1

aesthNETics

Technology has continued to scream past us over the last few decades. What is top of the line now may be completely obsolete in six months. As soon as you begin to understand a new technology, updates to it come out and make the entire learning curve start over. This has certainly been true for .NET developers. When .NET was introduced it was an amazing improvement over previous Microsoft programming languages. The mere fact that VB and C programmers could understand each other's code should say something. But the incredible number of new controls and gizmos could prove daunting to a lot of people. Before anyone could get comfortable with .NET 1.0, 1.1 was released. When people finally started getting comfortable with 1.1, 2.0 was released. And now, barely a year after the 2.0 release, there is a new 3.0 coming out. Some programmers love the flux and constant improvements to the functionality of .NET. They strive to learn all of the new features and enhancements to see how they can improve the projects they produce. Others try to keep up only so that their projects stay current and to see what impact the new frameworks will have on their existing applications. Either way, most folks want to focus on the ''under-the-hood'' pieces of the framework. They want to determine if the new .NET 2.0 GridView really is that much better than the 1.1 DataGrid (it is). They can immediately see the benefits of implementing `Try...Catch...Finally` statements in their code.

But what seems to happen too often is that these same programmers forget about, or at least put on the back burner, the concept of making these new powerful .NET pages look good; they forget about aesthNETics.

What Is aesthNETics?

aesthNETics simply refers to making .NET pages look good. Okay, that may be a bit simplistic. What does ''making .NET pages look good'' really mean? Does this simply mean ''aesthetically pleasing''? Beauty is in the eye of the beholder, right? The short answer is *no*.

For the longer answer, you have to consider websites you like and don't like. There is probably a consistent theme to those groups among most people. It may be hard to come up with a definitive

list of the features you want or expect when you access a website, but here are a few that would probably appear on most people's lists:

- You can find what you want easily.
- There is a consistent look and feel to every page you visit.
- Pages are laid out in an intuitive and logical manner.
- Graphics, if used, are not overpowering or obnoxious and not just used for the sake of using graphics.
- Colors used don't burn your retinas or make you immediately want to leave the site.
- You don't load the page and think ''Wow, there is just too much going on here.''
- The page doesn't take too long to load, even on slower connections.
- You don't immediately think ''Hey, I've seen this template before. A lot.''

In addition to these items, developers might think of how to provide the best experience for their patrons. Things like browser compatibility come in to play. You must at least decide which browsers you want to support and, subsequently, code your site to meet this standard. When thinking of browsers, though, you should remember that large populations of Internet surfers use text-based browsers or readers to access the content of your site. Have you coded your site for these people?

aesthNETics incorporates all of these ideas. aesthNETics means that you don't forget the look and feel of your site. It means that you don't just worry about the business logic and data of your site; you give equal consideration to making the interface user-friendly and pleasant. aesthNETics means that you remember to make the things your site patrons *can* see be as powerful as the things they can't.

The skill set for a good aethNETics programmer includes:

- A good understanding of web design and layout basics.
- An appreciation of colors and graphics. (Sometimes less really is more.)
- A working knowledge of cascading style sheets.
- A comprehensive knowledge of the tools in ASP.NET that allow for consistency among pages you create:
  - **Site Navigation** components for intuitive and consistent access to content.
  - **Master Pages** for structural layout.
  - **Themes** for consistent .NET components.

Using this book as a guide, you will learn to appreciate the design element of the websites you produce. You will learn enough about graphics to get you started on your quest to learn more. And, most relevant to .NET programmers, you will learn about the new features of .NET 2.0 that can help you with the aesthNETics of the sites you produce.

Don't misunderstand the scope of this book, though. It does more than outline specific .NET tools available to you as a web application developer. It shows you sound web design principles and tools that are not necessarily limited to .NET programs. This includes a discussion of Cascading Style Sheets and their standards, which are established and maintained by the Worldwide Web Consortium (W3C). You will also be given an overview of colors, images, and other universally relevant web design considerations. As the book progresses, though, you will see how to integrate these philosophies within Visual Studio 2005 and the components offered through the .NET 2.0 Framework. Many of the concepts discussed in this book will not be strictly .NET-specific. The idea of aesthNETics, though, is focusing and applying those concepts in your .NET applications.

Why Is aesthNETics Important?

You might think of aesthNETics as the presentation layer of a typical n-tier application. Many presentations regarding programming, and this is certainly true of .NET, go into great detail on how to make websites powerful and inundate them with the latest and greatest technologies. They introduce topics like AJAX and XML serialization and often go into real-world examples of implementing business logic into web applications you develop. What most presentations seem to lack, though, is a fair discussion of the presentation layer. The risk of this mentality is, quite simply, the perpetuation of really powerful websites that are really powerfully boring to look at.

Does the presentation layer matter? Of course it does. Does it matter as much as the business logic and data access layers in a typical 3-tier application? Well, that depends on who you ask. At one end of the spectrum, you might see the seasoned programmer who reads the most current periodicals and attends technical conferences in an effort to stay on top of the latest technological trends. He might say ''no, it's not nearly as important.'' But what about the customers? Sure, they might be appreciative of a 235-millisecond reduction in access speeds using the new cross-tab functionality of SQL Server 2005. Or maybe you can wow them with the fact that your Social Security field is validated through regular expressions and then encrypted in the database. But what do you think they will say if you are telling them all of this while they are looking at a white page with a couple of text boxes on it? This is often referred to as the ''halo effect,'' whereby initial perceptions are used to judge the other properties of an item. In regard to web design, this means that a potential customer will often make a judgment on an entire web application according to what they see in the browser in its initial load. If you have lost the customer before you begin discussing the behind-the-scenes merits, you will have a tough time getting them back.

To illustrate, assume that you have a client who wants a data dashboard application. They have some data in a spreadsheet that they first want converted to some sort of Enterprise solution. They then want to create functionality to automatically import data into the new database on a daily basis. Finally, they want to present the analysis of the data in a web application viewable on the corporate intranet. This is a high-profile project with a lot of potential accolades and recognition to the team that delivers the new system. As a result, two groups compete for the distinction of designing the new system and presenting it to the Executive Leadership team. The first team spends a lot of time making the data conversions and creating an automatic update system that should work within the parameters of the project requirements. They decide to forego spending much time on the interface of the project and just use some of the Microsoft Office graphing components to represent the data. They present to management the example in Figure 1-1.

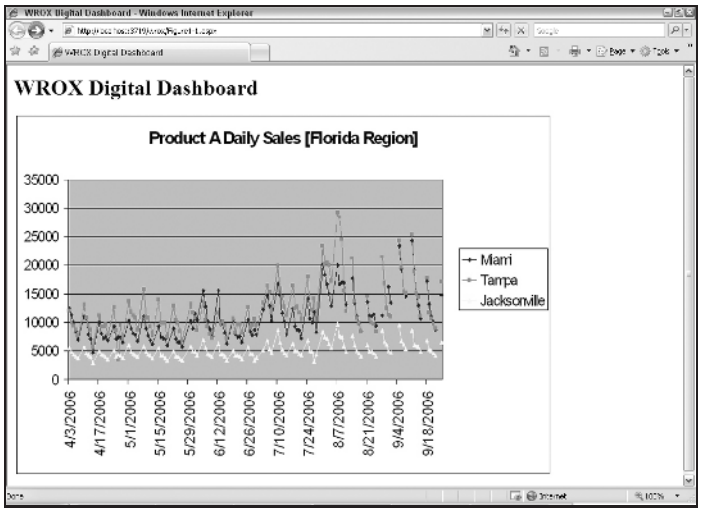

Figure 1-1

The second team also completes the initial data conversion but, in preparing for their presentation proposal, they spend time coming up with an aesthetically and intuitive layout that they hope will please the project management team. They decide to come up with custom graphics using Adobe PhotoShop. They decide to create custom graphing components using the `System.Drawing` namespace. They pay close attention to the color scheme of the site and make an attempt to have everything flow together. They do not spend the time at this point working on the automatic data upload piece. They present the example in Figure 1-2.

Figure 1-2

Which of these projects do you think, in a real-world situation, would win the nod from management?

Don't misunderstand; the presentation layer is not the only important component of a web project. In the above example, if it turned out that the second team was assigned this project but, soon after starting,

it became apparent that they were only capable of making sites look good and had no real concept of how to gather and analyze the data, management would not be happy. Making a site look good is by no means enough. The point of this example, though, is to show that the interface design is important and should not be ignored.

Evolving Expectations of Employers

In today's rapidly evolving technological market, programmers are being asked more and more often to become the ''Renaissance men'' of programming. This simply means clients and managers want these programmers to be able to handle all aspects of a web project from requirements gathering to application development to coordination of user testing to documentation. And, yes, even graphics and design. How many job postings have you looked at that had a laundry list of requirements that doesn't at least suggest the company wants you to do it all? Programmers already find it completely impossible to say ''I don't do database stuff; that is for the DBAs.'' As time goes on, you will find it increasingly hard to say ''I don't do graphics; that is for the designers.''

Even if that were not true, there are certainly other reasons to become a better web designer with regard to your .NET applications. For one, it can make you stand out among your peers and coworkers. Say that you are in a programming shop that does a lot of .NET web applications. Most of the programmers are pretty efficient at .NET. You are somewhere in the middle. You aren't the best of the group, but you certainly aren't the worst either. How can you distinguish yourself? You could certainly put the time in and become the best .NET developer in the group. But what else might you do?

Phillip Van Hooser, author of *Willie's Way: 6 Secrets for Wooing, Wowing, and Winning Customers and Their Loyalty* and an expert on leadership and customer service, has a simple and direct method for distinguishing yourself from the crowd. He says that all you have to do is just the smallest bit more than anyone else. You don't have to make huge leaps of improvement over everyone else; you only have to do slightly more. He explains that, if you are doing even slightly more than anyone else, you will be noticed and people will remember you.

So what can you do to be even the smallest bit better than the peers in your group? If you are proficient at web design and layout, and certainly if you can navigate your way around graphics programs like Adobe PhotoShop and create custom graphics, you will stand out. If you create applications that are aesthetically stunning, while others are bland and uninspired, you will be remembered.

Creating a Consistent Look and Feel for Your Website: aesthNETics Essentials

This book will attempt to outline some of the basic tools, primarily available in Visual Studio 2005, that .NET developers can use to make their sites appearance at the same level of the coding behind them. But if there is any hard-and-fast rule for web development, it is this: make the look and feel of your site consistent. There is probably no bigger annoyance than going to a site and having the front page look one way and then a content page look entirely different. This doesn't mean that page layout can't vary from page to page. But it should be the developer's goal to make the user feel that the site is consistent throughout every page. Sure, at times, you will have to stretch the header to accommodate large data tables. Or maybe you will have some sidebar links on some sections of the page and other links for other pages, depending on where the user is on the site, to accommodate a large site with many different

sections. But if the position of the links shift five or ten pixels to the left or right as the user navigates or if one page has a blue-based color scheme and another uses shades of brown, it will make the site look sloppy.

Of course, a first step in creating a consistent look and feel to your site is to actually establish the look and feel you want. While it can be argued that having a white page with text and various controls on it is, in fact, a look and feel (one which would be easily carried out through all pages), this doesn't really meet the standards of good aesthNETics. Although it may be hard to define exactly what the standards of good aesthNETics are, you can use the ''wow'' factor test to see if you are on the right track. If you show your project to your client and he or she says ''wow,'' you know you are probably following the aesthNETics standards. If the client doesn't say this or seems uninterested, you might want to rethink your approach. This can prove to be a challenge, since different clients have different ''wow'' factor criteria. It is your job, as an aesthNETics developer, to figure out what those criteria are. You should understand who your potential customers will be and how they will access the content you provide. You should have a basic understanding of website design and layout. To stand out from the crowd, you should have a basic understanding of graphics and color use and be familiar with the tools to help you create both. You should pick your colors and create your graphics early in the process so that you can decide the other necessary components of your site. A more detailed explanation of how this is done and what tools are available can be found in Chapter 2.

Once you have decided on a look and feel for your site, you need to figure out a way to carry it out throughout the pages of your site. Most applications have a template tool available to them. For example, you can easily find skins you can apply, or even create yourself, for applications such as Windows Media Player. Some web designer applications include them as well. But historically, .NET has not really had a clean way of making pages using a template.

That changed drastically with the release of the .NET 2.0 Framework. Master Pages are now a part of the Framework and should be a part of every .NET developer's arsenal of web design tools and tricks. Master Pages allow developers to create the structural template for the design of their website. In essence, this means that you can say ''I want the header to be at this position and the footer to be at this position and the navigation to go here and the content to go here'' and then save that as, for lack of a better word, a project template. Your other pages can then inherit the Master Page, and the content of those pages will fall into the content area created in your Master Page. This will be covered in more detail in Chapter 7. It is enough, for now, to understand that Master Pages can help you create a project template that can be imported into your web pages.

Cascading Style Sheets (CSS), which have been around for over 10 years, are another great way to create consistency throughout your site. With CSS, you can dictate the look and feel not only for the HTML elements on your site but also for XML, XHTML, SVG, XUL, and most derivations of SGML. If, for example, you want to say that the background color of your entire site is gray, you can do that with CSS. With the new Visual Studio 2005, a couple of tools were introduced to help you with creating accurate style sheets for your applications. There are also a couple of cool tricks for applying them to your site, as you will see in Chapter 4.

One shortcoming of using CSS, though, is that you can't use it effectively to create a consistent look and feel to ASP.NET elements. This means that with stylesheets alone you can't say ''I want every .NET label dropped on a page to be bold red Arial 12pt font.'' You can certainly create a CSS class that has that formatting to it and then reference it from the ''CssClass'' property of the label control. But what happens when you drop another label on the page? If you don't make a similar reference to the CSS class in the `CssClass` property of that label, it isn't applied; the label just uses the default settings for a label.

With the .NET 2.0 Framework, however, you can modify the default behavior for that label. You can tell your application that for every label dropped onto any of your pages, it should apply this particular CSS class. Even better than that, you can say that for every GridView control, the header will be blue, the row style will be gray, and the alternating row style will be white. When you drop a GridView on your page, it will just be formatted that way without you doing anything else. Furthermore, you can tell your application that in some situations it should make the GridViews all look as described above but, in other situations, it should make them look entirely different (based on profiles, browser specifications, and so on). You can also say to make some GridViews look this way and others look another way, based on a control ID. This is all done through the use of skins and themes, which will be discussed in Chapters 8 and 9.

Finally, developers need to make sure that their customers can easily get around their site and that the controls in place to navigate the site are consistent and intuitive. Historically, developers have tried to create Windows-style navigation controls by using JavaScript or third-party controls. The brave have even attempted to write their own navigation controls for their .NET projects. However, with the .NET 2.0 Framework, site navigation controls have been introduced to help developers create consistent and easy-to-use navigation elements on their sites. You want the drop-down navigation controls you see around the Internet (the kind that mimic Windows applications)? Those are now included in this control. You want to create a ''breadcrumb'' component to show site users where they are in the maze of pages that creates your site? That, too, is part of the default behavior now included in these controls. Developers can now create a `SiteMapDataSource` file that can be used throughout the entire site for navigation controls. Update the file, and all pages referencing it will be updated as well. Site navigation has come a long way with the 2.0 Framework, and you will learn how to use these features in Chapter 6.

Prerequisites

There really aren't a lot of prerequisites for learning good web design principles, whether in .NET or any other development platform. In fact, the mere fact that you have obtained this book and gotten this far into the first chapter shows a willingness to at least begin to think about design issues. Technically speaking, though, there will be a few assumptions about your ability level.

First and foremost, you should be at least fairly comfortable with HTML code. If the following code snippet looks completely foreign to you, the topics presented in this book will likely be confusing and hard to follow:

```
<strong>Hello world!</strong>
<br><br>
<p>This is my first paragraph for my first page!</p>
```

Obviously, if you have a familiarity with at least the elementary concepts of HTML, CSS, and good web design, you can breeze right through those sections. Similarly, if you have experience in developing .NET web applications (with any version), many of the new .NET concepts will be easier to understand. But there aren't any concepts that a dedicated reader can't grasp.

Another fairly crucial requirement is at least a cursory knowledge of .NET. It really doesn't matter if that knowledge is from .NET 1.0 or 1.1 for this level of discussion. A complete novice, though, will have trouble following some of the programming concepts presented. This book will not make attempts to explain in any great depth what a namespace or an object or a page class is. When pertinent, enough details about how those features interact with the topic at hand will be given. However, do not rely on the book to give a thorough breakdown of the .NET Framework. There are books that go into much broader explanations of the Framework; this book is specific to some exciting new features of the .NET 2.0

Framework. To get the most out of this book, the reader should be at a bare minimum familiar with object-oriented coding but really should have some level of experience in some level of .NET. Again, a lack of this experience will not be a total obstacle to getting something out of this text; but it will certainly be a hindrance (one only you can gauge).

To extend that thought, though, a familiarity with Visual Studio 2005 will really help your understanding of the concepts presented in the later chapters. Most of the code demonstrations are going to be performed in Visual Studio 2005. When introducing a new feature of Visual Studio, that feature will, of course, be explained in detail. However, if the concept is not new to Visual Studio 2005 and not necessarily pertinent to the discussion at hand, it will not be adequately explained to the person who has never seen the interface.

It should be noted that, for all of the demonstrations in Visual Studio 2005, you could recreate the functionality in Visual Studio Web Developer 2005 Express Edition. In fact, much of the learning done by the author was done through this free development tool. Obviously, if you have the wherewithal to purchase the full Visual Studio 2005 IDE, then you should. It has more features and is much more powerful. But if you are just getting started or are a hobbyist, there shouldn't be anything in this book that you can't do in the free version.

The biggest requirement, though, is a genuine interest in making the websites you develop more aesthetically pleasing. The desire to learn is something that shouldn't be downplayed and, if you have that desire, you can take advantage of the concepts in this book. The concepts presented here should be the foundation that every web developer builds on, not just the .NET folks. There are cool tricks that will be showcased that are specific to .NET, but the concepts are universal. If you develop on the web, you should get something out of this book.

Summary

The quintessential concept of aesthNETics is not really a new one. Web developers for years have struggled to make their sites look good while being powerful and useful. Thousands of books on the market talk about basic web design concepts. If you search for ''web design concepts'' in Google, you will get back more than 100,000,000 results (literally). There are hundreds of applications on the market that profess that they can turn the average Joe into a professional web designer. There are classes for elementary school students that teach the basics of web design. In short, there is no deficit of resources outlining the concepts of web design.

Where the deficit seems to come in is with the actual utilization of this information. Many developers, and, again, this is not specific to .NET developers, do not spend the time working on the aesthetics of the sites they develop. Throw a button control here, a couple of text boxes there, maybe try to implement a color scheme that was developed 10 years ago for the corporate intranet, and *bam*, you have a site.

What aesthNETics tries to push is using the powerful tools of the .NET 2.0 Framework to make your sites aesthetically pleasing. This book is focused on the .NET development community. Many of the concepts are universal, and other developers might find some good information in its content. But the primary audience is .NET developers. Most tools illustrated are ones that are specific to .NET 2.0 and Visual Studio 2005. While other platforms may have similar functionality, that is not really the point of discussion of this text. The point of this book is to make .NET developers better.

There are certainly many reasons to want to be better. The ever-evolving expectations of clients and managers is certainly one of the first and foremost reasons. Everyone expects today's developers to do it all. A typical job posting might include .NET experience, database experience, nUnit testing experience, project management experience, and even graphics experience. To stay competitive, developers need to keep learning. They need to broaden their horizons and challenge themselves to learn things that are potentially outside their comfort zone. And to the serious developer, *that* is the best reason to learn aesthNETics. The challenge should be the real appeal of learning new concepts. The difference between a good developer and a great developer is often simply the genuine desire to keep learning. As Steve McConnell stated in *Code Complete 2nd Edition*, ''The characteristics of a superior programmer have almost nothing to do with talent and everything to do with a commitment to personal development.''

So sit back, boot up your laptop, and begin your quest to be a better .NET developer.