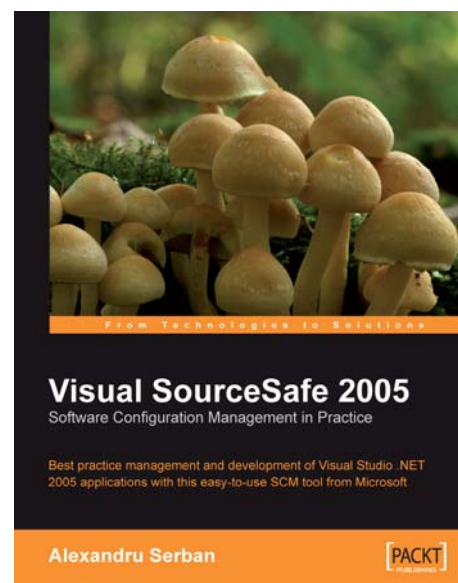




Visual SourceSafe 2005 Software Configuration Management in Practice

Best practice management and development of Visual Studio .NET 2005 applications with this easy-to-use SCM tool from Microsoft

Alexandru Serban



Chapter 3

"Creating a Service-Oriented Application"

In this package, you will find:

A Biography of the authors of the book

A preview chapter from the book, Chapter 3 “Creating a Service-Oriented Application”

A synopsis of the book’s content

Information on where to buy this book

About the Author

Alexandru Serban

Alexandru Serban is the founder and CEO of Unievo, a new software development company.

Previously, he worked as a .NET Software Architect with Softwin and Microsoft on extending Visual Studio for enterprise infrastructure projects. In 2004, he co-authored Pro .NET 1.1 Network Programming, Second Edition.

Alexandru has been driven by the computer revolution ever since he can remember. Now he plans to be a part of it.

When not planning to take over the world, he likes to drive and travel, in the summer to the sea and in the winter to the mountains, where he hits the slopes with his snowboard.

For More Information: <http://www.packtpub.com/visual-sourcesafe-2005/book>

About the Reviewers

Alin Constantin

Alin Constantin graduated from the Faculty of Automatic Control and Computers of the Politehnica University of Bucharest in 1997. He worked at Crinsoft S.R.L., developing hotel management and user interface automation software. Then in 1999 he joined Microsoft. For almost 7 years he focused on developing Visual SourceSafe and source control integration in Visual Studio.

Dragos Brezoi

Dragos Brezoi started programming to create an application for processing and adding extra effects to his guitar's sound. Several years later, he got a Masters Degree in Computer Science from the Politehnica University of Bucharest, and is now researching for a Ph.D. in Advanced Automatics. Dragos worked for several years in the industrial automation field as a programmer for PLC and DSP programming to SCADA, OPC, and DCS solutions. Dragos co-authored GDI+ Custom Controls with Visual C# 2005 (Packt Publishing, 2006), and he currently works for Motorola TTPCom Product Group (Denmark), developing a next-generation embedded software framework.

Jean-Baptiste

Jean-Baptiste Lab discovered computers at the age of 12, when he started writing demos in assembly to impress his friends. After a scientific-oriented basic education, he obtained a B.Sc. in Computer Science at Portsmouth University, UK in 1998, and went on to achieve an M.Sc. in Mathematics and Computing at the University of Besancon, France. Expatriated in Denmark, Jean-Baptiste has been working in the mobile phone industry since 2001, touching various fields spanning from GSM Protocol Stack simulation to software architecture, build systems, and configuration management.

For More Information: <http://www.packtpub.com/visual-sourcesafe-2005/book>

Visual SourceSafe 2005 Software Configuration Management in Practice

Software Configuration Management (SCM) is one of the first skills a serious developer should master, after becoming proficient with his or her development tools of choice. Unfortunately this doesn't always happen because the subject of SCM is not commonly taught in academic or company training.

Although software is not a material thing, as you cannot touch it, smell it, or taste it, building software can be as complex as building physical things such as cars or planes, if not more so. The main difference between the two worlds lies in the limitations you confront. In the world of developing software there are no physical limitations—the only limit is your imagination. However, all this freedom can have a downside. A good TV commercial once stated "Power is nothing without control"—if you do not control it wisely, it may start working against you. When developing software, you need to have a manageable team development effort, track and maintain the history of your projects, sustain parallel development on multiple product versions, fix bugs, and release service packs while further developing the applications.

This is where the concept of Software Configuration Management (SCM) comes into play, dealing among other things with source code versioning, tracking development evolution, building, and releasing. Putting it in simple terms, SCM is about getting the job done safer, faster, and better. While trying to keep the theory to a minimum, this book starts by teaching you what SCM is, why it is important, and what benefits you get by using it, either by working individually or by being part of a team. You will find this part very valuable if you're new to the concept of SCM, because you will be setting your base for understanding what happens in the rest of the book. Then the book concentrates on the Microsoft Visual SourceSafe 2005 SCM tool and the best practices used to manage the development and evolution of Visual Studio .NET 2005 applications. You will learn the theory by going through a journey, in which we will actually develop a new application, starting from designing its specifications and ending with releasing it and completing the Software Development Lifecycle (SDLC). You will learn how the SCM concepts are applied by Visual SourceSafe 2005 by developing Orbital Hotel, a Service-Oriented Application hotel reservation system.

You will learn how to use the team cooperation features in Visual SourceSafe 2005 with the help of John and Mary, two fictional team members who have been assigned to implement various project components.

The end of the book deals with SourceSafe administration tasks. It describes SourceSafe database creation, management, and maintenance, how to secure the database, how to create users and assign user rights, and how to manage projects and project settings.

Additional material on how to customize SourceSafe to suit your development style is available at <http://www.packtpub.com/visual-sourcesafe-2005/book>. You can visit Orbital Hotel online at <http://orbitalhotel.allexandruserban.com/>.

I hope you will find this book a great resource about Visual SourceSafe 2005, and I hope you will enjoy reading it as much as I enjoyed writing it!

For More Information: <http://www.packtpub.com/visual-sourcesafe-2005/book>

What This Book Covers

Chapter 1 teaches you the basic terminology and concepts used in the SCM world, and how SCM integrates in the Software Development Lifecycle.

Chapter 2 introduces you to Microsoft's SCM tool for small and medium teams: Visual SourceSafe 2005. You'll learn what this product is made of, and what new features and improvements it has over the previous versions.

Chapter 3 introduces Orbital Hotel, a hotel-reservation system application, which will be used in the next chapters as a case study for developing Visual Studio applications with SourceSafe. We will see what the best structure for Visual Studio solutions is when working under Source Control.

Chapter 4 discusses the various ways you can add a software project to the SourceSafe database. This is the first step you'll take when starting to develop an application under Source Control.

Chapter 5 covers the Source Control operations used daily in our development activities. We'll set up a new workspace and get the solution from the SourceSafe database. Then, we will add new files to the solution, check them in, examine their history, and get latest versions. We will also explore the team-cooperation models and see what are the differences between them, their advantages and disadvantages, and operations such as item comparison, undoing changes, file merging and pinning, and conflict resolution.

Chapter 6 teaches you how to access the SourceSafe server through the intranet or the Internet, in order to perform the necessary Source Control tasks. If you don't have an internet connection at the remote location, or if the local SourceSafe server is temporarily down, you can work offline, provided you already have the solution files on your remote machine. When a connection to the database becomes available again, you reconnect to the SourceSafe database and synchronize the changes. Depending on the database configuration and the Visual Studio plug-ins you use while reconnecting, there are some scenarios to consider for avoiding data loss. We will examine the possible scenarios that can lead to data loss and see how to avoid such situations.

Chapter 7 teaches you how to manage the software development lifecycle using SourceSafe. In the evolution of software products there are many milestones. We will see how to manage them using SourceSafe so that we can reproduce their specific configurations when needed. We will also talk about the build process and how a periodical build can catch integration problems early on. We will take a brief look at white-box and black-box tests and how they help in ensuring final product quality. Last but not the least, we will see how to maintain multiple product versions to be able to release service packs while continuing development towards the next versions.

Appendix A covers the installation steps for Visual SourceSafe 2005 and the configuration for remote access.

Appendix B describes how to perform SourceSafe database administration tasks such as creating and securing databases, managing database and Windows users, creating shadow folders, and configuring the services for the SourceSafe plug-ins in Visual Studio.

Appendix C discusses how to perform maintenance tasks on SourceSafe databases such as undoing user checkouts, changing the team version control model, locking, archiving, restoring, and running database maintenance tools.

For More Information: <http://www.packtpub.com/visual-sourcesafe-2005/book>

3

Creating a Service-Oriented Application

For the purpose of this book I could have used a simple "Hello World" type application that demonstrated Software Configuration Management with Visual SourceSafe 2005 and Visual Studio .NET 2005. However, I felt the need to give you as much value as possible, given the fact that the development process of building software is rarely so trivial and easy.

So, let's take a more realistic software development scenario. What I am going to build is a room-reservation system for the newly launched Orbital Hotel. As you well know, this is the very first space building, after the International Space Station, used for tourism, allowing people to enjoy a view of our blue planet and stars from their private rooms. OK, OK, the Orbital Hotel doesn't yet exist, but when it does, it must have a room reservation system anyway. Who knows, it might be this one.

I will build a prototype for a hotel reservation system outlining the way Software Configuration Management makes the job easier. Don't worry if you are not fully familiar with the technologies used. The purpose of this application is purely for reference, so you can sit back and relax.

For More Information: <http://www.packtpub.com/visual-sourcesafe-2005/book>

At this point I will use my time machine and get a screenshot for the final application so you can see how it will look like. Or, I can insert the screenshot after it finished. I think the first way seems more reasonable. This is what the public reservation site looks like:

The screenshot shows the Orbital Hotel website. At the top is a banner with a view of Earth from space and the text "Orbital Hotel". Below the banner is a navigation bar with links: Home, New Reservation, My Reservations, and a Login link. The main content area is divided into two sections: "Reservation period" and "Available rooms".

Reservation period

Arrival date:

< December 2006 >						
Su	Mo	Tu	We	Th	Fr	Sa
26	27	28	29	30	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31	1	2	3	4	5	6

Departure date:

< December 2006 >						
Su	Mo	Tu	We	Th	Fr	Sa
26	27	28	29	30	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31	1	2	3	4	5	6

Available rooms

Number	Type	Occup.	Price	Select
1	Regular	4	\$500.00	Details
2	Regular	5	\$600.00	Details
3	Regular	4	\$500.00	Details
4	Regular	4	\$500.00	Details
5	Regular	4	\$500.00	Details
6	Regular	4	\$500.00	Details
7	Regular	4	\$500.00	Details
8	Regular	4	\$500.00	Details
10	Appartment	8	\$1,000.00	Details
11	Appartment	8	\$1,000.00	Details
12	Appartment	8	\$1,000.00	Details

Room for 4 with wonderfull view

Reserve

If you like it, you can download the application from the book's website:
<http://orbitalhotel.alexandruuserban.com>.

Now let's get back to our time and start the development lifecycle on the Orbital Hotel product. The first phase is the specifications phase.

Specifications—Project Architecture

In order to build a software system, we need a list of requirements. What is the purpose of the system? What are the actions performed by the system and against the system? Who will use the system and how? The answers to these questions will let us identify the main parts of the system and the way these parts work together.

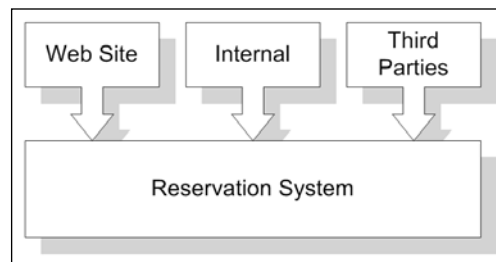
System Requirements

Let's take a look at the Orbital Hotel's reservation system's requirements. The purpose of the reservation system is to allow guests to make room reservations. There are several room types each having a number, occupancy, price, availability, description, and image. The reservations can be made by using the hotel's internet website, through the websites of travel agencies (third parties), or by making phone calls to the hotel's client service. Reservations can be also made by internal client service staff who receive phone calls from guests.

When guests use the hotel's website, they will create a user with a username and password and input their personal details such as first name, last name, address, city, zip code, state, country, phone, email address, and card number. Then they will choose a room and complete the reservation details such as arrival date, the number of nights they will be staying and the number of adults, teenagers, children, and pets. They will also be able to cancel their reservation.

When making a reservation over the phone, a guest will provide the same personal information and reservation details to the hotel's client-service staff. The staff will create a reservation for the guest using an internal application. The staff members will also authenticate using a username and password.

Travel agencies and other third parties must also be able to make hotel reservations.



Taking a big picture about the type of system we are going to build, what we need is an application design that will be as flexible as possible. It should provide us with a variety of options like reservations through phone calls, personal or third-party websites, smart devices like PDAs or cell phones, and so on. This is where we gather the specifications and plan the system architecture. In this phase we have to consider as many aspects as we can, based on our requirements and specifications.

So, let's see what the main existing application architectures are, and see what application architecture fits our requirements.

Application Architectures

The computer and computer programming history is a very short one in comparison with that of other industries. Although it is short, it has evolved and continues to evolve very rapidly, changing the way we live. Taking into account the architectures used at the beginning of computer programming, we can see an evolution from the single, powerful, fault-tolerant, expensive super mainframe computer applications, towards multiple, distributed, less expensive smaller machine applications, the personal computers.

During this evolution, three main application architectures can be identified:

- Compact application architecture
- Component application architecture
- **Service-Oriented Architecture (SOA)**

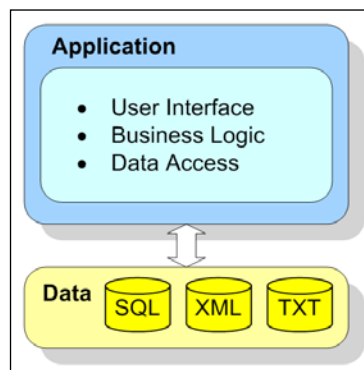
We are going to take a brief look at these application architectures and outline their characteristics.

Compact Application Architecture

During application development for the single mainframe, there was no clear separation between application layers and no reusable components were used. All the data access, business logic, and user interface-specific code were contained in a single executable program.

This traditional *compact architecture* was used because the mainframe computers had specific *proprietary* programming languages and formats for accessing and manipulating the data.

All the data access-specific procedures as well as the business logic and business rules code are written in this programming language. At the surface, a user interface is presented to the user for data visualisation and manipulation.



This application architecture works for applications that do not need data input from multiple sources and can be easily developed by a single programmer. However, this approach has several major disadvantages when it comes to building large-scale systems:

- Application components cannot be reused in other applications because they are *tightly coupled* and dependent on one another. Tight coupling means that in order for a piece of code to use another piece of code, it must have intimate knowledge about its implementation details.
- Being tightly coupled, a change to one component can affect the functionality of another, making debugging and maintenance a difficult task.
- The application is actually a black box; no one, except the main developer, knows what it is in there.
- Applying security is another problem because the user interface cannot be separated from the business logic components using security-specific mechanisms like authentication and authorization.
- Application integration is affected because the code is platform dependent. Integration between two such applications requires special and specific coding and can be difficult to maintain.
- Scalability issues are considered when the system grows and need to be scaled across several machines. Using this application architecture, scalability is not possible as you can't separate different application parts across different physical boundaries because of the tight coupling.

To address the issues with the compact application architecture, the *component-based application architecture* was developed.

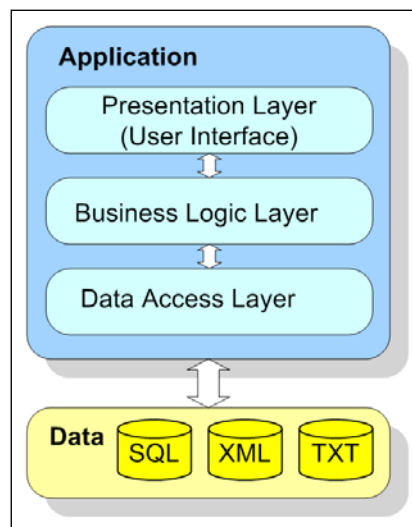
Component Application Architecture

In the component application architecture, the application's functionality is defined using *components*. A component is like a black box, a software unit that encapsulates data and code and provides at the surface a set of well-defined interfaces used by other components. Since a component only needs to support a well-defined set of interfaces, it can change the inner implementation details without affecting other components that use its external interfaces. Components that export the same interfaces can be interchanged, allowing them to be *reused* and *tight coupling* to be eliminated. This makes them *loosely coupled* because they don't need to know internal implementation details of one another.

This separation of application functionality using components allows the distribution of development tasks across several developers and makes the overall application more maintainable and scalable. In the Windows environment, the most used component application architecture is the **Component Object Model (COM)**.

Typically, components are grouped into logical layers. For example, an application uses the data access layer to access the different data sources, the business logic layer to process the data according to the business rules, and the presentation layer also known as the user interface layer to present the data to end users.

Using well-defined application layers allows for a *modular design*, component *decoupling*, and therefore the possibility for component reuse.



Data Access Layer

This architecture forms a chain of layers that communicate with one another. The base is the data access layer, which is responsible for querying, retrieving, and updating the data from and to different data sources while providing a uniform data view to the layers above.

Business Layer

Above the data access layer is the business logic layer. The business logic layer uses the uniform data provided by the data access layer and processes it according to the business rules it contains. The business logic layer doesn't need to know from what source and how the data was obtained. Its purpose is only data manipulation and processing.

Presentation Layer

At the top of the chain is the presentation layer or the user interface layer. Its purpose is to present the data processed by the business logic layer to end users and to receive input and commands from these end users. The presentation layer will propagate these commands down the chain to the business layer for processing.

Characteristics

The component application architecture solves many software problems and it has been used extensively in the past. But because software evolves continuously, new requirements introduce new challenges.

Let's suppose we have several applications on different platforms, each incorporating its presentation layer, business logic layer, and data access layer. We want to integrate them into a bigger distributed system, a system that spans across several heterogeneous environments. At some point, one application will need to access the data existing in another application. While components can work well in a homogenous environment on the same platform, for example COM in the Windows environment, problems appear in components working across several platforms. For example, it is very difficult for a COM component to be used from a Java application or vice-versa, mainly because they don't speak the same language.

Integration between two or more applications running on different platforms would require a middle component-dependent intercommunication layer that is expensive, difficult to build, and reintroduces *tight coupling* between systems, which is what we tried to avoid in the first place. Avoiding building this intercommunication layer would require that the data exchange between these applications be done by a person who will read the necessary data from the source application and write it into the target application.

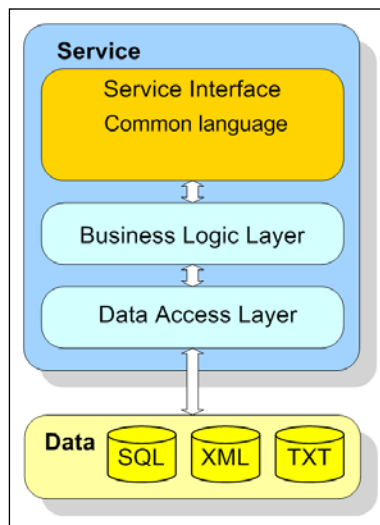
We need to integrate these systems, and maintain the loose coupling between them. What we need to do, is make these components understand each other, making them to speak the same language. This is where the concept of *services* and **Service-Oriented Architecture (SOA)** comes into play.

Service-Oriented Architecture

SOA describes an information technology architecture that enables distributed computing environments with many different types of computing platforms and applications.

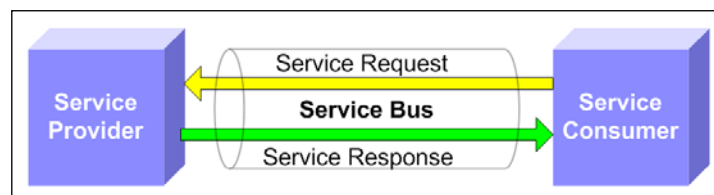
To enable distributed computing environments, SOA defines the concept of services. A **service** is a well-defined, self-contained unit of functionality, independent of the state of other services.

Let's see how services can be used to create distributed applications, integrate component-based applications, and make them communicate with each other. We keep our data access layer and business logic layer as they are, but we completely *decouple the presentation layer* so we can change it later without affecting the other layers. In order to expose the functionality of the business logic layer, we wrap it in a *service interface*. The service interface wraps the business logic layer components offering a *point of access* for any process that needs to access the business logic, whose functionality has now become a *service*.



Service-oriented architecture is basically a collection of services that communicate with each other. The communication can involve either simple data passing or it can involve two or more services coordinating some activity. Whatever the required functionality may be, we have now separated the functionality of applications into specific units, the services that we use to construct flexible, distributed applications.

Typically services reside on different machines. They are exposed to the outside world by service interfaces. A service provider provides its functionality using the service interfaces that are used or consumed by the service consumers. A service consumer sends a service request to a service interface and receives a service response. The following figure represents a typical service consumer-service provider request.



A service can be a service provider and a service consumer at the same time as it can consume other services. They communicate using a communication medium like a local area network for internal services or the Internet for external services. This communication medium is called a **service bus**.

We saw that services don't have a presentation layer as we've decoupled the presentation layer from the rest. This presents an advantage because we can now use any platform able to understand and consume the service to build a presentation layer. The service interface has to provide a standard and open way of communication, a common language that both service providers and service consumers can understand, regardless of the machine type they are deployed on, their physical location, and the language in which they are written.

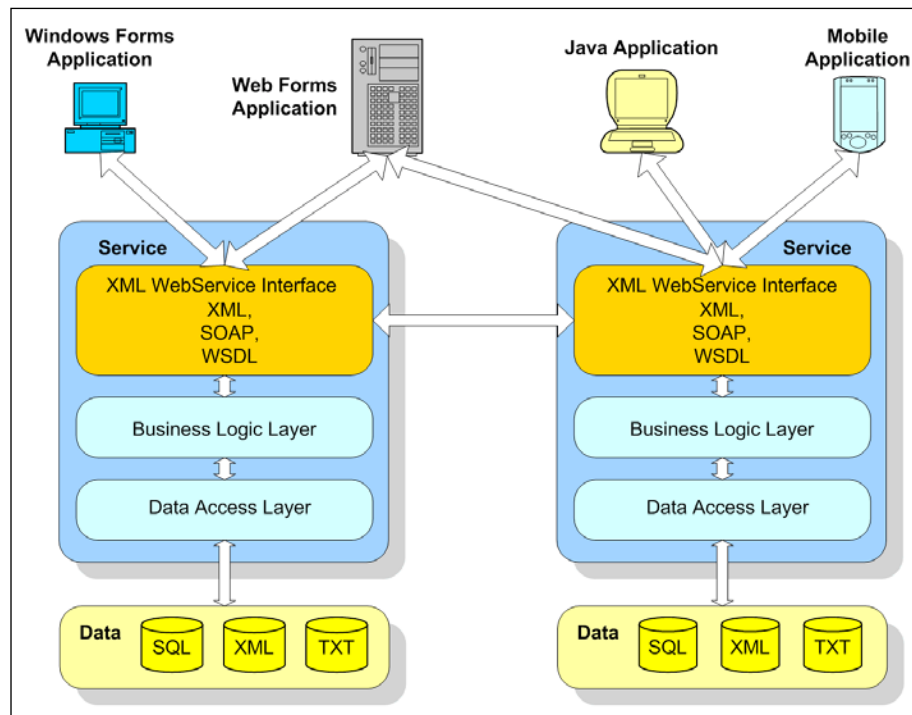
XML Web Services

In today's world, the communication standard used to connect services is achieved using **web services**. Web services are small, reusable applications that help computers with many different operating system platforms work together by exchanging messages. Web services are based on industry protocols that include **XML (Extensible Markup Language)**, **SOAP (Simple Object Access Protocol)**, and **WSDL (Web Services Description Language)**.

These protocols help computers work together across platforms and programming languages enabling data exchange between otherwise unconnected sources:

- *Client-to-Client*: Devices, also called *smart clients*, can host and consume XML web services, allowing data sharing anywhere, anytime.
- *Client-to-Server*: A server application can share data with desktop or mobile devices using XML web services over the Internet.
- *Server-to-Server*: Independent server applications can use XML web services as a common interface to share and exchange data.
- *Service-to-Service*: Systems that work together to deliver complex data processing can be created using XML web services.

The following figure shows an example of services exposed using web services, which deliver their functionality to a wide variety of platforms and applications.



Service-oriented architecture provides us with the maximum flexibility in building applications. Individual services define specific application functions and interact with one another to provide the entire business process functionality.

Using service-oriented architecture provides many benefits such as:

- **Encapsulation:** Just as an object encapsulates its internal implementation details inside while providing public methods to external objects, services encapsulate their internal complexity and implementation from the service consumers who don't have to know the internal details.
- **Mobility:** As services are independent and encapsulated, they can be deployed in any location. Since they are using the same standard communication language, they are accessed in the same way irrespective of their physical location or implementation details.
- **Parallel development:** A service-oriented application is built using several service layers and clients. These application components can be built in parallel by developers specialized in specific layer functionality, speeding up the development process.

- **Platform independence:** Service providers and service consumers can be written in any language and deployed on any platform, as long as they can speak the standard communication language.
- **Security:** More security can be added to a service-oriented application at the service interface layer. Different application components require different security levels. The security can be enforced by using firewalls configured to allow access only to the required service providers only by the required service consumers. In addition, by using **Web Service Enhancements (WSE)**, authentication, authorization, and encryption can be easily added.
- **Reusability:** Once a service is constructed and deployed, it can be used by any other service consumer without problems related to platform integration and interoperability.

Choosing an Application Architecture

Now that we have seen the existing application architectures, we must choose one that meets our project requirements. As you may have guessed by this point, the best application architecture we can use for our project is a Service-Oriented Architecture (SOA). The SOA allows us to build a distributed system, a system that has great flexibility and interoperability with other systems on other platforms. This will allow us to build the business logic functions and expose them as services that will be used by higher functionality layers.

Choosing an Application Platform

After choosing our application architecture, we must choose a platform capable of building and supporting it. For the purpose of our system we will choose the Microsoft .NET Framework platform and build the system using Microsoft Visual Studio.NET 2005 and Microsoft SQL Server as the back-end database for storing the data.

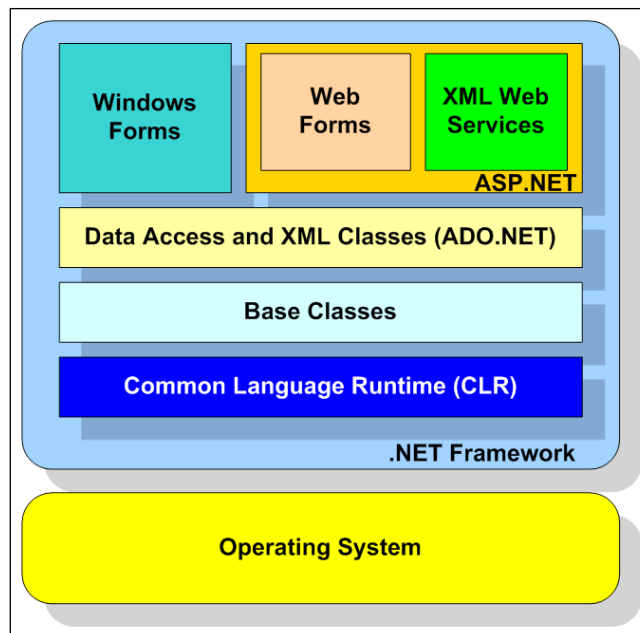
Microsoft .NET Framework

From a Service-Oriented Architecture point of view, the .NET Framework is the Microsoft strategy for connecting systems, information, and devices through software such as web services. .NET technology provides the capability to quickly build, deploy, manage, and use connected, security-enhanced solutions through the use of web services.

Intrinsically, the .NET Framework is an environment for development and execution that allows different programming languages and libraries to work together to create Windows-based applications that are easier to build, manage, deploy, and integrate with other networked systems.

The .NET core components are:

- **The Common Language Runtime (CLR):** A language-neutral development and execution environment that provides a consistent model and services to manage application execution that includes:
 - Support for different programming languages: A variety of over 20 programming languages that target the CLR, such as C#, VB.NET, and J#, can be used to develop applications.
 - Support for libraries developed in different languages: Libraries developed in different languages integrate seamlessly, making application development faster and easier.
 - Support for different platforms: .NET applications are not tied to a single platform and can be executed on any platform that supports the CLR.
 - Enhanced security: The .NET Code Access Security model provides a managed environment for application execution and security.
 - Automatic resource management: The CLR automatically handles process, memory, and thread management, enabling developers to focus on the core business logic code.
- **The Framework Class Libraries (FCL):** An object-oriented library of classes that extends a wide range of functionality including:
 - Support for basic operations: Input/output and string management, standard network protocols, and network standards such as TCP/IP, XML, SOAP, and HTTP are supported natively to allow basic operations and system connections.
 - Support for data access and data manipulation: The FCL includes a range of data access and data manipulation classes forming the ADO.NET technology that natively supports XML and data environments such as SQL Server and Oracle.
 - Support for desktop applications: Rich desktop and mobile client applications can be easily created using the Windows Forms technology.
 - Support for web applications: Thin web clients, websites, and web services can be created using web forms and XML web services technologies that form ASP.NET.



In the planning phase we've gathered the project requirements and specifications and we've also chosen an application architecture. The next phase is the design phase.

Designing the System

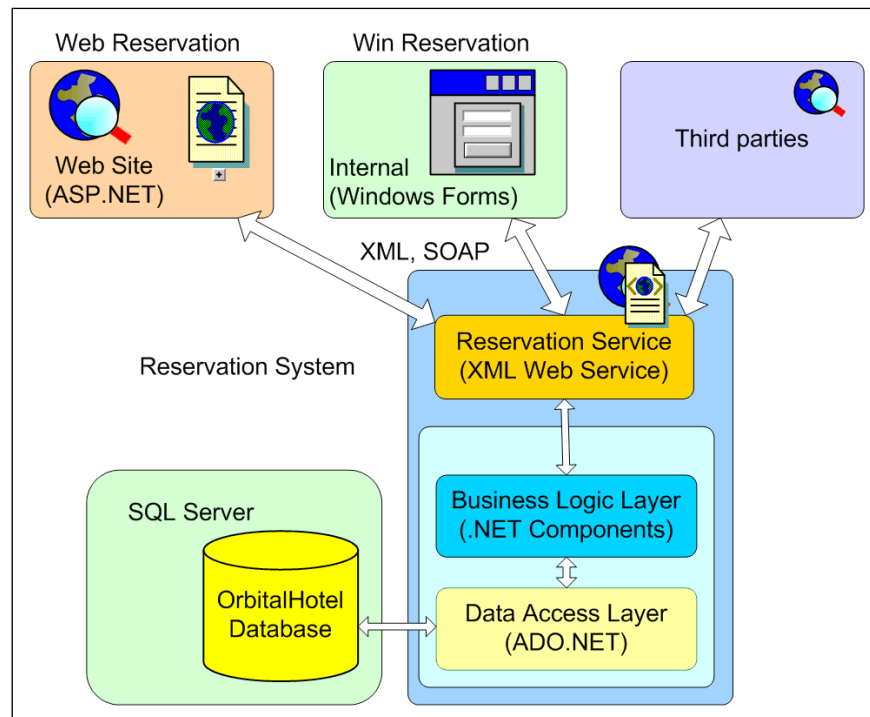
In the design phase, we will create an application design based on the application architecture, project requirements, and specifications. Gathering all the information needed to design the system is a difficult task, but the most important step is to start writing down the first idea.

System Structure

The system will be composed from the following main component categories:

- *Core* components (Data Access Layer, Business Logic Layer) forming the middle-tier component layers.
- *Web service* components (XML Web service) forming the Service Interface layer.
- *Website* components (ASP.NET website) forming the front-end *WebReservation* application, a web presentation layer.
- *Windows Application* components (Windows Forms Application) forming the *WinReservation* application, a Windows presentation layer.

The following figure illustrates the overall system structure, outlining each system component:



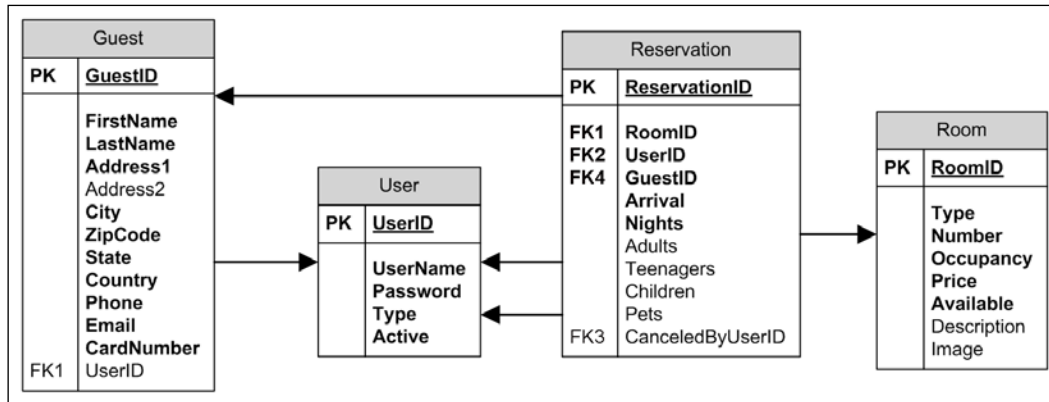
As we saw earlier, one major advantage of a service-oriented application is the decoupling of the presentation layer from the business logic layer. This allows for the business logic layer being exposed as a web service to be used by other third parties to integrate its functionality into their business process.

Database Structure

The back-end database is hosted by a Microsoft SQL Server system. According to the project specifications the internal database structure will be composed of the following database tables:

- User (Contains the user accounts)
- Guest (Contains the personal details of the guests)
- Room (Contains the details of each of the hotel's rooms)
- Reservation (Contains the details of the reservation made by each user)

The following figure illustrates these tables and the relations between them. The bold fields are mandatory (not NULL).



The User table contains the following rows:

UserID	The user identifier used as the primary key of the User table.
UserName	The user name.
Password	The user password.
Type	The user type, such as Guest, Internal, or ThirdParty.
Active	User accounts can be active or they can be deactivated according to business rules.

The Guest table contains the following rows:

GuestID	The guest identifier used as the primary key of the Guest table.
FirstName	The first name of the guest.
LastName	The last name of the guest.
Address1	The first line of the address of the guest.
Address2	The second line of the address of the guest. This field can have a null value.
City	The guest's city.
ZipCode	The guest's zip code.
State	The guest's state.
Country	The guest's country.

Phone	The guest's phone number.
Email	The guest's email address used for reservation confirmation.
CardNumber	The guest's credit or debit card number used to bill the guest.
UserID	Contains the guest's user identifier, if the guest has a user account. This field can have a null value if the guest's data is entered by a hotel staff member, and is a foreign key of the User table.

The Room table contains the following rows:

RoomID	The room identifier, used as a primary key of the Room table.
Type	The room type such as Single, Double, etc.
Number	The room number.
Occupancy	The room occupancy, containing the number of guests it can accommodate.
Price	The price of the room.
Available	The room availability. Some rooms may not be available due to repairs or other events.
Description	The room description. This field can have a null value.
Image	The room image. This field can have a null value.

The Reservation table contains the following columns:

ReservationID	The reservation identifier, used as a primary key of the Reservation table.
RoomID	The reserved room identifier, a foreign key of the Room table.
UserID	The identifier of the user that made the reservation, a foreign key of the User table. This is required because hotel staff can make reservations for guests too.
GuestID	The identifier of the guest for whom the reservation was made, a foreign key of the Guest table.
Arrival	The arrival date.
Nights	The number of nights for the reservation
Adults	The number of adults.
Teenagers	The number of teenagers.
Children	The number of children.
Pets	The number of pets.
CanceledByUserID	The identifier of the user that canceled the reservation, a foreign key of the User table.

After creating the back-end database structure we create the Visual Studio .NET solution structure for our project.

Visual Studio .NET Projects and Solutions

Before we start organizing the source code structure of our system, it is important to understand how Visual Studio .NET organizes and manages source code locally and by using a source control provider such as Visual SourceSafe.

Visual Studio .NET Projects

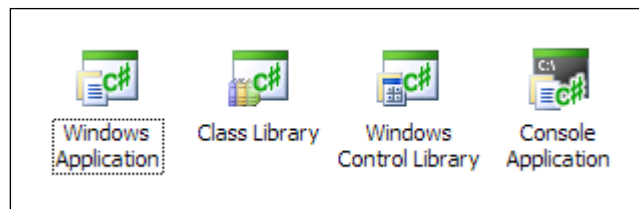
Visual Studio .NET uses projects to organize and manage the configuration, settings, and the build process that generates a .NET assembly. Assemblies are collections of types and resources that are built to work together forming a logical unit of functionality. They are the building blocks of any .NET application and form a fundamental unit of deployment, version control, reuse, and security.

Depending upon the project language, Visual Studio .NET projects have different file extensions such as .csproj for C# or .vbproj for Visual Basic .NET. Although there are many project types such as class libraries, console applications, Windows applications, websites, or web services, projects fall into two main project categories:

- Non-web projects
- Web projects

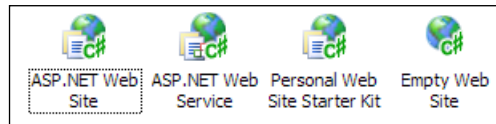
Non-Web Projects

Non-web projects, sometimes called Windows projects, are application projects that do not necessarily deliver content to web browsers and do not need a web server to run. These include projects like Windows applications, console applications, and Windows services. Non-web projects can run from any system folder without any additional configuration.




Web Projects

Web projects are application projects that deliver their content to web browsers and, in order to run, need a web server. In this category are websites and web services. Web projects usually live inside a web server's virtual folder and are usually referred to using a URL path.

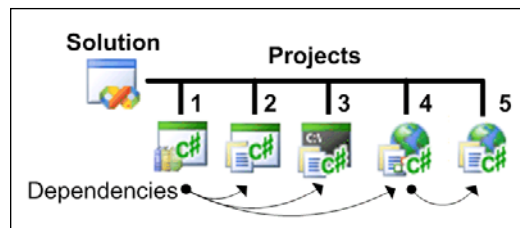


Visual Studio .NET Solutions

Because larger applications are built using multiple projects, Visual Studio .NET uses *solutions* to group projects together. Solution files have a `.sln` extension. Apart from grouping projects together, solutions maintain *project dependencies* controlling the order-dependent projects that are built.

[ A project can be part of one or more solutions but solutions can't be part of other solutions.]

The following figure shows a solution including a class library project (1), a Windows application (2), a console application (3), a web service (4), and a Windows application (5) project:



Projects will be built starting with the ones that are not dependent on other projects and continue on the dependency chain until all the projects are built.

Project 1 is not dependent on any other project. Projects 2, 3, and 4 are dependent on project 1. Project 5 is dependent on project 4. The solution maintains the dependency between projects so if we, for example, build project 5, project 1 and project 4 will be built first and project 5 will be built last. This ensures project 5 is built against the latest versions of the other referenced projects.

The solution that contains all the projects is called the *master* solution. The master solution ensures the final application is built by rebuilding the latest version of each individual project.

Partitioning Solutions and Projects

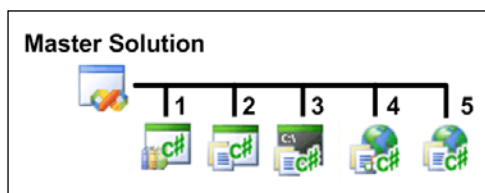
When dealing with solutions that contain a large number of projects developed by several teams, having a single master solution is not always the best option for development. In our example, the teams that work on project 1 don't need to have the other projects in their solution. Likewise, the teams that develop the Windows applications don't need to have the web applications in their solution. For this purpose, solutions and projects can be partitioned.

There are three main models for solution and project partitioning:

- Single solution
- Partitioned single solution
- Multi-solution

Single Solution

Using a single solution model is the easiest and the recommended way to contain all the projects in the application. The projects reference each other directly using project references instead of a file reference to a project assembly already built outside the system. This avoids assembly versioning because a referenced project is automatically rebuilt by Visual Studio .NET, if changes are made to it. Changing between solution configurations (release, debug) and application rebuilding is also very simple.

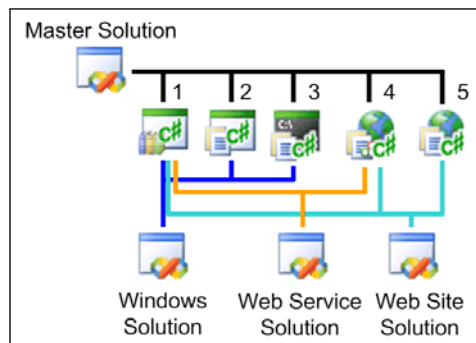


However, this model has its disadvantages when it comes to a large number of projects. For example, if we want to work on a single project in the solution we are forced to get the source code for all the projects. Minor changes to a base project trigger the rebuilding of all dependent projects. Unnecessary rebuilds for solutions containing many projects can be very time consuming.

Partitioned Solution

For larger applications where the master solution has many projects, we can eliminate the disadvantages associated with the single master solution by partitioning the projects using subsolutions and creating a partitioned solution.

Each subsolution contains the projects associated with a logical application sub-system. The following figure shows how related projects can be grouped into subsolutions:



The Windows solution contains the projects 1, 2, and 3, associated with the development of the Windows sub-system. The web service solution contains projects 1 and 4, associated with the development of the web service sub-system. The website solution contains projects 1, 4, and 5, associated with the development of the website sub-system. Note that project 1 is a part of three solutions while project 4 is a part of two solutions.

The master solution is used to build the entire system, containing all the application projects.

Each subsolution contains logically grouped projects that reference each other using project references. This presents all the advantages of project references and allows development on individual subsystems without the need to have all the projects in the solution.

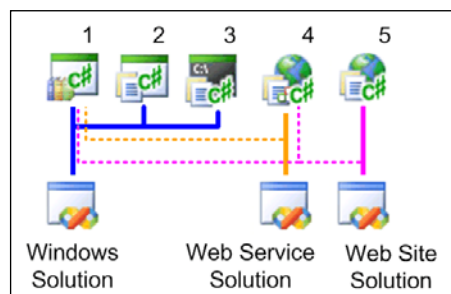
Note that we cannot group projects randomly. Projects must be grouped following the reference chain. In order to work on a top-level project we must include in the subsolution all the project referenced by the top-level project. For example, when working on project 5, we need to have project 4 and project 1 in the solution too, in order to use project references. But when working on project 1, which does not reference any other projects, we can create a subsolution containing only project 1.

Also, when adding new projects to the master solution, we must manually add them to the other subsolutions that reference these projects.

Multi-Solution

The multi-solution model is similar to the partitioned solution. The difference is that there is no master solution and projects outside a solution are referenced using external file references.

The following figure shows the multi-solution model:



The Windows solution uses project reference between the contained projects.

However, the web service solution containing only project 4, references project 1 using an external file reference, a reference to an already built assembly for project 1.

The same case applies for the website solution that contains only project 5. Project 1 and 4 are referenced using external file references to already built assemblies for project 1 and 4.

With this model, a project is included in only one solution. Adding or removing projects is easier as we don't have to add or remove them from every solution they are part of.

Projects can be grouped in any way, unrelated to the way projects reference one another. This allows for a system subdivision according to any criteria.

This model has its disadvantages too. Solutions use file reference instead of project reference. File references do not set up build dependencies and build order. The build process must be *handled separately* and adds more complexity when building the solution.

Best Practices for the Solution Physical Structure

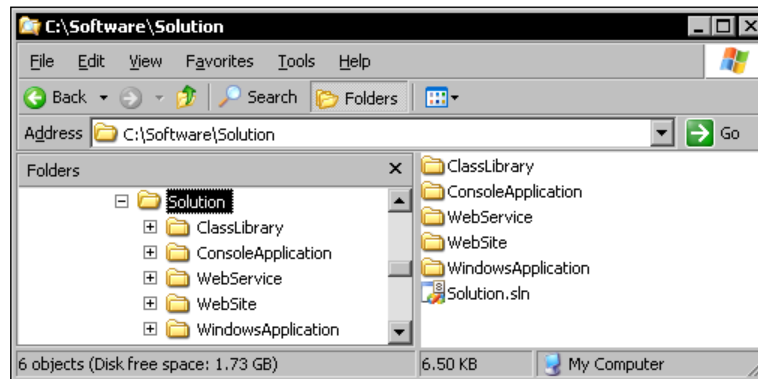
To ensure that the development and build processes work effectively in a team environment, it's essential to start with a correct solution folder structure that is consistent across all the development workstations and build servers.

Failure to create a well designed solution folder structure will result in problems in the later addition to a source control provider repository and the recreation on other machines.

Hierarchical Folder Structure

The best way to organize a solution in order to be consistent across source control repositories, workstations, and servers is to use a hierarchical folder structure, where the solution is the *root* and the projects are *sub-nodes*. This structure ensures there is a symmetrical correspondence between the physical solution structure in the workspaces and the structure in the source control repository.

The solution is created in a root folder and the individual projects are created in subfolders below the root folder. The **Solution** folder contains the master solution file `Solution.sln`. The individual project folders are under the **Solution** folder.



You can see that the web projects (**WebService** and **WebSite**) too are in this hierarchical structure and not under the default `C:\Inetpub\wwwroot` folder, which is the default root folder for Internet Information Services (IIS) web server. Let's see how this configuration is created.

Creating Web Projects

To maintain the hierarchical solution structure we must create all the projects in a specific solution under the solution's folder. For non-web projects this is achieved by default by Visual Studio .NET when creating such projects.

Web projects, however, can be created in multiple ways using different locations such as:

- File system
- Local IIS
- FTP site
- Remote site (using Front Page Server Extensions)



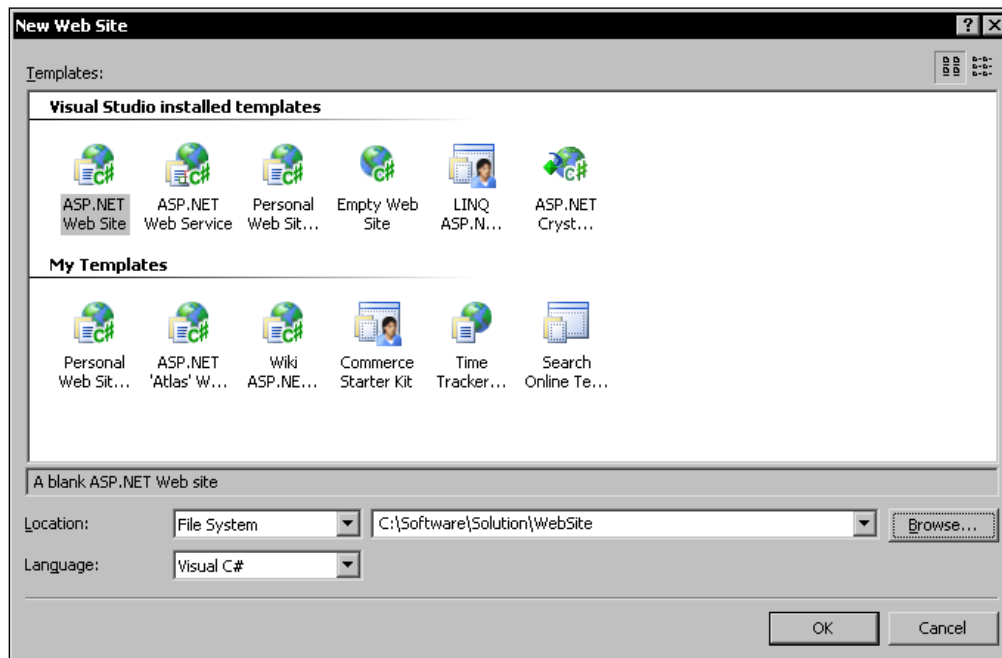
Visual Studio *does not* support source control integration when using FTP sites.

Source control integration for remote sites is supported, but this requires either installing Visual SourceSafe on the remote machine or using light locking mechanisms provided by FrontPage. Also additional configuration is required on the remote machine after creating the web projects.

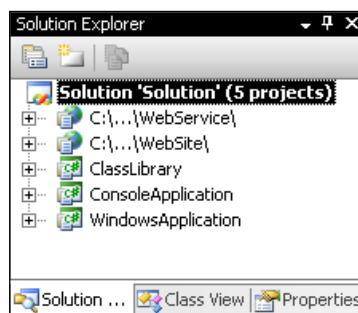
The best way to work with source-controlled web projects is to use one of the first two options, *File System* and *Local IIS*. A new web project can be added using the **Add New Web Site** dialog window by selecting **File | Add | New Web Site**. The **Add New Web Site** dialog window requests the type of the web project and its location.

Using the File System

The file system location is selected in the dialog window's bottom **Location** area using the left combo box. The file system path is specified using the right combo box and must be under the solution's folder:



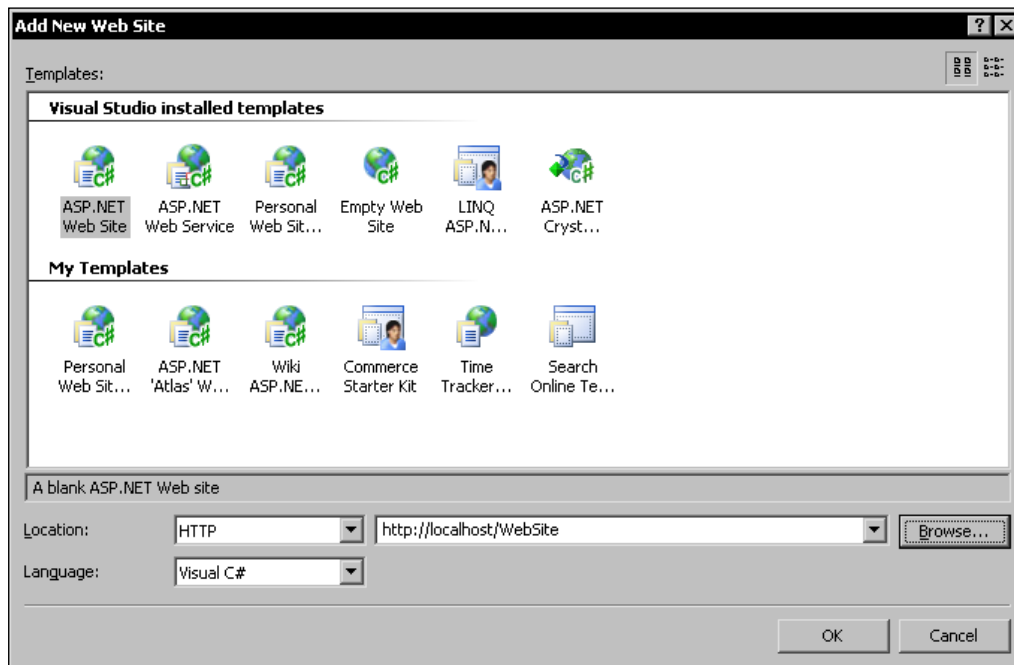
In the **Solution Explorer** window of Visual Studio .NET we can see the following solution structure:



Note that the names of the web projects represent the physical path on the local file system.

Using Local IIS

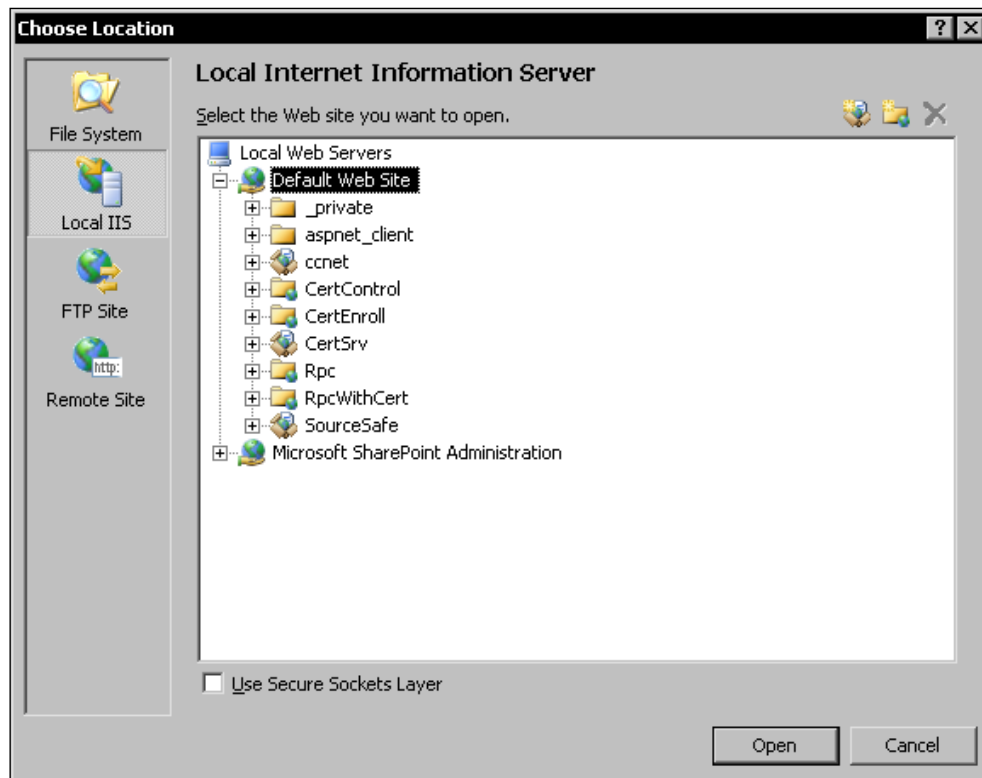
The local IIS location is selected in the dialog window's bottom **Location** area using the left combo box. The HTTP path is specified using the right combo box or clicking **Browse**:



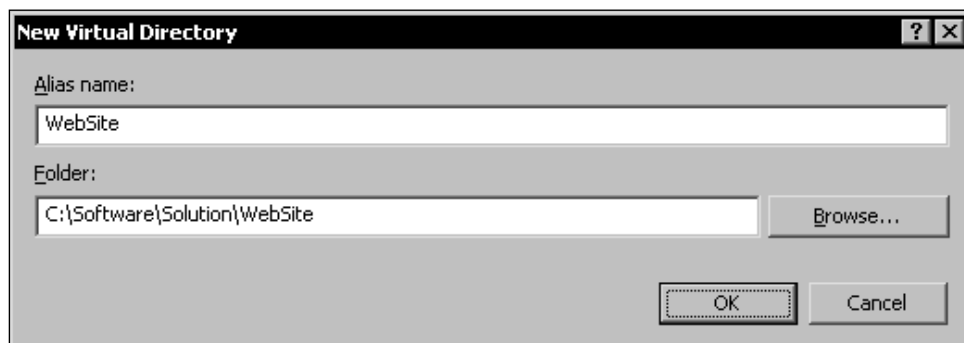
Note, however, that if we click **OK** in this state, the physical **WebSite** folder will be created under the **C:\Inteptub\wwwroot** folder, breaking the solution folder structure.

To create the folder hierarchically under the solution folder we must take additional steps and create a virtual folder that will map the physical folder in our folder structure.

After clicking on the **Browse** button another dialog window presents us the view of the local IIS **Default Web Site** structure:



We create a virtual folder by selecting the folder node under which we want to create the virtual folder, typically the **Default Web Site** node, and clicking the second button in the top right window area. Another dialog window asks us for an **Alias name** and a **Folder** path. We must specify a folder path *under* the solution's folder.

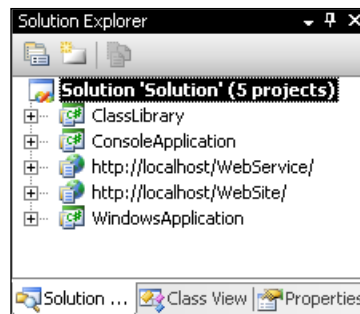




The **Alias name** and the **Folder** name must match.

We then click the **OK** button of the **New Virtual Directory** dialog and select the newly created virtual directory in the **Choose Location** dialog. To finish the operation we click the **Open** button of the **Add New Web Site** dialog window. The web project will be created in the virtual folder under the solution folder.

In the **Solution Explorer** window of Visual Studio .NET we can see this solution structure:



Note that the names of the web projects represent their URLs in the local IIS server.

File System versus Local IIS

The main difference between the file system location and the local IIS location is the web server used for running the web project when developing the project.

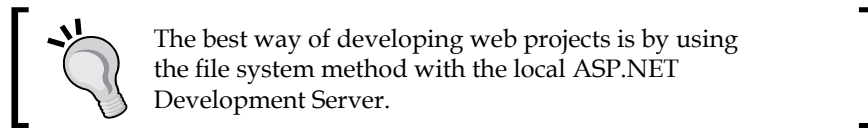
When using the file system location, the ASP.NET Development Server is used. Each web project in our solution is run by an individual ASP.NET Development Server. This web server is started automatically when we run the web project for the first time.



When using local IIS, the local Internet Information Services web server is used to run all the web projects in our solution.

While developing several web projects, using an individual web server for each one presents the advantage of being able to debug all of them at the same time. When using a single server, a debugging session affects all the web projects the server runs and only one project can be debugged at a time.

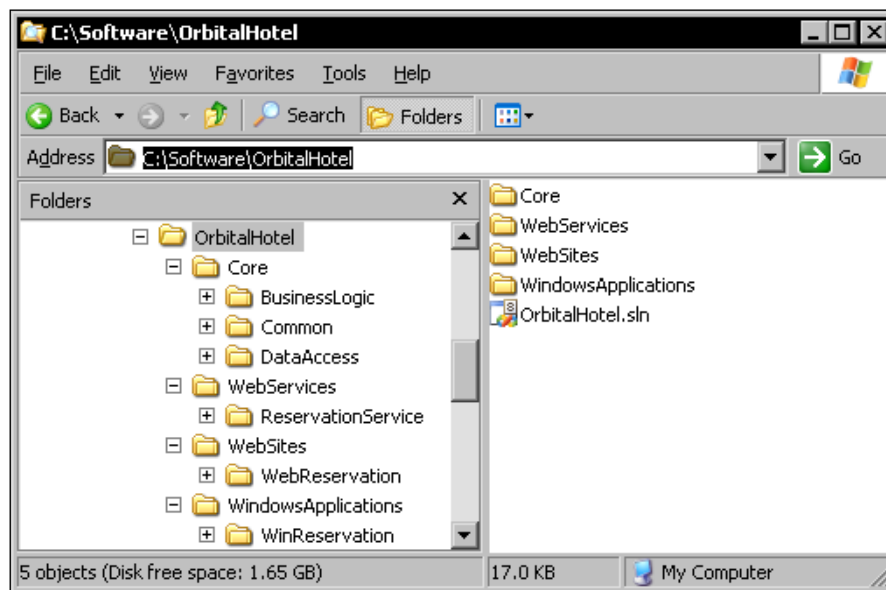
When solutions that contain local disk web projects are added to source control, Visual Studio automatically creates in the source control database a folder structure that will match the hierarchical structure on the local disk.



Creating the Orbital Hotel Solution

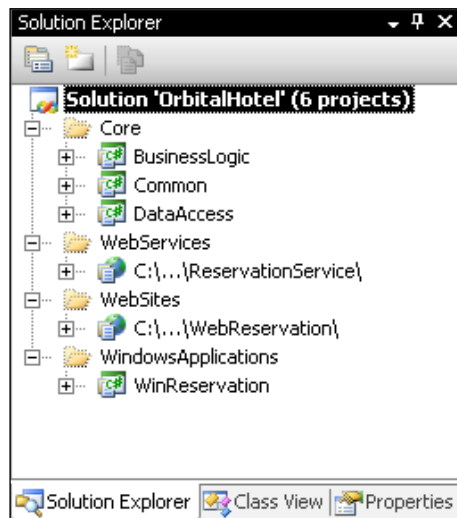
After considering the best structure of Visual Studio .NET solutions we can now create the solution structure for our Orbital Hotel reservation system product.

I will use a single (master) solution because our example has a small number of projects, and divide the solution folder structure into the following sections, mapping the system's design:



The **OrbitalHotel** folder is the root folder for the solution. Note an intermediary set of folders named **Core**, **WebServices**, **WebSites**, and **WindowsApplications** that contain the individual solution projects. These folders contain the main system components.

The following figure shows the detailed master solution structure in the Visual Studio .NET **Solution Explorer** window:



You will also notice that the web projects are created using the local file system instead of IIS. This configuration allows us to run the web projects without having to install IIS and configure virtual directories that match the physical project locations on the development machines and, as we saw earlier, we are able to debug multiple web projects at the same time. Also, using local file system web projects will help later to create a project structure in the source control database matching the structure on the local disk.

You can also see how the physical solution structure maps the logical solution structure in Visual Studio.NET.

Summary

In this chapter we started the development lifecycle for the Orbital Hotel product.

The first phase is gathering the system specifications. Once the specifications are clear we analyze them and decide which application architecture is the most appropriate for its implementation. For the Orbital Hotel application, service-oriented architecture is the most appropriate because it allows the greatest flexibility and interoperability.

We then moved on to the design phase and designed the system's structure and components. We saw that when creating the Visual Studio .NET solution, we must take into account the best structure for solutions that will be developed under source control by multiple team members.

The best structure is the hierarchical solution structure that maps directly into the source control database and is the same on all the development machines preventing binding problems especially for web projects. The best way to develop web projects is by using the local file system and the ASP.NET Development Server because they are created in the same hierarchical folder structure as the solution and because it avoids supplementary IIS configurations.

At the end, we've created the Orbital Hotel solution structure based on all the best practices we've seen in this chapter.

For More Information: <http://www.packtpub.com/visual-sourcesafe-2005/book>

Where to buy this book

You can buy *Configuring IPCop Firewalls: Closing Borders with Open Source* from the Packt Publishing website: <http://www.packtpub.com/visual-sourcesafe-2005/book>

Free shipping to the US, UK, Europe, Australia, New Zealand and India.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.



www.PacktPub.com

For More Information: <http://www.packtpub.com/visual-sourcesafe-2005/book>