**Introduction to Web Development**





[**http://msdn.microsoft.com/rampup**](http://msdn.microsoft.com/rampup)

Contents

[Introduction 3](#_Toc226013528)

[Computer Networks 3](#_Toc226013529)

[Network Addresses 4](#_Toc226013530)

[Browsers and Web Servers 5](#_Toc226013531)

[Hypertext Transfer Protocol (HTTP) 6](#_Toc226013532)

[Hypertext Markup Language (HTML) 7](#_Toc226013533)

[Request from Client Browser 7](file:///C%3A%5CUsers%5CPat%5CDocuments%5CDTC%5CModule%203%20-%20WebConfiguration.docx#_Toc226013534)

[Response from Web Server 7](file:///C%3A%5CUsers%5CPat%5CDocuments%5CDTC%5CModule%203%20-%20WebConfiguration.docx#_Toc226013535)

[Web Applications 10](#_Toc226013536)

[Summary 12](#_Toc226013537)

# Introduction

The Visual Studio Express Editions family provides a free, lightweight, and easy-to-learn suite of programming tools that are aimed at the hobbyist, novice, and student developer. Many people in this category will not have had any formal training in computer science, and indeed they may not have any programming experience at all. If you fall into this category, don’t worry – this guide is for you!

This beginner’s guide is designed for people with little or no prior knowledge of computer programming, who want to create Web applications and dynamic websites with the Visual Studio Express tools. If you already have significant Web programming experience, then you will probably not have much to learn from this article.

So what will you learn by reading through this guide? Well, this guide will introduce you to the fundamental concepts that will help you understand how to create Web applications. In this guide, you will learn the answer to questions such as:

* What is a network and how are computers connected?
* What is the Internet?
* How do browsers work?
* What happens ‘under the hood’ when I connect to a Web server?
* What is Hypertext Markup Language (HTML)?
* How do Web sites work?

The purpose of this guide is to help you understand the environment in which a Web application runs, which includes the Internet, Web servers, browsers, network communications, resource identifiers, and markup language.

Don’t worry if you are unfamiliar with these concepts – after reading this guide, you will know much more about them.

# Computer Networks

Almost as soon as there was more than one computer in the world, people have wanted to connect computers together. In essence, a computer network is any system in which two or more computing devices are connected together. Even a standalone computer connected to a printer is a kind of network. But for the most part, networks consist of interconnected computers that can exchange information between themselves.

There are two parts of any network that enables computers to exchange information. First, they need to be connected together by a link – this is usually a physical link such as a cable or a phone line, but it can also be a wireless link. Second, they need to have a commonly understood ‘language’ that each of them can use to unscramble the signals that each sends along the link to the other one. You can imagine this as if two people were on the telephone to each other: although they have a telephone connection, they also need to be able to speak the same language if they wish to understand what the other person says. In networking, the term for a common language is a **protocol**. The protocol used on the Internet is called Transmission Control Protocol / Internet Protocol, or TCP/IP for short. This protocol enables computers to exchange message ‘packets’ that contain either data or control information, such as acknowledgements of receipt. By exchanging data and control information, both computers can be sure that the information was transmitted successfully, or one of them can resend the information if a problem occurred.

If you consider the Internet as a network, it is obvious that the connection between any two computers does not need to be a direct connection. In fact, in between any two computers connected via the Internet, there are usually several intermediary devices that form part of the connection between the two computers. The most common of these devices are **routers**, and they act a bit like a telephone exchange. Routers are specialist computing devices that direct network traffic along the correct route towards their destination. You can imagine routers as network policemen at various intersections directing traffic. The diagram shows a network route that carries information between a computer and a Web server via routers.

As an aside at this point, you should note that no one actually owns the Internet as a whole. For a home user, the first few routers along a route are likely to be owned by a consumer-oriented Internet Service Provider (ISP). At the other end of the route, a Web hosting provider or a corporation usually owns the Web server itself and the nearest routers to the Web server. In between the ISP and the Web hosting provider, the routers that link these together form a backbone network. The Internet consists of many backbone networks linked together, each of which is usually commercial, educational, or government owned.

## Network Addresses

For routers to be able to determine the correct route between two computers, each computer needs to have an **address** sothat the routers know where to send the data. In networking terms, an address is a unique identifier for a given device. Even routers themselves have addresses – in fact, for any destination address (like a Web server), routers only need to know the address of the next nearest router to the destination, and they simply hand over responsibility for delivering the data to that router. Eventually the last router in the chain knows the Web server itself and can deliver the data to it.

On the Internet, an **Internet Protocol (IP) address** is a four-byte number, usually written in ‘dotted decimal’ notation; for example 10.43.172.77. Every device in the world that connects directly to the Internet has a unique four-byte IP address. This means that there are over four billion possible IP addresses (although around 20 million addresses are reserved for special uses). This may seem like a big number, but there are increasing numbers of computers connected to the Internet.

Even using dotted decimal notation, IP addresses are not very memorable. People tend to find it easier to remember names than numbers. For this reason, the Internet has a system that makes it possible for a Web site to have a ‘friendly’ name as well as a less memorable IP address. This is the **Domain Name System (DNS)**, which works by translating domain names into IP addresses. There are many DNS servers connected to the internet, and their job is to act like a telephone directory for the Internet – they look up the IP address for a given domain name in a big list of all the domain names. For example, when you type a Web address into your browser’s address bar, the following steps occur, as shown in the diagram:

1. The browser determines the target server name (**host name**) from the Web address.
2. Your computer uses DNS to retrieve the IP address of the target computer. It contacts the DNS server and passes the host name.
3. The DNS server searches its list of domain names for a matching record, and sends the IP address back to your computer.
4. Your computer then uses the IP address to establish communication with the Web server.

# Browsers and Web Servers

So far you have learned how computers connect to one another so that they can exchange information. Once the connection is established, the next stage is for data to flow between the two computers.

In most situations, the computer that initiates the exchange is called the **client**, and the computer that receives the connection is called the **server**. A computer program runs on the server at all times, listening for connections from clients. On the client computer, another program (such as a web browser) connects to the server whenever it requires information. For example, when you request a Web page, the browser makes a connection to the Web server for that page when you click the Go button.

In a typical client/server scenario, the client sends some data called a **request** to the server, and the server determines the nature of the request and formulates a **response**, whichit sends back to the client. For example, when an email program reads your email from a mail server, the following steps occur:

1. The client sends a user name and password to the server.
2. The server responds to say that the user name and password are accepted.
3. The client requests a list of emails that are on the server.
4. The server responds with a list of emails, not including the body of the email.
5. The client requests the body of a specific email (for example when you double-click on the email to view it).
6. The server sends the email body.

The client and server must have a common understanding of the contents of the request and response messages. We saw earlier how when the two computers made a network connection, they have common understanding, defined by the TCP/IP protocol, that enables them to send data and confirm that it has been delivered. On top of this **network protocol** sits another messaging protocol that specifies a common language for computer applications to talk to each other. The protocol between two computer applications is called an **application protocol**. You can imagine a network protocol like the regulations that govern how to construct a road, and an application protocol like the regulations that govern how to drive on the road.

The most common Internet application protocol is the **Hypertext Transfer Protocol (HTTP)**, which is the main protocol used in the World Wide Web. You may also have come across the **File Transfer Protocol (FTP)**, which you can use to transfer files from one computer to another.

## Hypertext Transfer Protocol (HTTP)

When a browser communicates with a Web server, it sends an HTTP request to the Web server. The contents of the HTTP request specify what action the client requires the server to take. HTTP supports several methods (sometimes called ‘verbs’) that the client can specify in the request; the most commonly-used methods are GET and POST. A browser uses the GET method when it requests a Web page, and it uses the POST method when it sends data to be processed, for example when you click the Submit or Search button on a Web page.

The following is an example of the information sent back and forth between the browser and the Web server when the browser sends a GET request for http://www.example.com/index.html. Don’t worry about the detail of the request and response; this is simply to give you a flavor of what is going on inside the HTTP protocol.

##  Request from Client Browser

GET /index.html HTTP/1.1

Host: www.example.com

Accept-Language: en

##  Response from Web Server

HTTP/1.1 200 OK

Date: Mon, 23 Oct 2006 12:05:20 GMT

Server: Microsoft-IIS/6.0

X-Powered-By: ASP.NET

Content-Type: text/html; charset=utf-8

Content-Length: 194

<html>

 <head>

 <title>Welcome</title>

 </head>

 <body>

 <h1>Welcome to Example.com</h1>

 <p>Welcome to the Example.com Web site. We hope you will find lots of examples here.</p>

 </body>

</html>

The first line of the request specifies the method (GET), the web page or resource (/index.html), and the protocol (HTTP 1.1). The following lines are request headers that indicate extra information to the Web server. The response contains a status line that includes the protocol (HTTP 1.1), a status code (200) and a text description of what the status code means (OK). The next few lines are response headers that contain extra information about the Web server and the Web page.

After the headers there is a blank line, followed by the actual content of the Web page. This is the real payload of the response, and it contains the information that the browser will actually display.

## Hypertext Markup Language (HTML)

The purpose of HTTP is simply to provide a common ‘language’ in which the browser and the Web server can exchange information about Web pages and other resources. The real substance of the exchange, when the browser requests a Web page, is the section of the response that describes how the page will appear inside the browser window.

The description of the Web page that the server sends is in a standard format, so that the browser can understand how the Web server wants it to display the page. This time the standard format is not in fact a protocol, because it does not define how two computers exchange information. It is actually a language, although the distinction is fine: there are rules about computer languages just as there are about a network protocols. The language that Web servers and browsers use to describe a Web page is **Hypertext Markup Language (HTML)**. HTML is a language that is used primarily to format data on a page. It doesn’t contain any advanced support for doing complex operations; it just serves to layout the contents in a readable way on the web page. When the browser receives an HTML page, it converts the HTML description into a screen display by a process called **rendering**. The browser reads the HTML instructions and “renders” the result to the screen.

HTML is a text-based language, which means that you can view and edit HTML in a standard text editor like Notepad. It consists of the text on the Web page, along with ‘markup tags’ that indicate to the browser how that text should be displayed. The markup specifies items such as the font to use for sections of text, where to display embedded images, and of course hyperlinks that enable you to link to different Web pages. You can look at the HTML for any Web page in your browser by viewing the page source. (To view the page source in Internet Explorer, click the View menu, and then click Source.)

This guide does not intend to teach you how to write HTML. There are many books and online resources that can help you to learn HTML. However, the document below is a short example of a simple HTML page, which demonstrates some important concepts.

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>

 <head>

 <title>Welcome</title>

 </head>

 <body>

 <img src="/images/logo.jpg" height="50px" width="200px" border="0" />

 <h1>Welcome to Example.com</h1>

 <p>Welcome to the Example.com Web site. We hope you will find lots of examples here.</p>

 <p>The full list of examples is on <a href="/example-list.html">this Web page.</a></p>

 </body>

</html>

HTML consists of a series of **tags** denoted by angle brackets around a tag name, such as the <html> tag in the example. At the end of the document is another tag, </html>. The forward slash indicates that this is the end tag that corresponds to the <html> tag at the start of the document. Everything between the two tags is called the <html> **element**. Inside the <html> element is a <head> element, which contains <title> and <link> elements. An HTML document consists of elements that are nested one inside another in this way.

Some of the elements define **attributes** inside the angle brackets as well as a tag name – the <a> and the <img> tags, for example. Attributes contain extra information about that tag that the browser uses to render the Web page. The <img> tag in the example above denotes that the browser should embed an image in the rendered Web page, and it should contain the following attributes:

* **src.** This refers to the **Uniform Resource Locator (URL)** of the image to embed in the page.
* **height.** This denotes how high the image should be in the rendered Web page – in this case, 50 pixels.
* **width.** This denotes how wide the image should be in the rendered Web page.
* **border.** This denotes the width of the border around the image – in this case, the border is set to 0 so browser should not display a border.

Note that the HTML document does not contain the image itself. Instead, the document contains only the URL of the image. When the browser renders this document, it performs the following steps.

1. The browser sees the <img> tag in the document and recognizes that it should display an embedded image.
2. The browser creates an HTTP GET request for the specified URL for the image file, and sends this request to the Web server.
3. When the server has responded by sending the image, the browser embeds the image in the rendered Web page.

# Web Applications

The scenario above shows how a browser downloads and renders a simple Web page. Let’s have another look at the HTTP request, so we can see what the server does when it responds to the request:

GET /index.html HTTP/1.1

Host: www.example.com

Accept-Language: en



1. The browser sends the GET request to the server. As you saw above, the first line of the request specifies the resource (/index.html) that the browser is requesting.
2. When the Web server receives this request, it locates the root folder of the Web site on its hard drive, looks for a file called index.html, and then reads this file from the disk
3. the Web server sends the contents of the file back to the browser inside the HTTP response.

So if the Web site root folder is C:\Website, then the Web server returns the contents of C:\Website\index.html. Likewise, when the browser requests /images/logo.jpg, then the Web server returns C:\Website\images\logo.jpg.

This system works very well if a Web site does not change its content very often. Every time the Web site owners want to change the content, they can upload a replacement for index.html (or whatever file needs to be changed).

However, the system is less efficient if the Web site contents change regularly, for example if the Web site displays data that changes regularly. In this scenario, it is not practical to constantly create HTML files that contain the data. Imagine if your Web mail application worked by reading files from the hard drive. Every time you deleted an email, the Web mail application would need a new HTML page to represent your Inbox, so that the Inbox no longer contained the deleted email. The Web mail application would therefore need to create a new HTML file on the hard drive to represent the new Inbox.

The solution to scenarios like this is to create a Web application that works with the Web server to create the page dynamically each time a browser requests it. Usually the information on the page is stored in a database. This kind of Web site is called a **dynamic** Web site, in contrast with a **static** Web site where the contents are not generated each time a Web page is requested.

When a browser requests a page from a dynamic Web site runs, the request cycle works as follows:

1. The first part of the cycle works the same as with a static Web site – the browser specifies the resource name inside an HTTP GET request.
2. After this, however, things work slightly differently. The Web server recognizes that the file extension is something different than .html – for example, the file extension might be .aspx, which is an ASP.NET file – and it calls the appropriate Web application to process the request.
3. The Web application reads the requested file from the file system.
4. Instead of returning the file directly, the Web application processes the file and runs any instructions that it finds within (see below).
5. These instructions may load data from a database or perform some other processing. The Web application dynamically generates some HTML according to the instructions in the file, and then returns the generated HTML to the Web server.
6. The Web server forwards this HTML content to the browser. The browser itself is unaware that any of this HTML generation has taken place – it just treats the HTML in the same way as if it were a static file.

The exact way that the Web application generates the HTML varies from application to application, as does the computer language that is used to specify the instructions to the Web application.

Typically, however, the file that contains the instructions is a mixture of regular HTML and special programming instructions. The HTML describes the appearance and structure of the Web page, and the programming instructions describe how to access the information in the database. When the Web server processes the file, it replaces the special programming instructions with regular HTML. The instructions tell the Web application how to generate the regular HTML which it will use to replace the instructions.

An example helps to illustrate how this works. Here’s how our static Web page might look if it were generated dynamically by using ASP.NET:

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>

 <head>

 <title><%=Page.Title%></title>

 <link rel="stylesheet" href="/style.css" type="text/css" />

 </head>

 <body>

 <asp:image id="imgLogo" ImageUrl="~/images/logo.jpg" height="50px" width="200px" BorderWidth="0px" runat="server"></asp:image>

 <h1><asp:label id="lblPageTitle" Text='Welcome to Example.com' Runat="server"></asp:label> </h1>

 <p>Welcome to the Example.com Web site. We hope you will find lots of examples here.</p>

 <p>The full list of examples is on <asp:hyperlink id="lnkExamples" NavigateUrl='~/example-list.html' runat="server"> this Web page.</asp:hyperlink></p>

 </body>

</html>

The detail of how this works is not important right now. However, you can see that certain parts of the original page have been replaced with different code that is not HTML (the tags that begin <asp: are not HTML). The ASP.NET Web application replaces these tags with valid HTML when it processes the page, so that the result would look similar to the static page you saw earlier. In the ASP.NET context, there would usually be some Visual Basic or C# code that would work with this page to create the dynamic results.

# Summary

In this guide you have learned the purpose of Web applications and the basics of how they work. You have seen how computers communicate together over the Internet, and how browsers work with a Web server to generate Web pages. You also learned how Web servers work, in both static and dynamic Web site scenarios.

If you want to use Visual Web Developer to create dynamic Web sites, you now understand the context in which the various components of your web application will operate. There are a few next steps that you should take:

* **Learn more about HTML**. Since HTML is the basis of everything that appears in a browser, you need a strong understanding of how it works to create great-looking Web sites.
* **Learn a programming language**. The two languages that you can use in Visual Web Developer are Visual Basic and C#. You should choose one of these languages and study it. You can find beginners guides about both of these languages here on MSDN.
* **Learn ASP.NET**. Once you understand HTML and a programming language, you can put the two together to create dynamic web applications. You should download and install Visual Web Developer, and then look at getting started by taking some of the lessons available here on MSDN.