate set of back-end systems. To achieve this goal the BDC provides a metadata model that provides a consistent environment for creating BDC enabled applications. In this lab exercise you will work with a predefined BDC application and find out where this application definition can be used to provide a value-add solution on top of the out of the box functionality provided by SharePoint 2007. You will begin with importing the BDC application into SharePoint. Next you will use various BDC enabled technologies such as the Web Part framework and SharePoint lists. You will finish the lab exercise by creating a custom Web Part that accesses the BDC through the BDC object model.

Exercise 1: Import an existing BDC application definition

In this exercise you will examine a pre-built BDC metadata definition which retrieves data from the **AdventureWorks** data-warehouse. Next you will import this application definition into SharePoint and examine the elements it contains using the browser. You will also modify the BDC security settings to allow non-administrative level users to view the data contained in the BDC application. The exercise will finish with the creation of a simple site and Web Part page to allow the BDC data to be displayed.

Examine the BDC metadata definition

The BDC metadata file is an elaborate XML definition containing the data types and operations available on the external data source. In this step you will examine a metadata file which contains information about products, product categories and product resellers from the **AdventureWorks** sample database.

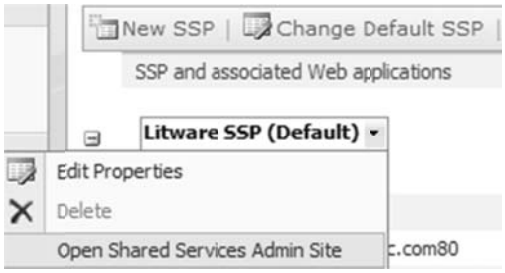
1. Open **Windows Explorer** and navigate to the following location: **C:\Student\Labs\16_BDC\Starter Files.**
2. Open the **AdventureWorksDWH.xml** BDC metadata definition in **Internet Explorer**. The metadata definition contains information about four entities, their relationships and their corresponding operations.

3. Begin by examining the root element called **LobSystem**. It contains three important fields. The **name** for the BDC application, **versioning information** and the **type** of application, database or web service.
4. Next examine the child elements of **LobSystem**.
    a. Look at the **Properties** node. It defines properties specific for this BDC application and can be thought of as an extra set of properties for the **LobSystem** element.
    b. Examine the **LobSystemInstances** node. This element defines the **connection** to the Adventure Works data-warehouse database.
    c. Take a look at the **Entities** element. An entity can be thought of as the **definition of a record** in the database (a table definition).
    d. Finish with the **Associations** element. This part of the definition is used to define the **relationships** between entities. This allows you to perform drill-down operations on business data.
5. Expand the **Entities** node and next expand the **Entity** node for the **Product** entity. Examine the child elements.
    a. Look at the **Properties** element, now used within an Entity element. It defines properties specific for the Product entity.
    b. The **Identifier** element is used to define the name of an identifying property. This name will later be attached to the fields of the entity to identify those fields as being part of the primary key for the entity.
    c. The **Methods** element is used to define the available operations on this entity and the data that the operation returns.
6. Expand the **Methods** element for the **Product** entity. Examine the structure that makes up an operation.
    a. The **Properties** element is again used as a container element for its parent, which in this case is the **Product** entity.
    b. The **FilterDescriptors** element defines the values to be used for filtering the retrieved items with the operation.
    c. The **Parameters** element defines the input and output parameters for the operation.
    d. The **MethodInstances** define the actual callable operations. The same Method definition can define multiple method instances. The **GetProducts** method can for instance return a single product or a set of products.

Import the BDC metadata definition

Now that a basic understanding has been reached of the contents of the BDC metadata definition, this definition can be loaded into the BDC environment. The shared service provider has a configuration page where you can administer the BDC.

1. Open **SharePoint 3.0 Central Administration** and navigate to the **Application Management** page.
2. On the **Application Management** page locate the **Office SharePoint Server Shared Services** group.
3. Click **Create or configure this farm's shared services** in order to navigate to the **Manage this Farm's Shared Services** page.
4. On the **Manage this Farm's Shared Services** page, open the ECB for **Litware SSP** and click **Open Shared Services Admin Site** in order to navigate to the shared services administration page.



*Note: you could also have selected the **Litware SSP** from the **Quick Launch**.*

5. On the **Shared Services Administration** page, click **Import application definition** in order to import a new BDC application definition into the BDC environment. All the BDC related options are combined within the Business Data Catalog group.

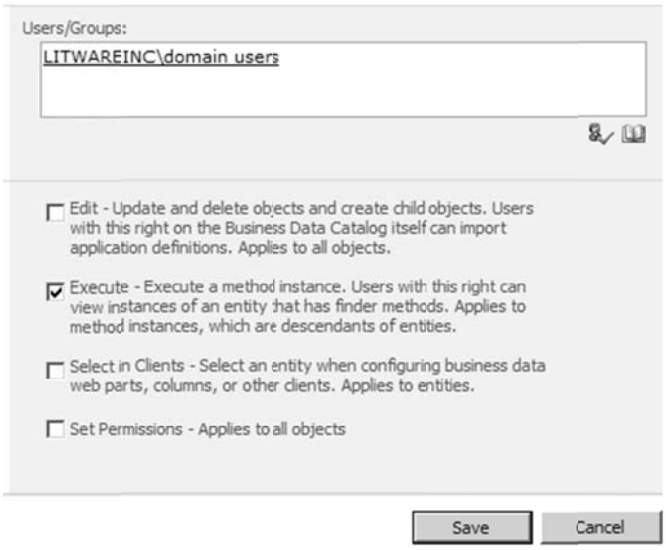
6. Import the BDC application using the following settings
    a. Application definition file is **C:\Student\Labs\16_BDC\Starter Files\AdventureWorksDWH.xml**
    b. File type is **Model**
7. Click **Import** to begin the process of importing a BDC application.
8. After the process is finished, click **OK** to be redirected to the **View Application: AdventureWorksDW** page. Review the information on this screen and leave it open for the next task.

Allowing user access to the BDC application

The imported BDC application is initially only available for administrative users. The BDC specific application management page allows the modification of these permissions to allow fine-grained control over who can access the BDC data and in what manner. First you will provide user access to

the BDC application imported in the previous step. Next you will provide user access to the BDC subsystem as a whole.

1. On the **View Application: AdventureWorksDW** page, click **Manage permissions** in order to change the default permissions of the BDC application.
2. On the **Manage Permissions: AdventureWorksDW** page, click the **Add Users / Groups** button in order to add permissions.
3. Add the following permissions:
    a. Users / Groups is **LitwareInc\Domain Users**
    b. Choose permissions is **Execute**



4. Click **Save** in order to attach the new permissions to the BDC application.
5. Go back to the **Shared Service Administration: Litware SSP** page. If your browser window is closed, you can take the following steps to go back to this page, or use the breadcrumb control available on most SharePoint pages.
    a. Open **SharePoint 3.0 Central Administration** and navigate to the **Application Management** page.
    b. On the **Quick Launch** select Litware SSP.
6. On the **Shared Services Administration: Litware SSP** page, click **Business Data Catalog Permissions** in order to control the user access permission to the BDC subsystem.
7. On the **Manage Permissions: Business Data Catalog** page, click the **Add Users / Groups** button in order to add permissions.
8. Add the following permissions:
    a. Users / Groups is **LitwareInc\Domain Users**
    b. Choose permissions is **Execute**
9. Click **Save** in order to attach the new permissions to the BDC subsystem.

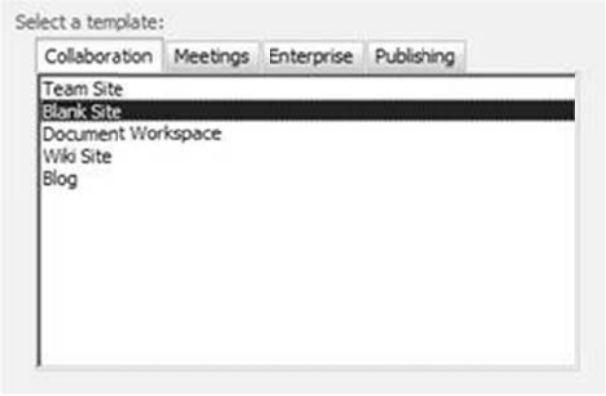
Exercise 2: Using BDC Web Parts

Now that the BDC application has been defined and security has been set, it is time to start making use of the built-in Web Parts for accessing BDC data. In this exercise you will use various Web Parts deployed with SharePoint 2007 to access and display hierarchical data. Before working with the Web Parts, a separate top-level site will be created for this exercise.

Create the BDC Web Part top-level site

While BDC Web Parts can be used on all sites that have the correct features activated, in this exercise you will create a separate top-level site to work with. This allows you to focus on the Web Parts and not worry about other details.

1. Open **SharePoint 3.0 Central Administration** and navigate to the **Application Management** page.
2. On the **Application Management** page, click the **Create Site Collection** link in order to navigate to the **Create Site Collection** page.
3. Create a new site collection using the following values.
    a. Be sure that the **Web Application** is set to **http://litwareinc.com** (not http://ssp.litwareinc.com).
    b. Title is **BDC Lab**
    c. Web Site Address is **http://litwareinc.com/sites/BDCLab**
    d. Template Selection is **Blank Site**, which you find under the **Collaboration** group.



    e. Primary Site Collection Administrator is **LitwareInc\Administrator**.
4. Click **OK** to start the process of creating the new site collection. When this process has completed successfully you will be presented with an administration page that links to the new top-level site. Navigate to the new top-level site.

Creating the main site layout

The new top-level site will contain a list of product categories and product subcategories. The final layout will allow a user to select a category and be presented with the list of subcategories for the selected category. The next task will expand on this model by altering the profile page for a

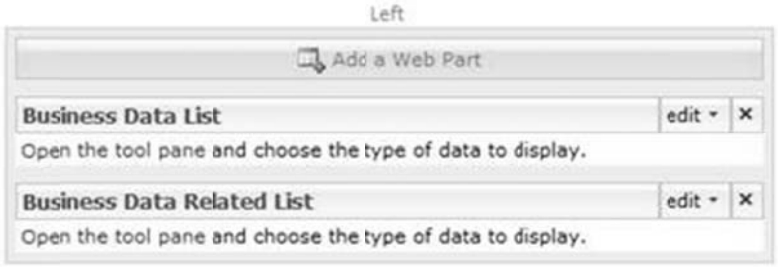
subcategory to also display all the products within that subcategory. Before using the BDC features on the BDC Lab site, first the BDC needs to be activated through a feature.

1. On the main page of the **BDC Lab** site, click the **Site Actions** button and choose **Site Settings** from the menu that appears. This will allow you to edit the Web Parts on the main page.
    a. On the **Site Settings** page, choose **Site collection features** under the **Site Collection Administration** group.
    b. On the **Site Collection Features** page, click **Activate** next to the feature named **Office SharePoint Server Enterprise Site Collection features** in order to provision the **BDC Web Parts** to the site collection.
2. Navigate back to the **Site Settings** page of the BDC Lab site, and choose **Site features** under the **Site Administration** group.
    a. Click **Activate** next to the feature named **Office SharePoint Server Enterprise Site Collection features** in order to provision the BDC Web Parts to the site.
3. Use the breadcrumb or URL to navigate back to the site. The URL to navigate to is http://litwareinc.com/sites/BDCLab
4. On the main page of the **BDC Lab** site, click the **Site Actions** button and choose **Edit Page** from the menu that appears. This will allow you to edit the Web Parts on the main page.

5. Click the **Add a Web Part** button available inside the **Left** Web Part Zone in order to add new Web Parts to the page.

6. In the **Add Web Parts to Left** dialog, choose the **Business Data List** and **Business Data Related List** Web Parts from the **All Web Parts** group. Click **Add** to close the dialog and add the Web Parts to the main page.

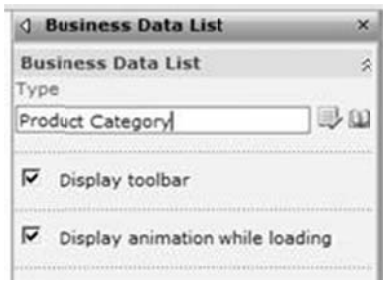
7. The **Left Web Part Zone** should resemble the following image. In the next task you will configure each of the two Web Parts.



Configuring the Business Data List Web Part

The first Web Part that will be using the BDC is the **Business Data List** Web Part. This Web Part can be used to display a list of records (BDC entities).

1. In the **Business Data List** Web Part, click **Open the tool pane** to configure the Web Part.
2. In the **Business Data List** tool pane, configure the Web Part with the following settings.
    a. Type is **Product Category**. You can find the **Type** property in the **Business Data List** group.



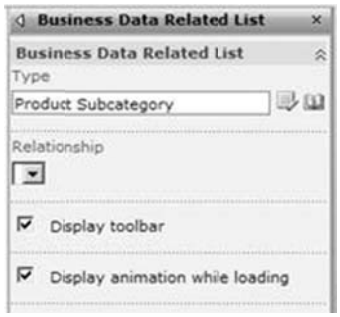
b. Click **OK** at the bottom of the tool pane in order to show the list of **Product Category** entities in the **Business Data List** Web Part.

3. In the **Business Data List** Web Part, click **Retrieve Data** to verify that the Web Part successfully retrieves data from the LOB system through the BDC. Note that the title of the Web Part has changed to reflect the type of data it is retrieving.

Configuring the Business Data Related List Web Part

The second Web Part on the page is a Business Data Related List Web Part. This Web Part will display related records based on the selection of the Business Data List Web Part, forming a master-detail hierarchy. In this sample the Business Data Related List Web Part will display a list of Product Subcategory entities based on the selected Product Category.

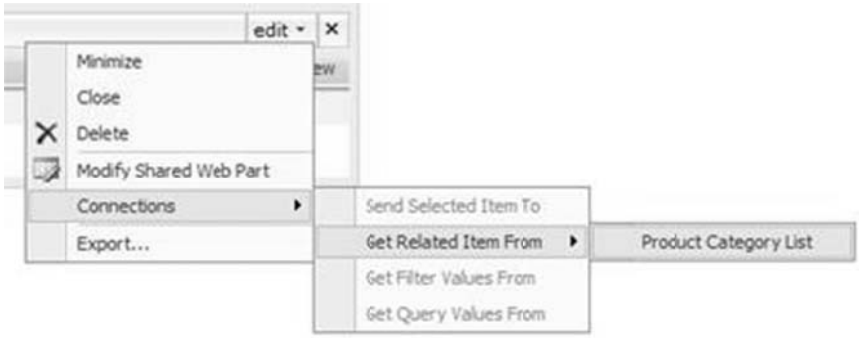
1. In the **Business Data Related List** Web Part, click **Open the tool pane** to configure the Web Part.
2. In the **Business Data Related List** tool pane, configure the Web Part with the following settings.
    a. Type is **Product Subcategory**. You can find the **Type** property in the **Business Data Related List** group.



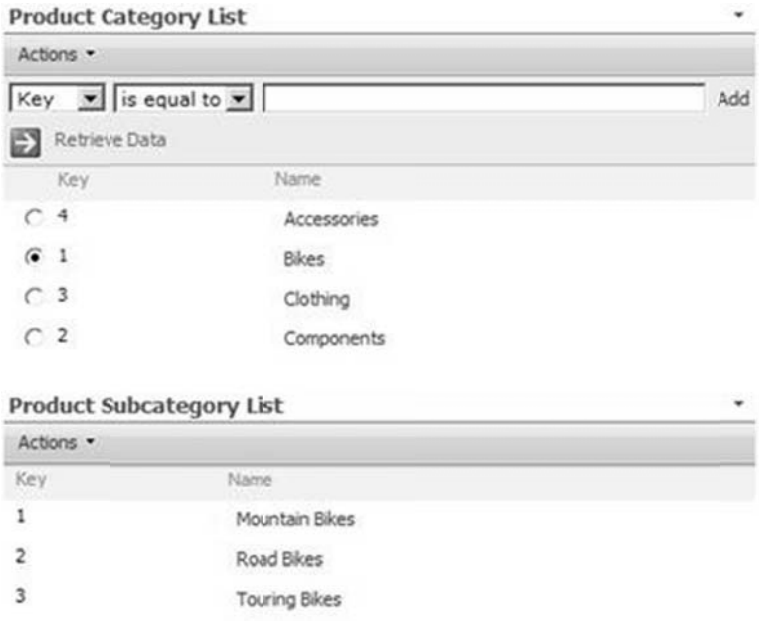
b. Press the **Check Types** button directly beside the **Type** field to verify whether you have entered the Product Subcategory correctly. This will also select the relationship to the Product Category.

c. Click **OK** at the bottom of the tool pane in order to show the list of **Product Subcategory** entities in the **Business Data Related List** Web Part. Note that the title of the Web Part has changed to **Product Subcategory List**.

3. On the **Edit** menu of the **Product Subcategory List** Web Part, choose **Connections**, next choose **Get Related Item From** and choose **Product Category List** in order to have the sub category list populated from the selected item in the Product Category Web Part. Note that this modifies the Product Category List Web Part which now shows selection buttons for each item displayed.



4. Verify that the final page looks like the following image. You can select an item in the **Product Category List** Web Part and display detail records in the Web Part listing sub-categories.



Creating the profile page

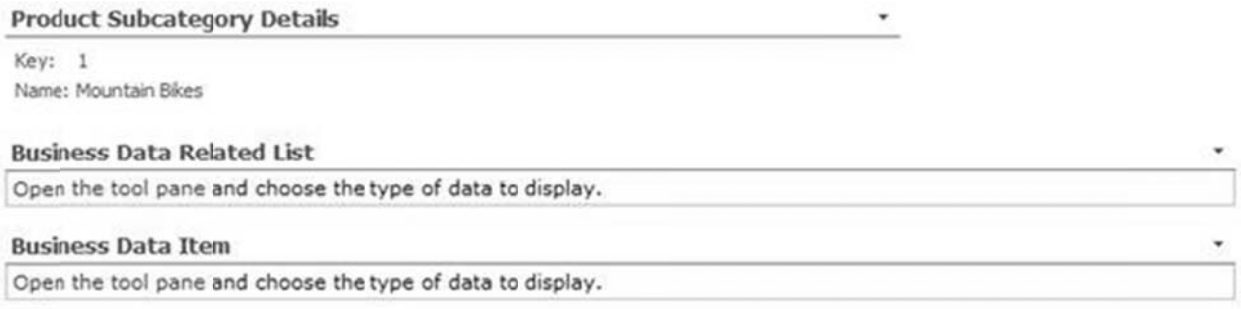
Each BDC entity has a special page where the details of the entity can be viewed. This page is called the profile page. You can modify the contents of these pages to display unique content along with the entity details. In this task you will modify the profile page for the **Product Subcategory** entity in order to display a list of products contained in the category, creating an extra level in the master-detail hierarchy.

1. On the item menu for an item displayed by the **Product Subcategory List** Web Part, choose **View Profile** in order to navigate to the profile page of the sub category. Note that this page is part of the shared service provider architecture. You can notice this from the URL you are being redirected to.

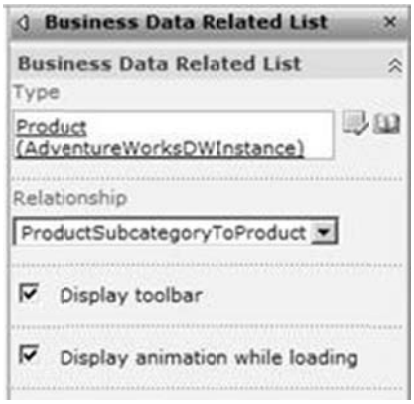
2. On the layout page of the Product Subcategory, click the **Site Actions** button and choose **Edit Page** from the menu that appears. This will allow you to edit the Web Parts on the profile page. When going to the edit mode, note the presence of the normally invisible Business Data Item Builder Web Part on the layout page. This Web Part takes the context on the query-string to retrieve the selected BDC entity.

3. Click the **Add a Web Part** button available from the **Bottom Zone** Web Part Zone in order to add new Web Parts to the page.
4. In the **Add Web Parts to Bottom Zone** dialog, choose the **Business Data Item** and **Business Data Related List** Web Parts from the **All Web Parts** group. Click **Add** to close the dialog and add the Web Parts to the main page.

5. The profile page should resemble the following image. In the next task you will configure each of the two Web Parts added to the profile page. The first Web Part will display a list of products for the selected sub category. The second Web Part will display the product details for a selected product. You can also view these details on the product profile page.

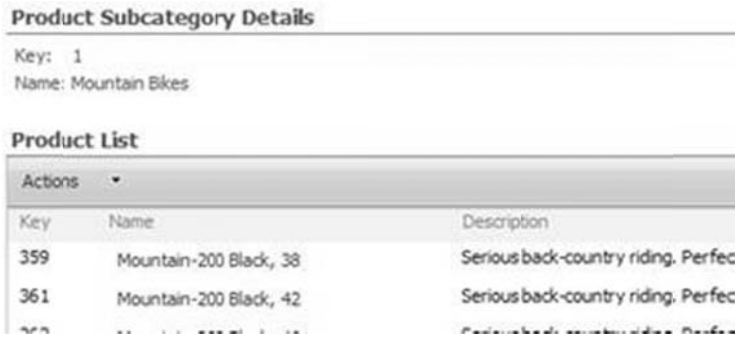


6. In the **Business Data Related List** Web Part, click **Open the tool pane** to configure the Web Part.
7. In the **Business Data Related List** tool pane, configure the Web Part with the following settings.
    a. Type is **Product**. You can find the **Type** property in the **Business Data List** group. Use the **Browse** button to choose the **Product** entity.



b. Click **OK** at the bottom of the tool pane in order to show the list of **Product** entities in the **Business Data Related List** Web Part. Note that the title of the Web Part has changed to **Product List.**

8. On the **Edit** menu of the **Product List** Web Part, choose **Connections**, next choose **Get Related Item From** and choose **Business Data Item Builder** in order to have the product list populated from sub category in context on the profile page. The Web Part should start retrieving data immediately.



9. Finish the profile page for the **Product Subcategory** entity by configuring the **Business Data Item** Web Part using the following details.

   a. Configured display entity is **Product**

   b. Add a connection from **Product List**

10. Verify that you can now select a product in the **Product List Web Part** and see the details in the **Product Web Part**.

Exercise 3: BDC List Column

The Business Data Catalog is available from many places within the SharePoint environment. In this exercise you will create a custom list which contains a BDC lookup column. You can use this column type to look up records in the BDC application and display the record inline in the list definition.

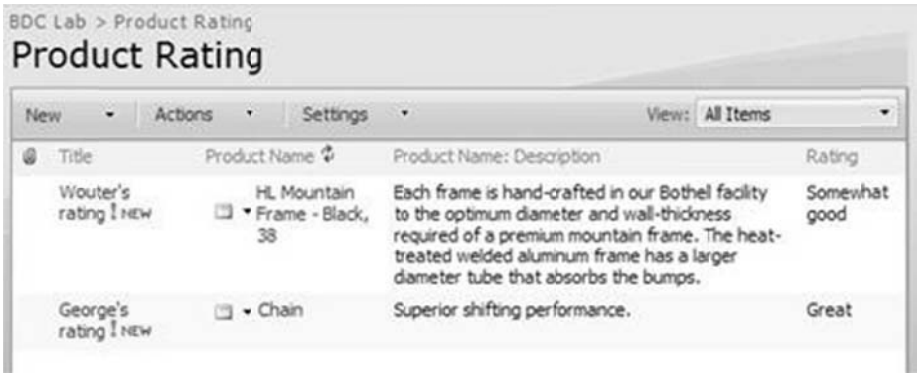
To create a BDC enabled list

The first step consists of creating a new custom list. You will add and configure the list columns to include a BDC column.

1. Open a browser window and navigate to the root site for this lab exercise located at **http://litwareinc.com/sites/BDCLab.**
2. On the **Site Actions** menu, choose **Create** in order to go to the content creation page.
3. On the **Create** page, in the **Custom Lists** group, choose **Custom list** in order to add a new list to the site.
4. On the **New list** page, use the following information to create the new list.
    a. Name is "**Product Ratings**"
    b. Description is "**Rates the AdventureWorks product line.**"
5. Click **Create** in order to create the new list and open the default list view.
6. On the **Product Ratings** page, open the list settings by choosing **Settings** and next **List Settings**.
7. On the **Customize Product Ratings** page, choose **Create Column** in order to add a new BDC column to the list definition.
8. In the **Create Column: Product Ratings** page, use the following information for the column definition.
    a. Column name is **Product**
    b. Column type is **Business Data**
    c. Require information is **Yes**
    d. Type is **Product**
9. Click **OK** to close the column creation page and add the new column definition to the list.
10. Repeat steps 7 through 9 using the following column definition.
    a. Column name is **Rating**
    b. Column type is **Choice**
    c. Require information is **Yes**
11. Choices are:
    a. **Very good**
    b. **Somewhat good**
    c. **Not so good**
12. Display choice is **Radio Buttons**

To view the BDC enabled list

1. Navigate to the list using the following location
   http://litwareinc.com/sites/BDCLab/Lists/Product%20Ratings
2. Add new items to the list and view the results. The following image displays how this can look on your SharePoint environment.



Lab 17: Excel Services and Report Center

Lab Time: 45 Minutes

Lab Directory: C:\Student\Labs\17_ExcelServices

Lab Overview: One of the difficulties of working with Excel workbook data is sharing the information contained in the workbook without multiple versions floating around. It is quite common to have an updated workbook created, but to still have users working on a previous version of the workbook, which of course creates incorrect data. This can be a major problem when the workbook is being used for mission critical calculations. In this lab you will explore the new Excel Services capability of MOSS to interact with data stored in workbooks on the server. By interacting with a server-based workbook you can prevent versioning issues and allow for greater accessibility to the data. Furthermore, you can utilize various features found in MOSS to ease the user interaction without requiring complex custom code.

In the first exercise, you will examine a mortgage calculation workbook which will be published to a SharePoint document library and driven through Excel Services. You will interact with the workbook on the server by providing new data for fields and calculations. In the second exercise, you will examine how pivot tables and charts behave when deployed on the server. The third exercise involves the addition of custom code to drive the mortgage calculator. This will allow you to access logic and data in back-end systems from inside a workbook.

Exercise 1: Publishing Excel Workbooks with Excel Services

In this exercise you will examine a prebuilt mortgage calculation workbook inside Excel. Next, you will deploy this workbook to a SharePoint **Report Library** and examine the behavior from inside a web-browser. Before deploying the workbook, the SharePoint environment needs to be setup. You will create a new site collection with a top-level site based on the **Report Center** site template. Next you will configure a **Report Library** as a **Trusted Location** to allow access to the workbook reports through Excel Services.

The **Report Center** template contains facilities for creating dashboard pages, **Key Performance Indicator** overviews, as well as providing a library type for reports. Trusted locations are used to ensure data integrity and quality as well as providing a centralized environment for managing resources and user access.

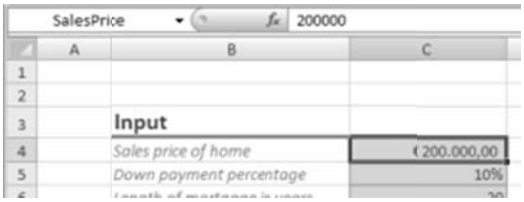
1. Start by examining the mortgage calculation workbook in Excel 2007. The mortgage calculation workbook is a simple workbook using the calculation engine of Excel to provide insight into the height of monthly mortgage payments based on house price and other variables.
    a. Open Windows Explorer and navigate to the following location: **C:\Student\Labs\17_ExcelServices\Starter Files**
    b. Open the **Mortage Calculator.xlsx** workbook in Excel. The workbook contains two worksheets. The **Mortgage Calculator** worksheet is used for calculating

mortgages. The second worksheet contains private instructions which should not be made publicly available.

c. Examine the **Mortage Calculator** worksheet. Alter the values for the mortgage calculation using the following information.
   i. Sales price is **$200,000**
   ii. Down payment is **10%**
   iii. Length of mortgage is **30 years**
   iv. Annual interest is **6.5%**

d. Check that the calculated monthly payment comes down to **$1,137.72.**

e. Take notice of the fields where you entered data. Besides being identified by a column / row index, the cells also have a name defined for them. This will allow you to specify values for these cells through Excel Services.



f. Examine the **Mortgage Guidelines** worksheet. It contains text which should remain outside of the public domain.

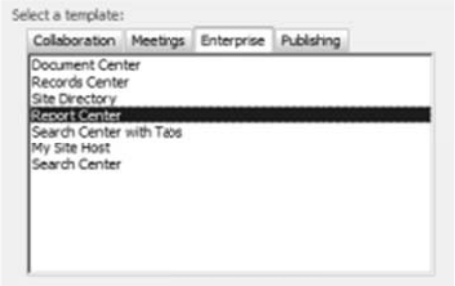
g. Keep **Excel** open, as you will return to it after finishing the next step.

2. Next, you will create a new site collection with a top-level site based on the **Report Center** site template. The Report Center site will be used to store workbook reports and provides other facilities common to a reporting environment. In our case, the site will be used to host the mortgage calculation workbook and the other workbooks used in later exercises.

   a. Open **SharePoint 3.0 Central Administration** and navigate to the **Application Management** page.

   b. On the **Application Management** page, click the **Create Site Collection** link in order to navigate to the Create Site Collection page.

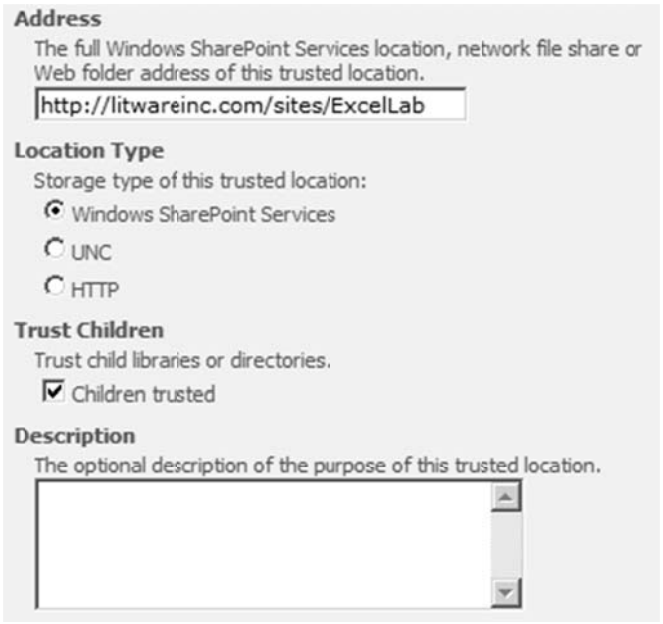
   c. Create a new site collection using the following values.
      i. Be sure that the **Web Application** is set to **http://litwareinc.com/** (Litware Public Site).
      ii. Title is **Excel Lab**.
      iii. Web Site Address is **http://litwareinc.com/sites/ExcelLab**.



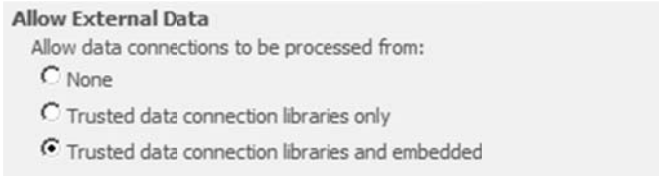
      iv. Template Selection is **Report Center**, which you find under the **Enterprise** group.
      v. **Primary Site Collection Administrator** is **LitwareInc\Administrator**.

d. Click **OK** to start the process of creating the new site collection. When this process has completed successfully you will be presented with an administration page that links to the new top-level site.

3. Now it's time to configure the URL for the Report Center site as a Trusted Location so that it's possible to publish workbooks there that will be loaded and rendered by Excel Services. This is a crucial step because Excel Services only works with workbooks it finds in trusted locations to manage access to resources.
   a. Open **SharePoint 3.0 Central Administration** and navigate to the **Application Management** page.
   b. Click **Litware SSP (Default)** on the **Quick Launch** in order to navigate to the shared services administration page.
      On the **Shared Services Administration** page, click **Trusted file locations** in order to manage trusted file locations for Excel Services.
   c. Click **Add Trusted File Location** to add the location of the Report Center site you created in the previous step of this exercise. The **Location** section is the first section on the page. Create the trusted file location using the following values.
      i. **Address** is **http://litwareinc.com/sites/ExcelLab**.
      ii. **Location Type** is **Windows SharePoint Services**.
      iii. **Trust Children** is checked



      iv. Scroll down and locate the **External Data** section. **Allow External Data** is **Trusted data connection libraries and embedded**.



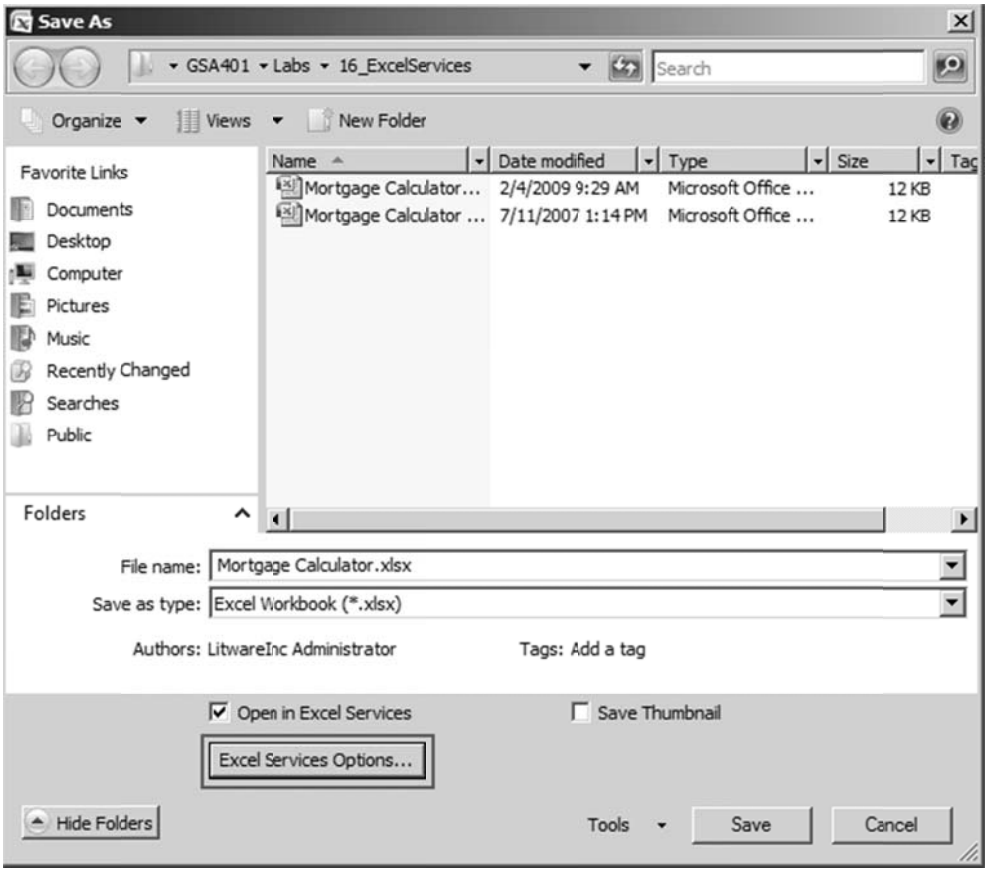
v. The last section is the **User-Defined Functions** section. Make sure **Allow User-Defined Functions** is checked.

NOTE: Trusting children automatically trusts all document libraries in the new Report Center site. The external data will be used in the next exercise to access data from a LOB database. The User Defined Function is a feature that will be used in the last exercise to drive the mortgage calculation using code in a .NET assembly.

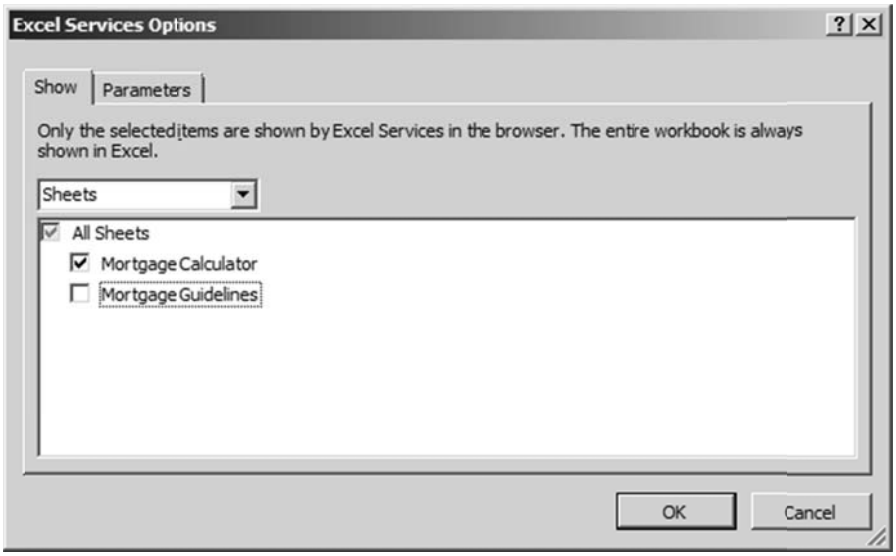
d. Press **OK** to add the new file location to the list of trusted file locations.
e. Open a new command window and issue the **IISRESET** command to reset the web server and allow the new trusted location to become immediately available.

4. Now it's time to publish the mortgage calculation workbook. The mortgage calculation workbook needs to be made available online to allow users to interact with the calculator without having Excel installed on their desktop, as well as allowing users to always work with the latest mortgage calculation formula.
   a. Go back to the **Mortgage Calculator.xlsx** which should have been left open in Excel from the previous exercise.
   b. On the **Office** Menu, click **Publish > Excel Services** to publish the workbook to SharePoint.

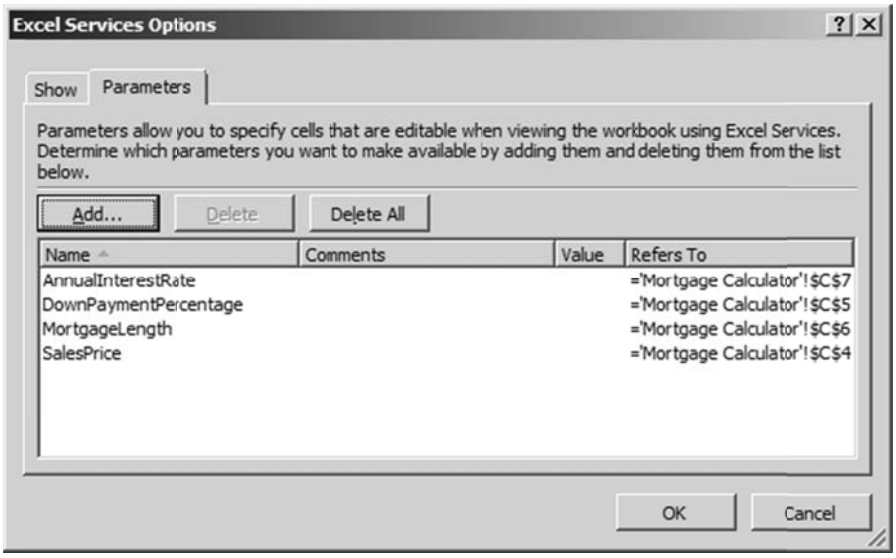
   c. In the **Save As** window, click the **Excel Services Options** button to configure the workbook before publishing it to Excel Services.



i. On the **Show** tab, choose **Sheets** to only allow certain sheets to be made available.
ii. Make sure that only the **Mortgage Calculator** worksheet is published by clearing the checkbox for all other worksheets.



iii. On the **Parameters** tab, review the parameters that are made available for modification through Excel Services. If all 4 of the parameters are not already in the listbox shown (**AnnualInterestRate**, **DownPaymentPercentage**, **MortgageLength**, and **SalesPrice**), then click the **Add** button, place a check next to any of the parameters that were not included already, and click **OK**.



iv. Click **OK** to close the **Excel Services Options** dialog.

d. In the **Save As** window, Click on **My Network Places** and double click on **ReportsLibrary on litwareinc.com**, then type **Mortgage Calculator.xlsx** into the **File Name:** textbox. If **ReportsLibrary on litwareinc.com** is not already there, then you need to make the connection available by doing the following [If it is there already, then click **SAVE** and move on to step e.]:
   i. Type or paste this link into the **File Name:** textbox: **http://litwareinc.com/sites/ExcelLab/ReportsLibrary/Mortgage Calculator.xlsx.**
   ii. Click **SAVE.**
e. In the **Choose Document Type** dialog, choose **Report** and click **OK** to publish the workbook.
f. The workbook is automatically opened in a web-browser. Keep this browser window open for the next exercise.

5. Finally, examine the mortgage calculation workbook in Excel Services. The view of the mortgage calculator workbook is nearly the same as the Excel-based view. The users can now interact with the workbook by providing values for the named cells. These values are entered in a different fashion than the Excel counterpart. Percentage values are a sample of this.
   a. Examine the **Mortage Calculator** worksheet using Excel Services. Alter the values for the mortgage calculation using the following information. Take note of the decimal separator character. It is locale specific and depends on the locale configured in SharePoint. Also take note of how percentages are noted, using the value '1' for 100%.

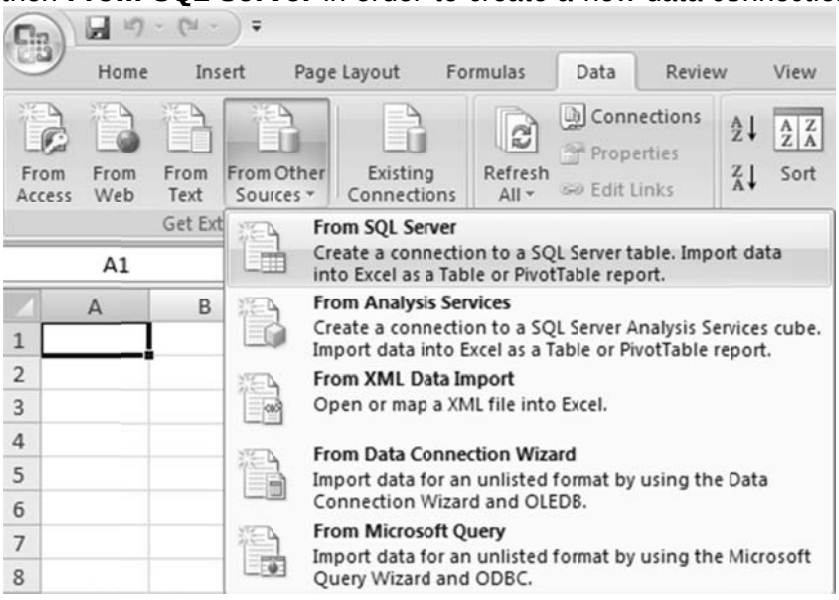
i. Sales price is **200,000**
ii. Down payment is **0.1**
iii. Length of mortgage is **30**
iv. Annual interest is **0.065**

b. Check that the calculated monthly payment comes down to **$1,137.72.**

Exercise 2: Publishing Excel Workbooks with a Data Connection

While the ability to publish a workbook online poses a great solution platform, many workbooks are not a self-contained unit. Instead they retrieve information from an array of external data sources such as the Microsoft SQL Server database engine and Analysis Services. In this exercise you will create a new workbook which rolls up information about sales based on the region, salesperson and year. Next the data will be displayed in a pivot-chart before publishing the spreadsheet to Excel Services.

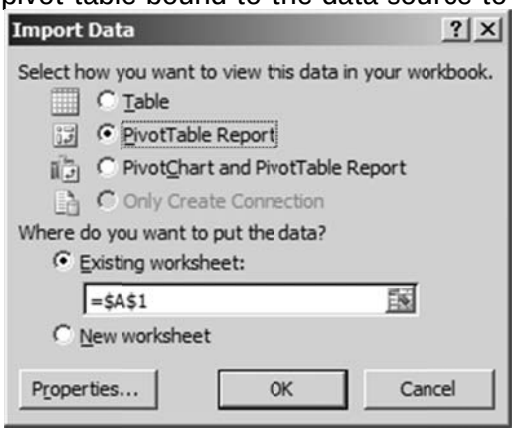
1. In this step you will create the sales overview using a pivot table. First, you will create a new workbook which will display a sales overview in a pivot table. Next, you will create a pivot chart that is defined on top of this table.
   a. Open a new instance of **Excel**. An empty workbook will be made available automatically. [**IMPORTANT**: Don't just open a new workbook in the same instance of Excel that you have been using. You must close and reopen Excel for publishing to work correctly later.]
   b. On the **Get External Data** group in the **Data** tab, click **From Other Sources** and then **From SQL Server** in order to create a new data connection to SQL Server.



   c. In the **Data Connection Wizard** dialog perform the following steps.
      i. In the **Connect to Database Server** step, enter **LITWARESERVER** for the server name. Leave the log on credentials to **Windows Authentication**. Click **Next** to navigate to the next step.

ii. In the **Select Database and Table** step, select **AdventureWorks2008** as the database and **vSalesPersonSalesByFiscalYears** view as the data source. Click **Finish** to complete the wizard and close the dialog.

d. In the **Import Data** dialog, select **PivotTable Report** and click **OK** to add a new pivot table bound to the data source to the first worksheet.



e. In the **Pivot Table Field List** pane, configure the pivot table according to the following information. If the pane does not appear, right-click anywhere in the pivot table and choose **Show field list**. Drag the fields from the field list into the four container areas at the bottom of the pane.

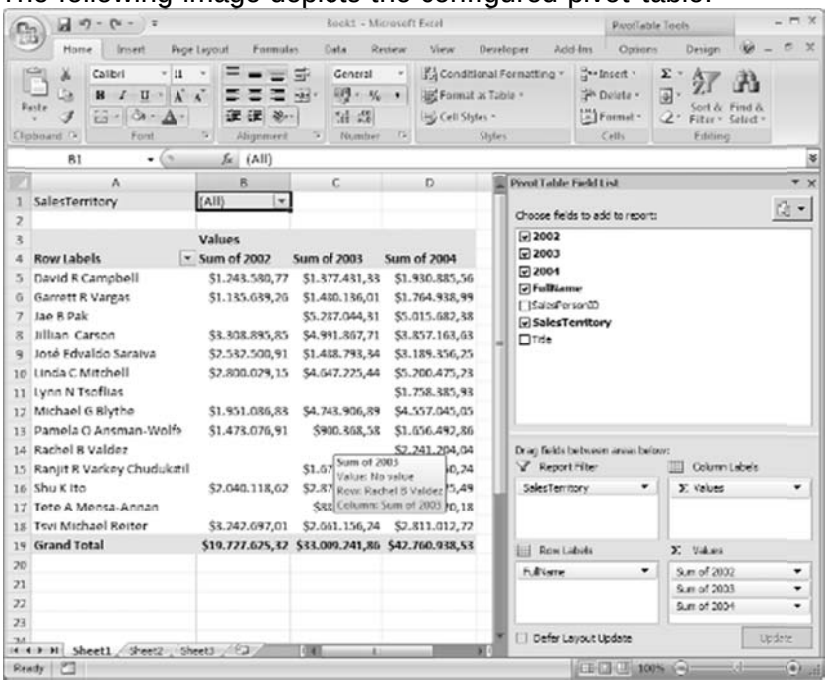
i. For **Row Labels**, add the **FullName** field

ii. For **Values**, add the **2002**, **2003** and **2004** fields. Configure each field to display the SUM instead of the default COUNT.

1. Click the field and choose **Value Fields Settings...**.
2. On the **Summarize by** tab, choose **Sum**.
3. Optionally change the number format to reflect the **Currency** type.
4. Click **OK** to close the settings dialog.

iii. For **Report Filter**, add the **SalesTerritory** field.

iv. The following image depicts the configured pivot table.



v. Select a cell in the pivot table. The next step will create a new pivot chart, and when doing so Excel will recognize that the chart needs to be bound to the selected pivot table. In the sample image above cell **B1** is selected, containing the report filter.

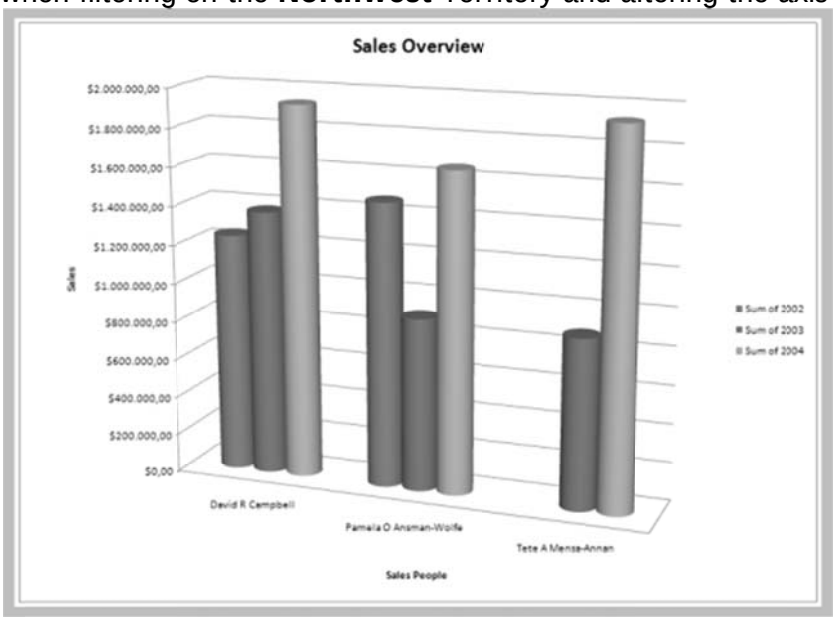
vi. Right-click the **Sheet1** tab, while keeping the cell selected. The tab can be found on the bottom of the Excel window. Choose **Insert** in order to open the sheet creation dialog.

vii. In the **Insert** dialog, choose **Chart** and click **OK** to insert a new chart sheet into the workbook. The chart should automatically display the records of the pivot table.

viii. Inside the **Type** group, on the **Pivot Chart Tools – Design tab**, click **Change Chart Type** to alter the chart type. Choose the **Clustered Cylinder** chart type.

ix. Optionally provide a value for the **SalesTerritory** filter to lessen the number of records displayed. The following image depicts how the chart can look

when filtering on the **Northwest** Territory and altering the axis labels.



x. On the **Office** menu, click **Save** in order to open the Save dialog. Choose **Sales Overview.xlsx** as the name for the document. Save this workbook to **C:\Student\Labs\17_ExcelServices\Lab**.

xi. Keep the workbook open as you will return to it in the following task.

2. Now it's time to publish the sales overview workbook to allow users to see a daily overview of the data in the sales workbook.
   a. Go back to the **Sales Overview.xlsx** that should still be open in Excel.
   b. On the **Office** Menu, click **Publish > Excel Services** to publish the workbook to SharePoint.
   c. In the **Save As** dialog, click **Excel Services Options** to configure the workbook before publishing it to Excel Services.
      i. On the **Show** tab, choose **Sheets** to only allow certain sheets to be made available.
      ii. Make sure that only the sheet containing the table and the one containing the chart are published by clearing the checkbox for all other worksheets.
      iii. Choose **OK** to close the Excel Services Options dialog.
   d. In the **Save As** window, Click on **My Network Places** and double click on **ReportsLibrary on litwareinc.com**, then type **Sales Overview.xlsx** into the **File Name:** textbox. Click **SAVE**.
   e. In the **Choose Document Type** dialog pop-up, choose **Report** and click **OK** to publish the workbook.
   f. The workbook is automatically opened in a web-browser.
3. Finally, examine the mortgage calculation workbook in Excel Services. You should observe that the view of the sales overview workbook is nearly the same on the web as in Excel. 3D charts are converted to their 2D equivalent and are rendered using full HTML.
   a. Examine the worksheet containing the pivot table in Excel Services. Alter the value for the **SalesTerritory** filter.
   b. Examine the chart sheet containing the pivot chart in Excel Services. Notice that the view is similar to the view provided by Excel, but differing slightly in layout.

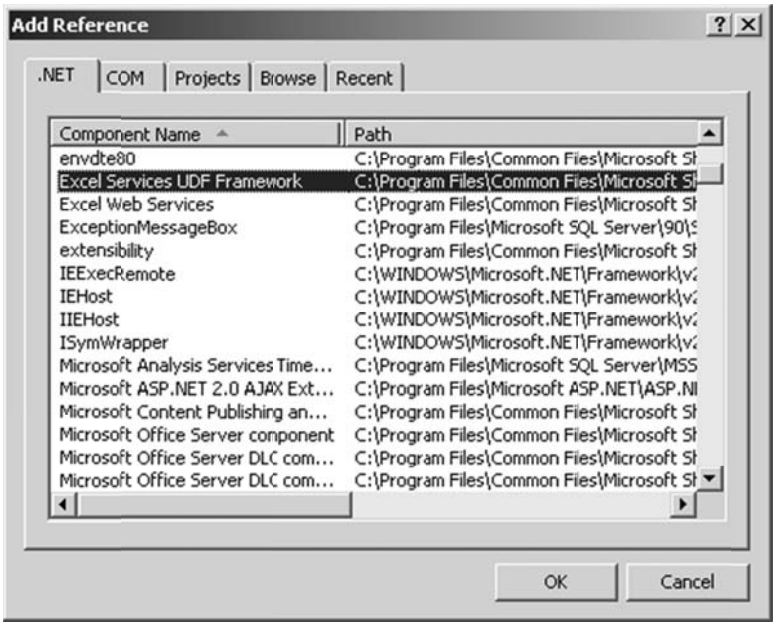
Exercise 3: Calling a method from a .NET Assembly from a Server-side Workbook

While the calculation engine provided by Excel is very elaborate and powerful, there might be times where there is a need to extend the functionality provided by Excel or to access previously created business logic. For standard client-side workbooks loaded inside a desktop version of Excel, one would resort to using a tool such as Visual Studio Tools for Office to embed custom .NET code right inside the workbook. A workbook hosted by Excel Services is not allowed to use these extensions since there is no client application to host the add-in runtime. Excel Services does allow a workbook to call into back-end code by using a User Defined Function (UDF). Using UDFs you can call a method on a class and return values back into the workbook. You can use values, cells, or cell ranges as the input for the UDF and also retrieve a similar set of values from the return value.

In this exercise you will move the mortgage calculation function of the first exercise into a new code class. This will allow the mortgage calculation to be shared by many different applications requiring mortgage calculation. The workbook will use a User Defined Function to call into the code class to retrieve the mortgage estimate.

1. Create a Visual Studio project to develop a new UDF class library. Before the workbook can access the extracted function, a new class library needs to be created and deployed into a location known to Excel Services.
    a. Open **Microsoft Visual Studio 2008** and create a new **Class Library** project named **MortgageCalculator**. Be sure to select **.NET Framework 3.0** in the dropdown box in the top right-hand corner. This is because .NET 3.0 is automatically included in MOSS 2007; however, .NET 3.5 is chosen by default in Visual Studio 2008, but may not be installed as a general rule in a company's farm. Create the project in the **C:\Student\Labs\17_ExcelServices** directory.
    b. In the Solution Explorer, right-click the **Class1.cs** node and click **Rename**. Name the file **Calculator.cs**.
    c. In the Solution Explorer, right-click the **References** node and choose **Add Reference** in order to open the **Add Reference** dialog.
    d. In the **Add Reference** dialog, choose the assembly with the component name of **Excel Services UDF Framework** to add a reference to

Microsoft.Office.Excel.Server.Udf.dll and then click **OK**.



e. In the **Solution Explorer**, double-click the **Calculator.cs** node in order to open the code file in the Visual Studio editor.
f. In the **Calculator.cs** code file, perform the following steps.
    i. Add a new public method to the **Calculator** class called **CalculateMortgage**. This method should take four parameters and return a **double** value.
        1. **SalesPrice** of type **int**.
        2. **MortgageLength** of type **int**.
        3. **DownPaymentPercentage** of type **double**.
        4. **AnnualInterestPercentage** of type **double**.
    ii. Hard-code a return value of -1, in order for the project to compile correctly. Later on you will provide a real implementation for the mortgage calculation method.
    iii. Add a new **using** statement that includes the **Microsoft.Office.Excel.Server.Udf** namespace. This namespace is part of the **Microsoft.Office.Excel.Server.Udf.dll** assembly referenced earlier.
    iv. Adorn the **Calculator** class with the **[UdfClass]** attribute.
    v. Adorn the **CalculateMortgage** method with the **[UdfMethod]** attribute.
    vi. The final class definition should look like the following code sample.

```
using System;
using Microsoft.Office.Excel.Server.Udf;

namespace MortgageCalculator {

  [UdfClass]
  public class Calculator {

    [UdfMethod]
    public double CalculateMortgage(int salesPrice,
                                    int mortgageLength,
                                    double downPaymentPercentage,
                                    double annualInterestPercentage) {
```



```
      // implementation
      return -1;
    }
  }
}
```

g. Compile the project by clicking **Build MortgageCalculator** on the **Build** menu. Make sure that the project compiles correctly. Repair any errors that might turn up.
h. In the **CalculateMortgage** method, provide an implementation for the calculation. The following sample displays a similar calculation as was provided earlier in the **Mortgage Calculator.xlsx** workbook.

```
double financed = (1 - downPaymentPercentage) * salesPrice;
int nrOfMonths = mortgageLength * 12;
double monthlyInterestRate = annualInterestPercentage / 12;

return financed * (monthlyInterestRate / (1 - Math.Pow((1 +
monthlyInterestRate), nrOfMonths * -1)));
```

i. Compile the project by clicking **Build MortgageCalculator** on the **Build** menu. Make sure that the project compiles correctly. Repair any errors that might turn up.

2. Now it's time to deploy the assembly DLL produced by your UDF class library project. The compiled UDF class needs to be placed in a location registered in Excel Services. You can register arbitrary locations on the file-system or use the Global Assembly Cache. In this lab, you will use a common UDF directory which will be registered with Excel Services.
    a. Open Windows Explorer and navigate to the root folder of the **C:** drive.
    b. Add a new folder named **C:\UDFs**. Since the files in this directory will be accessed by the IIS worker process for the hosting Web Application, the service account tied to this worker process, which will be servicing Excel Services requests, needs read access to this folder.
    c. Right-click the directory and choose **Properties**.
    d. On the **Security** tab, add the **litwareinc\sys-spWorkerProcess** account and give this account read access by doing the following:
        i. Click **Add**.
        ii. Type **litwareinc\sys-spWorkerProcess** in the **Enter the object names to select (examples)** textbox.
        iii. Click **OK**.
        iv. Make sure **litwareinc\sys-spWorkerProcess (sys-spWorkerProcess@litwareinc.com)** is highlighted.
        v. Check **Read** in the **Permissions for litwareinc\sys-spWorkerProcess** box.
        vi. Click **OK**.
    e. Copy the **MortgageCalculator.dll** assembly into the new UDFs folder. The **MortgageCalculator.dll** assembly can be found in your project directory at the following path **\MortgageCalculator\bin\Debug**.
3. Register the UDF class library with Excel Services. The new location of the UDF assembly needs to be trusted by Excel Services for the call into the UDF to work.
    a. Open **SharePoint 3.0 Central Administration.**
    b. On the **Quick Launch** select the **LitwareSSP** Shared Services Provider.

c. On the **Shared Services Administration** page, click **User-defined function assemblies** in order to manage UDF assemblies registered with Excel Services.
d. On the **User-Defined Functions** page click **Add User-Defined Function Assembly** in order to register a new UDF assembly with Excel Services.
e. On the **Add User-Defined Function Assembly** page, alter the values for the UDF assembly using the following information.
    i. **Assembly** is **C:\UDFs\MortgageCalculator.dll**.
    ii. **Assembly Location** is **File path**.
    iii. Click **OK** to add the assembly to the registered UDF assemblies.

4. Modify an Excel workbook to call the UDF method. This time the workbook itself will not contain the implementation for calculating a mortgage. Instead, the implementation will be leveraged from the UDF in the .NET you created in the previous steps.
    a. Open Windows Explorer and navigate to the following location: **\Student\Labs\17_ExcelServices\Starter Files**.
    b. Open the **Mortage Calculator UDF.xlsx** workbook in Excel. The workbook is similar to the earlier mortgage calculation form, only without the calculation logic.
    c. In cell C9, create a function that calls the **CalculateMortgage** function defined earlier in the class library. Use the defined names for the four value cells to pass parameters into the **CalculateMortgage** function. A sample of the function to use is as follows:

```
=CalculateMortgage(SalesPrice,MortgageLength,DownPaymentPercentage,AnnualInterestRate)
```

    d. Keep the workbook open as you will return to it in the next step.
5. Publish the UDF mortgage calculation workbook. This is the last step before you can view the result in Excel Services.
    a. Go back to the **Mortgage Calculator UDF.xlsx** workbook that should still be open in Excel.
    b. On the **Office** Menu, click **Publish** and then **Excel Services** to publish the workbook to SharePoint.
    c. In the **Save As** window, click **Excel Services Options** to configure the workbook before publishing it to Excel Services.
        i. On the **Show** tab, choose **Sheets** to only allow certain sheets to be made available.
        ii. Make sure that only the **Mortgage Calculator** worksheet is published by clearing the checkbox for all other worksheets.
        iii. On the **Parameters** tab, review the parameters that are made available for modification through Excel Services. If all 4 of the parameters are not already in the listbox shown (**AnnualInterestRate**, **DownPaymentPercentage**, **MortgageLength**, and **SalesPrice**), then click **Add**, place a check next to any of the parameters that were not included already, and click **OK**.
        iv. Click **OK** to close the **Excel Services Options** dialog.
    d. In the **Save As** window, type the following URL into the **File name** textbox: **http://litwareinc.com/sites/ExcelLab/ReportsLibrary/Mortgage Calculator UDF.xlsx**.
    e. Click **Save**.

f. In the **Choose Document Type** dialog, choose **Report** and click **OK** to publish the workbook.
g. The workbook is automatically opened in a web-browser.

6. Examine the mortgage calculation workbook in Excel Services. The view of the mortgage calculator workbook is nearly the same as the Excel-based view. The users can now interact with the workbook by providing values for the named cells. The calculation takes place on de server inside the custom UDF assembly, instead of through normal Excel functions.
    a. Examine the **Mortage Calculator** worksheet as it appears in Excel Services. Alter the values for the mortgage calculation using the following information. Take note of the decimal separator character. It is locale specific and depends on locale configured in SharePoint. Also take note of how percentages are noted, using the value '1' for 100%.
        i. **Annual interest** is **0.065**
        ii. **Down payment** is **0.1**
        iii. **Length** of mortgage is **30**
        iv. **Sales** price is **200,000**
    b. Verify that the calculated monthly payment comes down to **$1,137.72**

Lab 19: Building a Word 2007 Add-In

Lab Time: 20 Minutes

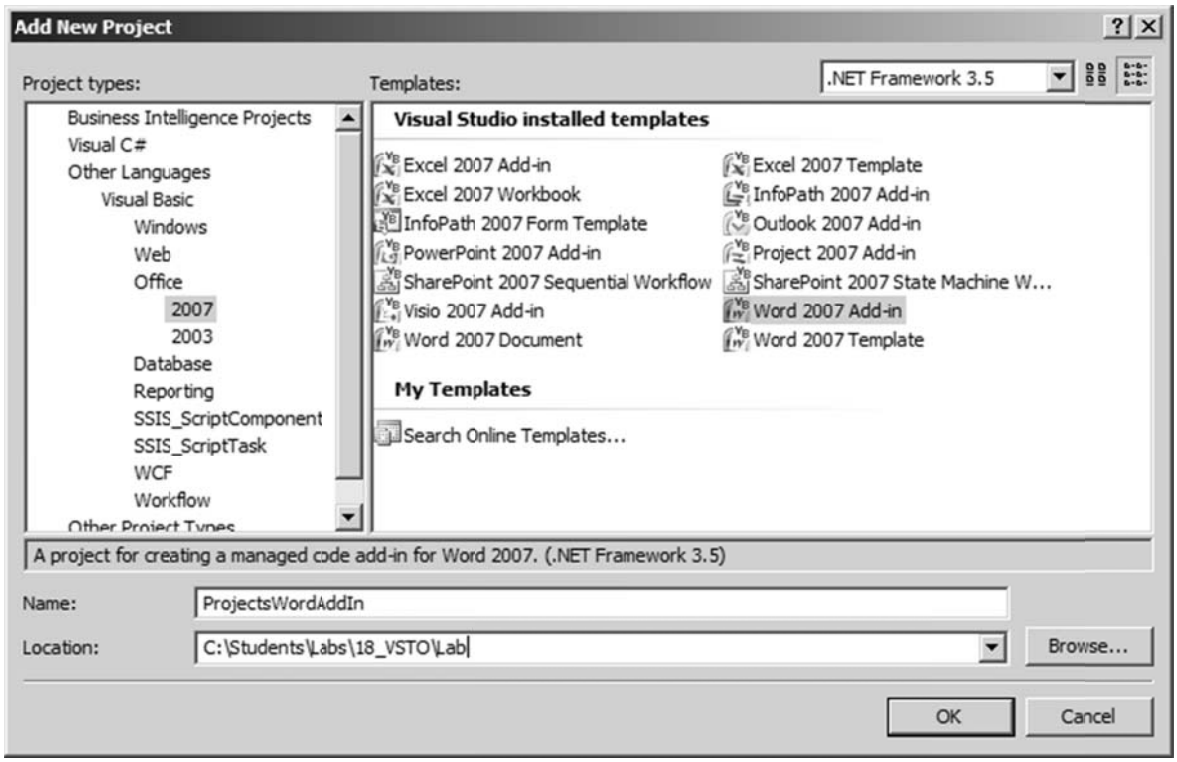
Lab Directory: C:\Student\Labs\19_VSTO

Lab Overview: In this lab you will create a Word 2007 add-in that displays a custom ribbon with one toggle button. This button will show or hide a custom task pane where the user can select a project from a dropdown list. This dropdown is populated from the Projects list you created in the first lab on custom branding. When the user selects a project from the dropdown, project details will be displayed in the custom task pane. The user can then decide to insert the selected project in the word document.

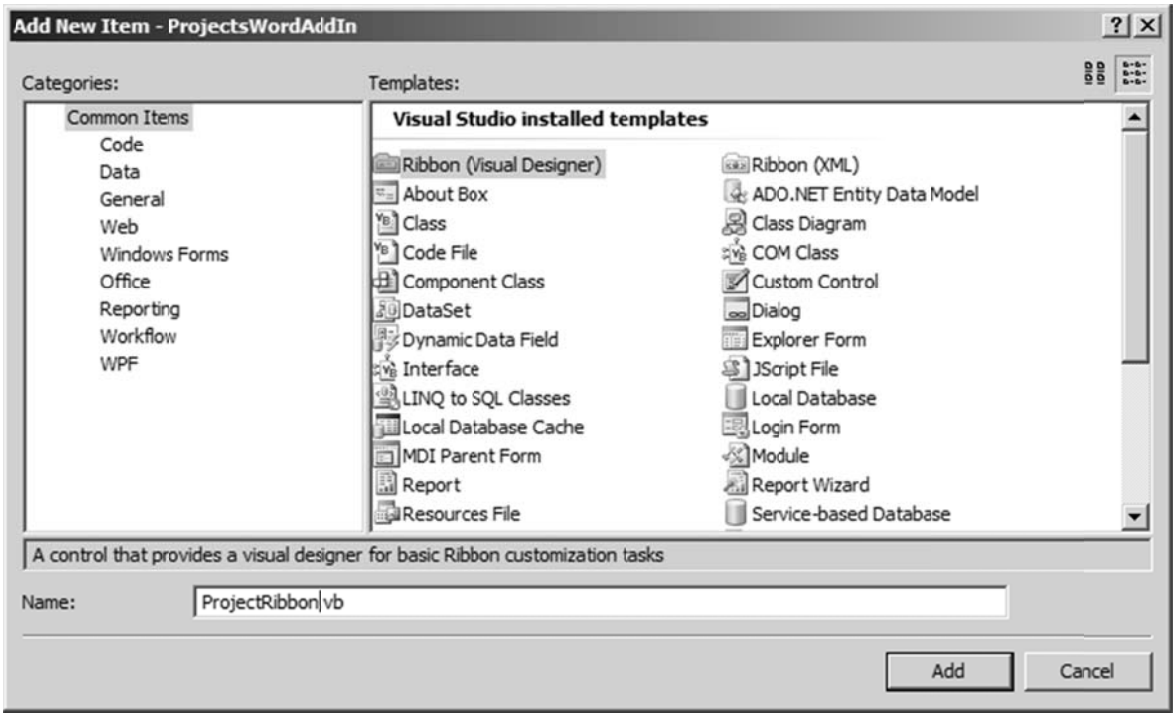
This lab contains directions to develop an Office Add-in using VB.NET.

Exercise 1: Create the custom ribbon for the Word 2007 Add-In

1. Launch Visual Studio and create a new project of type Office 2007 and choose the Word 2007 template. Give the project the name **ProjectsWordAddIn** and save it in the **C:\Students\Labs\19_VSTO\Lab**



2. Click the **OK** button to create the project.
3. Add a new item of type **Ribbon (Visual Designer)** and give it the name **ProjectRibbon.vb**. Click the **Add** button.



4. The ribbon designer opens with one tab and one group. Select the tab and set the following properties for the tab:
    a. Name: **LitwareTab**
    b. Label: **Litware**
5. Select the group and set the following properties for the group:
    a. Name: **LitwareGroup**
    b. Label: **Litware Projects**
6. Expand the **Toolbox** and expand the **Office Ribbon Controls** group. Drag a **ToggleButton** from the Toolbox into the group.
7. Select the button and set the following properties:
    a. Name: **ProjectsButton**
    b. ControlSize: **RibbonControlSizeLarge**
    c. Label: **Show Projects**
    d. OfficeImageId: **AddressBook**
8. In first instance we are only going to change the label of the button when the toggle button is clicked. Double-click the button to create an event handler for the **Click** event.

```
Private Sub MyToggleButton_Click(ByVal sender As System.Object, _
    ByVal e As Microsoft.Office.Tools.Ribbon.RibbonControlEventArgs) _
    Handles MyToggleButton.Click

End Sub
```

9. When the button is clicked we want to change the label of the button. Place this code in a separate method because later on it will be called from outside the ribbon:

```
    Public Sub RefreshToggleButton()
        If (MyToggleButton.Checked) Then
            MyToggleButton.Label = "Hide Products"
```

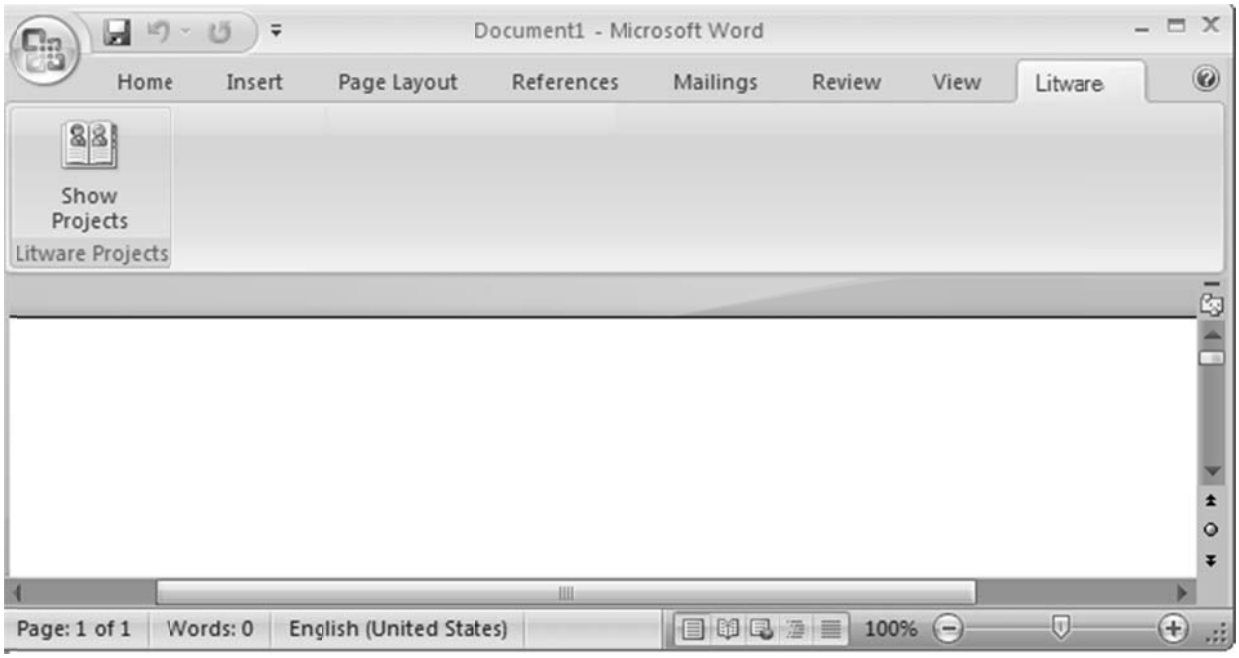


```
        Else
            MyToggleButton.Label = "Show Products"
        End If
    End Sub
```

10. Add the following lines of code in the Click event handler:

```
Private Sub MyToggleButton_Click(ByVal sender As System.Object, _
    ByVal e As Microsoft.Office.Tools.Ribbon.RibbonControlEventArgs) _
    Handles MyToggleButton.Click
      RefreshToggleButton()
End Sub
```

11. This is the easiest part of the project. Press F5 to build and run the project.
12. Word 2007 opens with a new document and your **Litware** tab should be visible.
13. Click the **Litware** tab and click the button. Notice that the label changes.



14. Close Word.

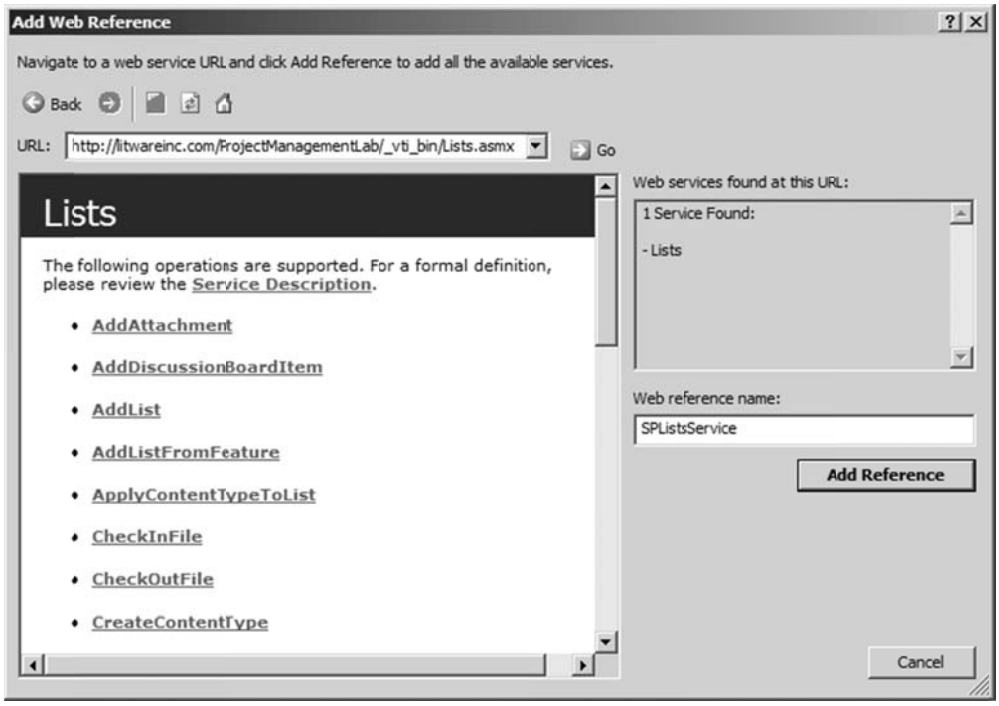
Exercise 2: Create the custom task pane

1. Custom task panes are developed in user controls. Return to Visual Studio and add a user control to the project. Give it the name **ProjectsPane**.
2. First you are going to design the user interface of the task pane.
    a. Set the **Width** property to **220** and the **Height** property to **400**. (The height doesn't really matter because it will take the height of the Word instance.

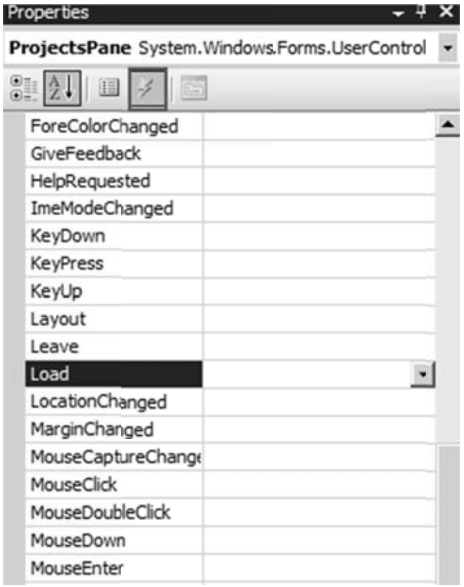
b. Add the following controls to the user interface so that it looks as follows:

Name	Label
ProjectLabel	Select a project:
ProjectComboBox	
NameLabel	Project name:
ClientLabel	Client:
ClientTextBox	
AmountLabel	Contract amount:
AmountTextBox	
BeginDateLabel	Begin date:
BeginDateTextBox	
EndDateLabel	End date:
EndDateTextBox	
ContractCheckBox	Contract signed
InsertButton	<< Insert

3. When the task pane is loaded, you want the dropdown to be populated with the projects available in the **Projects** list that you created on the **http://litwareinc.com/sites/ProjectManagementLab**. This data can be retrieved using the **GetListItems** method of the SharePoint **Lists.asmx** web service.
4. First add a reference to the Microsoft.SharePoint.dll. You can find this dll as **Windows SharePoint Services** assembly in the .**NET** tab of the **Add Reference** dialog.
5. You also need to add a service reference to the **Lists.asmx** web service.
    a. In Solution Explorer right click the project and choose **Add Service Reference**.
    b. Click the **Advanced** button at the bottom of the dialog.
    c. Click the **Add Web Reference** button at the bottom of the **Service Reference Settings** dialog.
    d. Enter the URL to the Project Management SharePoint site, completed with the path to the **Lists.asmx** web service. The SharePoint web services are accessible from the **_vti_bin** directory:
    http://litwareinc.com/ProjectManagementLab/_vti_bin/Lists.asmx
    e. Click the **Go** button to load the web service. The URL is entered correctly when the methods of the web service appear.
    f. Enter **SPListsService** as name for the web reference.



6. To avoid having to write all the plumbing code to retrieve the data, you can find a helper class in the **Starter Files** of this lab with the name **ProjectsHelper.vb**. Copy it to your project directory and include it in the project. Take your time to inspect the code. The helper file contains a class definition for the Project object and a helper class. The project data is downloaded using the **GetListItems** method of the web service. The result is an xml string that is parsed to a collection of project object using LINQ.
7. Return to the **ProjectsPane** in Designer mode. The projects dropdown from this custom task pane needs to be populated when the task pane loads.
    a. Go to the **Properties** box of the task pane and click the **Events** button.
    b. Locate the Load event and double-click it to generate the appropriate event handler.



c. First declare 2 class level variables: one for the list of projects and one for the selected project:

```
Dim projectList As List(Of Project)
Dim selectedProject As Project
```

d. Place your cursor in the **Load** event handler and add code to instantiate the helper class and retrieve the list of projects from the SharePoint web site:

```
Dim helper As New ProjectsHelper("Projects")
projectList = helper.GetProjects()
ProjectComboBox.DataSource = projectList
```

8. When a user selects a project from the dropdown the details should be displayed in the **TextBox** controls on the pane.
   a. Return to the **ProjectsPane** in Designer mode and double click the dropdown to generate the **SelectedIndexChanged** event handler.
   b. Add code to retrieve the selected project and populate the TextBox controls:

```
selectedProject = projectList(ProjectComboBox.SelectedIndex)
NameTextBox.Text = selectedProject.Name
ClientTextBox.Text = selectedProject.Client
AmountTextBox.Text = selectedProject.ContractAmount.ToString( _
       System.Globalization.CultureInfo.InvariantCulture) + "$"
BeginDateTextBox.Text = selectedProject.ProjectBeginDate.ToString()
EndDateTextBox.Text = selectedProject.ProjectEndDate.ToString()
ContractCheckBox.Checked = selectedProject.ContractSigned
```

9. When a user clicks the **Insert** button the selected project name should be filled out in the word document.
   a. Return to the **ProjectsPane** in Designer mode and double click the button to generate the event handler for the **Click** event.
   b. Add code to

```
Globals.ThisAddIn.Application.Selection.Range.Text = _
    String.Format("The project {0} is signed for an amount of {1}, " + _
    "starting {2} and ending {3}.", _
    selectedProject.Name, AmountTextBox.Text, _
    BeginDateTextBox.Text, EndDateTextBox.Text)
```

10. This finishes off the work to the custom task pane. Now you need to establish the interaction between the custom ribbon button and the custom task pane. In order to achieve this you have to change the code behind the ribbon button.
    a. Open the **ThisAddIn.vb** class.
    b. Declare a class level variable for the task pane:

```
Friend projectsTaskPane As Microsoft.Office.Tools.CustomTaskPane
```

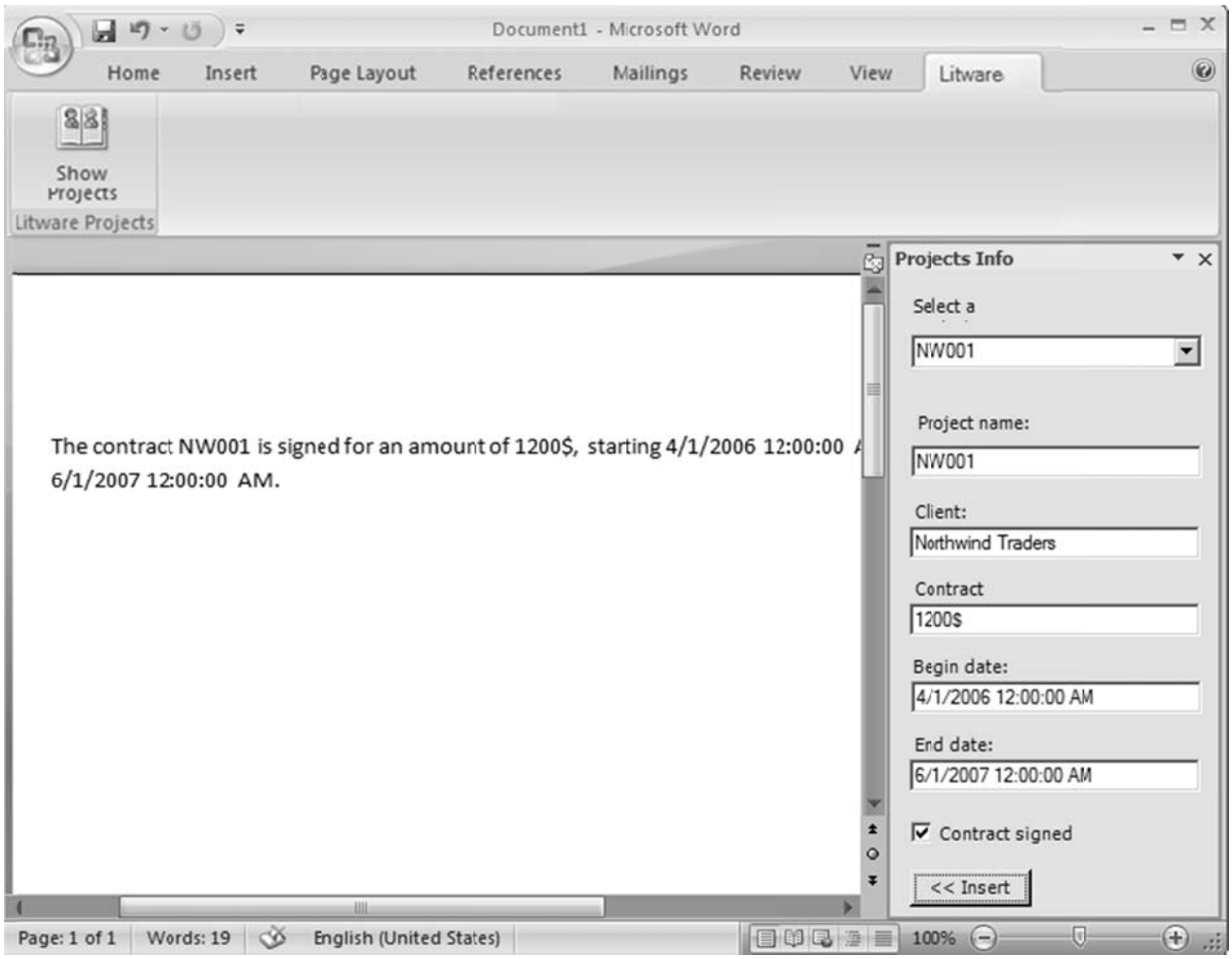
c. Locate the **Startup** event handler for the add-in and write code to add the custom task pane to the collection of custom task panes:

```
projectsTaskPane = Me.CustomTaskPanes.Add(New ProjectsPane(), "Projects
Info")
```


d. Open the **MyRibbon.vb** in code view and add the following to the Click event of the toggle button in order to:

```
Globals.ThisAddIn.projectsTaskPane.Visible = MyToggleButton.Checked
```

11. This concludes the work on the Word AddIn. Press F5 to build and run the project. A Word document should open.
12. Click the **Litware** tab and click the button. You custom task pane appears.
13. Select a project and verify the details displayed in the task pane.
14. Click the **Insert** button and notice the sentence in the word document, containing information on the selected project.
15. If you click the ribbon button again, the task pane will disappear.
16. Close Word.



17.

