

# 1

## A Framework for Enterprise Applications

The solution you will develop throughout this book is for a fictitious company whose human resources department needs to approve, deny, and report on vacation or holiday time requests by their employees. The solution will be developed using Visual Studio 2008 and implemented as an ASP.NET 3.5 application written in C# with SQL Server 2005 as the back end.

The concept is simple, and the solution is designed to be flexible enough that you can extend it for your own business needs. This chapter defines the requirements for the project and introduces how the solution will be architected. Each chapter details specific requirements and implements a solution in a three-layer architecture: the user interface (UI), the business logic layer (BLL), and the data access layer (DAL).

### Problem

Sue is the vice president of human resources and is currently using a combination of Excel and Word templates to manage employee vacation and personal time requests. It has come to her attention that many requests are not accounted for and are difficult to track down. She has received approval to build a system for the entire organization that will replace her current templates with an enterprise-wide application. As the project's sponsor, Sue has designated Mary as the main point of contact for the IT department in order to gather requirements. The following is an initial conversation that might occur when starting on this project, although you could find similarities with almost any project.

Mary: "My manager put me on this project but I really don't know anything about computers or building systems. They tried to build something a few years ago but it never went anywhere and the developer was fired. Let me explain a little bit about what we do. We have a Word template on the Z drive that everyone must fill out for vacation or holidays requests. When we sent our Excel

## Chapter 1: A Framework for Enterprise Applications

---

spreadsheet that keeps track of everyone's vacation balance to the managers last December, a lot of discrepancies came up. We were missing days that some managers forgot to send to us or requests had been cancelled and we were never notified. We need a database to replace the Word template."

Me: "So you want to automate the process of requesting vacation. I've used that template, so I know a little bit about the process, but I don't know what happens after my manager signs it. Can you tell me how the process is supposed to work?"

Mary: "When employees want to take a vacation or holiday, they are supposed to fill out the form, print it, sign it, and then hand it to their manager for approval and signature. The manager is then supposed to sign it and inter-office mail it to HR, where I check it against my Excel spreadsheet to ensure that the employee has enough days to cover the request. I then subtract the days in my spreadsheet. Also, people who want to take more than two weeks off at a time need to have their manager and a vice president sign off on the request. This really causes problems because the VPs are so busy they rarely send us the form."

Me: "Then you need a system that allows a user to request time off and then have a workflow built into the system for the manager, vice president, and you to approve or deny the request. What the user enters will determine how many levels of approval are needed."

Mary: "Well, yes. I guess we would need to be able to deny the request too, but we usually just throw them out. It probably would be a good idea to keep them around so we could refer to them."

Me: "You mentioned that one problem is that people cancel vacations and don't tell you, so would you like the system to allow users to cancel a request too? If so, I can have the application send you an e-mail notification that the request was cancelled. I could also send any approvers an e-mail when a request is in their queue for approval."

Mary: "Yes, that would be great. I could then adjust the balance in my Excel spreadsheet."

Me: "Could you send me that spreadsheet? I can likely recreate the spreadsheet as a report in the system that you could then export to Excel. This way, any requests or cancellations are tracked in the system and you won't have to maintain the Excel spreadsheet anymore."

Mary: "Yes, I can send it to you, but if I get rid of the Excel spreadsheet, then I need to keep track of the balances somewhere. I guess that would be in the database. Can you do that?"

Me: "I'm sure we can. You need to be able to enter a starting balance and then subtract any request from that balance and add back in any cancellations, correct?"

Mary: "Yes, but we also allow employees to roll over up to five days to the next year. Everything starts over on January first."

Me: "We can do that. Now, what about security? Do you need some people to have access to the reports, and others such as yourself to have access to everything?"

Mary: "I never thought about that but I definitely don't want Bob to be able to enter the starting balances. He should only be able to print reports and nothing else."

Me: "OK, so you need role-based security that allows groups to have either no access, read-only access, or edit access to screens and execute access to reports."

Mary: "I don't know what role-based security is but that sounds right."

Me: "What about an audit trail? Would you like to see any changes to a request or to security in the system? For example, if someone went into the application and gave someone access to a report, the system would capture that and you can print a report of all those changes. Would you like to track that?"

Mary: "I suppose so. I really didn't think about it but my manager would probably like it and those folks from QA that periodically audit our process would probably like to see that too. Could you do that?"

Me: "Yes, that can be done. What about viewing the requests? Do you need different views such as outstanding requests, cancelled requests, requests by manager, or requests by department? You don't have to know all the different variations now, but I could build the system to enable you to view the data in different ways without having to log a support ticket for a custom report. I could even put these views on a home page so you can see right away what needs action."

Mary: "A home page? Yes, I like that idea."

Me: "OK, I have enough information for now. I'll come back to you later to flush out the rest of the requirements. Can we agree that you need a system that has a workflow for processing requests, reporting capabilities, role-based security, a home page, e-mail notifications, querying capabilities, and an audit trail?"

Mary: "I'm not sure what querying capabilities are, but I trust you. Also, it must be easy to use and consistent. I don't want some screens that are green and others that are blue, with all different fonts."

Me: "We'll take care of it. I'll be back soon with more questions about your process but I have enough for now. I look forward to working with you."

Mary: "Thanks for taking the time. I hope you fare better than the last guy did."

In addition to the preceding requirements that came out of the discussion with Mary, there are also a few IT requirements for all applications within the organization. The IT department requires that nothing can be installed on any user's desktop, and all in-house built applications require the use of single sign on. IT should not be involved in administering the application except from the standpoint of supplying a server and providing an SLA for uptime.

Now that Mary has given us a general idea of her requirements, I'll show you how you can transform this into an application — from concept to design to development. I hope you stick around for the rest of the conversation with Mary; it will get fun.

## Design

As stated earlier, the solution consists of an ASP.NET 3.5 web application using the three-layer architecture. The following chapters expand on the general requirements and develop a fully functioning robust enterprise application. The following sections provide a brief outline of each chapter to give you a brief introduction to each topic.

# Chapter 1: A Framework for Enterprise Applications

---

Chapters 2 through 5 review the overall architecture of the framework, and these are the most crucial chapters to read before proceeding. Each chapter is detailed and uses new features in Visual Studio 2008. Chapter 6 and subsequent chapters demonstrate how Mary's requirements are implemented in the application and give you plenty of code you can use in your own applications. The final solution is available for download on the Wrox website, [www.wrox.com](http://www.wrox.com).

## **Chapter 2: The Data Access Layer**

Conceptually, the data access layer is the simplest layer of the traditional three-tiered architecture. Simply put, the data access layer calls stored procedures that reside in the database or executes dynamic SQL against the database. In prior versions of the .NET Framework, as the developer you were responsible for making sure that you knew which type of ADO.NET object should execute a stored procedure and what type of object should be passed back to the business layer. The debate about which object to pass back to the business layer is always heated, but the .NET Framework has given us a tool that could calm those debates.

This chapter introduces the new LINQ to SQL features and utilizes the built-in ORM Designer tool to create a set of entity classes that communicate with the database and are passed back to the business layer. Don't worry, stored procedures still work with LINQ to SQL, and I'll show you how.

## **Chapter 3: The Business Logic Layer**

The business logic layer, aka the middle tier, is where you apply the business rules against any data that should be saved to or deleted from the database. Simply creating a class that mimics the fields in a table is not enough. The business logic layer must protect the integrity of the data in the application by implementing business rules such as required fields, unique fields, limits, and calculations. If rules are broken, then the business logic layer must communicate them back to the caller and not bother the data access layer. This chapter describes the pattern for a set of base classes that encapsulate the business rules for all business objects in the middle tier. It also reviews options for creating lists of business objects and the base classes that are the building blocks for all lists in the application.

## **Chapter 4: The User Interface Layer**

The user interface is the only part of an application that the user sees, so it is important to keep it consistent and easy to navigate. Style sheets and ASP.NET themes can be used to control the fonts and colors of your application, but the developer has complete responsibility for the overall navigation of the system. If you have a Save button on the bottom-left corner in one screen and on the upper-right corner in another, the style sheet won't help you. This chapter reviews style sheets, master pages, nested master pages, and the UI framework that will be used throughout the application. This chapter also builds two custom server-side controls for a dynamic menu that will eventually be integrated with the role-based security built in Chapter 6.

## **Chapter 5: Exception Handling**

This is one area that is often overlooked when building an application, but it's one of the most important. I often see new developers completely forget to add exception handling to an application; or when they do handle exceptions, they aren't handled correctly or don't notify anyone when they occur. It is

always better to rely on the application to let you know when an exception occurs, rather than the user. This not only helps in the debugging process, but also makes you look like a more proactive developer. This chapter reviews the Microsoft Exception Handling Application Block, which I use in all my applications for exception handling. The set of classes it provides include logging and e-mailing capabilities, depending on the type of exception, and it is completely configurable.

## **Chapter 6: Role-Based Security**

Most business applications require role-based security. The business owner of an application usually wants certain groups to have full control of all screens, other groups to have full control of a few screens, and another group to have read-only access to a few screens. This chapter demonstrates a pattern for creating roles in a system and associating capabilities with those roles. Access can be categorized as “none,” “read only,” or “edit,” and can optionally be associated with an item on a menu that should or should not be shown to the user. Users are then associated with one or many roles. This pattern enables business owners to administer their own applications and not rely on IT or you to set up users and roles after you release your application to production.

## **Chapter 7: The Workflow Engine**

Simple workflows such as a request for vacation and a manager’s approval make up a large percentage of corporate applications. Microsoft developed the Windows Workflow Foundation (WWF) to specifically address this issue in order to give developers a foundation they can build upon in their own applications. Learning the WWF would require much more than a chapter in a book, but in this chapter you will learn a pattern you can use in your own applications, one that is simple to incorporate with a few tables and classes. You could use this pattern for applications that have any type of workflow with multiple states. Examples of such applications are approvals for vacation time, travel, network access, or even an issue-tracking system.

## **Chapter 8: Notifications**

Automated notifications can make a big difference in the value of an application, especially in a workflow-driven application. Systems should proactively notify users when a request is sent for their approval or their request is approved, denied, or unattended to for a certain length of time. In this chapter you will learn to build a Windows service application that monitors activity and sends e-mails to appropriate users based on events within the system. This not only exposes users to a notification service pattern, but also introduces building, debugging, and installing Windows services.

## **Chapter 9: Reporting**

The free Crystal Reports for .NET objects that come with Visual Studio are very powerful and can add a professional look and feel to your application. This chapter explains how to incorporate the Crystal Report objects for .NET to display your data in HTML, PDF, Excel, or Word. Again, a middle-tier pattern serves as the “brains” for fetching and manipulating the data before passing it to the user interface, where a Crystal Report object simply takes the data that was delivered and formats it as appropriate. This pattern enables you to reuse middle-tier objects for multiple report views and shows you how to dynamically set groups, or display headers, footers, or detail lines in the report.

### **Chapter 10: The Query Builder Control**

As soon as you give users access to reports or views of data, they are going to want to filter it based on a date, a user, status, and so on. Most of the time users want the same data but filtered in a different way. This chapter explains how to build a query builder server-side control that uses AJAX to enable users to dynamically create their own filters, which can be applied to views or any report that was designed in Chapter 9. This empowers users to extract the data they are interested in and manipulate it in Excel or simply display it in a PDF file.

### **Chapter 11: The Dashboard**

Dashboards are becoming standard in web applications, and the intrinsic web part controls that come with Visual Studio make it simple to create a useful home page that can display graphs, alerts, documents, and more — and put the most important data right in front of the user. This chapter reviews the web part controls and how they work, and builds a home page with a dashboard for the Paid Time Off application, including alerts for users when requests are pending approval.

### **Chapter 12: Auditing**

Auditing is important, if not mandatory, in most corporate applications. With the introduction of the Sarbanes-Oxley Act in 2002 and increasing government regulations in the pharmaceutical industry, auditing is an important feature that you will likely be asked to implement in your own applications. This chapter demonstrates a design pattern for developing a field-by-field audit trail with before and after values and a date\time stamp, and one that captures the user name for any changes to our vacation request application. You could easily adopt this pattern into your own application and save weeks of programming time needed to comply with the strict guidelines required by the government or your own organization.

### **Chapter 13: Code Generation**

A code-generation tool can automatically write the majority of the patterns used in this book. This custom code-generation tool is written in Visual Studio 2008 and enables the user to point to a database and a table to generate the stored procedures, data layer classes, and business classes. This will save developers a tremendous amount of time, as they won't be stuck coding the tedious pieces of the application and can concentrate on the business rules given to them by the user.

## **Solution**

As stated earlier, this solution uses a three-layered architecture: the user interface (UI), the business logic layer (BLL), and the data access layer (DAL). When building a solution in Visual Studio, you can include multiple project types in the same solution, making it easy to manage all three layers in one place. The UI layer consists of a web application and server-side controls. Essentially, anything that the user interacts with directly is considered part of the UI layer. UI developers or designers control the overall look and feel of the site, and typically excel at design, rather than programming. They focus on fonts, colors, and making sure the application meets the company's branding requirements.

# Chapter 1: A Framework for Enterprise Applications

The BLL contains all the business and validation rules that are the real “brains” of the application. For example, suppose you have a screen that allows the user to create a user in the system. The user’s name is required and must be unique when adding a new user. You could put logic in the web form that enforces these two rules. Now suppose a new requirement comes along: The application must import a comma-separated file of user names. The user name is still required and must be unique, so that logic would have to be duplicated in the import logic. If you were to put the logic in a class, then the web form and the import program could call the same code. Anytime new rules are added to the application, you would simply change the logic in the shared class. This makes code maintenance much easier.

The third layer is the DAL, which contains the logic to connect to the database, to call stored procedures, or to execute SQL statements. The DAL developer usually understands the intricacies of SQL and can optimize queries. One reason to create a DAL is that it is easier to migrate from one database type to another without affecting the rest of the application. Suppose you started out using an Access database for a small company. As the company grows, you realize that Access isn’t scalable enough and you need to migrate to SQL Server. If all of your database logic is in the DAL, then all you should need to change is the DAL. Another reason to separate logic into a DAL is because your application may need to support more than one database type. This is a common occurrence for third-party applications that need to support SQL Server and Oracle, depending on their client’s requirements.

Conceptually, the three-layer architecture looks like the diagram shown in Figure 1-1.

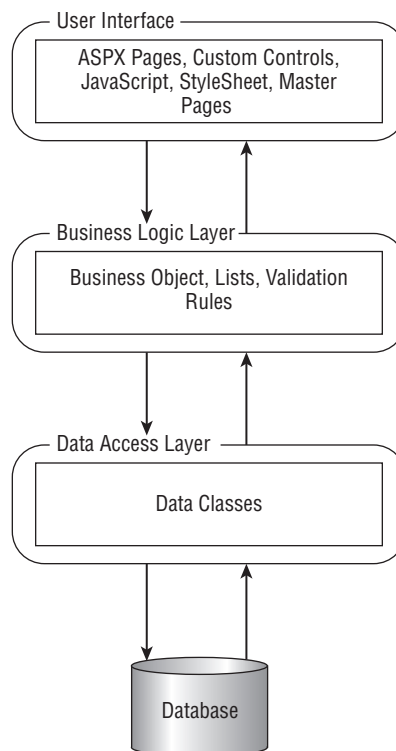


Figure 1-1

# Chapter 1: A Framework for Enterprise Applications

---

The Visual Studio solution for the application created in this book consists of an ASP.NET web application, a class library application for the business logic layer, a class library application for the data layer, and a control library for the custom controls. I also include the Microsoft Exception Handling Application Block, but you could just reference the dlls, rather than the project files.

The fictitious company's name is "Powered By V2." It is a good idea to create all your projects with a namespace that designates your company name in the beginning of the namespace. For this solution, all namespaces will begin with "V2." Follow these steps to build the Visual Studio solution that will be used throughout the rest of this book:

1. Start Visual Studio 2008 and select File ⇨ New Project from the menu.
2. Expand the Other Project Types item and click Visual Studio Solutions.
3. Enter **PaidTimeOffSolution** for the solution name.
4. Enter the location where you want the solution folder to be created in the Location box and click the OK button. By default, this is under the My Documents\Visual Studio 2008\Projects folder. An empty solution is created.

This solution will contain the ASP.NET web application and the class libraries that will be used to implement the Paid Time Off application. Visual Studio 2008 enables you to create a file-based website that uses its own built-in web server, but I used IIS for development because that is the environment where the application will be deployed. I want my development environment to mimic production as much as possible. If you were to add a website project to your solution at this point using http for the location, you would end up with a folder under the `Inetpub/wwwroot` folder. What you really want is for the folder and virtual directory to be created in the solution folder. To do this you first have to create the folder using Windows Explorer:

1. Navigate to the solution folder and create a folder called `PaidTimeOffUI`.
2. Launch IIS and create a virtual directory that points to this new folder and name it the same. In Visual Studio 2008, right-click on the `PaidTimeOff` solution in the Solution Explorer and select Add New WebSite, select ASP.NET Web Site, and change the location to http.
3. Click the Browse button and select Local IIS from the Choose Location dialog box. The virtual directory you just added should appear in the list.
4. Select the `PaidTimeOffUI` virtual directory and click the Open button. Make sure Visual C# is selected for the language and then click the OK button. This will add the `Default.aspx`, `Default.aspx.cs`, and `web.config` files and the `AppData` folder to the `PaidTimeOffUI` folder.

Now you can add the business logic layer project and the data access layer project to the solution just created:

1. Right-click on the `PaidTimeOffSolution` solution again and select Add New Project.
2. Select Class Library from the C# project types.
3. Enter `V2.PaidTimeOffBLL` for the project name. The location should be pointed to the solution folder.
4. Click OK. This will create a folder in the `PaidTimeOffSolution` folder for the BLL project and add the default project files to that folder.

Now repeat the same steps for the data access layer and name the project `V2.PaidTimeOffDAL`.

Next, add the control library by right-clicking on the solution file again and select Add New Project. In the left-hand side of the dialog, click the Web folder under the Visual C# node. Select ASP.NET Server Control as the project type and enter **V2.FrameworkControls** for the name. The location should still be the solution folder. Click the OK button.

Now, add a reference to `V2.PaidTimeOffBLL` to the website. To do this, right-click on the `PaidTimeOffFUI` project and select Add Reference from the menu. Click on the Projects tab and choose the `V2.PaidTimeOffBLL` project. Then add a reference to the `V2.PaidTimeOffDAL` to the `V2.PaidTimeOffBLL` project using the same instructions. Finally, add a reference to `V2.FrameworkControls` to the `PaidTimeOffFUI` project.

Congratulations! You have successfully created a three-layered solution in Visual Studio 2008.

## Summary

This chapter focused on presenting the business problem that will be solved by the application developed throughout this book. Each chapter dissects a part of the problem and discusses the design and the code in great detail, ultimately solving the problem. The goal is to give you a framework that you can use and enhance in your own applications to meet the requirements for your next application. Imagine how nice it would be to be able to start your next project with a complete set of controls, classes, stored procedures, and a data model for role-based security, reporting, dynamic queries, auditing, dynamic menus, and workflow — especially workflow! How many applications out there have approval processes? The workflow pattern defined in this book could be used by many applications for a variety of departments in your company. HR could have a hiring workflow, application development could have an issue-tracking workflow, finance could have a capital expenditure workflow, and the list goes on and on. I am excited to share my experience with you and I look forward to the feedback readers provide about how they have used this framework and improved upon it.

The adventure begins in the next chapter with an exploration of a new feature in Visual Studio 2008 called LINQ to SQL. You'll learn all about how you can incorporate LINQ in your data access layer to avoid a tremendous amount of repetitive code for making calls to the database. In addition, you'll never have to write another custom entity class again to pass data back to the business logic layer. It's all built into LINQ to SQL. Happy coding, my friend!

