NEAR EAST UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

.NET PASSPORT

Graduation Project
COM400

Student: Yakup Kalkan (20001067)

Supervisor: Mr. Kaan Uyar

Nicosia-2005

ACKNOWLEDGMENTS

"First, I would like to thank my supervisor Mr. Kaan Uyar for supervising my project. If there aren't his helps and advice, my project doesn't better then its situation. Under the guidance of him I successfully overcome many difficulties and I learned a lot about microsoft .net and .net passport. He always help me and ready to help me every time. I am so grateful."

"Second , I would like to thank to express my gratitude to Near East University for scholarship that made the work possible."

"Third, I would also would like to thank Kıvanç Eram and other friends for their advice and support."

ABSTRACT

The term “Microsoft .NET” refers to a massive effort on Microsoft’s part to get away from traditional software development and to build—with help from partners all over the industry—the Internet into a service-oriented software platform. The Internet will become a software platform with an API far richer than any operating system. Today’s applications rely primarily on operating system services. Tomorrow’s applications will use Web services to validate credit card purchases, check the status of airline flights, and perform other everyday tasks. Microsoft .NET services can be seem in any application on the web which are .NET Passport, .NET Alert and MSN Wallet. .NET Passport is examined in that project and ASP.NET is used .

ASP .NET is better for Microsoft .Net and is more complicated end easier then other web programming. It has a lot of advantages on the web.ASP.NET is improved with each passing day.

.NET Passport allows users to create a single sign-in name and password to access any site that has implemented the Passport single sign-in (SSI) service. In this project the basic implementation .NET Passport is written and all details are declared clearly.

TABLE OF CONTENTS

INTRODUCTION

Before getting deeply into the subject we will first know how Businesses are related to Internet, what .NET means to them and what exactly .NET is built upon. As per the product documentation from a Business perspective, there are three phases of the Internet. The First phase gets back to the early 1990's when Internet first came into general use and which brought a big revolution for Businesses. In the First phase of the Internet Businesses designed and launched their Website's and focused on the number of hits to know how many customers were visiting their site and interested in their products, etc. The Second phase is what we are in right now and in this phase Businesses are generating revenue through Online Transactions. We are now moving into the Third phase of the Internet where profit is the main priority. The focus here is to Businesses effectively communicated with their customers and partners, who are geographically isolated, participate in Digital Economy and deliver a wide range of services. How can that be possible? The answer, with .NET.

CHAPTER ONE

MICROSOFT .NET

1.1. What is .NET?

Many people reckon that it's Microsoft's way of controlling the Internet, which is false. .NET is Microsoft's strategy of software that provides services to people any time, any place, on any device. An accurate definition of .NET is, it's an XML Web Services platform which allows us to build rich .NET applications, which allows users to interact with the Internet using wide range of smart devices (tablet devices, pocket PC's, web phones etc), which allows to build and integrate Web Services and which comes with many rich set of tools like Visual Studio to fully develop and build those applications.

1.1.1. What are Web Services?

Web Services are the applications that run on a Web Server and communicate with other applications. It uses a series of protocols to respond to different requests. The protocols on which Web Services are built are summarized below:

UDDI: Stands for Universal Discovery and Description Integration. It's said to be the Yellow Pages of Web Services which allows Businesses to search for other Businesses allowing them to search for the services it needs, know about the services and contact them.

WSDL: Stands for Web Services Description Language, often called as whiz-dull. WSDL is an XML document that describes a set of SOAP messages and how those messages are exchanged.

SOAP: Stands for Simple Object Access Protocol. It's the communication protocol for Web Services.

XML, HTTP and SMTP: Stands for Extensible Markup Language, Hyper Text Transfer Protocol and Simple Message Transfer Protocol respectively. UDDI, WSDL and SOAP rely on these protocols for communication.

1.1.2. What is .NET Built On?

.NET is built on the Windows Server System to take major advantage of the OS and which comes with a host of different servers which allows for building, deploying, managing and maintaining Web-based solutions. The Windows Server System is designed with performance as priority and it provides scalability, reliability, and manageability for the global, Web-enabled enterprise. The Windows Server System integrated software products are built for interoperability using open Web standards such as XML and SOAP.

Core Windows Server System Products include:

SQL Server2000: This Database Server is Web enabled and is designed with priority for .NET based applications. It is scalable, easy to manage and has a native XML store.

Application Center 2000: This product is designed to manage Web Applications.

Commerce Server 2000: This powerful Server is designed for creating E-Commerce based applications.

Mobile Information Server: This Server provides real-time access for the mobile community. Now Outlook users can use their Pocket PC's to access all their Outlook data while they are moving.

Exchange Server 2000: This is a messaging system Server and allows applications on any device to access information and collaborate using XML.

BizTalk Server 2000: This is the first product created for .NET which is XML based and allows to build business process that integrate with other services in the organization or with other Businesses.

Internet Security and Acceleration Server 2000: This Server provides Security and Protection for machines. It is an integrated firewall and Web cache server built to make the Web-enabled enterprise safer, faster, and more manageable.

Host Integration Server 2000: This Server allows for the Integration of mainframe systems with .NET.

When developing real world projects if you don't know how to use the above mentioned Server's which are built for .NET based applications do not worry. Your System Administrator is always there to help you.

1.2. Introduction to .NET Services

Microsoft® .NET Services are Internet-enabled, user-centric services that provide the building blocks to develop your own powerful, connected Web applications. Microsoft .NET Services are built on open industry-standard protocols such as SOAP, XML, and Universal Description, Discovery, and Integration (UDDI). Because .NET Services work in heterogeneous technological environments, you can leverage your current technology investments and integrate .NET Services into any application that supports an XML Web Services programming model.

Connected applications provide a new level of convenience, increased value, efficiency, and ease-of-use to your customers. Applications that use .NET Services provide users with a powerful and consistent user experience across a variety of interfaces, from desktop computers to wireless devices. Microsoft .NET Services allow applications, devices, Web sites, and Web services to work together more effectively.

1.2.1. .NET Passport

Microsoft® .NET Passport is a suite of Web-based services that makes using the Internet and purchasing online easier and faster. Microsoft .NET Passport single sign-in (SSI) is used for user authentication. Kids Passport is designed to give parents the ability to manage how participating Web sites collect and store personal information about their children.

1.2.2. .NET Alerts

Microsoft® .NET Alerts is a message and notification routing service that makes delivering customer communications easy, and avoids the negative perceptions associated with junk e-mail. Content providers can send messages to customers who choose to receive them. Alerts are routed to Microsoft® Windows® desktops, cellular phones, wireless personal digital assistants (PDAs), or any e-mail address—all based on the customer's delivery preferences.

1.2.3. MSN Wallet

MSN® Wallet is an easy-to-implement, server-based, consumer wallet system designed to create a consistent and convenient online purchasing experience. With MSN Wallet, users can easily store and retrieve information commonly needed to complete online purchases, such as shipping addresses and methods of payment. Users also gain access to special promotions available only through the use of MSN Wallet.

CHAPTER TWO

Introduction ASP.NET

2.1. Introduction

ASP.NET, the next version of ASP, is a programming framework used to create enterprise-class Web Applications. These applications are accessible on a global basis leading to effecient information managment. The advantages ASP.NET offers is more than just the next version of ASP.

2.2. Overview of ASP.NET

ASP.NET is a unified Web application platform that provides the services necessary to build and deploy enterprise-class Web applications. ASP.NET offers a new programming model and infrastructure for more secure, scalable, and stable applications that can target any browser or device.

ASP.NET is part of the Microsoft .NET Framework, a computing environment that simplifies application development in the highly distributed environment of the Internet. The .NET Framework includes the common language runtime (CLR), which provides core services such as memory management, thread management, and code security. It also includes the .NET Framework class library, which is a comprehensive, object-oriented collection of types that developers can use to create applications.

ASP.NET offers the following benefits:

- Manageability: ASP.NET uses a text-based, hierarchical configuration system that simplifies applying settings to your server environment and Web applications. Because configuration information is stored as plain text, new settings can be applied without the aid of local administration tools. Any changes to the configuration files are automatically detected and applied to the application.

- Security: ASP.NET provides default authorization and authentication schemes for Web applications. Developers can easily add to, remove, or replace these schemes depending on the needs of the application.
- Ease of Deployment: An ASP.NET application is deployed to a server by simply copying the necessary files to the server. No server restart is required — not even to deploy or replace running compiled code.
- Enhanced Performance: ASP.NET is compiled code running on the server. Unlike traditional Active Server Pages (ASP), ASP.NET can take advantage of early binding, just-in-time (JIT) compilation, native optimization, and caching services right out of the box for improved performance.
- Flexible Output Caching: ASP.NET can cache page data, portions of a page, or whole pages, depending on the needs of the application. Cached items can be dependent on files or other items in the cache, or they can be refreshed based on an expiration policy.
- Internationalization: ASP.NET uses Unicode internally to represent request and response data. Internationalization settings can be configured for each computer, each directory, and each page.
- Mobile Device Support: ASP.NET supports any browser on any device. Developers use the same programming techniques to target new mobile devices that they use for traditional desktop browsers.
- Scalability and Availability: ASP.NET was designed to be scalable, with features specifically tailored to improve performance in clustered and multiprocessor environments. Moreover, processes are closely monitored and managed by Internet Information Services (IIS) and the ASP.NET CLR so that if one misbehaves, a new process can be created in its place, which helps keep your application available to handle requests.
- Tracing and Debugging: ASP.NET provides tracing services that can be enabled during debugging at both the application level and the page level. You can choose to view the information either in a page or using the application-level trace viewing tool. ASP.NET supports local and remote debugging with .NET Framework debugging tools, both during development and when the application is in

production. When an application is put into production, tracing statements can be left in the production code with no impact on performance.

- Integration with the .NET Framework: Because ASP.NET is part of the .NET Framework, the power and flexibility of the entire platform is available to Web applications. The .NET class library, as well as solutions for messaging and data access, are all seamlessly accessible from the Web. ASP.NET is language-independent, so developers can choose the language that best applies to the application. In addition, CLR interoperability preserves existing investments in COM-based development.
- Compatibility with Existing ASP Applications: ASP and ASP.NET can run side by side on an IIS Web server without interference; there is no chance of corrupting an existing ASP application by installing ASP.NET. Only files with a .aspx file name extension are processed by ASP.NET. Files with an .asp file name extension will continue to be processed by the ASP engine. You should note, however, that session state and application state are not shared between ASP and ASP.NET pages.

ASP.NET enables two features for distributed applications: Web Forms and XML Web services. These two features are supported by the same configuration and debugging infrastructure.

- Web Forms technology enables you to build powerful form-based Web pages. Web Forms pages use reusable built-in or custom components to simplify the code of a page.
- XML Web services that are created using ASP.NET enable you to access servers remotely. Using XML Web services, businesses can provide programmatic interfaces to their data or business logic, which in turn can be obtained and manipulated by client and server applications. XML Web services enable the exchange of data using standards such as XML messaging and HTTP across firewalls in client/server and server/server scenarios. Programs written in any language and running on any operating system can call XML Web services.

2.3. Why ASP.NET?

Since 1995, Microsoft has been constantly working to shift it's focus from Windows-based platforms to the Internet. As a result, Microsoft introduced ASP (Active Server Pages) in November 1996. ASP offered the efficiency of ISAPI applications along with a new level of simplicity that made it easy to understand and use. However, ASP script was an interpreted script and consisted unstructured code and was difficult to debug and maintain. As the web consists of many different technologies, software integration for Web development was complicated and required to understand many different technologies. Also, as applications grew bigger in size and became more complex, the number of lines of source code in ASP applications increased dramatically and was hard to maintain. Therefore, an architecture was needed that would allow development of Web applications in a structured and consistent way.

The .NET Framework was introduced with a vision to create globally distributed software with Internet functionality and interoperability. The .NET Framework consists of many class libraries, includes multiple language support and a common execution platform. It's a very flexible foundation on which many different types of top class applications can be developed that do different things. Developing Internet applications with the .NET Framework is very easy. ASP.NET is built into this framework, we can create ASP.NET applications using any of the built-in languages.

Unlike ASP, ASP.NET uses the Common Language Runtime (CLR) provided by the .NET Framework. This CLR manages execution of the code we write. ASP.NET code is a compiled CLR code instead of interpreted code (ASP). CLR also allows objects written in different languages to interact with each other. The CLR makes developement of Web applications simple.

2.3.1. Improved Performance and Scalability

- Compiled Execution: ASP.NET is much faster than classic ASP, while preserving the "just hit save" update model of ASP. No explicit compile step is required. ASP.NET automatically detects any change, dynamically compiles files if needed,

- Easy Migration Path: ASP.NET runs side by side on IIS with classic ASP applications on Microsoft Windows 2000 and Windows XP, and on members of the Windows Server 2003 family. You can migrate one application at a time, or even single pages. ASP.NET even lets you continue to use your existing classic COM business components.

2.3.4. New Application Models

- XML Web Services: XML Web services allow applications to communicate and share data over the Internet, regardless of operating system or programming language. ASP.NET makes exposing and calling XML Web services simple.
- Mobile Web Device Support: ASP.NET mobile controls let you target over 80 mobile Web devices using ASP.NET. You write the application once, and the mobile controls automatically generate pages for the requesting device.

2.3.5. Developer Productivity

- Easy Programming Model: ASP.NET makes building real-world Web applications dramatically easier with server controls that let you build great pages with far less code than classic ASP.
- Flexible Language Options. ASP.NET supports not only Microsoft Visual Basic Scripting Edition (VBScript) and Microsoft JScript but also more than 25 .NET languages, including built-in support for Visual Basic .NET, Microsoft C#, and JScript .NET.
- Rich Class Framework: The .NET Framework class library offers over 4,500 classes that encapsulate rich functionality such as XML, data access, file upload, regular expressions, image generation, performance monitoring and logging, transactions, message queuing, and SMTP mail.

2.4. ASP.NET Features

ASP.NET is not just a simple upgrade or the latest version of ASP. ASP.NET combines unprecedented developer productivity with performance, reliability, and deployment. ASP.NET redesigns the whole process. It's still easy to grasp for new comers but it provides many new ways of managing projects. Below are the features of ASP.NET.

2.4.1. Easy Programming Model

ASP.NET makes building real world Web applications dramatically easier. ASP.NET server controls enable an HTML-like style of declarative programming that let you build great pages with far less code than with classic ASP. Displaying data, validating user input, and uploading files are all amazingly easy. Best of all, ASP.NET pages work in all browsers including Netscape, Opera, AOL, and Internet Explorer.

2.4.2. Flexible Language Options

ASP.NET lets you leverage your current programming language skills. Unlike classic ASP, which supports only interpreted VBScript and JScript, ASP.NET now supports more than 25 .NET languages (built-in support for VB.NET, C#, and JScript.NET), giving you unprecedented flexibility in your choice of language.

2.4.3. Great Tool Support

You can harness the full power of ASP.NET using any text editor, even Notepad. But Visual Studio .NET adds the productivity of Visual Basic-style development to the Web. Now you can visually design ASP.NET Web Forms using familiar drag-drop-doubleclick techniques, and enjoy full-fledged code support including statement completion and color-coding. VS.NET also provides integrated support for debugging and deploying ASP.NET Web applications. The Enterprise versions of Visual Studio .NET deliver life-cycle features to help organizations plan, analyze, design, build, test, and coordinate teams that develop ASP.NET Web applications. These include UML class modeling, database modeling (conceptual, logical, and physical models), testing tools (functional, performance and scalability), and enterprise frameworks and templates, all available within the integrated Visual Studio .NET environment.

2.4.4. Rich Class Framework

Application features that used to be hard to implement, or required a 3rd-party component, can now be added in just a few lines of code using the .NET Framework. The .NET Framework offers over 4500 classes that encapsulate rich functionality like XML, data access, file upload, regular expressions, image generation, performance monitoring and logging, transactions, message queuing, SMTP mail, and much more. With Improved Performance and Scalability ASP.NET lets you use serve more users with the same hardware.

2.4.5. Compiled execution

ASP.NET is much faster than classic ASP, while preserving the "just hit save" update model of ASP. However, no explicit compile step is required. ASP.NET will automatically detect any changes, dynamically compile the files if needed, and store the compiled results to reuse for subsequent requests. Dynamic compilation ensures that your application is always up to date, and compiled execution makes it fast. Most applications migrated from classic
ASP see a 3x to 5x increase in pages served.

2.4.6. Rich output caching

ASP.NET output caching can dramatically improve the performance and scalability of your application. When output caching is enabled on a page, ASP.NET executes the page just once, and saves the result in memory in addition to sending it to the user. When another user requests the same page, ASP.NET serves the cached result from memory without re-executing the page. Output caching is configurable, and can be used to cache individual regions or an entire page. Output caching can dramatically improve the performance of data-driven pages by eliminating the need to query the database on every request.

2.4.7. Web-Farm Session State

ASP.NET session state lets you share session data user-specific state values across all machines in your Web farm. Now a user can hit different servers in the Web farm over multiple requests and still have full access to her session. And since business components created with the .NET Framework are free-threaded, you no longer need to worry about thread affinity.

2.4.8. Enhanced Reliability

ASP.NET ensures that your application is always available to your users.

2.4.9. Memory Leak, DeadLock and Crash Protection

ASP.NET automatically detects and recovers from errors like deadlocks and memory leaks to ensure your application is always available to your users. For example, say that your application has a small memory leak, and that after a week the leak has tied up a significant percentage of your server's virtual memory. ASP.NET will detect this condition, automatically start up another copy of the ASP.NET worker process, and direct all new requests to the new process. Once the old process has finished processing its pending requests, it is gracefully disposed and the leaked memory is released. Automatically, without administrator intervention or any interruption of service, ASP.NET has recovered from the error.

2.4.10. Easy Deployment

ASP.NET takes the pain out of deploying server applications. "No touch" application deployment. ASP.NET dramatically simplifies installation of your application. With ASP.NET, you can deploy an entire application as easily as an HTML page, just copy it to the server. No need to run regsvr32 to register any components, and configuration settings are stored in an XML file within the application.

2.4.11. Dynamic update of running application

ASP.NET now lets you update compiled components without restarting the web server. In the past with classic COM components, the developer would have to restart the web server each time he deployed an update. With ASP.NET, you simply copy the component over the existing DLL, ASP.NET will automatically detect the change and start using the new code.

2.4.12. Easy Migration Path

You don't have to migrate your existing applications to start using ASP.NET. ASP.NET runs on IIS side-by-side with classic ASP on Windows 2000 and Windows XP platforms. Your existing ASP applications continue to be processed by ASP.DLL, while new ASP.NET pages are processed by the new ASP.NET engine. You can migrate application by application, or single pages. And ASP.NET even lets you continue to use your existing classic COM business components.

2.4.13. XML Web Services

XML Web services allow applications to communicate and share data over the Internet, regardless of operating system or programming language. ASP.NET makes exposing and calling XML Web Services simple. Any class can be converted into an XML Web Service with just a few lines of code, and can be called by any SOAP client. Likewise, ASP.NET makes it incredibly easy to call XML Web Services from your application. No knowledge of networking, XML, or SOAP is required.

2.4.14. Mobile Web Device Support

ASP.NET Mobile Controls let you easily target cell phones, PDAs and over 80 mobile Web devices. You write your application just once, and the mobile controls automatically generate WAP/WML, HTML, or iMode as required by the requesting device.

2.5. Advantages Using ASP.NET

- ASP.NET drastically reduces the amount of code required to build large applications
- ASP.NET makes development simpler and easier to maintain with an event-driven, server-side programming model
- ASP.NET pages are easy to write and maintain because the source code and HTML are together
- The source code is executed on the server. The pages have lots of power and flexibility by this approach
- The source code is compiled the first time the page is requested. Execution is fast as the Web Server compiles the page the first time it is requested. The server saves the compiled version of the page for use next time the page is requested
- The HTML produced by the ASP.NET page is sent back to the browser. The application source code you write is not sent and is not easily stolen
- ASP.NET makes for easy deployment. There is no need to register components because the configuration information is built-in
- The Web server continuously monitors the pages, components and applications running on it. If it noticies memory leaks, infinite loops, other illegal software or activities, it seamlessly kills those activities and restarts itself
- ASP.NET validates information (validation controls) entered by the user without writing a single line of code
- ASP.NET easily works with ADO .NET using data-binding and page formatting features
- ASP.NET applications run fater and counters large volumes of users without performance problems

2.6. Differences between ASP.NET and Client-Side Technologies

Client-side refers to the browser and the machine running the browser. Server-side on the other hand refers to a Web server.

2.6.1. Client-Side Scripting

Javascript and VBScript and generally used for Client-side scripting. Client-side scripting executes in the browser after the page is loaded. Using client-side scripting you can add some cool features to your page. Both, HTML and the script are together in the same file and the script is download as part of the page which anyone can view. A client-side script runs only on a browser that supports scripting and specifically the scripting language that is used. Since the script is in the same file as the HTML and as it executes on the machine you use, the page may take longer time to download.

2.6.2. Server-Side Scripting

ASP.NET is purely server-side technology. ASP.NET code executes on the server before it is sent to the browser. The code that is sent back to the browser is pure HTML and not ASP.NET code. Like client-side scripting, ASP.NET code is similar in a way that it allows you to write your code alongside HTML. Unlike client-side scripting, ASP.NET code is executed on the server and not in the browser. The script that you write alongside your HTML is not sent back to the browser and that prevents others from stealing the code you developed.

2.7. Installing ASP.NET (IIS 6.0)

ASP.NET is supported on the Windows Server™ 2003 family, Windows 2000 (Professional, Server, and Advanced Server), and Windows XP Professional for both client and server applications.

A server running a member of the Microsoft Windows Server 2003 family can be configured as an application server, with ASP.NET as an option that you can enable while configuring the application server role. To deploy ASP.NET Web applications to a production server, you must be sure to enable the ASP.NET and IIS roles on the production server before you distribute the application.

If you want to install ASP.NET on a domain controller, there are special steps you must take to make the installation work correctly

ASP.NET, along with the .NET Framework version 1.1, is installed as a part of Windows Server 2003. You simply need to add it as a new program from Control Panel or enable it by using the Configure Your Server wizard.

To install ASP.NET on a server running Windows Server 2003 using the Configure Your Server wizard

1. From the Start menu, click Manage Your Server; in the Manage Your Server window, click Add or remove a role.
2. In the Configure Your Server Wizard, click Next, and in the Server Role dialog box, check Application Server (IIS, ASP.NET) and then click Next.
3. In the Application Server Options dialog box, select the Enable ASP.NET check box, click Next, and then click Next again.
4. If necessary, insert your Windows Server 2003 installation CD in the CD-ROM drive and then click Next.
5. When the installation is complete, click Finish.

To install ASP.NET on a server running Windows Server 2003 using the Add or Remove Programs dialog box

1. From the Start menu, point to Control Panel, and then click Add or Remove Programs.
2. In the Add or Remove Programs dialog box, click Add/Remove Windows Components.
3. In the Add or Remove Programs dialog box, click Add/Remove Windows Components.
4. When the Windows Components wizard has finished configuring Windows Server 2003, click Finish

To enable ASP.NET in IIS Manager on a server running Windows Server 2003

1. From the Start menu, click Run.
2. In the Open box in the Run dialog box, type inetmgr, and then click OK.
3. In IIS Manager, expand the local computer and then click Web Service Extensions.

4. In the details pane, right-click ASP.NET and then click Allow. The status of ASP.NET changes to Allowed.

2.8. Overview of ASP.NET Security (IIS 6.0)

Most Web sites need to selectively restrict access to some portions of the site. You can think of a Web site as somewhat analogous to an art gallery. The gallery is open for the public to come in and browse, but there are certain parts of the facility, such as the business offices, that are accessible only to people with certain credentials, such as employees. When a Web site stores its customers' credit card information in a database, for example, ASP.NET helps protect the database from public access. ASP.NET security addresses this and many other security issues.

ASP.NET, in conjunction with Internet Information Services (IIS), can authenticate user credentials such as names and passwords using any of the following authentication methods:

- Windows: Basic, digest, or Integrated Windows Authentication (NTLM or Kerberos).
- Microsoft Passport authentication
- Forms authentication
- Client Certificate authentication

ASP.NET helps control access to site information by comparing authenticated credentials, or representations of them, to NTFS file system permissions or to an XML file that lists authorized users, authorized roles (groups), or authorized HTTP verbs.

The topics in this section describe the specifics of ASP.NET security.

2.9. Architecture in ASP.NET (IIS 6.0)

This section provides an overview of the ASP.NET infrastructure and subsystem relationships, as they relate to the subject of security. The following illustration shows the relationships among the security systems in ASP.NET.

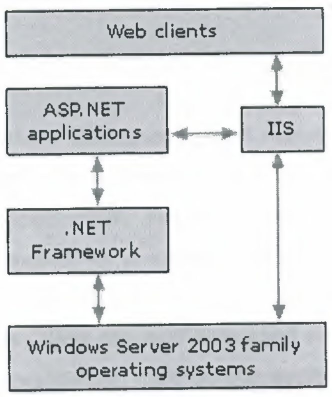


Figure2.9. – Security system in ASP

As the illustration shows, all Web clients communicate with ASP.NET applications through Internet Information Services (IIS). IIS deciphers and optionally authenticates the request. If Allow Anonymous is set to true, no authentication occurs. IIS also finds the requested resource (such as an ASP.NET application), and, if the client is authorized, returns the appropriate resource.

In addition to the built-in ASP.NET features, an ASP.NET application can use the low-level security features of the .NET Framework.

2.9.1. Integrating with IIS

When considering ASP.NET authentication, you should understand the interaction with IIS authentication services.

IIS always assumes that a set of credentials maps to a Microsoft Windows NT account and uses them to authenticate a user. There are three different kinds of authentication available in IIS 5.0 through IIS 6.0: basic, digest, and Integrated Windows Authentication (NTLM or Kerberos). You can select the type of authentication to use in IIS administrative services.

If you request a URL containing an ASP.NET application, the request and authentication information are handed off to the application. ASP.NET provides the two additional types of authentication described in the following table.

ASP.NET authentication provider	Description
Forms authentication	A system by which unauthenticated requests are redirected to an HTML form using HTTP client side redirection. The user provides credentials and submits the form. If the application authenticates the request, the system issues an authentication ticket in a cookie that contains the credentials or a key for reacquiring the identity. Subsequent requests are issued with the cookie in the request headers; they are authenticated and authorized by an ASP.NET handler using whatever validation method the application developer specifies.
Passport authentication	Centralized authentication service provided by Microsoft that offers a single logon and core profile services for member sites.

Table2.9.1. – ASP.NET Authentication Provider

2.9.2. Using ASP.NET Configuration Files

ASP.NET configuration, of which security is a part, has a hierarchical architecture. All configuration information for ASP.NET is contained in files named Web.config and Machine.config. Web.config can be placed in the same directories as the application files. The Machine.config file is in the Config directory of the install root. Subdirectories inherit a directory's settings unless overridden by a Web.config file in the subdirectory. In a Web.config file, there are sections for each major category of ASP.NET functionality. To see an example of the way in which the hierarchical configuration system works for security.

The security section of a Web.config file is organized as follows.

```
<authentication mode="[Windows|Forms|Passport|None]">

<forms name="[name]"

loginUrl="[url]"

protection="[All|None|Encryption|Validation]"

path="[path]" timeout="[minutes]"

requireSSL="[true|false]"

slidingExpiration="[true|false]">

<credentials passwordFormat="[Clear|MD5|SHA1]">

<user name="[UserName]"

password="[password]"/>

</credentials>

</forms>

<passport redirectUrl="internal"/>

</authentication>

<authorization>

<allow users="[comma separated list of users]"

roles="[comma separated list of roles]"/>

<deny users="[comma separated list of users]"

roles="[comma separated list of roles]"/>

</authorization>

<identity impersonate ="[true|false]"

userName="[domain\user_name]"
```

```
password="[user_password]"/>
<trust level="[Full|High|Medium|Low|Minimal]"
originUrl=""/>
<securityPolicy>
<trustLevel name="Full" policyFile="internal"/>
<trustLevel name="High" policyFile="web_hightrust.config"/>
<trustLevel name="Medium" policyFile="web_mediumtrust.config"/>
<trustLevel name="Low" policyFile="web_lowtrust.config"/>
<trustLevel name="Minimal" policyFile="web_minimaltrust.config"/>
</securityPolicy>
```

The default settings for these elements are shown in the following table.

Default value	Description
<allow roles= >	No default value.
<allow users="*">	All
<authentication mode="Windows">	The authentication mode cannot be set at a level below the application root directory.
<credentials passwordFormat="SHA1">	The hashing algorithm to be used on passwords.
<deny roles="">	Empty
<deny users="">	Empty
<forms loginUrl="login.aspx">	If you set the mode to **Forms**, and if the request does not have a valid ticket (cookie), this is the URL to which the request is directed for a forms-based logon.
<forms name=".ASPXAUTH">	Default cookie name.

Default value	Description
<forms path="/">	Path
<forms protection="All">	Type=[All|None|Encryption|Validation].
<forms requireSSL="false">	Specifies that an encrypted connection is not required to transmit the authentication cookie.
<forms slidingExpiration="true">	Specifies that sliding expiration is enabled.
<forms timeout="30">	Time-out in minutes. 30 minutes is the default.
<identity impersonate="false">	Impersonation is disabled by default.
<identity userName="">	Empty
<identity password="">	Empty
<passport redirectUrl="internal">	If you set the mode to **Passport**, and if the requested page requires authentication (anonymous users are denied access) but the user has not logged on with Passport, then the user will be redirected to this URL.
<trustLevel name="Full" policyFile="internal"/>	Default policy file for Full trust level.
<trustLevel name="High" policyFile="web_hightrust.config"/>	Default policy file for High trust level.
<trustLevel name="Medium" policyFile="web_mediumtrust.config"/>	Default policy file for Medium trust level.
<trustLevel name="Low" policyFile="web_lowtrust.config"/>	Default policy file for Low trust level.
<trustLevel name="Minimal" policyFile="web_minimaltrust.config"/>	Default policy file for Minimal trust level.
<user name="">	Empty
<user password="">	Empty

Table2.9.2. – Definition of web.config's elements

There are three major subsections to a Web.config file: authentication, authorization, and identity. The values for each of the security elements are usually set by overriding a section of the computer-level configuration file with a similar section in an application configuration file placed in the application root directory. All subdirectories automatically inherit those settings. However, subdirectories can have their own configuration files that override other settings.

ASP.NET configuration applies only to ASP.NET resources (those registered to be handled by Aspnet_isapi.dll). Unfortunately, ASP.NET configuration cannot provide authorization for non-Aspnet_isapi.dll resources, so TXT, HTML, GIF, JPEG, ASP, and other types of files are still accessible by all users, subject to IIS permissions. For example, although the ASP.NET resources in a directory might be restricted by a Web.config file, all users can still view the files located in that directory if directory browsing is turned on and no other restrictions are in place.

You can avoid this situation by explicitly mapping such files, but not directories, to Aspnet_isapi.dll using the IIS administration tool. However, there could be a performance impact if you do this.

You can use the <location></location> tags to specify a particular file or directory to which settings should apply.

2.10. How Security Works in ASP.NET (IIS 6.0)

Helping protect Web sites against unauthorized access is a critical, complex issue for Web developers. A successful system requires careful planning, and Web site administrators and programmers must have a clear understanding of the options for securing their site.

ASP.NET works in concert with the Microsoft .NET Framework and Internet Information Services (IIS) to help provide Web application protection. To help protect an ASP.NET application, you must perform the two fundamental functions described in the following table.

Security function	Description
Authentication in ASP.NET	Assures that the user is, in fact, who the user claims to be. The application obtains credentials (various forms of identification, such as name and password) from a user and validates those credentials against some authority. If the credentials are valid, the entity that submitted the credentials is considered an authenticated identity.
Authorization in ASP.NET	Limits access rights by granting or denying specific permissions to an authenticated identity.

Table2.10. – Security Function

IIS can also grant or deny access based on a user's host name or IP address. Any further access authorization is performed by NTFS file access permission's URL authorization.

It is helpful to understand how all the various security subsystems interact. Since ASP.NET is built on the Microsoft .NET Framework, the ASP.NET application developer also has access to all the built-in security features of the .NET Framework, such as code access security and role-based user-access security.

2.11. Data Flow in ASP.NET (IIS 6.0)

There are a number of different ways to design security into ASP.NET applications. This section describes the data flow for two common scenarios: impersonation and forms authentication using cookies.

2.11.1. Scenario 1: Impersonation

This scenario relies on Internet Information Services (IIS) authentication and Microsoft Windows NT® file access security to minimize security programming in the ASP.NET application itself. The data flow is shown in the following illustration.

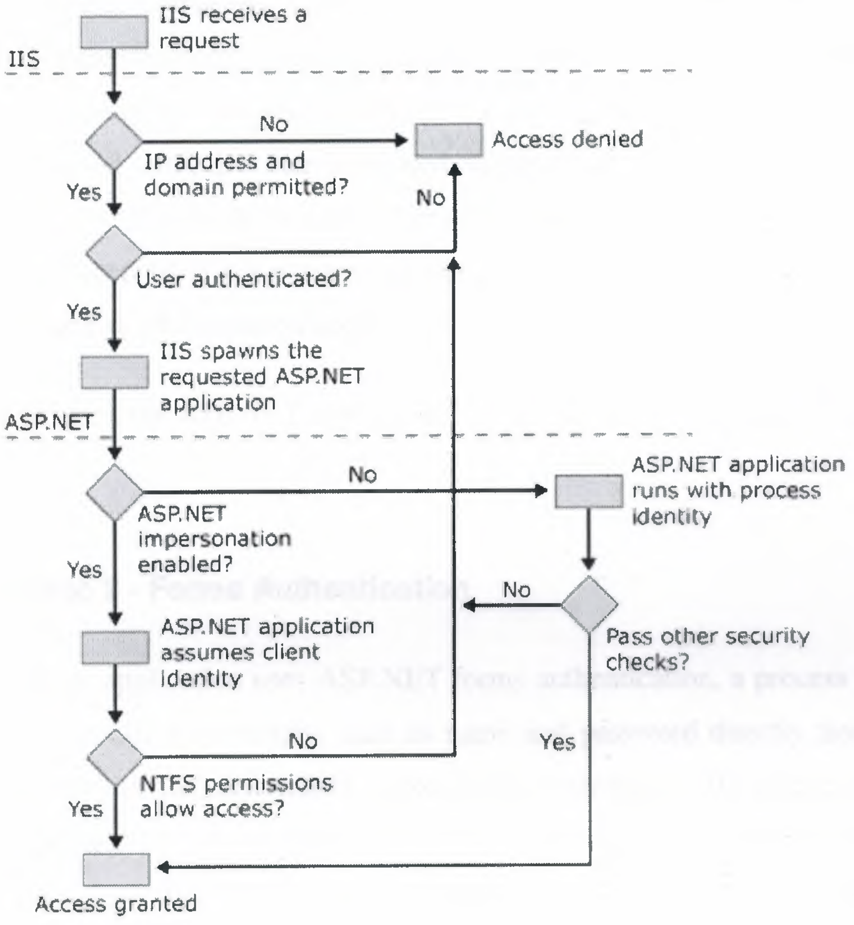

IIS receives a request
IIS
No
IP address and domain permitted?
Access denied
Yes
No
User authenticated?
Yes
IIS spawns the requested ASP.NET application
ASP.NET
No
ASP.NET application runs with process identity
ASP.NET impersonation enabled?
Yes
No
Pass other security checks?
ASP.NET application assumes client identity
No
Yes
NTFS permissions allow access?
Yes
Access granted

Figure2.11.1. – Data flow in impersonation

The illustration shows the following sequence of events:

- A request for access comes to IIS from a network client.

- IIS authenticates the client using basic, digest, or Integrated Windows Authentication NTLM or Kerberos).

- If the client is authenticated, IIS hands the authenticated request over to ASP.NET.

- The ASP.NET application impersonates the requesting client using the access token passed from IIS, and relies on NTFS file permissions for granting access. The ASP.NET application needs only to verify that in the ASP.NET configuration file, the impersonation-enable directive is set to **true**; no ASP.NET security code needs to be written.

Notice that if impersonation is not enabled, the application runs with the IIS process identity. For Microsoft Windows® 2000 Server and Windows XP, the default identity is a User account named ASPNET that is created automatically when ASP.NET is installed. For products in the Microsoft Windows Server 2003 family, the default identity is the Network Service account. If you want to restrict access, you must use some other means of authorization, such as URL authorization.

- If access is granted, the ASP.NET application returns the requested page through IIS.

2.11.2. Scenario 2 - Forms Authentication

In this scenario an application uses ASP.NET forms authentication, a process that enables the application to collect credentials such as name and password directly from the client requestor and make its own determination about their authenticity. IIS authentication is not used by the application, but IIS authentication settings are important to the ASP.NET forms authentication process. Unless you decide to reject all requests that do not meet the criteria for the enabled method of IIS authentication, you must enable the IIS **Anonymous Access** setting.

If you do not enable anonymous access in IIS, requests not meeting the criteria for IIS authentication will be rejected and never reach the ASP.NET application.

The data flow in this scenario is shown in the following illustration.

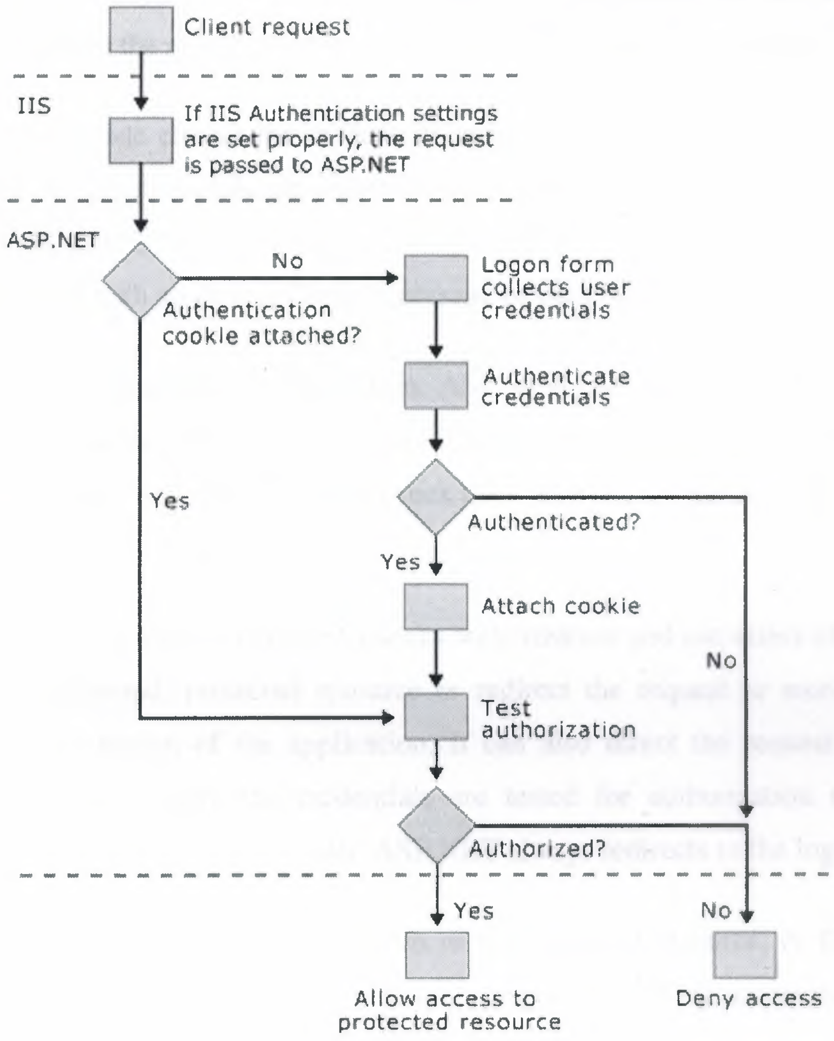

Client request
IIS
If IIS Authentication settings are set properly, the request is passed to ASP.NET
ASP.NET
No
Authentication cookie attached?
Logon form collects user credentials
Authenticate credentials
Yes
Authenticated?
Yes
Attach cookie
No
Test authorization
Authorized?
Yes
No
Allow access to protected resource
Deny access

Figure2.11.2. – Data flow in forms authentication

This illustration shows the following sequence of events:

- A client generates a request for a protected resource.

- IIS receives the request, and if the requestor is authenticated by IIS, or if IIS anonymous access is enabled, the request gets passed on to the ASP.NET application. Because the authentication mode in the ASP.NET application is set to forms in this case, IIS authentication is not used.

- If there is no cookie attached to the request, ASP.NET redirects the request to a logon page, the path of which resides in the application's configuration file. On the logon page, the client user enters the required credentials (usually a name and password).

- The application code checks the credentials to confirm their authenticity, usually in an event handler. If the credentials are authenticated, the application code attaches a ticket (as a cookie) containing the user name, but not the password. If authentication fails, the request is usually returned with an Access Denied message or the logon form is presented again.

- After a ticket is issued by the application, ASP.NET just checks the ticket for validity using a message authentication check. Applications do not need the credentials in the *.config files. In fact, ASP.NET does not check them after the cookie is issued, even if they are present.

- If the user is authenticated, ASP.NET checks authorization and can either allow access to the originally requested, protected resource or redirect the request to some other page, depending on the design of the application. It can also direct the request to a custom authorization module where the credentials are tested for authorization to access the protected resource. If authorization fails, ASP.NET always redirects to the logon page.

- If the user is authorized, access is granted to the protected resource; or the application might require an additional test of the credentials before authorizing access to the protected resource, depending on the design of the application.

CHAPTER THREE

.NET PASSPORT

3.1.Introduction

.NET Passport allows users to create a single sign-in name and password to access any site that has implemented the Passport single sign-in (SSI) service. By implementing the Passport SSI, you won't have to implement your own user-authentication mechanism. Users authenticate with the SSI, which passes their identities to your site securely. Although Passport authenticates users, it doesn't grant or deny access to individual sites i.e. .NET Passport does only authentication not authorization. Passport simply tells a participating site who the user is. Each site must implement its own access-control mechanisms based on the user's Passport User ID (PUID). Here is how .NET Passport Authentication works,

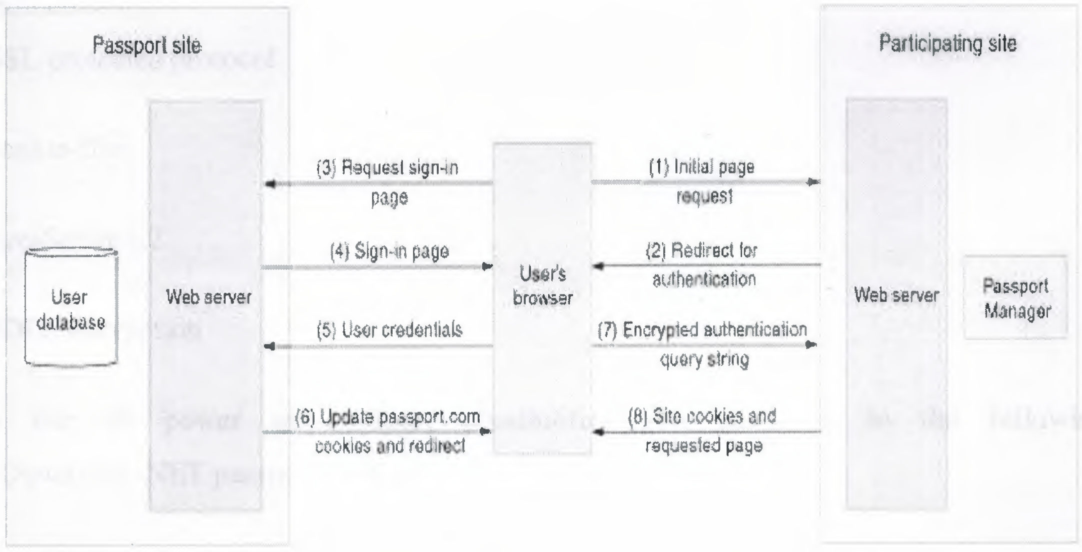


Figure3.1. - .NET Passport Authentication

First user requests any page from his web server. Since user is not authenticated, web server redirect its request for authentication with Sign-In logo. When user presses Sign-In button, request will go to Passport server for Sign-In page. Once the Sign-In page comes to browser, user will enter his authentication details like Passport ID and Password. When user credentials are submitted Credentials are validated in Passport server. Then Cookies are created in server and response is send to the browser with encrypted querystring. Now both cookies and querystring is having details about authentication. Once user is authenticate, he will be taken to page which is requested first.

3.2. Passport Authentication

Passport is a centralized authentication service created by Microsoft, you can use it at any participating web site. One important plus of this technology is that user does not need to remember login data for each site, as it often needed when your usual account already exists on a certain web site. Passport allows to resolve this problem due to using common user database, that is why at web sites supporting .Net Passport you will always enter one and the same login data: your e-mail and password.

Passport authentication uses standard Web technologies for of convenience and confidentiality:

- SSL protected protocol

- cookie-files

- JavaScript 1.2

- 3DES encryption

To use all power of Passport possibilities you need to do the following:
1. Download .NET passport SDK at:

http://msdn.microsoft.com/library/default.asp?url=/downloads/list/websrvpass.asp
2. Then you need to register your site with .Net Passport service:

http://go.microsoft.com/fwlink/?LinkID=9732
If you do not register you possibilities will be extremely limited and you will not be able to get expected result, for example, to logout, you will have to close all browser windows and delete all cookie-files with passport data after this.

3.3. .NET Passport System Requirements

This page describes the Web server and client computer requirements for the Microsoft® .NET Passport service.

The .NET Passport version 2.5 Software Development Kit (SDK) and Passport Manager version 2.5 require Microsoft® Windows® 2000 Server, Microsoft® Windows® XP Professional, or Microsoft® Windows® .NET Server.

3.3.1. Web Server Requirements

To code for the .NET Passport single sign-in (SSI) service using Passport Manager, your Web server must meet the following requirements:

3.3.1.1. Hardware

- X86 computer with a Pentium processor or faster
- 64 megabytes (MB) RAM or more
- Network card and a resolvable Domain Name Server

The .NET Passport SSI hardware requirements are the same as those for serving content on a Web site.

3.3.1.2. Software

- Microsoft Windows 2000 Server or Microsoft Windows XP Professional
- Microsoft® Internet Information Services (IIS) version 5.0 or later
- Microsoft® Internet Explorer version 4.01 with Service Pack 2, or Internet Explorer 5.01 or later

Installation and use on Microsoft Windows 2000 Professional is also possible, but recommended only for testing and development purposes.

- Web server capable of handling HTTP GET and POST requests

- Support for Secure Sockets Layer (SSL) certificates

3.3.1.3. Operations

- Microsoft .NET Passport-issued Site ID and encryption key
- Ability to handle SSL forms
- Ability to serve SSL Web pages

3.3.2. Client Computer Requirements

Microsoft .NET Passport users are able to use .NET Passport services with no client download and on all common browsers.

3.3.3. Browser Compatibility

The .NET Passport services require browsers that support SSL and cookies. For full cobranding support and optimal performance, JavaScript support (in an enabled state) is also required.

Microsoft .NET Passport uses several session cookies that are downloaded to the client computer during the user's session on the Web. The cookies facilitate the SSL sign-in and profile sharing to other Web servers that support .NET Passport. After the user has signed out of .NET Passport, the cookies are removed from the client computer and cannot be retrieved.

The .NET Passport SSI and core profile service has been explicitly tested on Internet Explorer 4.0 and later, Netscape Navigator/Communicator 4.08, 4.5, 4.6, 4.7 and 4.8, and MSN® TV.

The cobranding experience is best viewed on Netscape Navigator 4.08, 4.5, 4.6, 4.7 and 4.8, and Internet Explorer 4.0 or later, or on MSN TV.

Microsoft .NET Passport does not officially support Netscape 6.0 and 6.1, and, since the release of .NET Passport 2.1, no longer explicitly supports Netscape Navigator version 4.05.

3.4. Installing .NET Passport Encryption Keys

Key installation is currently handled through an out-of-band process: Microsoft® .NET Passport personnel send a key installation program to a requesting site as an executable file attachment to an e-mail message, or as a file on disk. The material submitted with the key installation program includes documentation in a Readme file that explains how to run the program and install keys. Save the documentation and the program file in case you need it again in the future (to reinstall or propagate the Passport Manager keys to new computers). Documentation for key installation provided here should match the documentation sent with the program file, but if there are discrepancies, follow the instructions sent with the program.

You should receive your encryption key within three to four business days from the time that you submit your registration. If you have not received the key within this time, contact Microsoft Product Support Services.

You will need two different encryption keys: one for your Development/Test site, and one for your Production site.

You must take steps to restrict access to this encryption key. This includes keeping it in a restricted place on your server and, if you have it on a disk, storing that disk in a restricted location, such as a safe. If you suspect your key has been compromised, contact Microsoft Product Support Services immediately.

The instructions for installing an encryption key differ slightly depending on whether you have previously installed an encryption key on your computer.

3.4.1. The Key Installation Program

The key installation program is a command-line executable file with two basic functions:

- To install the source material of an asymmetric triple-DES key into the server's registry.
- To specify and synchronize the stored key that Passport Manager should use to encrypt and decrypt communication with the .NET Passport network.

Each installation program is specifically compiled to be used by one and only one site and Site ID. The same program can be used to install keys to multiple servers (for example, in configuring a cluster of servers). The name of the program contains the Site ID for which it is intended, as well as the version of the key. For example, if your site's assigned Site ID is 1000, and this is the first time your site has registered and requested a Site ID and encryption key, the name of the key installation program for your site will be Partner1000_1.exe (where 1000 is your Site ID and 1 is the version of the key contained by the program).

3.5. Installing the .NET Passport SDK and Passport Manager

Before you install the Microsoft® .NET Passport Software Development Kit (SDK), make sure your system meets all of the minimum requirements.

You must be signed in as an Administrator when installing the .NET Passport SDK and Passport Manager on a Microsoft® Windows NT®, Microsoft Windows® 2000, or Microsoft Windows XP computer because installing the .NET Passport SDK changes registry settings and installs files to the system folder. Depending on the installation options you choose, running Setup may also stop and start the Microsoft® Internet Information Services (IIS) server process, install virtual directories, or install ISAPI filters to the default Web site root.

3.5.1 Installation Instructions

The following are instructions for starting the Setup program from CD-ROM, a share on a local area network, or the Web:

3.5.1.1. To install from a CD

- Click the Install option that automatically appears when you insert the .NET Passport SDK CD-ROM, and then follow the instructions on the screen.

If the Install option does not appear, or if you have already inserted the CD to browse this documentation, follow these steps:

1. From the Start menu, click Run.
2. Type d:\setup and click OK.

   Substitute the correct drive name for your CD-ROM drive if other than "d:".

3.5.1.2. To install from a share

- Run Setup.exe in the supplied installation path or share.

3.5.1.3. To install from the Web

1. Download the .NET Passport SDK 2.5 installation files.
2. Choose Run this program from its current location and click Yes to install.

   If you are using a Netscape browser, download the SDK to your computer and run the executable locally.

Setup provides you with several installation options. The primary option is to specify which .NET Passport environment this .NET Passport SDK installation should use. In most cases, you should choose the default (Preproduction). If you are deploying Passport Manager to multiple servers in anticipation of going live to Production, or if you are testing isolated servers against Production, the Production option may be appropriate.

If the server on which the .NET Passport SDK is being installed has an earlier version of the .NET Passport SDK installed and has undergone extensive .NET Passport-related configuration already, the Keep existing configuration settings option may be appropriate. In this case, you may wish to select the components to be installed.

By default, the check box that allows you to specify each component to be installed is not selected. This is because Setup chooses default groups of components that are appropriate for each setup type and environment. However, you can select this box to verify and confirm each component for any of the three setup options. The default component lists for each installation option are shown in the following table.

	Dev	Prod'n	Existing
Component			
Sample Sites—Installs the "Adventure Works" Active Server Pages (ASP) sample site as a local virtual directory (VDir) on the server. This VDir is by default "/PassportExample" but can be changed using administrative consoles later. At a later phase in Setup, you can configure this VDir to be created in a specific Web root other than the default Web site root configured on the server. Optional component.	✓		✓
Documentation—Installs only the Readme file specific to the release. All other documentation exists on the .NET Passport Web sites. Optional component.	✓		✓
C++ Support Files—Installs various header and library files required for C++ implementation. These files are not required for normal operation of the **Passport Manager** object on the server or for implementation in script. Optional component.	✓		✓

Passport Manager			
Simple Test Site—Installs the "Simple Test" ASP sample site as a local VDir on the server. This VDir is by default "/PassportTest" but can be changed using administrative consoles later. At a later phase in Setup, you can configure this VDir to be created in a specific Web root other than the default Web site root configured on the server. This directory contains only a few files and can be useful in verifying that the initial configuration of either Development or Production servers is correct. For more information, see Test Site. Optional component. **Note** Use your Preproduction (PREP) .NET Passport to sign in to the test site. Disabling the sample site installation prevents the use of the manual refresh function of the Passport Manager Administration utility. The automatic refresh of the Passport Manager will continue to function as expected.	✓	✓	✓
Passport Dynamic-Link Libraries (DLLs)—Installs the core DLLs that support the **Passport Manager**, **Passport Crypt**, **Passport FastAuth**, **Passport LookupTable**, and **Passport Factory** objects. Also installs *Msppfltr.dll*, an ISAPI filter that is installed to the default Web server root and is required for .NET Passport authentication interactions. Required component.	✓	✓	✓
Administration Tools—Installs the Passport Manager Administration utility and support files, plus an initial version of the Component Configuration Document (CCD) file. Required component.	✓	✓	✓
Other Components			
Dictionary Files—Installs the dictionary files required to look	✓	✓	✓

up friendly strings for GeoIDs, using the **Passport LookupTable** object. This is a potential operational requirement because several .NET Passport profile attributes use GeoIDs as their representation. By default, these dictionary files are installed in the directory specified on installation in a Dictionaries subfolder, with multiple subfolders underneath corresponding to a Web-style language-locale directory structure. For more information, see GeoID Dictionaries. Required component. (Dictionary files may be removed if you decide not to do any reading or writing of profile information in GeoID form.) Does not appear as a component in the custom component dialog.

Table3.5.1.3. – Component list

3.5.2. Deploying Passport Manager

Installing the .NET Passport SDK always installs the Passport Manager server-side object that provides the application programming interface (API) for most of the sign-in and profile service implementation done by a participating site. The .NET Passport SDK can be installed on single or multiple computers used by participating site developers as they integrate .NET Passport sign-in services with their site's existing code.

If you wish to deploy Passport Manager to live Web servers and you wish to install only the bare minimum—the object and files to support your .NET Passport-related code when it has been developed—choose the Production environment option when installing. You may wish initially to install the "/PassportTest" sample site, which can be used for a quick "smoke test" of basic Passport Manager functionality, but remove the VDir after such testing is complete. For more information about using the sample site for testing, see Test Site in SDK.

3.5.3. Uninstalling Passport Manager and the .NET Passport SDK

Uninstall is handled by InstallShield. Uninstall reads the .isu file and removes all unmodified components that were installed initially. Components modified since the initial installation are left in place.

To remove Passport Manager and .NET Passport SDK files

1. From the Start menu, point to Settings, click Control Panel, and then click Add/Remove Programs.
2. From the list box, select Passport Manager, click Remove, and then click Yes to confirm.

- Read the Readme file related to the SDK version being installed. The Readme file contains late additions to the .NET Passport SDK documentation and new details or instructions about installing the .NET Passport SDK and software.
- Reinstalling the .NET Passport SDK will save any existing configuration settings of a previous Passport Manager installation. To be safe, save these settings first, using the Save menu features of the Passport Manager Administration utility. The InstallShield uninstall program removes most .NET Passport components. However, it does not change the IIS configuration settings or remove Access Control Lists (ACLs) on any files or directories, or any files that you have made any changes to (such as modified Sample Site files).
- Reinstalling the SDK may require that previously installed encryption keys be reinstalled..

3.5.4. HTTP-only Cookie Support in Passport Manager 2.5

Cross-site scripting attacks can expose sensitive information about the users of the Web site. In order to help mitigate the risk of cross-site scripting, a new feature has been introduced in Microsoft® Internet Explorer 6. This feature is a new attribute for cookies which helps prevent them from being accessed through client-side script. A cookie with this attribute is called an HTTP-only cookie. Any information contained in an HTTP-only cookie is less likely to be disclosed to a hacker or malicious Web site. New installations of Passport Manager version 2.5 will enable the HTTP-only property in all .NET Passport cookies. When upgrading from an earlier version of Passport Manager, the Setup application will not enable this functionality.

The registry key and value that enable HTTP-only cookies are not imported or exported through the Passport Manager Administration utility.You must manually create a registry entry for each site you want to use this option.

To enable or disable the HTTP-only cookie feature after installation, change the value for the following registry key:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Passport\NotUseHTTPOnly

Set to "0" to enable the HTTP-only property in .NET Passport cookies. Set to "1" to disable the HTTP-only property in .NET Passport cookies. In a multi-site configuration, the value must be set for each site:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Passport\Sites\{YourSiteNames}\NotUs eHTTPOnly

3.6. SSL Sign-In

Secure Sockets Layer (SSL) sign-in provides a way to minimize vulnerability to replay and dictionary attacks. Sending the domain-authority cookies in the header made it remotely possible to intercept them at a packet-sniffing or gateway level, and then to make an attempt to authenticate in another Microsoft® .NET Passport domain as long as the site's time-window requirements were satisfied. Similarly, intercepting the *t* and *p* parameters of a query string and submitting them back to the same site would also spoof a user as far as a site's calls to the IsAuthenticated method are concerned, and would satisfy authentication for the remainder of the time window.

The base .NET Passport service has a vulnerability to a concerted brute-force attack against a user's credentials. The Login servers for .NET Passport implement a slow-down mechanism to discourage casual attempts to gain access to a user's account by repeatedly guessing passwords. This functionality is intended to provide a moderate level of security combined with a reasonable level of usability without creating a vulnerability to a denial of service attack directed against user credentials.

SSL sign-in eliminates these vulnerabilities by adding features to the sign-in process. The IsAuthenticated, LogoTag2, and AuthURL2 methods of Passport Manager and related interfaces accept a new parameter that allows your site to indicate one of the two new levels of authentication security:

- The first level (called *SSL Required*) requires the use of SSL communication for all authentication iterations and also determines whether your domain Secure cookie should be decrypted and checked against the user's .NET Passport Unique ID (PUID) in the Ticket. If there is a mismatch, the user is not considered authenticated. The user should then be sent to the SSL sign-in server to be reauthenticated.

- The second level (called *Security Key*) requires that the user select a secondary credential, known as a Security Key, which is used to sign in to participating sites that require it. This Security Key is considered to be strong because it requires the compromise of the base credentials in order to attack it (effectively increasing the overall protection of the base credential on a protected site), and because it is protected by a lock-out mechanism that goes into effect after five failed attempts. The Security Key does not automatically unlock and requires the user to go through a reset process to regain access to protected sites.

3.6.1. SSL Required

The SSL Required level of SSL sign-in includes the following features to eliminate the packet-sniffing window of vulnerability.

- The Login server URL pages are served from a secure domain if a SSL sign-in is requested by a site. Previously, the sign-in process used a secure address only when silently redirecting to the network-side page used to set the Ticket-Granting cookie.
- SSL sign-in now writes a new "Secure" cookie in the HTTPS header both into passport.com and into your domain. The Secure cookie contains the encrypted PUID (MemberIDHigh and MemberIDLow attributes of the core profile) that uniquely identifies a .NET Passport user. Packet sniffing cannot extract the Secure cookie from the query string and cannot use it to spoof a user.
- When SSL sign-in is used, if the PUIDs in the Ticket and Secure cookies do not match (or no Secure cookie exists), no cookies are copied; the SSL sign-in user interface (UI) is presented, through which the user must resubmit credentials. This prevents the case in which cookies submitted in headers to the Login server could be copied and used on other sites.
- Upon completion of any SSL sign-in, Passport Manager writes the Secure cookie into your domain as an HTTPS cookie. If SSL sign-in is requested, the return URL given must therefore be an HTTPS URL. The Ticket and Profile are still written as nonsecure cookies so that they can still be accessed by non-HTTPS pages on your site if desired.

- The .NET Passport application programming interface (API) will continue to extract the required timestamping information from the regular Ticket. However, if you call the IsAuthenticated method to check for Ticket validity within the time window, you can specify *SecureLevel*=10. Returning True now necessitates that the last sign-in has written the Secure cookie, and that the PUID in the Secure cookie must match that in the nonsecure Ticket.
- Secure and nonsecure versions of the ***t*** and ***p*** parameters submitted to your site, as well as the cookies in headers, are physically different even though they are encrypted in the same key. Unscrupulous users cannot simply submit the nonsecure Ticket and Profile cookies they captured and use them to connect to the secure Login server and get secure Ticket and Profile copies back. The Secure cookie is never transmitted as a query string parameter.
- An efficient use of SSL sign-in would be to request SSL sign-in on the initial check for authentication when you send the user to the Login server. Subsequent checks for authentication can use IsAuthenticated, which does not require a Login server redirect, specifying that the Secure cookie should be checked. Even if someone captured ***t*** or ***p*** parameters to your site, or managed to submit captured cookies in the header to the Login server, the authentication would fail because there would not be a Secure cookie to match the most recent Ticket.

3.6.2. Security Key

The Security Key level of SSL sign-in was introduced in .NET Passport version 2.0 to support sites that had even higher security requirements than handling sign-in over secure channels. This level includes all of the features of SSL Required, plus additional security features.

The first time a user attempts to access a site where the Security Key is indicated, a Security Key selection page is displayed. The user is required to select a four-character Security Key along with three different secret question-and-answer combinations. The Security Key is used by the user to sign in to any SSL sign-in-enabled site that has 'request

Security Key' level security. The secret questions and answers are used if the user needs to change or reset the Security Key.

3.6.3. Why Use SSL Sign-In?

Your site might use SSL sign-in if you are already using fairly long time windows (such as the 10,000-second default) and thus do not expect to be making many further reauthentication checks. Longer time windows increase the opportunity for replay attacks, but help to provide better performance on a site because of fewer required redirects back to the Login server. You could also use SSL sign-in if successful .NET Passport authentication provides your users access to extensive personal or sensitive information, such as transactional abilities or the ability to edit personal data above and beyond the .NET Passport profile.

3.6.4. Requirements for Using SSL Sign-In

The primary requirement for using SSL sign-in is that your site be capable of serving pages using HTTPS protocol. This requires that your site has established and signed certificates. Microsoft .NET Passport servers already use HTTPS in order to write secure Ticket-Granting cookies. Requiring that your users have HTTPS-capable browsers does not increase the requirements for your users and will not intrude in the user experience any further, as long as your certificates are signed and in order. If you are using cobranding on the Login server and other network servers and specifying SSL sign-in requirements, cobranding elements must also be available using HTTPS. Cobranding with SSL sign-in is limited to the logo image URLs you specify when registering your .NET Passport participating site. This limitation is designed to increase the security of the transaction.

3.6.5. Possible Disadvantages of Using SSL Sign-In?

The possible disadvantages of using SSL sign-in are primarily in terms of performance. Serving pages over HTTPS transmits more data because of the added infrastructure of the key exchange, which can take more time to construct and serve at the server end. HTTPS pages also take more time to be loaded and rendered at the client end. If your site does not expose extremely sensitive user data on the basis of .NET Passport authentication, or already uses *TimeWindow* parameters to assure that sensitive data pages are difficult to access through replay attacks, the standard HTTP Login server and standard **Passport Manager** calls are probably adequate for your needs and will be more efficient.

3.7. SSL Certificates

Secure Sockets Layer (SSL) is the most common client-server encryption schema used on the Web today. Most browsers support SSL transactions, and enabling SSL on a server platform is usually simple.

For Microsoft® .NET Passport single sign-in (SSI) service or for obtaining core profile information, SSL support is not strictly required. However, SSL is required to implement the SSL Required level of SSL sign-in.

Your site will need to obtain and install SSL certificates for proper cobranding support under certain circumstances. SSL is also a requirement for cobranding registration pages, because your cobranding information renders in the same page as the .NET Passport-initiated user interface (UI). Microsoft .NET Passport's portion must be served HTTPS so that the user's password is always passed using encryption.

HTTPS is required for cobranding because of the way browsers behave when presented with a page that contains both secure and nonsecure elements. Some browsers give a specific "mixed-content" warning message to protect users from framing or "spoofing" situations, where data could be captured or redirected to some other non-SSL domain hosted within an SSL frame. Because this warning message interferes with the end-user experience, all .NET Passport server pages that use cobranding perform prerendering checks for mixed content and discard any cobranding material supplied by your site that would cause a mixed-content warning to appear. Microsoft .NET Passport defaults are rendered so that the pages are either 100% SSL content or 100% non-SSL content, but never a mixture that triggers the warning.

If you are implementing Kids Passport, the Account Data and Account Removal pages should also be served HTTPS.

3.7.1. Installing SSL Certificates on a Server

This documentation does not discuss obtaining or installing SSL certificates, because the particulars involved vary, depending on which server your site uses and also on whether your site supports its own certificate service or relies on one of the commonly used certification authorities. If you are using Microsoft® Internet Information Services (IIS), see "Certificate Wizard" in the IIS documentation.

3.7.2. Installing SSL Certificates on a Browser

This documentation also does not discuss how to install or approve an SSL certificate on a client browser. Browser users must specifically accept any certificate not issued by a trusted root domain, and the list of certification authorities potentially varies with each browser and each version thereof. Sites operating live in the Production environment should hold certificates issued by one of the commonly accepted certification authorities. If it is necessary to install or accept a specific certificate (for example, if the browser is being used to view a site using a test certificate not issued by a common certification authority), consult the documentation that comes with that particular browser and review the procedures for accepting or importing SSL certificates.

3.8. Passport Authentication Provider in ASP.NET (IIS 6.0)

Passport authentication is a centralized authentication service provided by Microsoft that offers a single logon and core profile services for member sites. This benefits the user because it is no longer necessary to log on to access new encrypted resources or sites. If you want your site to be compatible with Passport authentication and authorization, this is the provider you should use. This topic provides some introductory material about Microsoft .NET Passport and the ASP.NET support for it. For more information, see the

Passport documentation located at http://www.passport.com. In order to access the documentation, you must get a Passport and register.

Passport is a cookies-based authentication service. A sample transaction conversation using Passport authentication might look similar to the following:

1. A client issues an HTTP GET request for a protected resource, such as http://www.contoso.com/default.aspx.
2. The client's cookies are examined for an existing Passport authentication ticket. If the site finds valid credentials, the site authenticates the client. If the request does not include a valid authentication ticket, the server returns status code 302 and redirects the client to the Passport Logon Service. The response includes a URL in the query string that is sent to the Passport logon service to direct the client back to the original site.
3. The client follows the redirect and requests the original resource again, this time with the Passport cookie.
4. The Passport logon server presents the client with a logon form.
5. The client fills out the form and does a POST back to the logon server, using Secure Sockets Layer (SSL).
6. The logon server authenticates the user and redirects the client back to the original URL (http://www.contoso.com/default.aspx). The response contains an encrypted Passport cookie in the query string.
7. The client follows the redirect and requests the original protected resource again, this time with the Passport cookie.
8. Back on the originating server, the PassportAuthenticationModule detects the presence of the Passport cookie and tests for authentication. If successful, the request is then authenticated.

3.9. Setting Up .NET Passport in IIS 6.0 (IIS 6.0)

Before setting up Microsoft .NET Passport authentication on your Web sites in a production environment, you are required to test IIS against .NET Passport preproduction servers. By working through this process, you confirm that your IIS server and the .NET Passport server are communicating correctly, your site(s) is registered with .NET Passport (which might involve signing forms and agreements), and your Web sites have the required site IDs. You must complete each process for every Web site you want to enable with .NET Passport authentication.

This topic includes the following information:

- NET Passport Environments
- Configuring for Preproduction (PREP)
- Configuring for Production
- Passport Manager Administration Utility

With .NET Passport authentication on members of the Windows Server 2003 family, the default .NET Passport SecureLevel setting is 10. This means that new sites using .NET Passport authentication (and default settings) require an Secure Sockets Layer (SSL) server certificate. You can change the SecureLevel setting for a site by changing a registry value.

Using Registry Editor incorrectly can cause serious problems that require reinstalling the operating system. Because Registry Editor bypasses the standard safeguards that prevent you from entering settings that are conflicting or likely to degrade performance or damage your system, exercise caution when making changes to the registry. Microsoft cannot guarantee that problems resulting from the incorrect use of Registry Editor can be solved.

To change the SecureLevel setting for the default Web site

Enter a new value in the registry for the following key:

- HKEY_LOCAL_MACHINE\Software\Microsoft\Passport\SecureLevel

To change the SecureLevel setting for any Web site other than the default Web site

Enter a new value in the registry for the following key:

- HKEY_LOCAL_MACHINE\Software\Microsoft\Passport\Sites\<*Site Name*>\SecureLevel

3.10. .NET Passport Environments

Setting up .NET Passport on your Web sites involves testing and configuring IIS against the following three .NET Passport environments:

- Default installation environment: By default, the Microsoft .NET Passport SDK and Internet Information Services (IIS) (with the Passport Manager object) are configured for testing. The test or default installation environment uses a site ID of 1 (one), and has a default encryption key instead of a private, site-specific key.
- Preproduction: The Preproduction (PREP) environment enables sites to verify their development efforts against .NET Passport servers without access to real-world .NET Passport user identifications and profiles. While in PREP mode, prospective .NET Passport participating sites can manipulate data, create new users, and run other tests against the user base without affecting existing deployed .NET Passport sites.
- Production: The .NET Passport Production environment is shared by all working and approved .NET Passport participating sites after they are deployed to the public.

3.10.1. Testing and Preproduction (PREP)

You can determine whether your site is in the default installation environment by checking the Site ID field in the Passport Manager Administration utility. If the value in the Site ID field is 1, your site is in the default installation environment. While your site is in this environment, you can perform an initial evaluation of the single sign-in service (SSI) and test any dynamic content on your site against the Passport Manager object. You can develop your Web site and any applications on your site before registering your site and requesting a site ID, or while you are waiting for the site ID and key after registration. After you have a site ID, there should be no reason to use the default installation environment again. The restrictions imposed by this mode are intended only to prevent developers from accessing certain .NET Passport features before having signed the necessary agreements or contracts.

The preproduction environment requires a .NET Passport preproduction logon account. To set up a preproduction logon account.

3.10.2. Limitations in the Default Installation Environment

The following operational limitations exist in the default installation environment:

- Cannot read full core profile data or profile cookies: A Web site running in the default installation environment indicates that the administrator has not yet signed the necessary agreements that specify a site's requirements for privacy. For this reason, a site running in the default installation environment cannot access all .NET Passport user information. .NET Passport user information displayed on a test site contains several fields of default placeholder values, as generated by the logon server.

- Cannot use co-branding: Co-branding support is dependent on several URLs that you must provide when registering as a .NET Passport participating site. Until you register and provide these URLs, the co-branding logo and text exist as placeholders. For details about how to implement co-branding after your site has registered as a .NET Passport participating site, see ".NET Passport Cobranding Overview" in .NET Passport SDK Help.
- Cannot sign users out from sign-out page: On a registered .NET Passport site, when members choose to sign out by clicking the .NET Passport sign-out link, they are redirected to a central page that enables deletion of all .NET Passport cookies from all of the sites the member visited during the session. This is not the case in the default installation environment, so .NET Passport cookies remain on the user's browser until the user closes the browser entirely. If the .NET Passport user chose to save their password (thus making all session cookies persistent cookies), .NET Passport cookies written in the test site's domain are still not deleted. For this reason, when testing browser behavior in the default installation environment, you may occasionally need to quit the browser to reproduce a clean .NET Passport sign-in.

3.10.3. Configuring for Preproduction

Real-world .NET Passport users probably do not have existing e-mail names within the PREP environment. Part of your site's development and testing effort might require that you first create a store of users within the PREP environment and then use these accounts for testing against the .NET Passport authentication and profile-access portions of your site's code. The .NET Passport server code that runs the services in the PREP environment is essentially identical to the code that runs *live* in the Production environment; Production and PREP are released in tandem. This means that *going live* should be a matter of simply stopping the servers on a site currently running against the PREP environment, reconfiguring Passport Manager on those servers to run against the Production environment instead, and then restarting the servers.

For more information about getting a PREP site ID, see "Registering Your .NET Passport Site" in the .NET Passport SDK (starting with version 2.1) Help. For more information about PREP Passports, see "Get a PREP Passport" and "Sign in to PREP Passport" in .NET Passport SDK (starting with version 2.1) Help. Also, see "Going Live: Deploying Passport Manager and Site Code."

Upon completion of the registration process, you will see a confirmation page that indicates your new .NET Passport sign-in name. This is the name you will use while in the development phase.

3.10.4. Configuring for Production

All Microsoft .NET Passport users already have an account in this environment. Users of Microsoft MSN Internet Access and MSN Explorer, as well as users who registered directly at the Passport.com Web site, also have passports in this environment. Within the Production environment, sites can expect to handle many real-world .NET Passport users who might already have been authenticated at various other Passport participating sites.

To deploy the .NET Passport code of your site, the Passport Manager server-side object must be reconfigured to work against the Production environment. To deploy against the Production environment, your site must also register for a site ID in the Production environment, and must supply some configuration information about your site that is specific to the Production deployment. Approval of a site ID in the Production environment might require some basic remote checking of your site's .NET Passport implementation by the .NET Passport team, and might also require some additional contracts or agreements.

Do not place co-branded URLs in a virtual directory on which .NET Passport authentication is enabled. If a co-branded URL resides in a virtual directory on which .NET Passport authentication is enabled, client requests are challenged to authenticate before the co-branded information is passed to the client. This can result in a situation where a client might not be able to authenticate and view the desired page.

To deploy against the .NET Passport Production environment, see "Registering Your Passport Web Site" in the .NET Passport SDK (starting with version 2.1) Help. To disable the default installation environment and enable normal operation, enter a different site ID in the Site ID field and commit changes. Note that if you change the site ID to something other than 1, a matching encryption key must also be installed. The key, as well as the unique site ID assigned to any particular participating site, can be obtained only by registering as a .NET Passport participating site.

Be sure to reconfigure your Windows XP clients for .NET Passport production mode as well. For more information, see ".NET Passport SDK: System Requirements" in the .NET Passport SDK (starting with version 2.1) Help. More specifically, read the section titled "Client Computer Requirements."

3.10.5. Passport Manager Administration Utility

The Microsoft Passport Manager Administration utility is a graphical user interface (GUI) alternative to editing the registry when changing Passport Manager object settings. The most common use of the Passport Manager Administration utility is to change the default settings. For details about how defaults can be set and affect the behavior of Passport Manager methods, see "Setting Passport Manager Defaults" in .NET Passport SDK Help.

To access the Passport Manager Administration utility

1. From the Start menu, click Run.
2. In the Open box, type msppcnfg.exe, and click OK.

3.11. Passport Manager Administration Utility

The Passport Manager Administration utility is a graphical-interface alternative to editing the registry to change Passport Manager settings. The most common use of the Passport Manager Administration utility is to change the default parameter values used by PassportIdentity methods if optional parameters are omitted when the methods are called. The term "Passport Manager" refers to the interface with Microsoft® .NET Passport servers that is controlled in ASP.NET code by instances of the PassportIdentity class.

Although some parameters of .NET Passport methods are overloaded and can be considered optional, these methods (for example, PassportIdentity.LoginUser) still generate URLs or results that reflect default values when called. These default parameter values can be used to provide consistent site-wide values, such as the required time window within which all users must be authenticated.

The following are the primary defaults that affect PassportIdentity implementation:

- Time Window

  Change this value to the default time window you wish to set. The time window specifies how old a Ticket can be before the IsAuthenticated method returns False for an otherwise valid Ticket. It also qualifies how old a Ticket as submitted to the Login server can be without requiring a refresh for purposes of the LoginUser, AuthURL2, or LogoTag2 methods. The default as installed is 1800 seconds (equal to 30 minutes).

- Force Login

  Change this value to be either True (checked) or False (unchecked).

  True specifies that the Ticket as read by the IsAuthenticated property must represent a refresh in which the user physically enters a password, not a silent refresh, in order for IsAuthenticated to return True itself. A True default for Force Login also changes the behavior at the Login server for any user sent there by URLs derived from the LoginUser, AuthURL2, or LogoTag2 methods. Specifically, the Login server does not silently refresh and always asks for a password if the existing Ticket is past the specified time window.

  A False default for Force Login means that any Ticket within the time window is acceptable either to the IsAuthenticated property or to handling at the Login server as accessed by means of the AuthURL2 or LogoTag2 output URLs (or a LoginUser redirect).

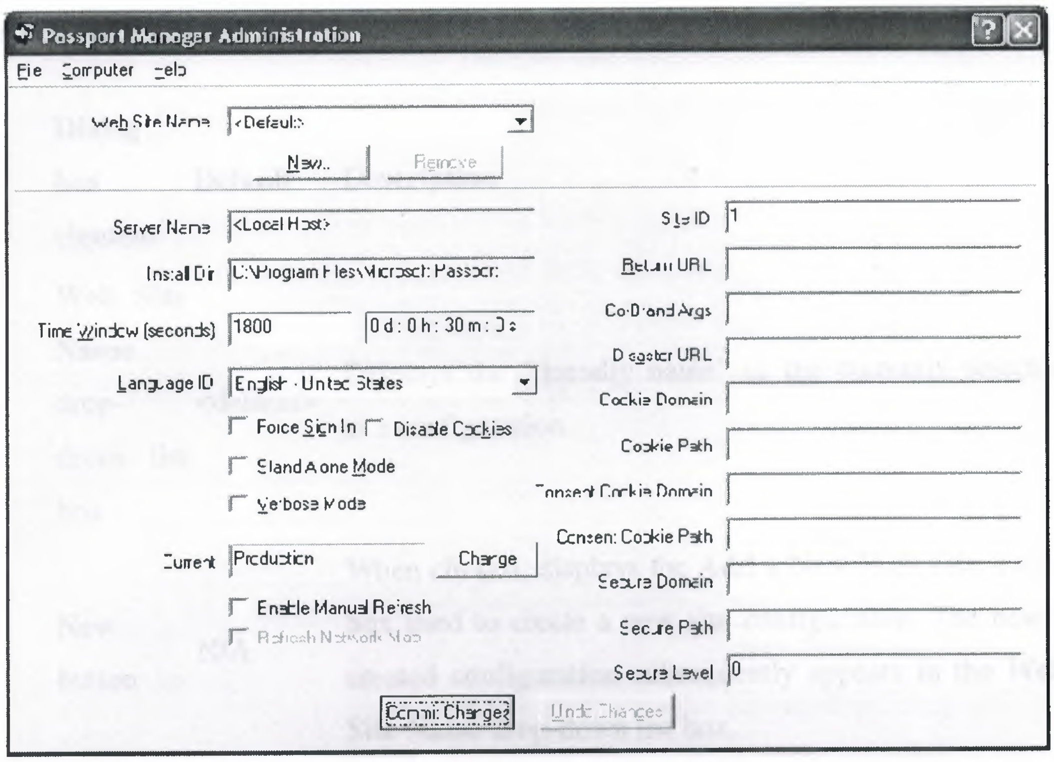
- Language ID

  If a server on your site is dedicated to a particular language or locale, it may be useful to set the Language ID on that server always to be a consistent value rather than declaring the locale ID (LCID) in each method call. The Language ID declares the language that .NET Passport service pages (such as Sign-in and Registration) render in when obtained with the URL results of the LoginUser, AuthURL2, LogoTag2, or GetDomainAttribute methods. There may still be reasons to declare an LCID by method call or user access if your site expects to handle multiple languages and uses either .NET Passport profile information or browser sniffing to determine the user's probable language choice.

To run the Passport Manager Administration utility

1. Click Start, point to Programs, and then to Microsoft Passport.
2. Click Passport Manager Administration Utility.

The following figure shows the main user interface of the Passport Manager Administration utility.



Web Site Name: <Default>
Server Name: <Local Host>
Install Dir: C:\Program Files\Microsoft Passport
Time Window (seconds): 1800 | 0 d : 0 h : 30 m : 0 s
Language ID: English - United States
☐ Force Sign In ☐ Disable Cookies
☐ Stand Alone Mode
☐ Verbose Mode
Current: Production
☐ Enable Manual Refresh
☐ Refresh Network Map
Site ID: 1
Return URL:
CoBrand Args:
Disaster URL:
Cookie Domain:
Cookie Path:
Consent Cookie Domain:
Consent Cookie Path:
Secure Domain:
Secure Path:
Secure Level: 0

Figure3.11. - Passport Manager Administration utility

3.11.1. Passport Manager Administration Utility

The following table describes the various elements of the user interface (UI) and their default values (if applicable).

Dialog box element	**Default**	**Description**
Web Site Name drop-down list box	<default>	Displays the "friendly name" of the currently selected site configuration.
New button	N/A	When clicked, displays the **Add a New Web Site** dialog box used to create a new site configuration. The newly created configuration subsequently appears in the **Web Site Name** drop-down list box.
Host Name text box	Blank	Displays the host name of the currently selected site configuration. Not displayed for the default site's configuration.
IP Address text box	Blank	Displays the IP address of the currently selected site configuration. Not displayed for the default site's configuration.
Remove button	N/A	When clicked, removes the site configuration currently selected in the **Web Site Name** drop-down list box.
Server Name text box	Localhost	Displays the name of the server hosting Passport Manager. (The value is read-only here, but can be set using the **Computer** menu).
Install Dir	c:\Program Files\Micr	Displays the directory where the .NET Passport Software Development Kit (SDK), but not the Passport DLL, is

text box	osoft Passport	installed (a read-only value).
Time Window combo box	1800	Used to fill in defaults for the *iTimeWindow* parameter used in **PassportIdentity** object methods **LoginUser**, **AuthURL2**, **IsAuthenticated**, and **LogoTag2**, among others. *iTimeWindow* specifies the maximum duration allowed between either silent or forced manual sign-in to a participating site. *iTimeWindow* must be >=100 and <=1000000.
Languag e ID combo box	1033, English	This drop-down list box allows you to set the language preference sent by the **PassportIdentity** object on requests to the Login server. This becomes the default value of the *iLangID* parameter, also called the LCID, used by **PassportIdentity** object methods **AuthURL2** and **LogoTag2**, among others. Users see different localized text at the Login server depending on this LCID. If the LCID you require is not available, you can add it by selecting all the text in the text box and then typing in a new LCID number. If the LCID is included in the Passport Manager Administration utility support code, this will display the language name next to the new LCID.
Force Sign In check box	Unchecked	Used to fill in defaults for *fForceLogin* parameter used in **PassportIdentity** object methods **LoginUser**, **AuthURL2**, **GetIsAuthenticated**, and **LogoTag2**, among others. Specifies whether user sign-ins falling outside of *iTimeWindow* are allowed to be silent or require the user to enter the password again.
Disable Cookies	Unchecked	Disables the use of cookies with the **PassportIdentity** object. If you disable cookies, all profile information

check box		must be passed page-to-page using the query string. This practice is not recommended, because it requires a large amount of query string handling, and writing code to handle requests at the HTTP level.
Stand Alone Mode check box	Unchecked	When checked, sets the Passport Manager installation to stand-alone mode, which is for cases in which all Login servers at .NET Passport are down. Stand-alone mode treats all existing user cookies as valid and does not contact the Login server or compare timestamps, effectively disabling any application programming interface (API) methods that would ordinarily reject a user with an old Ticket.
Verbose Mode check box	Unchecked	When checked, sets this Passport Manager installation to verbose mode, which helps to enable better debugging through a text log of all **PassportIdentity** method calls and operations. Verbose mode should be used only to debug specific problems, because it slows performance and generates a large text log if left on for extended periods.
Current (environment) controls	Depends on initial installation	Use this control to reconfigure the environment that Passport Manager will run against.
Change button	N/A	Click this button to open the **Choose Environment** dialog box in which you can select an environment from one of the options in the Environment section.
Enable Manual Refresh check box	Checked	Sets a registry entry (**NSRefresh**) when checked. This allows the passporttest Web site (http://localhost/passporttest/default.asp) to receive the latest version of the *Partner.xml* file. You cannot get the

		latest *Partner.xml* from the passporttest Web site without setting this registry entry because the default.asp page checks the registry entry prior to instantiating **PassportIdentity** object and calling **Refresh**.
Refresh Network Map check box	Unchecked	When checked, this will cause the Passport Manager Administration utility to launch the passporttest Web site on localhost when the **Commit Changes** button is pressed. The appropriate query string parameters are passed to the site, which will instruct it to download the *Partner.xml*. After clicking **Commit Changes**, a MessageBox is displayed indicating that the Network Map is about to be updated. Clicking **OK** will cause the passporttest site to be launched as: `http://localhost/passporttest/default.asp?Refresh=True&Env=Prep&NewID=False`
Site ID text box	1	Displays the participating Site ID number. This Site ID qualifies all communication with the .NET Passport servers. The .NET Services Manager will provide your participating site with an executable program that can be run on each Passport Manager-enabled server to install site-specific encryption keys. At this time, you should set your Site ID to the value provided in instructions sent with the key installation program, and this is generally the only time that the **Site ID** value should be edited. Attempting to change this value in the Passport Manager Administration utility will cause a warning message to be displayed. The initial Site ID of 1 means that this **Passport Manager** is running in test mode.
Return URL text box	Blank	Used to fill in defaults for the *strReturnURL* parameter given in **PassportIdentity** object methods **LoginUser**, **AuthURL2**, and **LogoTag2**, among others.

Cobrand Args text box	Blank	Used to fill in defaults for the *strCoBrandedArgs* parameter given in **PassportIdentity** object methods **LoginUser**, **AuthURL2**, and **LogoTag2**, among others.
Disaster URL text box	Blank	Specifies the URL used to replace any .NET Passport network server URL in cases in which the Passport Manager is configured to run in stand-alone mode.
Cookie Domain text box	Blank	The domain to which the **PassportIdentity** object should write Ticket and Profile cookies. Typically this should be the same as the value given by **Request.ServerVariables("SERVER_NAME")** for a page that uses this **PassportIdentity** object.If you are writing cookies to a subdomain of your own domain, you should precede the domain path with a dot (".") character. For example, if your site is shopping.example.com and you want the **PassportIdentity** object to set cookies in example.com, set the **Cookie Domain** entry to .example.com (note the preceding dot, a requirement for some Netscape browsers).
Cookie Path text box	Blank	Within the domain, the path to which Ticket and Profile cookies are written.
Consent Cookie Domain text box	Blank	The domain to which the **PassportIdentity** object should write Consent cookies. Typically, this should be left blank, or at least be the same as the value given by **Request.ServerVariables("SERVER_NAME")** for a page that uses this **PassportIdentity** object. Consent cookies are written by **Passport Manager** only if you specifically inform your .NET Passport representative that you intend to enable a "property" model, where

		several properties share a Ticket for authentication but each property is distinct for purposes of Kids Passport and consent. If Passport Manager is writing the Consent cookie, Consent domain entry should not match the Cookie domain, and should instead be written to a tertiary domain. Each property must be distinct for purposes of establishing unique consent status. The Consent cookie written to the tertiary domains will contain the consent status granted each individual property. The domain must still fall within the root domain specified at registration time.
Consent Cookie Path text box	Blank	Within the domain, the path to which Consent cookies are written.
Secure Domain text box	Blank	The domain to which the **PassportIdentity** object should write Secure cookies. Secure cookies are used as verification for SSL sign-in when calling the **IsAuthenticated** property.
Secure Path text box	Blank	Within the domain, the path to which Profile cookies are written.
Secure Level text box	Blank	Displays the default per-site security level of the sign-in. Valid values are: 0 (or blank) Sign-in UI is served HTTP from the .NET Passport domain authority (default). Even using this option, there will be an intermediate

		transition to HTTPS on the .NET Passport server side to enable writing a Secure cookie that is set by domain authorities for the persistent sign-in option.
		10 Sign-in UI is served HTTPS from the .NET Passport domain authority. Requires that return URL be an HTTPS URL; otherwise, the authentication will fail.
		100 Sign-in UI is served HTTPS from the .NET Passport domain authority, and sign-in process now requires submission of a security key in addition to password. Requires that return URL be an HTTPS URL; otherwise, the authentication will fail. For more information, see SSL Sign-In.
Commit Changes button	N/A	Click to assign values in current Passport Manager Administration utility controls to the Passport Manager and the associated **PassportIdentity** instances, and assign these values to the underlying registry keys. You must click **Commit Changes** in order to actually make any configuration changes.
Undo Changes button	N/A	Click to redisplay saved registry values. This does not undo any changes committed to the registry. Create .ppi files as backups if there is a need to revert to previously saved or overwritten registry settings.

Table3.11.1. – Dialog box table

- Consistency between servers

When changed, many settings in the Passport Manager Administration utility display a message window informing you that changes to specific Passport Manager values should be consistent across servers in your site. In general, a site's Passport Manager-enabled servers should all have the same Passport Manager Administration utility settings, so that cookies written by one installation will not react differently when the user returns to the site and rotates to another server. You can use the Select Server command from the Computer menu to access and configure any remote server that is accessible on the LAN and already has Passport Manager installed.

- Cookie path and domain

  Make sure that all required installations of Passport Manager on your site can both read and write to a common path and domain location. If they cannot, cookies set by one PassportIdentity instantiation may not be able to read the cookies set by another. A symptom of this would be that a user does not always appear to have a Ticket even though he or she signed in to .NET Passport and your site before.

- Saving text-file versions of Passport Manager Administration utility settings

  From the File menu, the Save and Save As commands save all current Passport Manager Administration utility values as a text file with the extension ".ppi," but they do not commit the values to the registry. (To commit values to the registry, click the Commit Changes button.) These text (.ppi) files are useful for configuring multiple servers in exactly the same way without having to extract the required keys to make a .reg file. From the File menu, choose Open to load a previously saved .ppi file and assign these values to the various controls in the Passport Manager Administration utility. A common use for this feature would be to save all settings on a single computer as a .ppi file, use the Computer menu to access a remote server's configuration, load the saved .ppi file, click Commit Changes to commit those settings to the remote computer, and repeat as necessary to propagate all settings identically across multiple computers.

- Cobrand Args and Return URL

  Setting Cobrand Args and Return URL as site-wide defaults does not make sense in most cases; these should probably be left blank and set on each method call. Other attributes as set in the Passport Manager Administration utility do not necessarily affect PassportIdentity method defaults directly.

3.12. .NET Passport Cookies

Using the **Passport Manager** object, Microsoft® .NET Passport reads and writes cookies that are used to persist authentication and profile information on the user's computer in order to provide the user with the experience of an authentication session.

3.12.1. Domain-Authority Cookies

The following five cookies are written to the .passport.com domain. None of these cookies can be accessed directly by a participating site.

Common Name	**Label**	**Description and Contents**
Profile	MSPProf	Encrypted with .NET Passport key. Contains each of the core profile attributes.
Secure	MSPSecAuth	Set when the SSL sign-in feature is enabled on your site. Contains a token that allows Passport Manager to help verify that the SSL sign-in process was invoked.
Ticket	MSPAuth	Encrypted with .NET Passport key. Contains the .NET Passport timestamps, saved-password flag, key version verification, and any flags set by network servers.
Ticket-Granting	MSPSec	Sent using HTTPS protocol for all browsers that allow HTTPS cookie writes. Contains the SSL-encoded .NET Passport Unique ID (PUID) and password, used for silent sign-in.
Visited Sites	MSPVis	Used by the Login server to compile the list of sites that must be signed out from when the user clicks any sign-out link. Each new .NET Passport participating site visited has its Site ID written to this cookie. No encryption.

Table3.12.1. - Command name

3.12.2. Participating Site Cookies

The following two cookies are written to the domain and path to which the participating site's Passport Manager object is configured. By default, cookies are written to the participating site's root domain. Subdomains can be used, by entering values in the Cookie Domain and Cookie Path text boxes of the Passport Manager Administration utility. The participating site is responsible for deleting all cookies, including those in subdomains

Common Name	Label	Description and Contents
Consent	MSPConsent	Encrypted with .NET Passport key. Contains the user's consent status. This cookie is set in the domain and path specified in the **Consent Cookie Domain** and **Consent Cookie Path** fields in the Passport Manager Administration utility.
Profile	MSPProf	Encrypted with .NET Passport key. Contains each of the core profile attributes.
Secure	MSPSecAuth	Encrypted with .NET Passport key. Set when the SSL sign-in feature is enabled. Contains a token that allows Passport Manager to help verify that the SSL sign-in process was invoked.

Table 3.12.2. - Common name

3.13. .NET Passport Unique ID

For the purpose of unique identification in a site's internal, private database, Microsoft® .NET Passport users should be referenced by their .NET Passport Unique ID (PUID), which is a combination of two .NET Passport profile attributes, MemberIDHigh and MemberIDLow, and is reflected as a 16-character string in the HexPUID property of the PassportIdentity class.

The PUID is used for three reasons: security, uniqueness, and the lack of support for 64-bit unsigned integers in commonly used development languages.

Although the user's sign-in name does uniquely identify users for the purpose of signing in to .NET Passport, for security reasons, the sign-in name is not stored anywhere in a .NET Passport profile, and is therefore unavailable for method calls. The sign-in name is verified, but not distributed to participating sites. Allowing sign-in names themselves to be stored in any database other than the .NET Passport central database introduces the risk of misuse or theft of users' personal data.

The PUID is also the only .NET Passport profile element that is guaranteed to be unique. The MemberName attribute, for example, lacks such a guarantee, as many people have the same name, and so is inappropriate for this purpose. The PreferredEmail is not a required attribute, so it cannot be used as a unique identifier. Furthermore, sign-in names may potentially be recycled or even reassigned, but the network-side unique identifiers (and thus the resulting PUID) are never reused.

The PUID is generated in two parts for the user's profile due to more common support for 32-bit than for 64-bit data types. Although this requires that the two values be combined, it avoids forcing developers to write applications only in languages with 64-bit support. It is strongly recommended that participating sites use a common derivation algorithm throughout an application for constructing a PUID.

The use of a common algorithm for deriving PUIDs from the MemberIDHigh and MemberIDLow values is particularly important if several participating sites share a common private database but do not necessarily share a complete code base. There are several derivations that will produce a unique PUID value when combining the two 32-bit values. For implementations in C# and for databases, the best alternative is usually to store the PUID as a native 64-bit data type. That is, the string value returned by the HexPUID property of the PassportIdentity object should be converted into an unsigned long integer.

Microsoft® Visual Basic® and Visual Basic Scripting Edition (VBScript) do not support an unsigned integer data type or a true `INT` type of 64-bit size. This complicates the task of implementing a derivation if both C# and script are used to read and write PUIDs to a common database. The preferred solution is to represent the PUID not as a true numeric value, but as a string representing the concatenation of the hexadecimal character representation of the two 32-bit values. This is why the HexPUID property is of type `string` rather than type `ulong`. Members 0-7 of the resulting vector are always the MemberIDHigh component and members 8-15 of the string vector are the MemberIDLow component.

3.14. Enabling .NET Passport Authentication in IIS 6.0 (IIS 6.0)

When .NET Passport is enabled, requests coming into IIS must contain .NET Passport credentials either on the query string or within a cookie. The credentials also have to be valid, meaning the ticket has not expired. If IIS does not detect .NET Passport credentials, requests are redirected to the .NET Passport sign-in page.

.NET Passport uses cookies, which contain information that can be compromised. However, .NET Passport authentication can be used over a Secure Sockets Layer (SSL) connection, which reduces the potential of replay attacks.

You must be a member of the Administrators group on the local computer to run scripts and executables. As a security best practice, log on to your computer by using an account that is not in the Administrators group, and then use the runas command to run your script or executable as an administrator. At a command prompt, type runas /profile /User:*MyComputer*\Administrator cmd to open a command window with administrator rights and then type cscript.exe*ScriptName* (include the script's full path and any parameters).

Procedures are follow:

To enable .NET Passport authentication on a Web site

1. In IIS Manager, expand the local computer, expand the Web Sites folder, right-click the Web site on which you want to enable .NET Passport authentication, and click Properties.
2. Click the Directory Security tab.
3. In the Anonymous access and authentication control section, click Edit.
4. Select the .NET Passport Authentication check box. There are fundamental differences in the way .NET Passport validates user credentials, so .NET Passport cannot be used with other authentication methods. When .NET Passport authentication is selected, all other authentication methods are unavailable.
5. Click **OK**.

CONCLUSION

At the beginning the microsoft .NET is explained and is examined. This project answers the what the .NET , ASP.NET and how the .NET Passport is implementing on the web in real life. When these questions are answered, some details of windows and IIS are searched and declared in the project..NET Services are considered to understand clearly the system of Microsoft .NET then Windows Server System is represented to learn what .NET is boult on.

ASP.NET is considered becouse of the implemention .NET Passport is written with the it. ASP.NET is choosed to write the web application because it is better then other languages. The adventages and features of ASP.NET is introdused and examined one by one. For the .NET Passport the ASP.NET Security is declared and explained how it use. How the ASP.NET is declared in IIS that is written step by step. Some command in ASP.NET are examined to understand the .NET Passport implementation which is written in project. Special arrangment and declaration is written about IIS 6.0 in the project.

.NET Passport Architecture is illustrated and explained how it work on web clearly in the project. System requirement is considered for .NET Passport, all details are written for the implementation. At the end, implementation .NET Passport is written and illustrated in the project.

RERERENCES

http://www.microsoft.com

http://www.passport.com

http://www. xlinesoft.com

http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS

http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/1f74eac5-9005-4f91-9137-f63b73eefde8.mspx

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpgenref/html/gngrfASPNETConfigurationSectionSchema.asp

http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/1f74eac5-9005-4f91-9137-f63b73eefde8.mspx

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/passport25/NET_Passport_VBScript_Documentation/Testing_And_Troubleshooting/Troubleshooting/tshoot3.asp

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/passport25/NET_Passport_VBScript_Documentation/NET_Passport_Fundamentals/Installation.asp

Microsoft .NET Passport SGK, NET Passport Service Guide Kit version 2.5

Mesbah Ahmed, Chris Garrett, Jeremy Faircloth, Chris Payne, ***AS P.NET** Web Developer's Guide,* 800 Hingham Street Rockland, MA 02370, 2002

Danny Ryan and Tommy Ryan, *ASP.NET Your visual blueprint for creating Web applications on the .NET Framework,* Hungry Minds, Inc. 909 Third Avenue NY 10022, 2002

Scott Mitchell, Bill Anders, Rob Howard, Doug Seven, Stephen Walther, Christop Wille, and Don Wolthuis, *Draft,*SAMS, 201 West 103rd St., Indianapolis, Indiana, 46290 USA,2001

Jeff Prosise, *Programming Microsoft .NET,* Microsoft Press, 2002

APPENDIX A
WEB CODES AND FIGURE

4.1. Web.Config

Passport\Web.Config

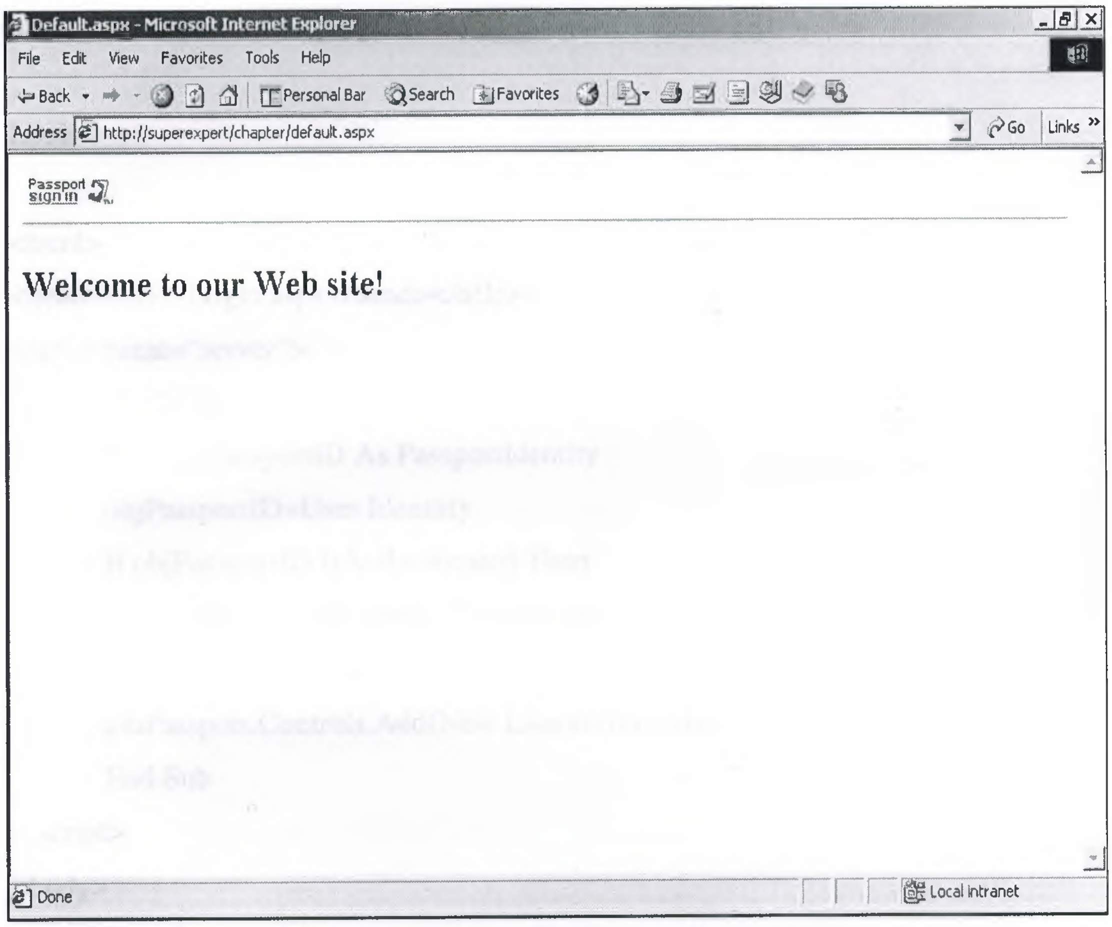
```
<configuration>
<system.web>
<authentication  mode="Passport">              → The Authentication type is declared.
<passport redirectUrl="/Login.aspx"/>
</authentication>
</system.web>
</configuration>
```

4.2. Default.aspx

Passport/Default.aspx

```
<html>
<head><title>Default.aspx</title></head>
<body>
<script runat="server">
        Sub page_Load
        Dim objPassportID As PassportIdentity


        objPassport=User.Identity
        If objPassportID.GetFromNetworkServer Then
                Response.Redirect(Request.Path)
        End If
        plhPassport.Controls.Add(New LiteralControl(objPassportID.LogoTag2()))
        End Sub
</script>
<asp:PlaceHolder ID="plhPassport" runat="Server"/>
<hr>
<h2> Welcome to our Web site!</h2>
</body>
</html>
```


Default.aspx - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Personal Bar
Search
Favorites
Address
http://superexpert/chapter/default.aspx
Go
Links
Passport sign in
Done
Local intranet

Welcome to our Web site!

Figure 4.2. Default page

4.3. Login.aspx

Login.aspx

```
<html>
<head><title>Login.aspx</head></title>
<script runat="server">
        Sub Page_Load
        Dim objPassportID As PassportIdentity
        objPassportID=User.Identtity
        If objPassportID.IsAuthenticated Then
                Response.Redirect("Default.aspx")
        End If
        plhPassport.Controls.Add(New LiteralControl(objPassportID.LogoTag2))
        End Sub
</script>
<body>
<h2> Please Login:</h2>
<p>You have requested a restricted resource.<br>
To continue, please login by clicking the following button:
</P>
<asp:PlaceHolder ID="plhPassport" runat="server" />
</body>
</html>
```

4.4. News.aspx

News.aspx

```
<html>
<head><title>News.aspx</title></head>
<script runat="server">
        Sub Page_Load( s As Object, e As EventArgs)
        Dim objPassportID as PassportIdentity
        objPassportID=User.Identity
        If objPassportID.GetFromNetworkServer Then
                Response.Redirect(Request.Path)
        End If
        plhPassport.Controls.Add(New LiteralControl( objPassportID.LogoTag2()))
        If objPassportID.IsAuthenticated Then
                pnlAuth.Visible=True
        Else
                pnlAnon.Visible=True
        End If
        End Sub
</script>
<asp:Placeholder ID="plhPassport" Runat="Server"/>
<hr>
<asp:Panel ID="pnlAuth" Visible="false" Runat="server">
<h2>Customized News</h2>
<p> News customized only for you...</p>
</asp:Panel>
<asp:Panel ID="pnlAnon" Visible="False" Runat="Server">
<h2> Anonymous News </h2>
<p>News for anonymous users....</p>
</asp:Panel>
</body>
</html>
```