Part I
Getting Acquainted

1

Introducing DotNetNuke Skinning

DotNetNuke (DNN) is a dynamic, database-driven web application that leverages Microsoft ASP.Net to construct a full-featured website in minutes. As an open source product, the barrier to entry is low in terms of dollars; however, the wealth of features can seem daunting at first. DotNetNuke uses Internet Information Services (IIS), the web server included in Windows, and SQL Server 2005 Express Edition, a free database server available for download from Microsoft out of the box. As you migrate a DotNetNuke website out to a hosted web server on the Internet, you'll probably implement a SQL Server 2000 or SQL Server 2005 database for a little more horsepower.

DNN offers plenty of features to solve common business problems. Graphic design and custom module development both have input on the usability and construction of a website. There's a certain balance to strike between beauty, familiarity, and discoverability.

The Origins of DotNetNuke

DotNetNuke's history stems from the IBuySpy Portal that promoted the ASP.Net version 1.0 Framework. Shaun Walker continued the concept over the subsequent years. *Professional DotNetNuke 4: Open Source Web Application Framework for ASP.NET 2.0* (Wiley Publishing, Inc., 2006) is the most recent book written by Shaun and several other folks close to the DotNetNuke Core Team. This book is available at `www.wrox.com` and is a worthy addition to your web development library. The first chapter provides a firsthand detailed account of the history and evolution of DotNetNuke as both a code base and the people involved.

Why Use DotNetNuke

The web development world of today is very different from its roots. The expectations of a modern, engaging website have leapt since the last turn of the century. Plus, few people possess expert level skills in every technology that touches a feature-rich site, and few have the ability to move a sizeable project from conception to implementation. You can be thankful for the large community of DotNetNuke enthusiasts that support our efforts along the way.

As of July 24, 2007, the DotNetNuke website, `www.dotnetnuke.com`, listed the following statistics:

- 6,483 downloads yesterday
- 440 new members yesterday
- 459,350 registered users

The community is one of DNN's most valuable assets. As a worldwide project, vast numbers of people contribute to the common storehouse of DNN knowledge and experience. The platform is under active development, and this development shows no signs of slowing. New versions of the core framework are released every few months on average, since version 4.0 was released on November 7, 2005. Ordinarily, this frequent release cycle would be a cause for alarm for any level-headed technologist in charge of maintaining an available system. The core DotNetNuke team addresses this concern by applying lengthy testing procedures prior to a release. Enabling error-free upgrades to the latest version is one of their most heavily protected features.

When a new version is released, visitors access the DotNetNuke site `www.dotnetnuke.com`, authenticate, and download the latest version as a zip file. This file is unzipped over the top of an existing system. The system discovers the new version and automatically upgrades itself, including any database updates. Of course, responsible administrators make necessary data backups and perform the upgrade in a test environment first.

A company, department, or club can utilize document management and collaboration features in DNN to improve productivity and information availability at an internal level. Externally, this platform is an effective tool for marketing the organization to customers as a public website. DNN can be customized and branded through skinning to reflect the message and nature of the organization.

Public websites are solid candidates for a customized DNN skin. These types of sites have a real need to develop brand equity or carry on an existing brand. Visitors come to the site for its content — the text, photos, and diagrams on the page. Everything else inside the browser helps draw the appropriate mood including fonts, colors, positioning, and background effects. A custom DNN skin is an asset that transforms the website into a unique experience and augments the content, which is the most important aspect of the website. Although the design can be stunning and elegant, the content is what provides the real value to the visitors.

Intranet applications have different priorities from applications available to the public. However, the considerations of each share common ground. The administrator of the DNN website should have a good idea about the desired experience for the visitors to the website.

Intranet applications might share an overall brand for the company and permit a divisional or departmental level sub-brand by emphasizing colors in the brand's color palette or a similar take on a theme.

What Is Skinning?

Skinning is the act of applying a design to the website content. If the various roles for building a website are taken to an extreme, a copy writer produces the words as the core of the page, and a graphic artist develops the mood, textures, tones, and imagery. As you develop a custom DotNetNuke skin, you'll carefully parse each of these inputs and produce a thoughtful web page that balances the ideas of all the stakeholders. DotNetNuke has a well-defined system for separating each of these concerns.

The task of skinning DotNetNuke exists somewhere between a graphic designer role that creates an original idea and a developer role that writes custom code to extend the website. Skinning DNN can be approached by either specialty, or by someone who enjoys wearing both hats.

This book assumes the important tasks of creating website copy and an original graphic design for a skin have already been accomplished, and now the goal of implementing the design inside of DotNetNuke is upon you. You'll learn common patterns and practices for implementing skins over the course of the text. The discussion of creating a graphic composition is outside the scope of this book.

Before you dive into using DNN, it's important to understand some important parts of the system.

DNN Modules

A DotNetNuke module provides a specific feature to the website, such as document management. DotNetNuke has several *modules* included in the default installation. If you can think of an interactive feature for a website, then chances are someone has built a module that does something similar to what you have in mind. If you can't find it commercially or freely available online, then you might have discovered a great niche for selling a new module in the DotNetNuke marketplace as well as some inspiration for building your own component.

A module can be added to any page in the website at any time. A page can hold multiple instances of a module, too. For example, the Documents module allows authorized users to upload documents and tag them with meta information like a title and a description. This module displays the documents as a list in the specified sequence and offers a tracking feature to gauge the popular downloads. A Documents module on the left side of a page might list the training documents for an organization and the right side of the page can hold another Documents module listing supporting information. Security can be configured such that all employees have access to the Documents module on the left side, but only managers have access to the Documents module on the right side.

Many more commercial modules are available online in the $20 to $200 price range. This makes DotNetNuke a worthwhile consideration for a dynamic and feature-rich website with little, if any, out-of-pocket expenses.

The sheer number of out of the box modules in DotNetNuke makes it an excellent option for an instant intranet application. At the end of this chapter, you'll have all the information you'll need to explore around the full installation of a standard DotNetNuke site. Be sure to explore all of the default modules when time permits. There are so many modules that you may find exactly what you're looking for.

Perhaps the most popular module is the Text/HTML module. It displays whatever content was typed into its HTML editor. Although the Text/HTML module is a flexible and familiar component in most DNN

installations, it doesn't have any interactive features outside of the HTML editor it displays while in edit mode. The administrator doesn't need to have HTML skills. The editor helps him or her enter content and style it with a wide array of buttons in the toolbar. The Text/HTML module, like any other module in DotNetNuke, can have its own RSS feed and custom print button if it makes sense within the context of the module. You might enable the print button for a Text/HTML module that contains instructions for how to get to your office, but you might disable the print button for a Text/HTML module that just contains hyperlinks to other pages on your site.

Learning About Containers

A DotNetNuke container envelops a module on a one-to-one basis. Every module has a container, although the module can be configured to hide the container that wraps around it. A container traditionally shows the title of the module, common buttons, and the raw content of the given module, whatever that might be.

Imagine you downloaded a free container file online named Roundy. This container draws a nice background with rounded corners around each module that's configured to use it. The Documents and Text/HTML modules on a page are both configured to use the custom container named Roundy. Because each instance of a given container on a page refers to a single physical file named `Roundy.ascx`, it's easy to make updates cascade across the entire site; you only have to edit a single container file. Later on in this book, you'll build your own custom containers.

Containers help enable consistency around different modules on the same page and across the entire site. Different modules can be configured to have the same container applied to them. It's not uncommon to configure all the modules on a page to use the same container.

Every module has a module settings button. It's the job of the container to show this button to authorized users and hide it when the visitor is not authorized to click it. Administrators can click on the module settings button and navigate to the standard module settings page. This is just one example of the responsibilities assigned to a DotNetNuke container.

Learning About Skins

A DotNetNuke skin is the glue that holds a large number of parts together. Understanding it can be a little rough at the start, but let the information wash over you until the hierarchy begins to make sense.

In DotNetNuke, a given page has only one skin. Different pages can have different skins, and they can all be configured to use the same skin if that is your wish. Skins can be assigned on a page-by-page basis, or they can simply inherit the default skin. If you never assign a skin at a page level, then the site can change dramatically by simply installing a new skin file and making it the default skin. All of the pages that inherit the default skin will instantly take on the look of the new skin. In my experience, most sites use at least three skins: one for the home page layout, one for the landing page layout, and one for the detail page layout. Of course, there are always outliers.

DotNetNuke modules are dropped into panes. The lines around a pane are visible for a viewer logged in as an administrator, but they're invisible for normal users browsing the site. A pane can happily exist

without any modules, or it can be overflowing with them; the administrator decides which modules to drop into a pane. The author of the skin merely determines the quantity of panes on a page and where they appear on the page. Panes expand vertically to fit the number of modules inside of them — an important consideration for anyone building their own custom skin.

For example, a landing page that has several detail pages underneath it in the website hierarchy might use a skin with panes arranged in a three-column layout. The detail page might use a skin with panes arranged in a two-column layout. A collection of HTML and CSS rules specify the arrangement of panes on a web page. These arrangement decisions take several roles into account, including the copy writer, the graphic artist, and the DotNetNuke developer.

Developing Skins

DotNetNuke utilizes the Microsoft ASP.Net platform at its foundation, but you have no direct need to compile code, learn object-oriented programming, or launch sophisticated development tools such as Visual Studio.Net 2005.

DNN skin developers spend the majority of their time using their favorite text editor and a web browser. After all, A DNN skin is just a text file. The majority of the work revolves around proper consideration and usage of HTML, CSS, and embedding DotNetNuke components that do interesting things similar to "include" files in Classic ASP or PHP websites. None of this work requires a compiler or two pots of coffee — but it can if that's your thing!

The idea of iterative development lends itself well to the task of skinning DotNetNuke. One of the fundamental tenets of modern software development is the notion of splitting a large task into smaller and smaller chunks. This helps to isolate and identify dependencies as well as reduce the number of things to keep straight in your head, so you can focus on the task at hand.

The sense of accomplishment on a regular basis can go a long way toward problem solving. A task that is allocated several weeks to accomplish might consume several days of pointless thrashing and wasted time before anyone notices. It's easier to spot problems with shorter milestones. They prevent developers from spinning their wheels too long.

When you're building a custom container or designing a unique skin, you shouldn't go too far before checking it in the browser to see if it's behaving as you expect. Start with the smallest file possible, then slowly add to it and check it regularly for errors. If you do this consistently, you'll never get too far along before noticing and correcting a mistake in your files.

Installing Skins

The DotNetNuke administrator has web-based utilities available for installing or updating skins. The various files used to implement a skin are compressed into a single zip file and installed on the DotNetNuke site through a file upload component on a web form.

Once the skins are installed, they can be modified directly by using Windows File Explorer and making file modifications in a text editor. If the DotNetNuke installation is on a remote computer, such as a third-party hosting provider, then an FTP-based approach might suit you better. You learn all about skin installation in Chapter 4, "Exploring Skins."

Browsing the DNN-Blue Skins

A default installation of DNN displays the home page using a skin named *DNN-Blue*. Figure 1-1 shows part of the home page using the DNN-Blue skin.

This skin uses a horizontal menu structure at the top of the page. There are several modules in this page, but Figure 1-1 doesn't give you too much of an idea about where the panes are located. That information is hidden to the normal visitor.

The container around each module applies a consistent blue rectangle. Each container includes the title of the module at the top. The module in the center of the page with the header "Welcome To DotNetNuke®" is an instance of the Text/HTML module. The administrator used the HTML Editor in DotNetNuke to enter the markup into the system so it would render as you see it here.

Figure 1-2 shows the files that make up the DNN-Blue family of skins. It might look like a lot of files to get your head around but there are actually four different skins in this folder. The folder name indicates the family of skins. The files inside a given folder usually have slightly different takes on a common theme. Technically, only one file is required in a given folder to implement a skin. Later on, you'll make your own skin and become very familiar with the contents of a skin folder.

Figure 1-1

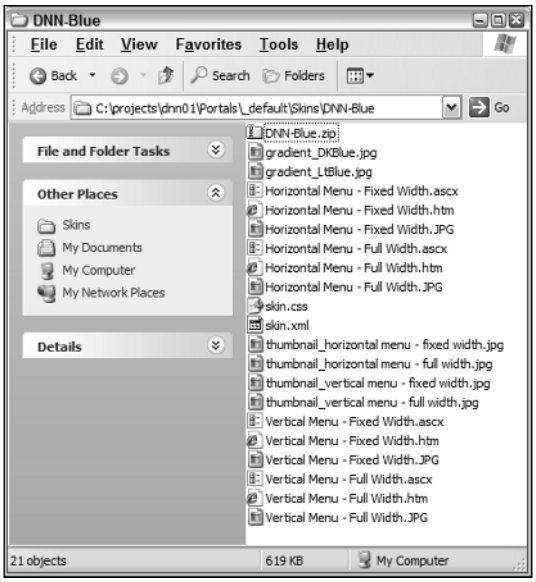

Figure 1-2

You dive deeper into the contents of a skin file in Chapter 4, "Exploring Skins." For now, it's sufficient to know that the DNN-Blue skin comes in four different flavors. By default, DotNetNuke assigns the skin named Horizontal Menu – Fixed Width to a new site. Like the filenames of each skin, the following list describes the characteristics of each distinct skin in the DNN-Blue family:

- Horizontal menu with a fixed width layout
- Horizontal menu with a full width layout
- Vertical menu with a fixed width layout
- Vertical menu with a full width layout

Feel free to explore these text files and get a glimpse of what's going on to produce the default home page displayed in the browser. As you can see in Figure 1-2, the files are located in `<website root folder>\Portals\_default\Skins\DNN-Blue`.

Cascading Style Sheets

Like other modern websites, DotNetNuke makes extensive use of Cascading Style Sheets, or CSS. This technology provides a great deal of flexibility for site design. Rather than hard-coding presentation instructions directly inside the markup of the content, sites that use CSS can omit presentation

instructions altogether or simply label specific blocks of HTML with names. These names are attached to rules in the CSS file, and the rules indicate features such as the color and size of the text.

Before CSS gained the momentum it has today, sites would make extensive use of HTML tables to align and position text. The content would also be littered with instances of the `<font>` tag to define colors or various font sizes. This type of presentation markup contributes to the difficulty in making site-wide changes. Plus, massive manual changes are prone to error. It's hard for a human to make the same change over and over again in a slightly different context without messing up. Major changes done with CSS are usually limited to just one file. One small change to a single CSS rule can affect the entire site. This single change can be easily tested and reverted if the result was not palatable.

The graphic designers use programs such as Photoshop and Fireworks to create composition as one giant image. The web page is created by applying HTML and images sliced out of the composition. The developer uses tools such as Microsoft Visual Studio.Net to build new modules.

You can become a specialist in skinning with a hybrid of these skills. Developing custom DotNetNuke skins is possible when you have some understanding of HTML and cascading style sheets. Most people have a little more experience on either the creative side or the technical side. Either experience is useful in becoming proficient at skinning.

CSS Zen Garden

The CSS Zen Garden website, `www.csszengarden.com/`, is an excellent example of how CSS can transform a web page in a radical manner. Figure 1-3 shows the home page.

Figure 1-3

This site celebrates the creative capabilities of cascading style sheets. It's a great place to visit on a regular basis to draw some inspiration. The home page describes the purpose of the site and how to participate. This organization encourages people who are passionate about CSS to submit their cross browser designs to promote the flexibility and creativity of the technology.

The home page of the website renders the content in the default style and the side bar lists recent designs submitted by CSS enthusiasts. You can click on any of the links in the side bar to see the same HTML markup under the guise of a different theme. It's interesting to compare the emphasis applied by different styles and observe what a given designer is signaling as the most important content on the page. This is the crux of website development: the creative integration of design and content to produce a compelling web page.

These designers create customized CSS files and image files that transform the display of the standard HTML content into prolific representations of a particular style. The CSS file renders the image files as background images. Few sites can demonstrate the creative capabilities of cascading style sheets in such a succinct manner. Take a look at the HTML source for any of the designs and get a look at the HTML tags that can be targeted through CSS.

The most grueling part of this process is arriving at an idea for an interesting website theme. Once the idea is solidified, the follow through is accompanied with several options for achieving the same goal. As with most things in software development, myriad ways exist to accomplish a given task. The array of web browsers and their varied HTML rendering capabilities add a special multiplier to the complexity you'll encounter.

CSS and Web Browsers

Modern web browsers have now reached a respectable level of consistency. Each popular browser continues to have its own unique traits, but to the credit of browsers as a group, they're good stewards in the implementation of established web standards. The list of specific differences in how various browsers render the same HTML and CSS has been reduced enough to allow a respectable amount of authoring by folks who don't eat, sleep, and breathe the cascading style sheet lifestyle.

Internet Explorer 7 is a major leap forward from its predecessor. Firefox 2 is also popular in the tech savvy circles. Macintosh users enjoy the experience of Safari as the default browser. Most browser development teams have a blog that demonstrates their passion as well as the challenges they face in building a great product. You would do well to keep a couple of them on your radar to stay abreast of the current developments.

There are many more browsers. For the purposes of this book, I'll limit the discussion to the aforementioned three. You should know that the vast majority of your users will have a great experience, and a tiny percentage of your visitors might have a near-great experience in other web browsers. You can now feel relieved that the days of Internet Explorer 3 and Netscape Navigator 4 are long behind us, and get on with building great looking sites.

Building a Website with DNN

In this section, you'll briefly cover some common steps in configuring a DotNetNuke website. Figure 1-4 shows a standard Documents module added to a page with a DNN-Blue skin.

The module displayed in Figure 1-4 applied the default container. Containers are just text files, so after a few minutes of editing this file in your favorite text editor, it can look like the page shown in Figure 1-5.

The header has a rounded background, and the printer icon has moved to the top. Notice the two palm trees in front of the header, but behind the table, just to snaz it up a bit. You'll work toward a more comprehensive set of goals to create a customized skin for DNN throughout the remainder of the book.

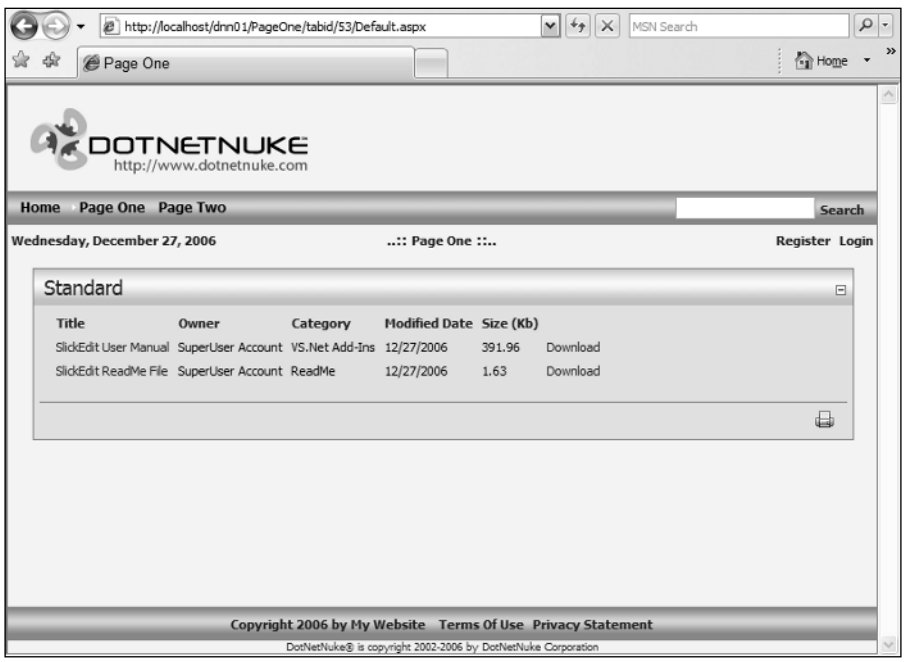

Figure 1-4

Skinning Made Easy

Custom module developers are proficient in a .Net language. Most choose Visual Basic.Net or C#, and they can create a custom DNN module that solves a particular need for a DotNetNuke installation. Similar to the challenge facing a graphic designer, the module developer has the burden of identifying the next great idea. As DotNetNuke skin designers, you don't need to learn a programming language or work with complex business logic, but you still need to keep reaching for that next great idea for a site.

Recent additions to the default set of modules in the current DNN release include the Gallery, Blog, and Store modules. These are excellent additions to the default installation package available on the `www.DotNetNuke.com` website. These modules were once just ideas. When they became real, they were deemed useful enough by the DNN core team to be included in the default installation. It's hard to think of a greater compliment than to have your skin or module included in the default installation of such a widely used platform.

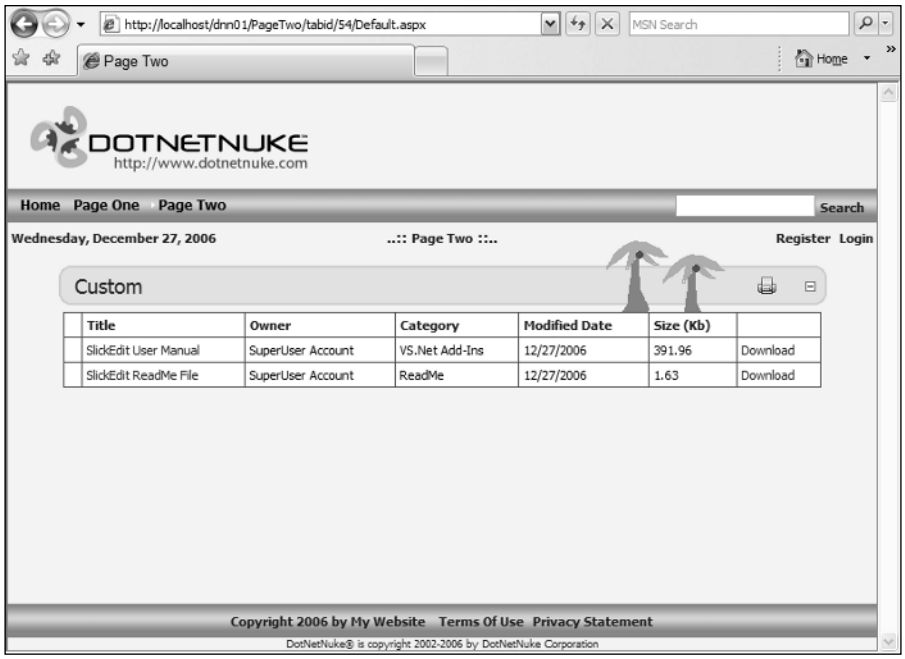

Figure 1-5

Site Map Diagram

When you are building a new site, it's helpful to create a site map diagram that lists all of the web pages in a simple hierarchy. There might be several pages that are cross-linked throughout the website. For the purposes of this discussion, the site map has a simple hierarchy where each subpage is limited to only one parent page. Figure 1-6 is an example of a site map that clearly projects the depth and breadth of the website to be constructed.

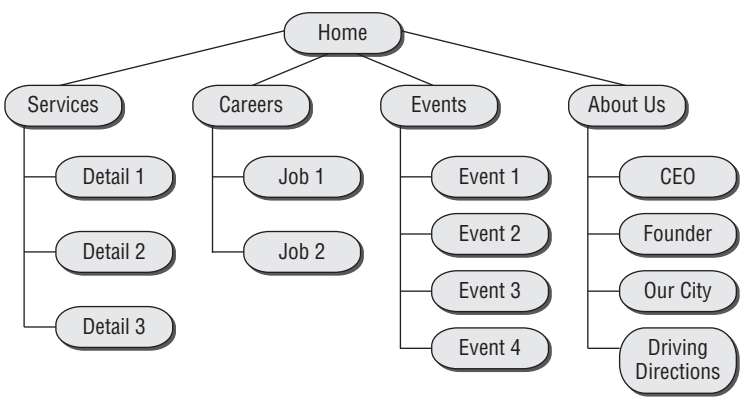

Figure 1-6

Each page in the site map is classified as one of three basic types: the home page, a landing page, or a detail page.

Home Page

The home page is pretty easy to spot; there's just one at the top. All other pages are subordinate to the home page. The skin for the home page can have a very unique design because it's charged with setting the mood for the entire site. It also has to advertise the content under all of the pages. To that end, it might have very few panes and rely on wide, compelling photos in the skin to invite the user into the site. Although this might sound like a great task to start with on a new site, remember that search engines scan for keywords on your site. These keywords are probably on other pages in your site and they oddly become the first page visitors see on your site.

Landing Pages

Most landing pages have the home page as an immediate parent, and they link to multiple child pages. The general notion of the landing page is to provide a summary or overview of the pages below it. Depending on the site, landing pages might come in a few flavors. From the site map diagram, patterns will begin to emerge, and consideration can be given to the number of unique landing page and detail page designs. Unless each landing page will make a radical departure from the design of its siblings, it's OK just to have one graphic composition of the landing page and note the color or photo variations along each branch in the site map.

Detail Pages

Detail pages have a very narrow focus by design. They usually require a wide pane in the skin for a table of documents or lots and lots of text. Because the visitor is deep within the site when they arrive at a detail page, a breadcrumb is a nice navigation feature to give them some context about their location in the site map.

If you're building a public website and not an internal business application, you can assume visitors are coming to the site to read content. Detail pages make up the majority of the site, followed by landing pages and the one home page, so you would do well to spend the majority of your time building a solid detail page, and then working backward.

I've seen an enormous amount of time spent on the home page, which has precious little content. At the same time very little conscious effort was given to the detail page layout and the voice of the content. The content on the detail pages supplies the real value to the website. Because public sites are scanned by search engines and visitors are linked to detail pages with the matching keywords instead of the home page, it seems odd to spend any time on the home page until the rest of the site is fully baked.

Parent-Child Page Relationships

Parent-child relationships are interesting in DNN. On a typical static website, where every page is represented by a unique file on the hard drive, the landing page is special. By convention, it's the default file in the directory. For example, if the URL `www.company.com/services/` were typed into an address bar, the default file in the directory would appear in the browser.

It's the job of the web server to know the default file in the directory so it can be served up when no specific file is requested. The Windows web server, IIS, has an editable list of default files. When no specific file is requested, IIS traverses the list in top-down order and serves up the first matching file it discovers in the directory. By convention, the file is named `default.aspx` in an ASP.NET environment.

In DNN, the parent-child relationship is virtual, not physical. Suppose a website has the following URL that displays information about the company:

```
www.company.com/about-us/default.aspx
```

If this were a DNN site, the web server would not have a physical file named `default.aspx`; in fact, it would not even have a physical directory named about-us. This is the magic of DotNetNuke and its URL rewriting feature.

ASP.Net supports a feature that allows the web server to intercept a web request, perform and look up in the database, and potentially redirect the web request to another page on the site without ever telling the visitor. This can be a very powerful and valuable feature, and DotNetNuke takes full advantage of it. In fact, there's basically only one web page in the entire site. Its name is `default.aspx` and it's located in the root directory. All of the modules and various features are loaded dynamically into this page and the URL rewriting feature makes it appear as though a site might have hundreds of pages.

Setting the Parent and Child Pages

On the plus side, the URL rewriting feature allows pages to move around on the site with ease. The relationship can be redefined at any time because the change occurs in the database, where DNN stores most of its information. Figure 1-7 shows a portion of the DotNetNuke administration page that indicates the parent of the given page. Changing this value will move the location of the page in the navigation. Because the field is in a drop-down list, there's a chance you'll pick the wrong parent, but you won't mistype it.

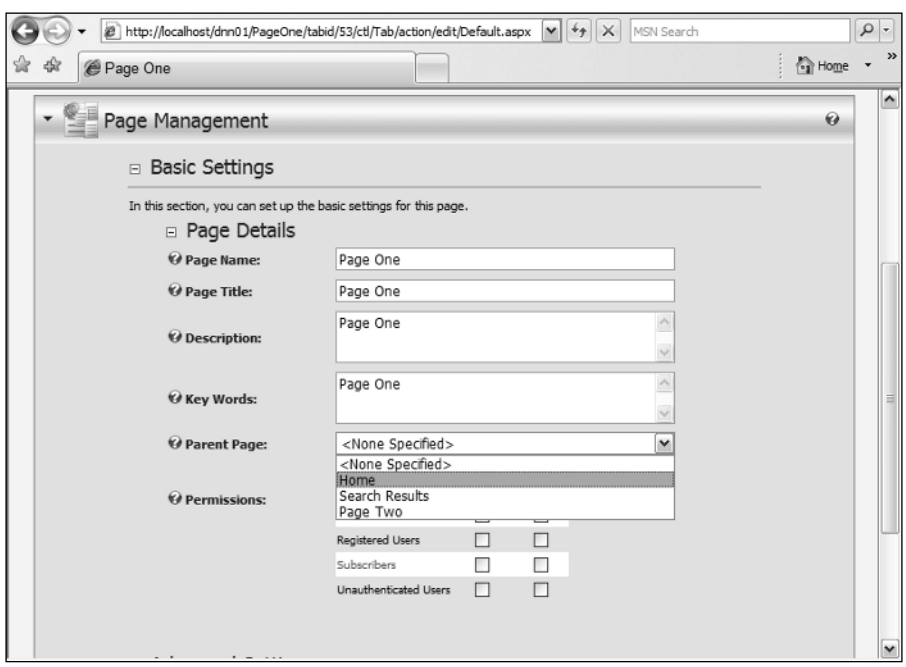

Figure 1-7

This dynamic parent-child association feature takes a burden off the shoulders of the build team. During the course of website development it's common to move pages around much like furniture in a new apartment. Because all of the relationships are managed through a web-based interface, there is less risk of losing a file, misnaming the file, or other oversights.

Installing DNN Automatically

Provided you have all of the prerequisite software available, DotNetNuke can install itself. Presuming the web server and database server are already on a computer, these are the basic steps to install DotNetNuke:

1. Download and unzip the bits from `www.dotnetnuke.com` into a folder.
2. Inside IIS, the web server in Microsoft Windows, create a new website that points to the folder in Step 1.
3. DotNetNuke will create and modify files on disk, so verify the appropriate file permissions are set on the folder.
4. As with most other ASP.Net web applications that use a database, verify that the database connection in the `web.config` file is pointed at the appropriate database — in this case, a new (empty) database that you created or one created for you on the corporate network.
5. Launch a web browser, and answer a few questions in the DotNetNuke Installation Wizard.

These installation steps are covered in much greater detail in Chapter 2, "Installing DotNetNuke." For now, just keep the basic installation concepts in mind.

Upon the initial web request, DotNetNuke will detect the need to execute the auto-installation process. As DNN traverses the list of installation tasks, the browser displays the growing list of completed steps. The system first installs the core framework, then upgrades to the current version, and finally extends itself by installing a default set of modules. Based on the speed of the computer, all of this takes place in just a few minutes.

Figure 1-8 shows a page displayed after a successful installation. The middle of this long web page has been cropped to show the starting and ending messages. The bottom of the page indicates the entire installation process took just over a minute to execute locally on a run-of-the-mill laptop. Clicking the bottom hyperlink will navigate to the default home page of the DotNetNuke installation.

Several installation options and considerations can be exercised before installation and post-launch, yet the basic software package remains approachable to people just starting their exploration of web-based systems. When the situation calls for it, the system has enough layers to peel back and customize in order to satisfy the extensibility and customization requirements of complex systems. The last chapters of this book examine some complex scenarios that you'll gradually build toward.

DotNetNuke has matured into a feature-rich application that spans the needs of many scenarios. At the same time it has stayed true to its origins by remaining approachable on many levels, including custom designs through skinning.

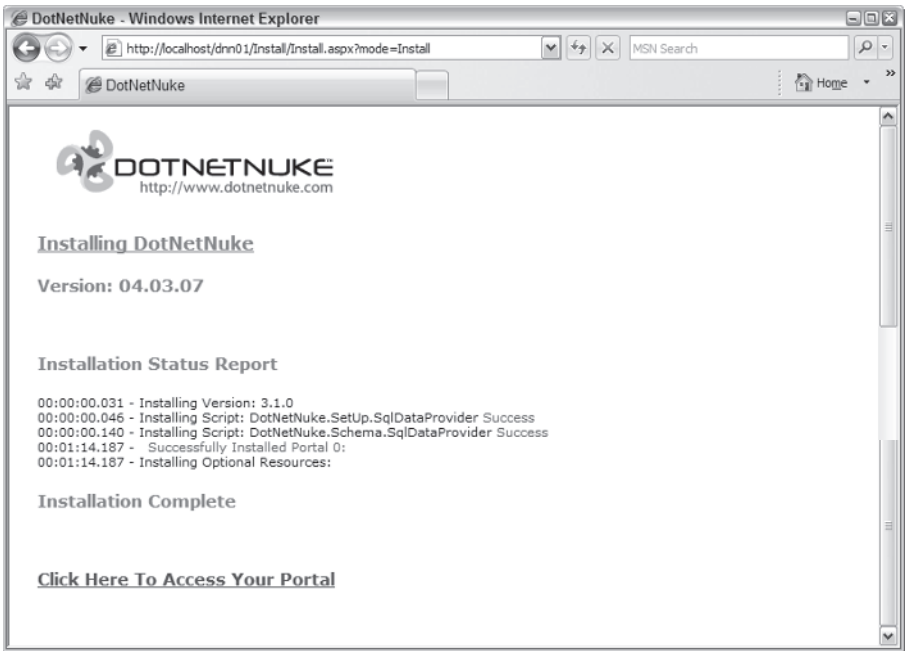

Figure 1-8

Summary

This chapter discussed a broad overview of the DotNetNuke framework and the tangential aspects of creating a database-driven ASP.Net website. The following chapters are much narrower in scope. This chapter introduced you to the following items:

- The roots of DotNetNuke and its popularity statistics
- What skinning DotNetNuke is and how it relates to graphic design and custom module development
- Modules, containers, and skins as the components of DotNetNuke
- The installation process