Application Templates for Windows SharePoint Services 3.0: Under the Hood

June 2007
The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

This article is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS DOCUMENT.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

2007 Microsoft Corporation. All rights reserved.
# Table of Contents

Application Templates for Windows SharePoint Services 3.0: Under the Hood... i

Introduction................................................................. 1

**Introduction to Tools and Technologies Used**.......................... 4

- Windows SharePoint Services 3.0 Capabilities................................. 4
- SharePoint Designer: The Premier Tool for Building SharePoint Applications................................................................. 6
- Visual Studio............................................................................ 7
- Microsoft Office Access 2007 ....................................................... 7
- Approaches and General Techniques .......................................... 7

**Application Architecture and Development Methodology**.............. 9

- Define the Functional Requirements for the Application.................. 9
- Validate Your Data Model.......................................................... 11
- Determine the Components and Their Relationships..................... 11
- Start Building the Application Components.................................. 12
- Determine the Customizations Needed in the Application.............. 13

**Implementing Common Design Patterns**.................................... 15

- Using Custom Forms.................................................................... 15
  - Task Based Customization.......................................................... 16
  - Navigation.................................................................................. 17
  - Assigning Roles to Users............................................................. 17
  - Other Examples of this Design Pattern........................................ 17

- **Controlling Action Flow**......................................................... 18
  - Other Examples of this Design Pattern........................................ 20

- Using Parent-Child Relationships............................................... 20
  - Creating a Default Link between a New List Item and an Existing One.. 20
  - Other Examples of this Design Pattern........................................ 22

- **Using Workflow**.................................................................... 22
  - Workflow Usage Considerations.................................................. 23
  - Using SharePoint Designer to Build a Custom Workflow.................. 24
  - Other Examples of this Design Pattern........................................ 24

- **Using Dashboards**................................................................. 24
  - Other Examples of this Design Pattern........................................ 29

- **Building Templates**.............................................................. 30
  - Site Definitions.......................................................................... 30
  - Site Templates.......................................................................... 30
  - Using Site Definitions versus Site Templates............................... 31
  - Guidance for Creating Site Templates and Site Definitions........... 32
Introduction

Microsoft® Windows® SharePoint® Services 3.0 is a technology of Windows Server® that offers an integrated portfolio of collaboration and communication services. It is also a platform for developing Web-based business applications. Taking advantage of this capability, Microsoft has developed forty Application Templates to provide out-of-the-box solutions to address the needs of specific business processes such as coordinating a Help Desk or tracking marketing campaigns, as shown in the example in figure 1.

Microsoft developed these freely downloadable application templates to be usable immediately after deployment. However, customers and partners can also use these application templates as the starting point for more customized solutions, or they can use them as teaching examples as they build their own sophisticated Windows SharePoint Services applications using Microsoft Office SharePoint Designer 2007.

The purpose of this paper is to describe how Microsoft developed the application templates, identifying best practices for how to work with core capabilities within both Windows SharePoint Services and SharePoint Designer, with the goal of empowering customers and partners to create their own applications. The paper is not a substitute for the Windows SharePoint Services 3.0 SDK, nor is it primarily a developer resource. Developers should use the SDK for understanding generally how to extend Windows SharePoint Services 3.0.

This paper is meant to be a resource for a new breed of site designers. Because Windows SharePoint Services and SharePoint Designer make it possible to build so much...
application functionality through the UI, advanced development skills are not required to build rich applications. To be sure, this paper does describe some custom code implementations for particularly tricky design patterns, but the overall methodology should be accessible to non-developers and is presented with that audience in mind. Developers may want to quickly read through the early sections on tools and methodology and pay more attention to the description of the design patterns and the specific examples of how to implement those design patterns.

In terms of paper structure, the first section provides an overview of the capabilities within Windows SharePoint Services 3.0 and SharePoint Designer that are used in building applications. The next section describes, in general, the methodology that Microsoft used across all of the application templates. The methodology is a common-sense approach that leads to a real examination of the purpose of the solution, how it will be used, and by whom, and which pieces of technology get you the farthest without having to write custom code. It ends with a process for identifying areas where custom code or other custom work will be needed.

The next section of the paper, “Implementing Common Design Patterns,” focuses on how to use the strengths of both Windows SharePoint Services and SharePoint Designer to address common application design requirements, such as how to create custom actions in a list. This is the core of the paper, and it describes the approaches to the design patterns that recur across all the application templates (and, indeed, any applications that you might come up with). This section provides examples for each of five design patterns, including prescriptive guidance for working with the SharePoint UI or with SharePoint Designer and some custom code.

The last section of the paper, “Building Templates,” describes how to actually create a template file using SharePoint Designer. It also covers other issues, such as localization.

When you finish this paper, you should have a good understanding of how to design and architect an application, how to begin by building a site directly in Windows SharePoint Services, including building linked lists, custom columns, libraries, workflows, and so on, how to then open the site in SharePoint Designer to make further customizations, create custom forms, add custom code to change certain behaviors, create custom workflows,
and so on, and, finally, how to create the application template itself and deploy it for usage.
Introduction to Tools and Technologies Used

There are a number of technologies and tools that come together to make it possible to create applications more easily than ever before. On the technologies side, new capabilities such as workflow support mean that a site designer does not have to write code to bring workflow to an application. On the tools side, SharePoint Designer, Microsoft Visual Studio®, and other tools make it possible to use less and less code (often no code) to do things that used to be very difficult to accomplish.

In general, Microsoft has pursued a strategy across all these things that moves more and more of the difficult plumbing effort into the infrastructure itself, making it possible to be more a designer and less a developer. In other words, Microsoft has already done much of the hard work once, so that you can simply use those capabilities in your applications in an intuitive way through UI.

For the purpose of preparing to understand how the various technologies and tools come together in the application creation process, the following sections discuss Windows SharePoint Services 3.0, SharePoint Designer, and other technologies specifically in terms of relevant new features and abilities. For more comprehensive discussions, review the resources at the end of this document.

Windows SharePoint Services 3.0 Capabilities

Windows SharePoint Services 3.0 introduces some powerful new capabilities (for a comprehensive list, see the Windows SharePoint Services 3.0 Evaluation Guide). The following new capabilities and features are particularly relevant to building custom applications, and you will see many of them mentioned again in later sections:

- **Libraries and Lists** - Windows SharePoint Services 3.0 introduces a number of new library and list types, which can be used as the basis for applications. New library types include the slide library, a library specially designed to store and manage reusable PowerPoint slides, a data connection library, and others.

- **Content Types** - Content types are a core concept used throughout Windows SharePoint Services. Content types are designed to help users organize their
SharePoint content in a more meaningful way. A content type is a reusable collection of settings that can be applied to certain categories of content. Content types enable you to centrally manage and reuse the metadata and behaviors of a document or item type. For example, you can associate workflows and events to a content type, rather than having to add workflows and events to multiple documents or libraries.

- **Site Columns** – Site columns provide a central, reusable model for column definition. When you create a site column, each list that uses this column has the same definition, and you do not have to reproduce the column in each list.
  Site columns provide a way for end-users to pick from a predefined set of columns which might be useful in their list. So, not only can they be used to define columns centrally for well-known list templates, but they provide users with a path to use special columns which can have custom meanings.

- **Workflow** – In Windows SharePoint Services, a workflow enables you to attach a business process to items in SharePoint Products and Technologies. This process can control almost any aspect of an item in SharePoint Products and Technologies, including the life cycle of that item. For example, you could create a simple workflow that routes a document to a series of users for approval. Typically, a site designer or developer will create specific workflows. Site designers can use Microsoft Office SharePoint Designer 2007 to create workflows by using the Workflow Wizard environment, and developers can use Visual Studio 2005 to create more powerful and complex workflows.

- **Feature Framework** – Windows SharePoint Services contains a new structure called a “feature.” A feature packages SharePoint elements that help a user accomplish a particular goal or task. A feature contains one or more elements. An element is an atomic Windows SharePoint Services concept. Windows SharePoint Services features provide an entire framework that you can leverage as a developer to provide custom functionality for Windows SharePoint Services solutions. Features also provide administrators with an easy way to add or remove packaged pieces of functionality.
• **Event Enhancements** – Events fall into two major categories:
  
  o List events—core events, including changes, additions, and removals of list items and list columns (schema changes)
  
  o Simple site events—deletion of sites and site collections

Events are either synchronous “before” events, denoted by the “XYZing” name format, or asynchronous “after” events, denoted by the “ABCed” name format.

• **Mobile Device Access** – Windows SharePoint Services provides new capabilities that allow lists to be rendered appropriately on mobile devices. When a user browses to a Windows SharePoint Services site by using a mobile device, their Web browser will be re-directed to a mobile-specific version of the site that renders site content and lists in a format that is most suitable for the device.

**SharePoint Designer: The Premier Tool for Building SharePoint Applications**

SharePoint Designer is specifically designed to help you create and customize Web sites and workflows built with SharePoint Products and Technologies (Microsoft Windows® SharePoint Services and Microsoft Office SharePoint Server 2007). It provides tools that IT professionals and solution creators need to develop SharePoint-based applications and workflow solutions that enhance organizational agility and business process automation. Using SharePoint Designer, you do not have to use traditional procedural coding languages or techniques to do the following:

• Create no-code data view/forms on a variety of data sources—such as XML files, SQL databases such as Microsoft SQL Server™ 2005, and Web Services

• Create sophisticated, dynamic, no-code workflows

• Perform page layout and design

• Create master pages

• Edit and apply cascading style sheets (CSS)

• Build Web Part pages and connect Web Parts to create sophisticated business applications

http://www.microsoft.com/sharepointapps
Visual Studio

Visual Studio 2005 can be used for adding custom code to applications or for creating custom workflows. You can use the Visual Studio 2005 Designer for Windows Workflow Foundation to create workflow templates and custom workflow activities. You can include code in your workflow, as well as design forms to be used by the workflow to communicate with the workflow users during association and runtime.

Visual Studio 2005 Extensions for Windows SharePoint Services 3.0 is a free download that bundles together a set of tools for developing custom SharePoint applications using Visual Studio. The bundle includes Visual Studio project templates for Web Parts, site definitions, list definitions, and a stand-alone utility program, the SharePoint Solution Generator, which generates a site definition project from an existing SharePoint site. The program enables developers to use the browser and Microsoft Office SharePoint Designer 2007 to customize the content of their sites before creating code by using Visual Studio.


Microsoft Office Access 2007


Approaches and General Techniques

In the context of the development methodology described in the next section, the basic overall steps for creating an application are:

1. Decide whether you’ll need a site definition or a site template
2. Create the core site in Windows SharePoint Services
3. Use SharePoint Designer to open the site, make modifications, etc.
4. Use Visual Studio, if necessary, to create additional custom code, custom workflows, etc.
Application Architecture and Development Methodology

As with any development project, architecting and building a Windows SharePoint Services application will have a greater chance for success if you follow a proven methodology. This section describes the methodology Microsoft used in designing all of the downloadable application templates. There will be nothing surprising in the approach here, especially for experienced developers, but it does take into account some of the specifics of the SharePoint environment, and it does reflect the lessons learned by Microsoft, so it will be of value in this context. Again, this section will be most useful for non-developers who know how to use Windows SharePoint Services and SharePoint Designer.

In brief, the methodology starts with thinking generally about what the application needs to do, who needs to use it, and so forth. Then it looks into more detail at how the data in the application needs to flow, where it needs to be stored, and what relationships there are among various pieces of data. With a good understanding of the data model and the usage scenarios, the methodology calls for jumping in and starting to build out a rough version of the application in Windows SharePoint Services, building lists, libraries, workflows, and so on, in an iterative manner, to the point that it starts to look and behave generally as desired. Finally, the methodology calls for identifying and composing the tweaks and customizations needed to make the application really fit your business process needs.

Define the Functional Requirements for the Application

While it is not necessary to have a highly detailed technical specification document before building a Windows SharePoint Services solution, it is necessary to have a good idea of how the application needs to function. The previous point may seem obvious, but many developers have experienced the pain of the mismatch between what stakeholders think they want in an application and what they actually need for the business process to
succeed (of course, stakeholders usually realize this AFTER they see the nearly finished application!).

All of this is simply to say that starting at the beginning means gathering the functional requirements and thinking about what the application needs to accomplish. For example, if you need a project tracking solution, you will want to identify, at a minimum, the following:

- **What are the actors/roles in the business process?** – in this case, a project owner creates a project and maintains information about tasks, issues, and so on, and task owners have issues and tasks assigned to them and need to interact to complete their jobs. Managers need to see roll-up information about overall project status.

- **What are the UI needs for the different actors?** – in this case, the project owner, task owners, and the manager all need different views relevant to their activities. The task owners ought to be able to see all issues assigned to them, for example, while the project owner ought to be able to see ALL past due issues.

- **What does the business process look like?** – in this case, a project owner creates a project, creates milestones, tasks, and budget entries, then tracks progress over time. The project owner has frequent access across all pieces of information, while task owners need to act on data when something is assigned to them.

- **Where is the data?** – will you only use data in SharePoint, or will you need access to external data (from a database, through a Web service, through the Business Data Catalog, and so on), and will you need to store data outside of SharePoint?

- **What are the relationships among the data?** – in this case, we need to have a project item, a milestone item, and task and issue items, and they have a logical hierarchy. Users are data points as well, and so are things like budget, number of days, and the like.

http://www.microsoft.com/sharepointapps
A great way to find the answers to most of these questions is to literally draw pictures on a whiteboard, iterating through the business process a few times and with a few variations (creating a few project, assigning tasks to two or three people across projects, and so on). Conceptually, the project tracking application is a pretty simple design, and the requirements are pretty clear at this point.

**Validate Your Data Model**

While the data model has already been discussed, it is so important as to require a deeper analysis. Misunderstanding the data model will likely require you to make dramatic, system-wide changes later on, so it is important to get it right.

It is important to understand that there are a number of related pieces of information, as well as a number of pieces of meta-information that describe and define where in the business process a given piece of information resides. In the project tracking case, we need to know more than that there is a project, a milestone, and tasks and issues. We need to know that a project is the top-level item and that it can have multiple milestones. Projects can also have multiple tasks and issues.

**Determine the Components and Their Relationships**

At this point, we have a good functional description of the application. The next step is to plan the architecture and decide how to use the various technologies and capabilities of SharePoint and SharePoint Designer. Every application will involve some combination of the features and capabilities described in the "Introduction to Tools and Technologies Used" section, including:

- Lists and Lookups
- Custom list views (Grouping, Sorting, Filtering, etc., through the SharePoint UI)
- Workflow
- Custom pages and custom Data Views in SharePoint Designer

The lists and lookups really implement the data model. So, in the context of the project tracking application, you would use one list for projects, another list (with a lookup field to the project list) for milestones, another list with a lookup field to the project list for
issues, and so on. For custom page views, you may simply want to set the default view to group on status and sort by health.

Overall, another white board exercise is in order here, where you simply create a mockup of the application, identifying the technologies used for each piece.

**Start Building the Application Components**

Again, it is not necessary to have a very precise technical specification document at this point. It is more important, assuming you have a good handle on the basic data flow and UI requirements, to simply begin building.

Literally, this means creating a new site using one of the out-of-the-box templates in Windows SharePoint Services, or creating your own site template ahead of time and building a new site instance based on that. For many applications, you'll likely just use the **Team Site** or **Blank Site** template as your base.

In the project tracking application, the first thing to do is to create four lists (Project List, Projects Tasks, Project Issues, and Project Milestones). Then, create custom columns for each of those lists. Windows SharePoint Services lets you create a number of different column types, including **Choice**, **Number**, **Person or Group**, and **Lookup** (information already on this site). The Projects List list in Microsoft's **Project Tracking** application template uses many of these, as shown in the graphic below:

![Figure 2: Custom Columns in a List](http://www.microsoft.com/sharepointapps)

The **Health** column is based on the **Choice** column type, the **Budget** column is a **Number** column (using currency), and so on.
In the Project Milestones list, a **Lookup** column is used so that a milestone can be associated in a child-parent relationship with a project. Creating a lookup column is simply a matter of picking the list in which you want to look, and then picking the column from that list (SharePoint populates the drop-down automatically):

![Figure 3: Defining a Lookup column to take information from another list](image)

At this point, you could work in some application logic, building a simple workflow to notify members when new projects are created. You could also begin create custom views within lists using the SharePoint UI to sort and filter to your preference.

**Determine the Customizations Needed in the Application**

Functionally, much of the project tracking application has been implemented with just the SharePoint UI. At this point, we can already see what things work but not well enough, what things can’t be done through the SharePoint UI, and so on. Some areas where we’ll want to take advantage of SharePoint Designer are:
• **Application logic in out-of-the-box workflow** – the workflows you can use out of the box in SharePoint must specify either a user or a group to whom to assign a task. For our project tracking application, we need a way to dynamically determine a recipient for a workflow action, based on the **Owner** property in an issue or task. We can use SharePoint Designer to build a dynamic no-code workflow for this solution.

• **Dashboard and manager views** – we need a way to show a combined view of all projects, all budgets, and so on. We can easily build no-code SharePoint Web parts to show joined and combined information for a dashboard.

• **Parent-child relationships** – there are a few ease-of-use issues when creating tasks. Perhaps we would like to have a task default to a particular project, depending on what page we are on. We can use SharePoint Designer to add logic to automate linking of one item to another.

Microsoft used the same iterative approach while building the forty application templates, and a set of consistent approaches, with consistent best practices, decision trees, and so on, were developed to optimize for what should be done natively in Windows SharePoint Services environment and what should be done using SharePoint Designer and other tools. The next section of this paper discusses these approaches, identifying five “design patterns” and the ways that they ought to be implemented, citing examples from many of the application templates along the way. As such, the remainder of this document is more detailed and deeper technically than the discussion about methodology.
Implementing Common Design Patterns

In this section, we will describe the five basic design patterns in the context of how Microsoft actually used them. Thus, we will refer to a number of different examples across the templates, including code samples and some step-by-step guidance where appropriate. Developers will find this section and the remaining sections on how to actually build an application template most useful, but it is written to be accessible to non-developers as well.

To start, the design patterns are:

- **Custom Forms** – Creating custom look and feel that gives guidance for action at certain point in the business process (this includes only exposing the appropriate properties that are allowed to be modified or are relevant at a given stage)
- **Action Flow** – Creating well-defined actions that lead to the right place for the right actor (for this, we need good navigation control that is sensitive to the action being taken and the actor taking the action)
- **Parent-Child Relationships** – Creating a relationship between two or more SharePoint lists for lookup and reference
- **Workflows** – Using SharePoint Designer to create a Windows Workflow Foundation-based business process
- **Dashboards** – Using Web parts to create an unified view of scattered information throughout the site in one place

**Using Custom Forms**

Custom forms play a vital role in customizing the data input user interface in SharePoint. Just as custom list views are used to present list data in different ways, custom forms are used to present different methods of capturing user input. The need to create such custom forms may arise for various reasons, depending on the business need.
Task Based Customization

Business processes require users to carry out very specific actions that affect business data at different points in the process. A good business process solution makes it easy for users to do their work by exposing the correct information at each stage in the process.

SharePoint includes an edit form with each list by default. However, this form may not show the best set of fields for a particular stage of a business process, and it will not arrange those fields in a way that makes the required action clear. In order to ensure that users are able to easily edit the correct information when carrying out an action, you can create a custom form for each action.

The Bug Tracking solution uses custom edit forms so that at each stage of the bug tracking process, it is immediately clear what information should be entered for a bug item. Each action in the bug tracking process has its own form. For example, when a user chooses to resolve a bug, they are directed to the resolution page (Resolve.aspx). This page allows them to enter the bug resolution and the name of the person who resolved the bug. The user clicks on the Resolve button to submit their modifications. This provides an action flow that is easy to use and helps eliminate errors or missing information.

Creating a custom edit form is relatively easy to do using Microsoft Office SharePoint Designer.

1. Create a copy of the default EditForm.aspx and rename it.

   **Important:** It is important that you create a copy of the default EditForm.aspx and rename it as the first step in this process. If you perform step 2 on the original EditForm.aspx page, you will irreversibly break the list.

2. Delete the list form Web part in your new edit form.

3. Insert a data view that includes the data you want to expose (click **Insert**, **SharePoint Controls**, **Custom List Form**).
4. Using the Insert method, SharePoint Designer automatically creates a save button for the form. The code below shows an example of custom HTML for the save button.

```html
<input type="button" value="Save" name="btnSave" onclick="javascript:{ddwrt:GenFireServerEvent('__commit;__redirectsource')}/"/>
```

**Navigation**

Once you have created pages corresponding to your roles, you still need to provide a way for users to quickly access the relevant dashboard. In the Event Planning solution, we did this by using an XSL template to create a link to the correct dashboard for the current user based on their role.

One limitation of this technique is that if you add more roles to your solution, providing custom dashboards to those roles will require the creation of additional .aspx pages.

It is important to note that this technique does not add any additional control over who has access to information on the site. Any user can still theoretically view any information in the solution.

**Assigning Roles to Users**

There are a variety of ways to create roles and assign them to users. The best solution to this challenge will vary greatly according to the requirements of a particular application or organization. SharePoint does not provide a way to provision role assignments out of the box. In some cases, it may be sufficient to provision roles after a site has been deployed or to allow users to assign roles to themselves as needed.

In the Event Planning solution, users can sign up for the role that best applies to them. This is done through the custom Web part on the front page of the Event Planning workspace.

**Other Examples of this Design Pattern**

For another server admin example of using custom forms, see the Lending Library template. For a site admin example, see the Manufacturing Process Management template.
Controlling Action Flow

A real world web application that is modeled after a business process is rarely contained in a single web page or a Web Part. Instead, it spans several such components, each responsible for a distinct step in the business process. Hence, navigating smoothly and easily between the components of an application becomes a key aspect of the application design. One key design pattern, then, is to control action flow by binding business process actions to navigation, so that taking a specific action leads the user to the appropriate next page or next action.

One method of controlling action flow is to use SharePoint’s Custom Action capability, which allows you to add your own actions to the Embedded Context Button for items in a library. One limitation of this capability is that actions must be hard coded and cannot be parameterized by the name of the item or some other dynamic value.

Another method, which allows you to incorporate dynamic actions into your business process, is to create custom SharePoint lists and use “computed fields.” The Bug Tracking application template uses a custom Bugs list to track bug status and information. Using computed fields, Microsoft was able to incorporate custom links into the list view, so that a user can take action on bugs, such as “Activate” or “Resolve”, and each of these actions has navigation that brings the user to the proper custom form (see previous design pattern) for doing what is necessary to activate or resolve the bug.

Computed fields are a feature of SharePoint that allows you to apply a custom rendering pattern as a column in a SharePoint list. Thus, a business process action (activating a bug) is exposed as a button in a SharePoint column that navigates the user to the correct custom form to carry out that business action.

The computed field supports conditional logic to choose whether an action is displayed. For example, if the bug is already activated, we do not show the activate button.

A computed field is added to a SharePoint list by adding a Field element to the schema XML file that defines the list.

```xml
<Field ID="{EA1D0509-767B-4576-ABEF-FC66647037B9}" Name="ActivateBug" Group="_Hidden" Type="Computed" Sortable="FALSE" Filterable="FALSE"
```
This code shows the XML for the **Activate** computed field. This field shows a clickable orange status link if the bug has not been activated. The user can click the link to go to the activate form for this bug.

http://www.microsoft.com/sharepointapps
The **FieldRefs** element includes a field reference to the status field in the bug list. This reference enables us to do custom rendering of this computed field based on the bug’s status.

The **DisplayPattern** field of a computed column contains the computations and rendering pattern for the column. In this example, we use an if-then-else statement to determine if the status of the bug is active. If the bug is active, we display nothing in the Activate field. If the bug is not active, then the HTML in the Else element is displayed. This HTML code is the image and link that the user can click to activate the bug.

**Other Examples of this Design Pattern**

For another server admin example of controlling action flow, see the Lending Library template. For a site admin example, see the Employee Activities template.

**Using Parent-Child Relationships**

Business solutions often require data to be viewed and used in the context of a relationship to other data. For example, as we have seen with the project tracking application, Projects, Tasks, Issues, and Milestones each stored in their own SharePoint lists. Each task, issue, and milestone is associated with an item on the Project list. Maintaining the relationship between a list and the “child” information contained in other lists can pose a challenge.

**Creating a Default Link between a New List Item and an Existing One**

One common challenge is encountered when a user wants to create a list item that is related to an item in a parent list. In the multiple project tracking solution, this occurs when a user attempts to create a task that is related to an existing project. Out of the box, there is no mechanism in SharePoint to automatically create a relationship between a list item that is being created and one that exists.

From the project details page (DispForm.aspx) in the multiple project tracking application template, a user can create new tasks under that project. The new task creation page (NewForm.aspx) contains a dropdown menu where the user can select the parent project for this task. To make this dropdown default to the parent project displayed on the page
the user navigated from, we pass the ID of the parent project to NewForm.aspx in the querystring. Next, using JavaScript, we parse the Project ID value out and select the relevant project in the dropdown box.

This code shows the link from DispForm.aspx. Notice that the projectID is included in the query string.

When NewForm.aspx is loaded, JavaScript parses the project ID from the query string and changes the project dropdown so that the correct project is selected.

The JavaScript is contained in a Content Editor web Part on NewForm.aspx. The Web Part is set to have no chrome, so that it does not appear on the page unless the page is in edit mode. The JavaScript is placed inside the content editor area.

```javascript
_spBodyOnLoadFunctionNames.push("fillDefaultValues");

function fillDefaultValues() {
    var qs = location.search.substring(1, location.search.length);
    var args = qs.split("&");
    var vals = new Object();
    for (var i=0; i < args.length; i++) {
        var nameVal = args[i].split("=");
        var temp = unescape(nameVal[1]).split('+');
        nameVal[1] = temp.join(' ');
        vals[nameVal[0]] = nameVal[1];
    }
    setLookupFromFieldName("Project", vals["ProjectID"]);
    setLookupFromFieldName("Milestone", vals["MilestoneID"]);
}

function setLookupFromFieldName(fieldName, value) {
    if (value == undefined) return;
    var theSelect = getTagFromIdentifierAndTitle("select","Lookup",fieldName);
    if (theSelect == null) {
        var theInput = getTagFromIdentifierAndTitle("input","",fieldName);
        ShowDropdown(theInput.id);
        var opt=document.getElementById(theInput.opt);
        setSelectedOption(opt, value);
        OptLoseFocus(opt);
    } else {
        setSelectedOption(theSelect, value);
    }
}```
function setSelectedOption(select, value) {
    var opts = select.options;
    var l = opts.length;
    if (select == null) return;
    for (var i=0; i < l; i++) {
        if (opts[i].value == value) {
            select.selectedIndex = i;
            return true;
        }
    }
    return false;
}

function getTagFromIdentifierAndTitle(tagName, identifier, title) {
    var len = identifier.length;
    var tags = document.getElementsByTagName(tagName);
    for (var i=0; i < tags.length; i++) {
        var tempString = tags[i].id;
        if (tags[i].title == title && (identifier == "" || tempString.indexOf(identifier) == tempString.length - len)) {
            return tags[i];
        }
    }
    return null;
}
</script>

The _spBodyOnLoadFunctionNames.push command adds the fillDefaultValues function to the list of scripts to run at load time.

The fillDefaultValues function captures the project ID or milestone ID from the querystring and passes it to setSelectedOptions. setSelectedOptions sets the value of the dropdown menu to the ID for the parent project.

Other Examples of this Design Pattern

For another server admin example of using parent-child views, see the Inventory Tracking template. For a site admin example, see the New Store Opening template.

Using Workflow

Windows SharePoint Services hosts the Windows Workflow Foundation to enable adding workflow logic to an application. SharePoint Designer has a powerful rules-based design tool for customizing and adding conditional logic around the out-of-the-box workflows.
You can also create more complex and customized workflows with code using Visual Studio with the Visual Studio extensions for Windows Workflow Foundation. In any case, there are some considerations for when and how to use workflow, as follows.

**Workflow Usage Considerations**

Workflow is a good solution for asynchronous operations. In the Lending Library solution, workflow is used when a user proposes a new library asset. One of the reasons that workflow is a good choice here is that the process between the proposer and approver is not synchronous.

Workflow also works well for scheduled events. In the lending library solution, workflow is used to handle checkout and due date reminders for assets. Once an item is checked out, the workflow automatically sends a due date reminder when the item becomes due.

Workflow can be used as a means to simplify an otherwise complicated programming task on the server. The Timecard Management application is an example. This application tracks the duration of an employee’s work on a particular task by calculating the time difference between punch in and punch out timestamps. Normally, keeping track of the timestamps would require logging in these two activities to the server (probably to a database) – a notable development effort. Instead, this application relies on one simple workflow step. When the user punches in, the workflow creates an entry in the Time Log list and sets the Start Time value to the current timestamp. Similarly, when the user punches out, the workflow updates the associated row in the same list and sets the End Time value. The duration of the user’s work is simply the difference between the End Time and Start Time values, which is reflected in the calculated column labeled Hours.

Workflows execute asynchronously. If you are creating a solution where the results of an action must be completed immediately before navigating to the next page, workflow may not be the best solution. For example, in the bug tracking solution, workflow is not used to handle bug activation and resolution. One of the reasons for this design choice is that changes in bug status need to be reflected in the user interface immediately after a user takes action on a bug.
Using SharePoint Designer to Build a Custom Workflow

Creating a workflow using SharePoint Designer is a code-free experience. With a site open, you simply click File, New, Workflow, and then follow the wizard. The wizard lets you create sophisticated rules and actions and is integrated with SharePoint lists and libraries so that you can use fields and values directly from SharePoint to drive your workflow logic.

For example, suppose you want to create workflow action that sends an email to a task owner whenever a new task or issue is created. In the Project Tracking template, Owner is a custom column in the SharePoint list, so the workflow can use this column’s value and then dynamically determine the actual email address at run-time.


Other Examples of this Design Pattern

For another server admin example of using workflows, see the Job Requisition and Interview Management template. For a site admin example, see the Clinical Trial Initiation and Management template.

Using Dashboards

Among the many benefits of SharePoint’s Web Part infrastructure is the ability to display information scattered through a SharePoint site or externally in one summary page, called a dashboard. One common business need in shared environments is role-based dashboards, which can provide a view of the relevant information for a role or person. Such dashboards can take advantage of powerful built-in capabilities, such as Web Part filtering and audience targeting to display information based on WHO is viewing the page.

You can also go beyond these capabilities and create separate pages for separate roles, so that you can truly customize not only the information presented, but also the layout, and other aspects of the page. Such customized role-based pages are used extensively through the application templates and can, of course, include filtering and audience targeting as well.
The Lending Library application, for example, makes use of role based dashboard to present information that is most relevant to the user based on the user’s role. The main page of the site has two available pages – one targeted at the library user (the default page) and one for the librarian.

Similarly, the Help Desk application has three dashboard views available on the main page: Service Rep Home, Knowledge Mgr Home, and Service Rep Mgr Home. The user selects the hyperlink most relevant to their role. The Web Part view displayed in the dashboard also filters contents to be displayed relevant to the user. This is done by applying a filter to the view. The following steps illustrate how such a filter is applied to an existing list view:

1. Navigate to the list view page in browser.
2. Locate the drop-down control in the list toolbar and select the view you want to modify, if the view is not already selected.
3. Go to the same drop-down control again and select **Modify this View**.
4. In the **Edit View** page, scroll down to the Filter section. Set the filter value as follows:
   - Show the items when column **Created by** is equal to [Me] **Or** When column **Customer** is equal to [Me].
5. Click **OK**.

The Service Rep Mgr Home page of the Help Desk application, shown in the figure below, features two dashboard Web Parts that summarize service requests by status and priority.
These are data view Web Parts that reflect data contained in a list called Service Requests, which is also a part of the Help Desk Solution site. To best understand how these Web Parts were created, open the site in SharePoint Designer and load the page HelpDeskManager.aspx that contains these Web Parts. View the page in split view mode and you will see that the code behind the Web Parts is a mix of XLST, HTML and CSS markup languages.

Let us examine how the bar graph bars on the Web Part were created. We will take the Overall Service Request Priority Web Part as an example. In the SharePoint Designer menu, selecting Insert, SharePoint Controls, Data View reveals the Data Source Library task pane. This task pane lets us browse through various data sources currently accessible to the site, including lists already in the site. In our case here, we expand the
section labeled **SharePoint Lists** and click on the list name, **Service Requests** to indicate that this list will be the source of our data. In the context menu that pops up, we select **Show Data**. This opens a new panel in the task pane area called **Data Source Details** which reveals all the fields and sample data stored in the list **Service Request**. From this panel, we highlight the desired field names we wish to show in the Web Part, in our case, the **Priority** field, and choose **Insert Selected Field as Single Item View** (see figure below).

Note that our goal here is to show a summarized count of all tasks grouped by priority. This means, we need to list out the options available in the **Priority** field only once then show a count of each priority value in the list alongside. Hence we opt to show the priority values as **Single Item View** as the starting point of our customization. Selecting **Multiple Item View** instead would expose all rows in the list.

![Figure 5: Creating a Dashboard Page in SharePoint Designer](image)

This inserts a data view Web Part in the page at the cursor point. However, the Web Part simply displays one row of data, and not a bar graph summarizing service requests by priority. Fortunately, since the Web Part is rendered using HTML and XSL code, we are
free to customize the code to tailor our needs. The following section highlights some of the key areas of this customization to convert this Web Part into a bar graph.

We know our source data list has three possible priority values: (1) High, (2) Normal, and (3) Low. Hence, we declare an XSLT variable for each value as follows:

```xml
<xsl:variable name="High" select="count(/dsQueryResponse/Rows/Row[normalize-space(@Priority) = 'High'])" />
<xsl:variable name="Normal" select="count(/dsQueryResponse/Rows/Row[normalize-space(@Priority) = 'Normal'])" />
<xsl:variable name="Low" select="count(/dsQueryResponse/Rows/Row[normalize-space(@Priority) = 'Low'])" />
<xsl:variable name="AllTasks" select="count(/dsQueryResponse/Rows/Row)" />
```

Note that the XSL parameter @Priority refers to the name of the data source field. Since the code to graph each of the three priority values is similar, we focus on just graphing the first priority value. To get the percentage value, we define a new variable, percentHigh, to calculate the percentage value of high priority requests:

```xml
<xsl:variable name="percentHigh" select="$High div $AllTasks" />
```

The code that creates the actual bar is an XSL template, as shown below:

```xml
<xsl:template name="ChartRow">
  <xsl:param name="RowName"></xsl:param>
  <xsl:param name="Value"></xsl:param>
  <xsl:param name="PercentValue"></xsl:param>
  <tr>
    <td class="ms-formbody" width="125px" style="vertical-align:middle">
      <xsl:value-of select="$RowName"/>
      <xsl:value-of select="$Value" />
        ddwrt:nbsp-preserve="yes" disable-output-escaping="yes">
        &nbsp;
      </xsl:text>
      <xsl:call-template name="percentformat">
        <xsl:with-param name="percent" select="$PercentValue"/>
      </xsl:call-template>
    </td>
    <td>
      <table width="100%" >
        <tr>
          <td width="{round($PercentValue*100)+1}%" height="15px" class="ms-selected">\n```

http://www.microsoft.com/sharepointapps
Note that the bar itself is a table with two cells. The width of the first cell, which represents the width of the bar in the graph, is determined by the PercentValue variable, represented in the code as:

```
    <td width="{round($PercentValue*100)+1}%" 
```

This cell also has a CSS style class, ms-selected, applied to it. This style class is defined in the core.css style sheet file as follows:

```css
.ms-selected {
    background-position: left top;
    color: #000000;
    background-image: url("/_layouts/images/filedialogselected.gif");
    background-color: #FFE499;
    border-top: 1px solid #FFE499;
    border-bottom: 1px solid #FFE499;
    background-repeat: repeat-x; }
```

The background image, filedialogselected.gif, is what makes the cell (i.e. the bar) appear yellow.

**Other Examples of this Design Pattern**

For another server admin example of using dashboards, see the Call Center template. For a site admin example, see the Business Performance Reporting template.
Building Templates

Application templates in Windows SharePoint Services come in two types: site definitions and site templates. The forty application templates are a mix of these two. Both types of templates are available to the user to choose from in the Create New Site form, and both function very similarly to the end user. However, the method of creating these templates and making them available for use in the site creation form is very different.

Site Definitions

In the most basic sense, a site definition is a collection of XML files, assemblies, and .aspx pages that dictate how the site based on it will be structured and what the underlying application in the site will do. Since the base XML and .aspx files are accessible through the file system, they can be cloned and modified easily to re-create new site definitions, making site definitions highly customizable.

After a site is provisioned (created) out of a site definition, changes made to the site definition files in the file system can still propagate to the provisioned site. However, changing site definition files once sites are provisioned is not supported by Microsoft. If any of the pages in the provisioned site are modified using an external editor such as SharePoint Designer, the page loses its connection with the site definition in the file system. Instead, the page is saved in SharePoint’s database system, and is known as a “customized” or “unghosted” page.

If you need to customize an existing site definition, your approach would be to start with a renamed copy of the site definition. Instead of modifying an existing one. To implement changes to an existing site definition after sites have already been provisioned form it, you need to create and deploy a site definition upgrade solution package. For details on how to do this, see the corresponding articles in the WSS 3.0 SDK.

Site Templates

A site template is a SharePoint site packaged up into a single file that can be unpacked to create new sites with similar structure and content. This means that in order to create a site template, we need to have an existing SharePoint site as a starting point. Any site
customization – page layouts, style sheets, images, master pages, documents, lists, list contents – can be captured in the template.

The process of capturing an existing site into a template can happen right from the site’s administrative tasks (under Site Settings) or from SharePoint Designer. Once created, the site template is stored in the site template gallery of the current site collection. The file has an .stp extension. This file can be downloaded from the site template gallery and migrated to another site collection or to a different server environment. The latter part of this section delves into the details of creating and using a site template.

Using Site Definitions versus Site Templates

As we just discussed above, site templates are actually derived from site definitions. When deciding between creating a new site definition or a site template, consider the following:

- **Complexity of the application** - If your needs are mostly cosmetic, such as layout changes and image manipulations to an existing site, site templates are the way to go. On the other hand, if you wish to add new Web part definitions, or want to use custom code or computed fields, you should create a custom site definition.

- **Your access level in the server** - Do you have access to the entire web server? Or just to a particular site collection? Creating and deploying a site definition requires accessing the server’s file system. If you don’t have this access, you’re limited to creating a site template at the site collection level that you have access to. Bear in mind that this access requirement does not apply to site administrators - once deployed, any one with rights to create a new site will have access to either type of application template.

- **Frequency of future updates/changes** - Changing a site template does not affect sites already created from it—only newer sites created after the changes are affected. Deploying a site definition upgrade solution package affects all sites already created from it.
Guidance for Creating Site Templates and Site Definitions

Creating site templates and site definitions involve varying amounts of complexity. The basic techniques are described in the next sections.

Creating a Site Template

As discussed earlier, a site template is actually a SharePoint site that is packaged for reuse. This packaged file exists in the site template gallery at the site collection level. The site templates in the gallery are available for creating new sites at all child site levels of the site collection. The following steps guide you on creating a new site template from an existing site:

1. Open the existing site in SharePoint Designer and ensure the layout and contents in the site are as you want.
2. Click File, Export, SharePoint Site Template. This takes you to the Site Settings Web page.
3. Provide a file name, title and description for the template.
4. Optionally, check the Include Content option if you want data in lists and document libraries included in the template. You should also check this option if you want to include workflows since workflows are actually contents stored in a document library.
5. Click OK. This creates a template file out of the site with an .stp extension and places the file in the site template gallery of the parent site.

From the site template gallery, you can click on the template name to download the .stp file locally to disk. From there, you can upload the file to another site template gallery.

Creating a Site Definition

Unlike site templates, site definitions are stored in the file system. Each site definition is located in its own folder at %CommonProgramFiles%\Microsoft Shared\web server extensions\12\TEMPLATE\SiteTemplates on the server. The site template folder houses various site elements such as .aspx and .html files as well as the
associated resources such as images and javascript files. ONET.XML – the core site definition file that specifies the various configurations and modules for the site definition, is stored in a subdirectory labeled “XML”.

Site definitions are registered with SharePoint and made available via the WEBTEMP XML files. All the WEBTEMP XML files are located at %CommonProgramFiles%\Microsoft Shared\web server extensions\12\TEMPLATE\<LCID>\XML where <LCID> is the location ID such as 1033. The actual name of the XML file is prefixed with WEBTEMP (for example, WEBTEMPBT.XML).

Basically, creating a new site definition consists of two major steps – setting up the site definition folder, and creating the WEBTEMP XML file that will register the site definition with SharePoint. The first step can be done by cloning an existing site definition folder and changing its contents to suit business requirements. The ONET.XML file contains elements that specify how various parts of the site pages are provisioned, such as navigation bar, document templates and list templates. The Configurations element specifies the lists and modules that are created by default when the site definition is instantiated. The following excerpt is from the ONET.XML of the Bug Tracking site definition, and shows a portion of the Configurations element:

```xml
<Configurations>
  <Configuration ID="0" Name="Default">
    <SiteFeatures>
      <!-- BasicWebParts Feature -->
      <Feature ID="00BFEA71-1C5E-4A24-B310-BA51C3EB7A57" />
      <!-- Three-state Workflow Feature -->
      <Feature ID="FDE5D850-671E-4143-950A-87B473922DC7" />
      <!-- TSA Fields and Content Types -->
      <Feature ID="75A0FEA7-CD50-401e-AF0E-782F3662A299" />
    </SiteFeatures>
    <WebFeatures>
      <!-- TeamCollab Feature -->
      <Feature ID="00BFEA71-4EA5-48D4-A4AD-7EA5C011ABE5" />
      <!-- MobilityRedirect -->
      <Feature ID="F41CC668-37E5-4743-B4A8-74D1DB3FD8A4" />
      <!-- Bug Tracking Categories List -->
      <Feature ID="75A0FEA7-42E8-4527-8313-F63C4C49A7E6" />
      <!-- Bug Tracking Bugs List -->
      <Feature ID="75A0FEA7-2D1E-451a-B445-16BC346D7D8E" />
      <!-- Bug Tracking Bugs List Instance -->
    </WebFeatures>
  </Configuration>
</Configurations>
```
Note that this site definition uses numerous features such as Bugs List, Bug Categories List, Mobility Redirect, etc. Such features are listed under the SiteFeatures and WebFeatures elements with feature GUIDs. The GUIDs are found in the Feature element inside the corresponding Feature.XML file located in the %CommonProgramFiles%\Microsoft Shared\web server extensions\12\TEMPLATE\FEATURES folder. Please refer to Microsoft Windows SharePoint Services 3.0 SDK for more details on customizing ONET.XML.

The second step to creating a custom site definition is to create the file WEBTEMP*.XML file in the %CommonProgramFiles%\Microsoft Shared\web server extensions\12\TEMPLATE\<LCID>\XML folder. The Templates element of this file specifies the configurations that can be used to instantiate the sites created from the site definition. The following excerpt illustrates the format of WEBTEMPbt.XML file, which is the configuration file used by the Bug Tracking site definition:

```xml
<Feature ID="75A0FEA7-2D1E-451a-B445-16BC346D7D8F" />
...
<!-- Post Provisioning Event Handler -->
<Feature ID="75A0FEA7-B0EF-434e-90D6-CE997D970564">
  <Properties>
    <Property Key="ZonedWebPartsUrlList" Value="$Resources:core,lists_Folder;/Bugs/Resolve.aspx,$Resources:core,lists_Folder;/Bugs/Activate.aspx,$Resources:core,lists_Folder;/Bugs/Close.aspx"/>
  </Properties>
</Feature>
</WebFeatures>
</Configuration>
</Configurations>

<Configuration ID="0" Title="Bug Database" Hidden="FALSE" ImageUrl="/_layouts/images/stsprev.png" Description="A site for teams to track bugs in their shared software projects." DisplayCategory="Application Templates"
</Configuration>
</Template>
</Templates>

http://www.microsoft.com/sharepointapps
Note that the Name attribute of Template node above must match the "*" in the file name, WEBTEMP*.XML. Also, note that the DisplayCategory attribute of the Configuration element dictates which tab the configuration will appear on in the Template Selection section of the Create Site Web page. You can create your own tabs by putting your own values in this attribute.

Once the site definition files have been created and placed in appropriate folders in the file system, restart the IIS service. The new site definition will be available for selection in the Select Site template page in SharePoint. Site definition files can also be packaged up as a solution file for easy migration to another SharePoint environment or re-deployment in the same environment. A solution file is a cabinet file with a .WSP extension (for Web solution package). It contains features, Web parts, assemblies, class resources, etc. used by the site definition. You can use the makecab.exe tool create a .WSP file. Please refer to the SharePoint Services 3.0 SDK on how to create a solution file.

Localizing a Site Definition

Localization of a site definition is facilitated through the use of resource files and culture detection. While a typical ASP.NET 2.0 application localization is done during compile time, SharePoint site localization is done during site provisioning. Among the application templates, all 20 site definition templates have been localized in 10 languages. The site definitions themselves are created in a language-neutral fashion while the string literals are stored in resource files (.resx). If you wanted to add localization to your own site definition, or add support for a new language to an existing site definition, you can do so by creating a new resource file. The file goes in the Resources directory located at %CommonProgramFiles%\Microsoft Shared\web server extensions\12\Resources.

This file is essentially an XML file, editable in any text editor. The following steps show how you can create a new localization file for use in a site definition.

1. Browse to the Resources directory mentioned above and locate the existing .resx file that you want to localize to a new language.
2. Clone the file and give it the same name, except for the language-culture name (example tsa-en-us.resx to tsa-es-es.resx).

3. Open the cloned file in a text editor.

4. Set the lcid code on the second line of the file to represent the language. For example, the following code sets the language to Spanish:

   ```xml
   <!-- _lcid="3082" _version="12.0.5006.3000" _dal="1" -->
   ```

5. Scroll down to the part where you see the `data` elements followed by `value` elements in a pair. This is where you pair up the resource name with localized string value. Modify the string values in the `value` element for your locale. The following code example shows a resource string for a `create new customer` action in Spanish:

   ```xml
   <data name="Action_NewCustomer">
     <value>Crear un Nuevo cliente</value>
   </data>
   ```

6. Save the file to the Resources folder with a name like `myCustomResource.es-es.resx` (the actual name depends on the language-culture name for which the resource file is intended). This file is now ready to be referenced by the site definition files in your application.

In order for the localized site definition to show up as an available template in the create site page for that language, add a WEBTEMP file to the appropriate `<LCID>` directory as well. Refer to the earlier section, “Creating a Site Definition” in this document on how to add a new WEBTEMP file.

Localized resource references work differently in `.aspx` files than they do in XML files. For example, references in `.aspx` files are evaluated at runtime, while XML file references are evaluated when the web is instantiated. To access the XML elements in the resource file in a site definition files, the following syntax is used: `$Resources:myCustomResource, DataName`. For example, to consume a resource string from an `.aspx` file, the markup may look something like this:

```xml
<div>
</div>
```
<asp:Label runat="server" Text="<%@Resources:myCustomResource, Action_NewCustomer %>">
</asp:Label>
Summary

Together, Windows SharePoint Services 3.0 and SharePoint Designer 2007 provide the powerful tools you need to build and customize interactive, workflow-enabled applications. Whether customizing one of the forty downloadable application templates or building your own, this paper describes the proven methodology and best practices that even non-developers can use to succeed.

If you haven’t already done so, the first next step to take is to download the application templates and start using them and opening them in SharePoint Designer. Next, use the resources in the Resources section below to assist you as you take on your own application.
Resources

For more information on Windows SharePoint Services 3.0 and Office SharePoint Designer 2007, please refer to the following resources:

Microsoft Windows SharePoint Services 3.0:
http://www.microsoft.com/sharepoint

Application Templates for Windows SharePoint Services:
http://www.microsoft.com/sharepointapps

Microsoft Office SharePoint Designer 2007:
http://office.microsoft.com/sharepointdesigner

Microsoft Office SharePoint Designer 2007 Product Guide

For more developer-focused resources, please refer to the following:

Microsoft Windows SharePoint Services 3.0 SDK

Visual Studio 2005 Extensions for Windows SharePoint Services 3.0

Visual Studio 2005 Extensions for Windows Workflow Foundation

Office SharePoint Server 2007 Developer Portal
http://msdn.microsoft.com/office/server/moss

http://www.microsoft.com/sharepointapps
Developer Posters for Office System


Office 2007 Logical Architecture Diagram