Developing Solutions with EPiServer

by Rolf Åberg

mailto:info@episerver.com
http://www.episerver.com

Developing Solutions with EPiServer

ISBN 91-631-5351-3

Published by:

ElektroPost Stockholm AB
Finlandsgatan 38
SE–164 74 KISTA
Sweden

Tel. +46 (0)8–444 19 30
Fax +46 (0)8–444 19 59

First published 2004

Författares Bokmaskin, Stockholm 2004

Set in 11/13 pt Garamond, headings in Tahoma.

Contents

Table of Contents

Foreword

Our main focus at ElektroPost has always been the person, man or woman, developing information solutions, in short the developer. Having this focus has been key to EPiServer's huge success which we find both humbling and inspirational in our continued work.

It is the ability of the Web developer to understand and make the most of any development product that enables them to create the best and most innovative customer solutions. Several hundred Web developers are creating information solutions with EPiServer on a daily basis. Thanks to them EPiServer is today one of most wide-spread Web content management systems in the Nordic countries. We're also able to see a growing interest from organisations and companies elsewhere. We hope that this book can become a stepping stone in the effort to spread knowledge and information to new groups of developers and new market places.

The information solutions market place is crowded by many suppliers and products displaying an array of more or less mature offerings. Many customers and consulting companies are actively seeking a solutions platform to which they can make a long-term committment. We know that EPiServer is the product to fulfill those needs.

A book cannot work miracles but it will make your development efforts easier. Our goal is to shed light on EPiServer development and, hopefully, to serve as a source of both inspiration and knowledge in your daily work. It's high time to open up your senses to the potential and possibilities that are EPiServer 4.

Welcome to the EPiServer Universe, be sure to also visit our Web site at http://www.episerver.com

Per Rask, Managing Director
ElektroPost

We would like to take this opportunity to thank everyone at ElektroPost who has worked hard to make EPiServer a success:

Anna Olsson, Daniel Maurer, Erik Skagerlind, Fredrik Tjärnberg, Gøran Hüllert, Henrik Alm, Janne Westberg, Joakim Ribb, Johan Olofsson, Linus Ekström, Magnus Stråle, Maria Hedlund, Mats Hellström, Mikael Runhem, Odd Simon Simonsen, Per Bjurström, Per Rask, Roger Eriksson, Roger Wirz, Ruwen Jin, Steve Celius, Susanne Magnusson, Tina Runhem, Øyvind Wabakken Hognestad.

1
Windows Development and ASP.NET Basics

EPiServer 4 is a Web content management system based on Microsoft .NET in general and ASP.NET in particular. Through extensive use of templates it's very easy to get a comprehensive Web site up and running quickly without any programming at all. However, to reap the full benefits of EPiServer 4 some software development might be needed. The aim of this book is to make you a successful EPiServer 4 developer.

In this, the first chapter, we'll begin with an introduction to Windows application development and ASP.NET overview.

If you're thinking about skipping this chapter at least take a good long look at figure **1-6** on page 10.

How Does EPiServer Work Its Magic?

EPiServer, being an ASP.NET application in itself, benefits from Microsoft .NET Framework and its rich API set. As you can imagine though, there's a lot of EPiServer involvement necessary to transform an EPiServer Web Page as it was created in EPiServer Edit mode into the HTML page viewed in a Web browser. To fully appreciate this we begin by repeating a few Windows and ASP.NET concepts.

ASP.NET Web Forms Are Event-Driven, Making Them Appear Much Like Windows Forms

Ever since the twentieth century, when Windows was little more than a fancy file handler, it has been a message passing system. Messages were passed between Windows and Windows applications, and of course between the various parts of Windows. Messages were often created by translating both software and hardware events. When the mouse was moved the resulting hardware interrupt was captured by Windows which translated it into a message and passed this message to the pertinent application for proper handling.

Before Visual Basic was introduced this message handling was prominent when developing Windows applications: according to developer lore these applications centred around a central message loop which processed messages from its message queue (supplied by Windows). With the arrival of Visual Basic the programming paradigm changed and the message loop wasn't needed in the foreground any more. Instead developers would create event handling functions

which were invoked by Windows whenever something happened. The event-driven programming model was born.

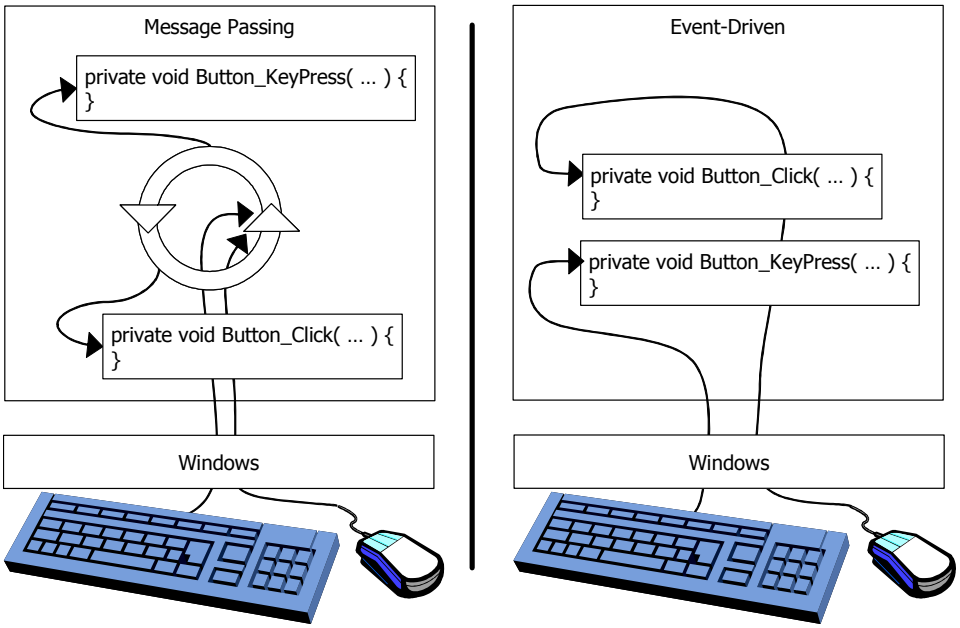


Figure 1-1: Comparing Message Passing, to the left, and Event Driven programming, to the right.

In figure 1-1 we see a message passing Windows application on the left and an event-drive Windows application on the right. In the message passing Windows applications hardware and software events were translated by Windows into messages. These messages were sent to the pertinent application and handled in the message handling loop. The message handling loop mainly comprised a C language while statement surrounding a very long switch statement.

Example 1-1: Pseudo code for the message handling loop in early Windows applications.

```
MSG msg;
while ( GetMessage( &msg, NULL, 0, 0 ) ) {
    switch ( msg.message ) {
        case WM_LBUTTONUP :
        case WM_RBUTTONUP :
        case WM_MBUTTONUP :
            Button_Click( msg.MousePos );
            break;
        case WM_KEYUP :
            Button_KeyPress( msg.KeyPressed );
            break;
    }
}
```

So, earlier Windows programs spent their time in the message loop waiting for messages to arrive.

For event-driven applications the story is very different. There's no 'visible' message loop, there's no unpacking of strangely packed attributes and there's no interpreting of messages. Instead event-driven applications have a lot of event handling code, routines that are 'automagically' called by the run-time system when a hardware or software event occurs. The fingerprint of the event handling routines, i.e. their return data type (if applicable), number of and data type for formal attributes is determined by Microsoft when they design the development and run-time environment for the computer language (how it interacts with Windows). From this it follows that when there are no events to handle an event-driven application is simply idling away, doing absolutely nothing.

One way to look at event-driven programming, with a message passing twist, would be to say that the message handling loop is moved from every application into a run-time system which closely interacts with Windows. It is the run-time system that calls the event handling routines and passes the pertinent arguments.

The event-driven paradigm is very much alive in Microsoft .NET Framework, which is obvious in both Visual Basic .NET and C#.

Translate the Event-Driven Model to the Web, Do Not Create an X Window System

With the success of the event-driven programming model, it was only natural for Microsoft to also use that for Web-based applications. An interesting and widespread system that uses the event-driven model in a network is the X Window System. The system centres around an intelligent and powerful server with X Terminals connected to it over the network. X Terminals aren't supposed to be able to do very much on their own – everything is processed on the server. This means that the mouse cursor is controlled by the server. When the mouse is moved a message is passed from the X Terminal to the server which calculates the new mouse cursor position and passes back a message to the terminal which eventually redraws the mouse cursor at the new position. This is fine in a high-speed network and with a limited amount of clients, but what if we were to deploy a Web application using the same technique?

In doing this there's a very big performance issue to consider: mouse movements cannot be expected to be handled by the server! And in ASP.NET they're not. In fact, the standard Web Forms Controls (Button, Check box, Label, Radio button, etc.) that are available for Web Forms may look just like their Windows Forms Controls cousins, but there are far fewer events defined for them (that they can act upon). A comparison between a Web Forms Button control and a Windows Forms Button control reveals that the Web Forms Button control has a mere eight events where as the Windows Forms Button control has 57 events. The only events they have in common are Click and Disposed. So, none of the Drag events, Key and Mouse events from the Windows Forms Button control are available for the Web Forms Button control, and this holds true for all Web

Forms controls. Of course, there's always the possibility of handling all processing on the client, so bypassing the whole issue. But that's not the route Microsoft chose for ASP.NET. They realised that a lot, as in very much, could be won from letting the server handle most, if not all, of the processing.

Handling Events on the Server

One of the major benefits of ASP.NET is that it provides server controls, controls whose processing is handled by the server, not on the client.

Example 1-2: Declaration of an ASP.NET Button control.

```
<asp:Button id="AspNetButton" runat="server" Text="ASP.NET Button"></asp:Button>
```

The code in example 1-2 shows what the HTML code looks like when an ASP.NET Button object has been added to a Web Form (or Web User Control). What makes it interesting is the attribute 'runat' which decides that this particular instance of an ASP.NET Button should be processed on the server. Adding an HTML Button object looks like this in HTML mode of Visual Studio .NET:

Example 1-3: Declaration of an HTML Button control.

```
<INPUT type="button" value="HTML Button">
```

So, we've decided to handle processing on the server, now we need to maintain some kind of visual consistency for the user. In other words, how do we make sure that user input isn't erased as we go back and forth between the client and server and that we can change control attributes programmatically? In ASP .NET the answer is two-fold: wrap everything in a form and preserve view state for ASP.NET controls such as Labels.

Using Forms Help Preserve Visual Consistency

When Visual Studio .NET is used to create a Web Form, all controls are automatically enveloped in a form.

Example 1-4: Form created by Visual Studio .NET.

```
<form id="Form1" method="post" runat="server">
...
</form>
```

The form itself is considered to be an ASP.NET object, as you might guess from the 'runat' attribute seen in example 1-4.

When an ASP.NET Button is 'clicked' the whole contents of the form is shipped off to the server for processing. For ASP.NET controls this processing takes place in the compiled code-behind file.

Controls that allow the user to change an attribute, such as filling in text in a text box or checking a check button preserve their setting by simply changing the HTML that is posted back to the client to reflect the current 'settings'.

ASP.NET View State Variable ViewState

For some ASP.NET controls, predominantly the Label control, ASP.NET utilises a special variable called '__VIEWSTATE' to preserve the control's visual state. But the view state variable isn't limited to handling state values belonging to controls, it can also be used to handle state values for the page as a whole. As the view state is passed between the client and server with the form both have up-to-date information about the view state. This means that you're guaranteed that code on the server can access the view state contents without any performance penalty – it never triggers a server–client round-trip. This also holds true for any code you might want to have processed on the client side. Having said that, we cannot simply assume that using the view state variable is a panacea for solving all problems. As the number of view states contained in the view state variable grows, so does its own size, increasing the transfer times between Web client and server.

The contents of view state variables are preserved in a hidden input HTML field.

Example 1-5: Actual View State variable.

```
<input type="hidden" name="__VIEWSTATE"
    value="dDwtNDM1NDIwNzU7Oz5aI7A1H+3YFiX9HB/U6X6zksDC/w==" />
```

Apparently the view state variable contents aren't encrypted but an HMAC-SHA1 (Hashed Message Authentication Code-Secure Hash Algorithm is a method to calculate secure check sums) digest is appended that protects it from tampering. The resulting text is then Base64 encoded.

Example 1-5 shows a very short view state string; they can easily grow to several kilobytes. The example EPiServer Web site has a view state of about 6000 bytes when the start page is first loaded.

EPiServer uses the view state variable. Values, settings, in EPiServer (EPiServer.WebControls.Property) have a standard boolean attribute EditMode which controls whether the value, setting, should be rendered when the EPiServer Web Page is viewed in Edit mode or only in the regular View mode. EditMode is part of the control's view state. EditMode is used in, e.g. the Profile Web User Control (Profile.ascx) which ships with EPiServer.

Example 1-6: Wrapping view state EditMode in a property function in Profile.ascx.

```
public bool EditMode {
    get { return ViewState[ "EditMode" ] == null ? false : (bool) ViewState[ "EditMode" ]; }
    set { ViewState[ "EditMode" ] = value; }
}
```

Avoiding Re-Initialising with Every Form Posting: IsPostBack

A third important piece in the ASP.NET foundation is the ability to decide whether the current page load results from a post back or if it is the first time the Web Form is being rendered. This is implemented using a boolean Page attribute

IsPostBack. Since IsPostBack is a standard attribute of System.Web.UI.Page all EPiServer Page Template Files also have access to it (since System.Web.UI.Page is part of the inheritance chain for Page Template Files).

IsPostBack is used to load initialisation data only once, among other uses.

Example 1-7: Typical use of IsPostBack in EPiServer solutions.

```
private void Page_Load( object sender, System.EventArgs e ) {
    if( ! IsPostBack ) {
        DataBind();
    }
}
```

Example 1-7 shows DataBind being called in the Page_Load function. While this is fine, it bears the sign of a fall-back solution to letting the individual control objects handle their own data binding.

Events Are Handled in Compiled Code on the Server

One of the ASP.NET breakthroughs is the ability to separate presentation and code. ASP.NET even has physical separation: presentation objects go in one file and the supporting code is put in its own file.

ASP.NET Web Forms files have the standard extension 'aspx' and its supporting code file has the same name and a second extension that is unique for the computer language used, e.g. 'aspx.cs' for C# (Visual C# .NET) and 'aspx.vb' for Visual Basic .NET. (ASP.NET Web User Controls use the same double extension pattern, the visual parts file has an 'ascx' extension and the code file uses 'ascx.cs', etc.)

'Code-Behind File' Is Microsoft's Term

Microsoft has dubbed the supporting code file 'code-behind file'.

Example 1-8: C# code in the code-behind file to handle the Click event for an ASP.NET Button.

```
private void AspNetButton_Click( object sender, System.EventArgs e ) {
}
```

Compare this with the declaration for a Button in a Windows Forms application.

Example 1-9: C# code in the source file to handle the Click event for a Windows Forms Button.

```
private void WindowsFormsButton_Click( object sender, System.EventArgs e ) {
}
```

As you can see they're identical, which should mean, and indeed does mean, that you handle the click event in the same way irrespective of where it came from, an ASP.NET Web Forms application or a Windows Forms application.

When Compiled Code-Behind Files Form a Dynamic Link Library (DLL)

When the code-behind files for an ASP.NET application are compiled, a Windows Dynamic Link Library (well, of course it's really a Microsoft .NET Framework Dynamic Link Library) is created and put in the bin (a folder named bin). The next time this code is needed it's already compiled and the whole process should be faster.

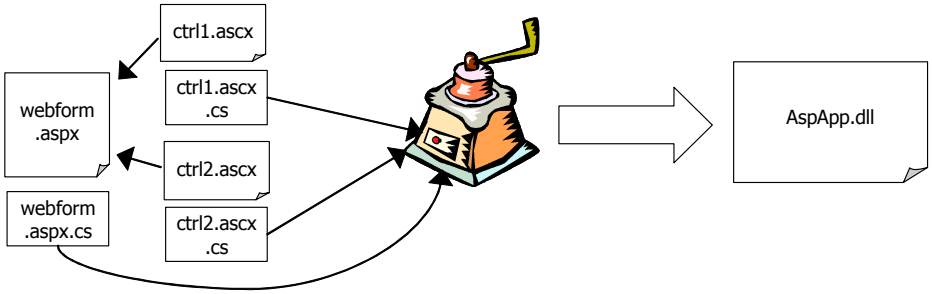


Figure 1-2: Code-behind files for an ASP.NET application are compiled into a dynamic link library file.

If we use figure 1-2 as an example we see that the application, called AspApp, is comprised of one ASP.NET Web Form and two ASP.NET Web User Controls, each with a corresponding code-behind C# file. When the application is compiled in Visual Studio .NET, each of the three source code files, Webform.ascx.cs, ctrl1.ascx.cs and ctrl2.ascx.cs, is compiled into a single Dynamic Link Library File, AspApp.dll, and put in the application's bin folder on the Web server.

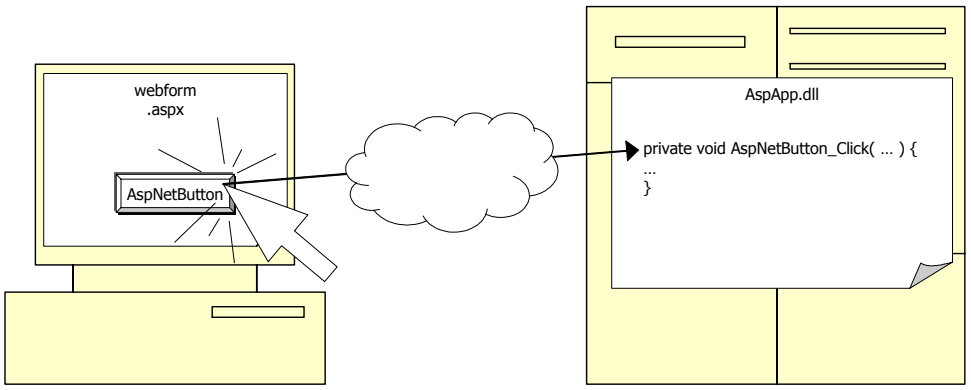


Figure 1-3: Clicking an ASP.NET Button on a client results in handling code being called in the code-behind DLL on the Web server.

The ASP.NET Magic: Going from a Web Form to an HTML Page

When Internet Information Services, IIS, receives a request for an HTML page it simply returns the page. When it comes to ASP.NET Web Forms, aspx pages, IIS calls on help. So, who do you call? Well, this is configured for IIS.

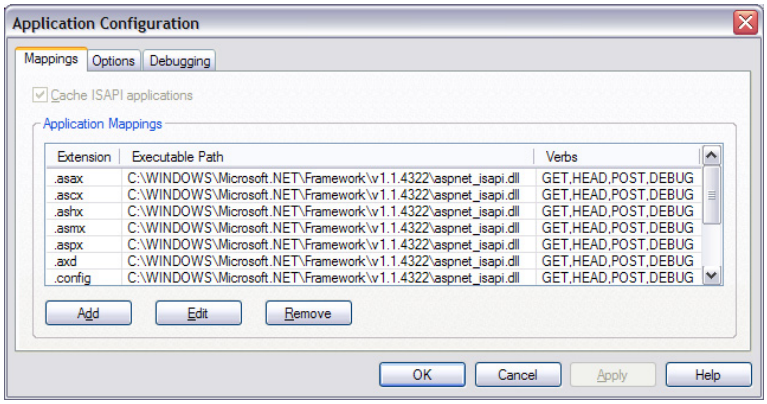


Figure 1-4: Linking file extension to handler (picture edited).

As can be seen in figure 1-4, ASP.NET Web Forms, aspx files, are handled by ASPNET_ISAPI.DLL for IIS.

So the request for an aspx files is handed off to ASPNET_ISAPI.DLL, then what? Basically the request is passed from ASPNET_ISAPI.DLL to the ASP.NET worker process ASPNET_WP.Exe (W3WP.Exe for IIS 6.0), which reads and processes the pertinent aspx file. This processing also involves compilation of code-behind files. After everything is processed, HTML is passed back to the client via the same route the request took.

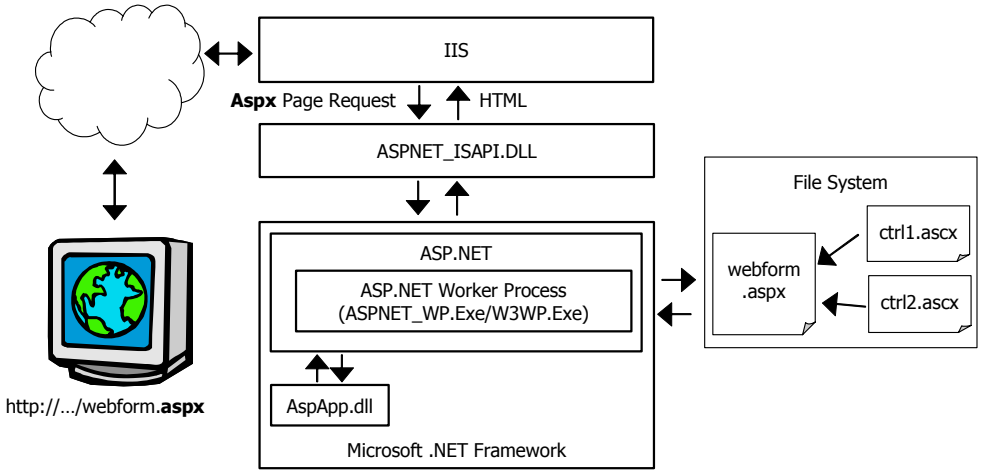


Figure 1-5: The ASP.NET chain from client to server and back.

The picture in figure 1-5 is based on a few assumptions:

- There is an ASP.NET Web Form, webform.aspx, with two ASP.NET Web User Controls, ctrl1.aspx and ctrl2.aspx
- There's a code-behind file for each of the aspx and ascx files compiled into a dynamic link library, AspApp.DLL
- The user types in the URL, or clicks on a link pointing to, http://*Web server*/*path*/webform.aspx

Going from an EPiServer Web Page to an HTML Page in the Visitor's Browser

Plain vanilla ASP.NET Web Forms and ASP.NET Web User Controls inherit from System.Web.UI.Page and System.Web.UI.UserControl. This guarantees that wiring as described in figure 1-5 is available.

EPiServer Page Templates and Framework Definition Files are not plain vanilla ASP.NET object classes since they do not directly inherit from the same classes as do Web Forms and Web User Controls. Instead they inherit from EPiServer.PageBase.

Table 1-1: Inheritance for EPiServer object types.

EPiServer Object Type	*Main Class Inherits From*
Framework Definition File	EPiServer.WebControls.ContentFramework, a descendant of EPiServer.UserControlBase
Page Template File	An EPiServer.PageBase descendant, often EPiServer.TemplatePage
Web User Controls	EPiServer.UserControlBase

EPiServer doesn't forego the ASP.NET goodies, all of the EPiServer classes mentioned in table 1-1 are descendants of relevant ASP.NET classes, such as System.Web.UI.Page.

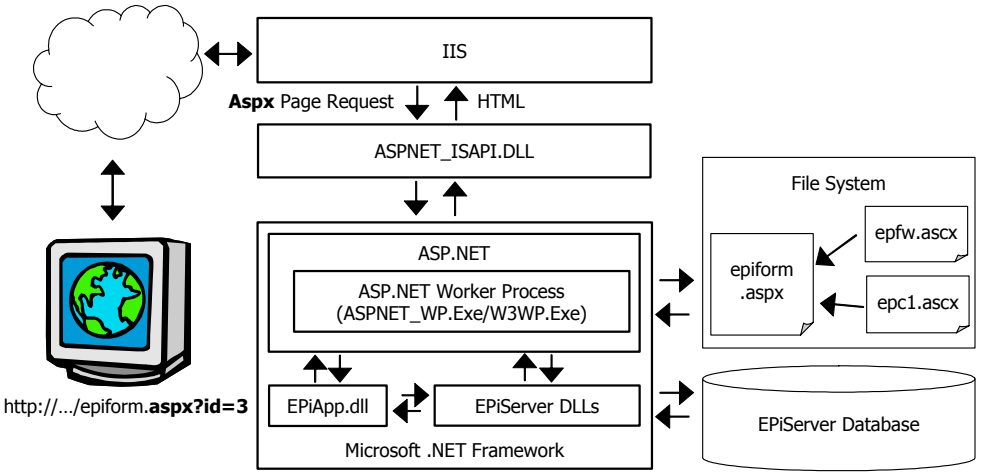


Figure 1-6: The ASP.NET and EPiServer chain from client to server and back.

The differences between figures 1-5 and 1-6 aren't great, but they are important:

- An EPiServer Page Template Page, epiform.aspx, is used
- There's a Framework Definition File, epfw.ascx, used by the Page Template File
- The code-behind files have been compiled into EPiApp.DLL
- The call is made not simply to 'epiform.aspx' but to 'epiform.aspx?id=3'

The rather innocuous looking call to 'epiform.aspx?id=3' is responsible for triggering a lot of EPiServer action. Remember that no EPiServer Web Pages are stored on disk, they are merely a combination of a Page Type and current contents for the various EPiServer values (called Page Properties) defined for the particular Page Type which was used as a basis for the Page. One way of viewing the EPiServer operations is as follows:

1. The Page Template File is parsed and compiled (only occurs the first time the page is executed).
2. The contents of the Framework Definition File are read and inserted into the Page Template File.
3. The OnInit method of the controls inserted in step 2 is called.
4. The OnInit method of the Framework Definition File is called.

5. The OnInit method of the Page Template file is called, causing page data to be loaded, user access permissions checked, etc.
6. The Page Load event for the Page template File is triggered.
7. The Page Load event for the various controls initialized in step 3 is triggered.
8. The HTML results are sent back to the caller.

The list above is really rather fascinating. It looks so simple and effortless on paper when in reality many years of hard work have been required to make both ASP.NET and EPiServer realities. One of the more interesting steps in the list, from an EPiServer point of view, is number 5 which perhaps needs a more detailed explanation. The processing at this point involves a lot of the EPiServer run-time infrastructure (mostly located in EPiServer.dll). If you look at figure 1-6 you can see the cooperation between ASP.NET, EPiApp.DLL, EPiServer DLLs, File System and the Database. All of these systems are active during this step in the processing. These are some of the processing sub-steps performed by EPiServer components at this point:

- Access control lists for Web Pages are checked for the current user identity
- EPiServer Content controls are enumerated and their contents are retrieved from the database tables

You'll Be Seeing EPiServerSample.DLL a Lot, Not EPiApp.DLL

In the example above we assumed that the code-behind files for the EPiServer solution we created were compiled into EPiApp.DLL. When you install EPiServer, the solution will have the name EPiServerSample and compilation will produce a Dynamic Link Library called EPiServerSample.DLL.

2
EPiServer Overview and Operation

Developers need to know about basic operation of an EPiServer Web site, so we'll discuss that but first an overview of EPiServer 4.

EPiServer Overview

As a content management system EPiServer has been designed with four crucial objectives in mind:

- Make the resulting Web site fast for the viewer
- Make the Web site contents easy to change and maintain
- Make the Web site easy to administer, whilst providing powerful functions when needed
- Make EPiServer functions comprehensive and easy to use for developers

We at ElektroPost are confident that these goals will become apparent as you learn more about EPiServer.

EPiServer 4 Is Only the Latest Incarnation

EPiServer 4 comes from a line of successful EPiServer products. The first version was released commercially in 1997. Version 3, the immediate predecessor to version 4 was object based, but not object oriented (much like Visual Basic before Visual Basic .NET). Being based on Microsoft .NET and ASP.NET, EPiServer 4 is object oriented within the specific limitations of Microsoft .NET, e.g. there's no multiple inheritance in Microsoft .NET.

Given the long lineage of EPiServer and the vast possibilities of Microsoft .NET we at ElektroPost hope that you will enjoy using EPiServer as much as we've enjoyed creating it!

The goals we set for EPiServer 4 included:

- Greater simplicity for EPiServer developers
- Implement EPiServer 4 also as an XML Web Service to promote interaction with other systems

- Increased performance utilising the new cache functions in Microsoft .NET and EPiServer

EPiServer 4 itself is written in C#.

Relation Between EPiServer and Internet Information Services, IIS

Being based on ASP.NET, EPiServer 4 demands the use of Internet Information Services, IIS, 5 or 6 for Web server.

EPiServer 4 Is Backwards Compatible

Backwards compatibility is really not something developers embrace. However, the people using our products are always very grateful when we do make something new work well with our previous products. Heeding this, EPiServer 4 is fully backwards compatible with existing customer solutions.

If the customer so wishes, an existing solution can be gradully upgraded to EPiServer 4. Part of the credit for this goes to Microsoft .NET Framework which allows for both ASP and ASP.NET techniques to cohabit the same application.

EPiServer's View of People

During use of a typical EPiServer 4 Web site, three roles can be distinguished: Web site viewers, Web site editors and Web site administrators. The Web site viewers are the people using the information on the Web site. Editors are charged with creating and maintaining content and the administrators are ultimately responsible for administering the Web site, including making sure that editors have access to the pertinent part of the Web site contents. As EPiServer is created for, and used by, organisations and companies of all sizes, this division is very flexible: often editors are also administrators.

Add to this the EPiServer 4 developer and we have a full complement of roles. As a developer you will certainly play all parts from time to time.

Division of Responsibilities between Roles

When discussing the division of responsibilities between roles for Web site development and maintenance we concentrate on three roles: Web site administrators, Web site editors and EPiServer developers. In short, Web site administrators are very much in the driver's seat as they alone decide what elements should make up the Web site. Developers act as producers of Web site elements for the administrators to make available to the Web site editors.

Typically, but in no way demanded by EPiServer, a Web site administrator is either an IT employee or a 'power user', as the role of the administrator is somewhat technical. As maintaining the Web site's contents is crucial, Web site editors are simply the people who would otherwise would be responsible for information dissemination in the organisation.

EPiServer DLL and Executable Infrastructure

The EPiServer DLLs that have been alluded to, e.g. in figure 1-6, make up the EPiServer infrastructure, i.e. the parts that aren't implemented as ASP.NET Web User Controls.

Table 2-1: EPiServer DLLs and executables.

File Name	*Purpose*
ElektroPost.Licensing.dll	License handler.
ElektroPost.Win32.dll	File access permissions handling, IIS meta base interface and compression support.
EPiServer.CodeBehind.dll	Supporting logic for the built-in Web forms and user controls, e.g. Admin and Edit modes and ASP.NET Web User Controls.
EPiServer.dll	Main module, by far the biggest file at 800+ kilobytes.
EPiServer.Enterprise.dll	Support library for Web site import and export.
EPiServer.Install.dll	Installation support.
EPiServer.Scheduler.dll	EPiServers Scheduler support.
EPiServer.Scheduler.WKTL.dll	Defines the interface used to communicate between EPiServer and the scheduler service (Well-Known Type Library, WKTL).
EPiServer.SchedulerSvc.exe	Windows Service to handle scheduled jobs.
EPiServer.Workflow.dll	EPiServer work flow handler.
ldapper.dll	Connector for Directory Services based on Lightweight Directory Access Protocol, LDAP, such as Microsoft Active Directory.

EPiServer Operation

EPiServer Is a Trifurcated Application

We've already stated and hinted at it several times: EPiServer comprises three different but equally important parts, listed here in relation to usage frequency:

- View mode
- Edit mode

- Admin mode

View mode is the mode enjoyed by visitors to the Web site. It's the 'normal' Web server mode.

Edit mode is used by EPiServer Editors, the people responsible for adding and maintaining content for the Web site.

Admin mode, of course, is for EPiServer Administrators, the people who busy themselves with matters such as 'who can do what to which page?'. Admin mode is also used by developers since it's most often they who create Page Types from the Page Templates they've created themselves in Visual Studio .NET. Whether the Page Type or the Page Template is created first doesn't matter, either way is fine.

All Modes Are Multi-User

Its rather obvious that a Web server would allow multiple simultaneous viewers, but it might not be obvious that EPiServer also allows multiple Editors and Administrators to work at the same time.

In day-to-day operation, Administrators and Editors go about their chores. Administrators mainly busy themselves with access permissions and logins whilst Editors merrily add new, change or remove old content to/from the Web site.

From an EPiServer developer point of view the most important thing about what Administrators and Editors do is Page Types and their Properties. Typically this forms only a small part of the Administrators' work load but all of what Editors do.

The Most Important Admin Mode Assignment

Developers are charged with maintaining a list of available Page Types and their properties. These Page Types are little more than Page Template Files, possibly with some added extra properties. Properties are either built-in or created by Administrators. As properties are created Developers may provide default values.

Editor Efforts Are Crucial

Editors are the users of Page Types and their properties, they are the ones supplying properties' initial content and, perhaps, changing and deleting that content. As content is entered or changed, the database table tblProperty is used to read from and write to (as well as several other database tables).

Admin Mode

EPiServer Admin mode is available to accounts that are members of the EPiServer group WebAdmins or the local Administrators group, directly or indirectly.

There are two ways to enter Admin mode in EPiServer:

- Append '/admin' to your Web site's URL, e.g. 'http://widgets.info/admin', and navigate there; after logging on, you're in Admin mode
- Navigate to your Web site and log on (using whatever means available, e.g. if you've included a log-on tool button, such as), then right-click and select Admin mode from the shortcut menu

Once you're logged on, you'll have access to Admin mode and Edit mode from View mode on the shortcut, right-click, menu.

The tool-bar in the upper left corner of the left pane in Admin mode is somewhat frugal as it contains only three tools.

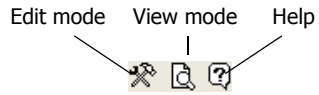


Figure 2-1: Admin mode tool-bar (EPiServer version 4.2).

In Admin mode you have access to a lot of tools in the left pane together with all Page Types that have been defined.

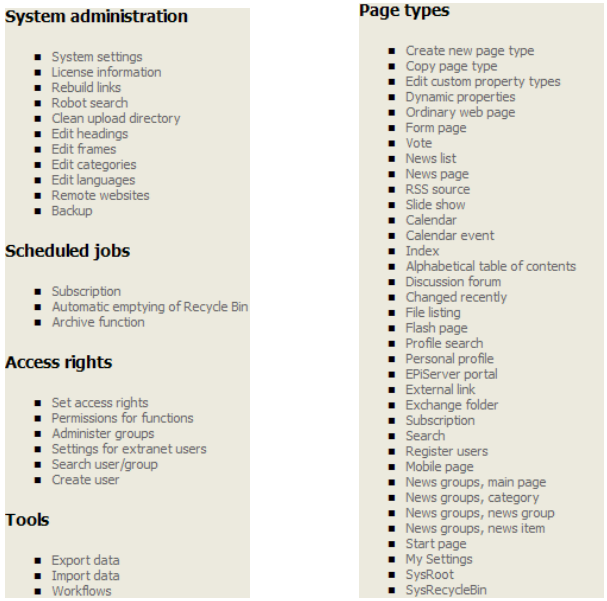


Figure 2-2: Tools and Page Types in the left pane of EPiServer 4.2 Admin mode (pictures have been re-arranged).

Names of Property Data Types Are Different in Admin Mode

The names of Property Data Types are a little different in Admin mode; keep this in mind when you add Properties in Admin mode.

Table 2-2: EPiServer Property Data Types.

Data Type Name	*Data Type Name in Admin mode*
Boolean	'Selected/not selected'
Category	'Category selection'
Date	'Time/Date'
FloatNumber	'Floating point number'
Form	'Form'
LongString	'Long string (>255)'
Number	'Integer'
PageReference	'Page'
PageType	'Page type'
String	'String (<= 255)'

Edit Mode

EPiServer Edit mode is constructed with ease of use as an important goal, as Edit mode will be used mainly by non-developers. Just after installation of EPiServer, accounts that are direct or indirect members of the group WebEditors have access to Edit mode. (Edit mode access permisionss are controlled by administrators.)

Entering Edit mode in EPiServer is very similar to entering Admin mode:

- Append '/edit' to your Web site's URL, e.g. 'http://widgets.info/edit', and navigate there, after logging on you're in Edit mode
- Navigate to your Web site and log on (using whatever means available, e.g. if you've included a log-on tool button, such as ), then right-click and select Edit mode from the shortcut menu

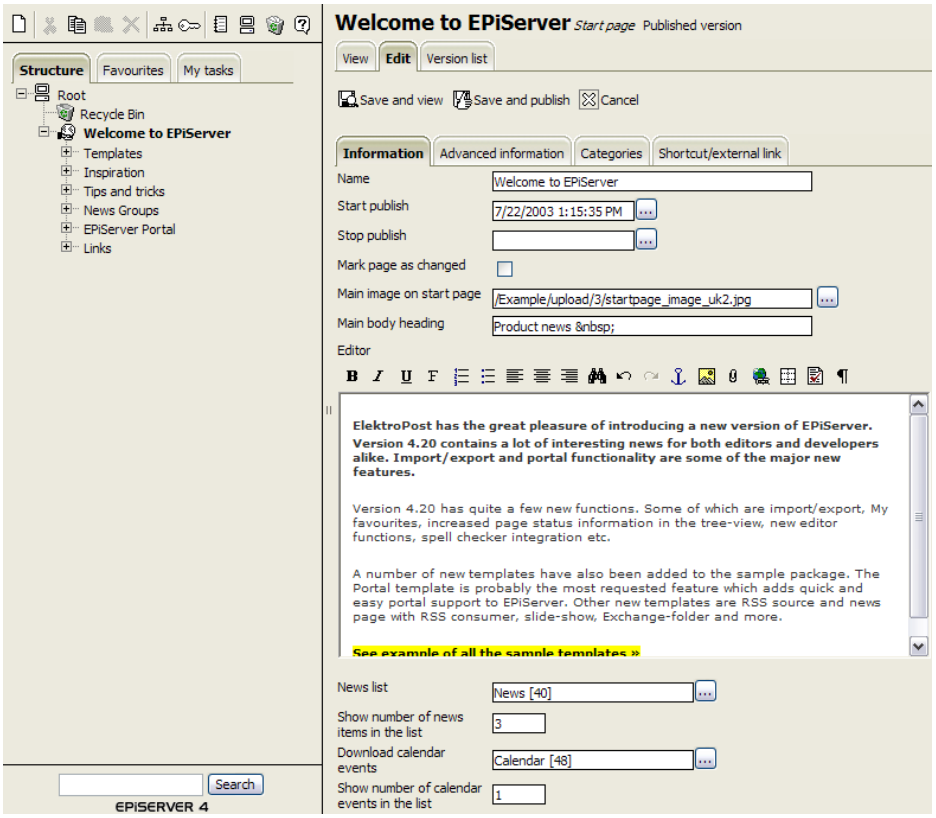


Figure 2-3: EPiServer Edit mode for the example Web site, start page being edited.

Figure 2-3 depicts EPiServer Edit mode. The screen dump was captured during editing of the Start page for the example Web site. Worth noting is the editing area with its tool-bar. This powerful editor is automatically invoked for every EPiServer Property of type LongString.

Naming Web Pages

The name of a Web Page is often used in links, HTML anchor tags, so never give the pages quirky names; instead use names such as 'EPiServer Now with Oracle Support'.

Tools

Once you're logged on, you'll have access to Admin mode and Edit mode from View mode on the shortcut, right-click, menu. More tools are available by right-clicking on a Web Page name, or another object in the left pane, and selecting from the menu. The two most important tools available on the shortcut menu are 'Create new' and 'Edit'. Editing may also be initiated simply by clicking on the name of the Web Page.

The tool-bar in EPiServer 4.2 Edit mode has eleven tool buttons.

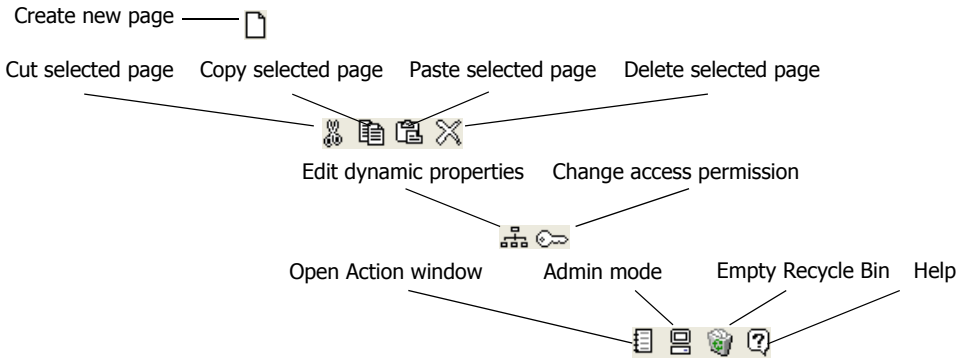


Figure 2-4: EPiServer 4.2 Edit mode tool-bar.

Page Tree

The Edit mode Page Tree is a hierarchical representation of the all-important Web Pages that the EPiServer Editors have created. The hierarchy of the Web Page Tree is also the hierarchy of the Web site as presented to viewers. It is also important when using some EPiServer controls, such as menu controls, since they use the Web Page Tree hierarchy directly as menu hierarchy.

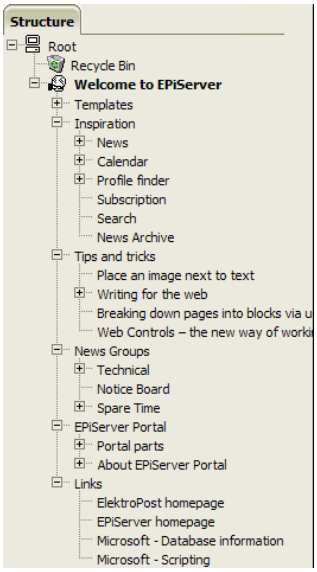


Figure 2-5: The Web Page Tree in EPiServer Edit mode (from the example Web site).

Sort Order in The Page Tree

The position of a Web Page in the Web Page Tree is determined by two things: the hierarchical level of the Web Page and what sort order is used. The sort order and possibly the sort index of the Page determine its position relative to other

Web Pages at the same hierarchical level. You have a number of different sort criteria to choose from, ranging from 'Last created first in list' to 'According to sort index'. As 'Last created first in list' is the default, it's most convenient if you create the Web Pages in reverse order to the desired sort order, i.e. start with the Web Page you wish to have last in the list.

EPiServer Handles Versions Automatically

Whenever a Web Page is altered and subsequently saved, EPiServer regards this as a new version of the Web Page. Versions are easy to handle but take up space in the database, so Editors should make sure to avoid keeping more versions than necessary.

Built-In Properties in Edit Mode

As mentioned earlier some of the built-in properties are handled in Edit mode.

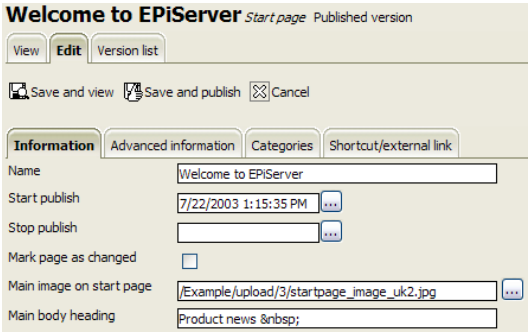


Figure 2-6: The Start page for the example Web site in Edit mode.

Figure 2-6 shows some of the Properties on the Start page for the example Web site. Among the properties there are four built-in ones:

- Name
- Start publish
- Stop publish
- Mark page as changed

The Page Chain, Tools and Techniques Used

When a Web site is being constructed, the developer pretty much single-handedly performs all the duties of developing Framework Definition Files, Page Template Files in Visual Studio .NET, creating Page Types and Properties in EPiServer Admin mode and Web Pages in EPiServer Edit mode. Later, when the Web site has reached the production phase the developer can concentrate on the development

chores and creating new as well as updating existing Page Types. A figure summarising the contributions of developers and editors might look like this:

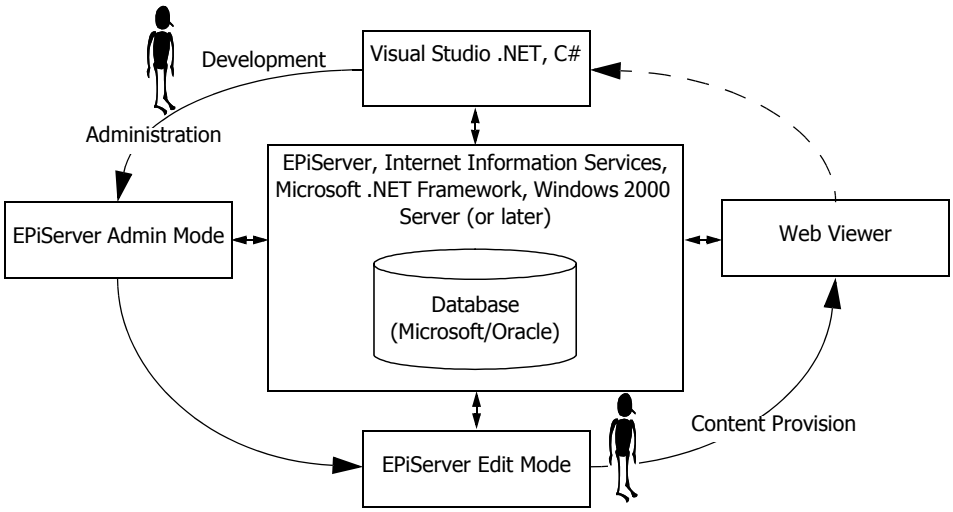


Figure 2-7: Techniques and tools used in the operation of an EPiServer site.

Figure 2-7 considerably underestimates the number of editors. It's not unusual for a single Web site to have more than a hundred editors.

Suppose we start with the Web viewer, after all, that's why we're here really, as all efforts centre around providing the best possible Web site from the customer's perspective. Viewers use their favourite Web browser, which doesn't have to be Internet Explorer since ASP.NET and EPiServer make it possible to be browser agnostic by using mostly pure HTML and some JavaScript on the client.

When the layout and other requisites of the site have been determined, the developers go to work producing Page Template Files using Visual Studio .NET. They then switch to EPiServer Admin mode and make Page Types out of the Page Template Files, the Properties of which are then exposed to the work of the good Editors. In many web site projects Page Types are created before Page Templates, as programming is secondary to structure.

3
Developing with EPiServer 4: Basic Insights

In this chapter these topics are presented in varying degrees of detail:

- EPiServer structure and object model
- EPiServer development

All examples in the book are written in Microsoft C#, but we've been careful to choose examples that can also be written in Visual Basic .NET, or indeed any other Microsoft .NET compatible language.

In order for you to be successful in studying this book and its examples we believe that a working knowledge of Microsoft .NET Framework, ASP.NET and C# is necessary. A lot of what you need to know about EPiServer is presented in the book.

EPiServer Structure and Object Model

One of the things that EPiServer customers like the most about the product is the fact that there's is a clear separation of content and structure. This makes it possible to, among other things, use content in several structure scenarios.

EPiServer implementations centre around four very important classes of both ASP.NET and non-ASP.NET objects:

- ASP.NET Web User Controls as Framework Definition Files
- ASP.NET Web Forms as Page Templates
- Page Types created from Page Templates
- Web Pages created from Page Types

As you might imagine, making Framework Definition Files, Page Templates and so on, is not the only way to use for Web User Controls and other ASP.NET features in EPiServer solution development.

Framework Definition Files

Framework Definition Files are ASP.NET Web User Controls (ascx files) used to create the visual, static, layout of a Web site. Framework Definition Files and Page Template Files form the EPiServer Content Framework, the preferred

method to separate layout from content. We'll be looking at and discussing the EPiServer Content Framework a lot in this chapter and also in the following chapters.

Regions Are Central

Developing with EPiServer centres around the concept of the region. An EPiServer region is two things: an area on a Web page and an EPiServer object bound to that area. The object is instantiated from the class Region in the name space EPiServer.WebControls. This area can be filled with text, HTML elements, Web User Controls or blanked out entirely. We will be discussing regions more below and certainly using them a lot in the chapters to follow.

Relationship Between Page Templates, Page Types and Web Pages

As we are for the most part dealing with text and image based information, it's only natural to be handling pages containing text and images. In EPiServer parlance, there are three different types of pages:

- Page Template, ASP.NET Web Form (aspx file), using a Framework Definition File and possibly other Web User Controls
- Page Type, EPiServer object (Page Template with EPiServer values, settings)
- Web Page, EPiServer object, instance of Page Type, containing text and other objects added by Editors

As your organisation starts using EPiServer to create your Web site, you'll probably notice that a lot of Web Pages are created from the same Page Type and that some Page Types may be created from a single Page Template.

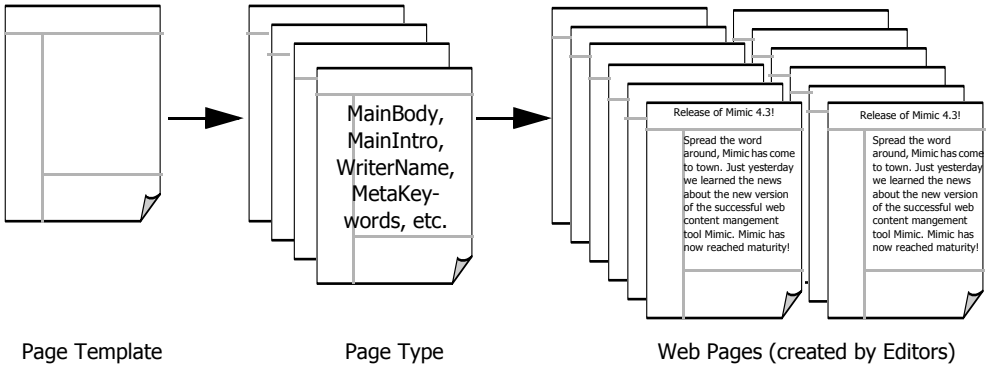


Figure 3-1: Relationship between Page Templates, Page Types and Web Pages.

The relationship between Page Templates, Page Types and Web Pages as depicted in figure 3-1 is certainly a very dynamic one. There are no laws stating that a Page Template can only be used to create four Page Type or that you must create at least eight Web Pages from one Page Type. In fact, it's most common that a

single Page Type is created from every Page Template. What is carved in stone though is that Web Pages can only be created from Page Types which, in turn, can only be based on Page Templates.

Page Templates and Framework Definition Files are created using Visual Studio .NET, Page Types and Web Pages are made in EPiServer Admin mode and Edit mode, respectively.

Framework Definition Files and Page Templates Are Created in Visual Studio .NET

As both Framework Definition Files and Page Templates are ASP.NET objects, they are created using Visual Studio .NET. When creating them, a Framework Definition File's class, although being a Web User Control, is set to inherit not from System.Web.UI.UserControl but from EPiServer.WebControls.ContentFramework. In the same manner, the main class in a Page Template File inherits from EPiServer.TemplatePage instead of System.Web.UI.Page.

That's two of the few things a developer has to consider when starting to develop for and with EPiServer. 'Using' the appropriate name spaces is also important. It turns out that the name spaces EPiServer and EPiServer.Core go a long way.

Page Types Are Created in EPiServer Admin Mode

Having created the Framework Definition Files and Page Templates for a Web site, the developer next opens the Web site and logs on to Admin mode. EPiServer Page Types can only be created in EPiServer's Admin mode, nowhere else. A simple, and yet correct, definition of an EPiServer Page Type is: an ASP.NET Web Form with added EPiServer Properties. Thus we understand the need to use EPiServer Admin mode to create Page Types as only EPiServer knows how to deal with EPiServer Properties and affix them to Page Types. A property is defined by its name and data type and the Page Type it is connected to. Properties are only created and connected to Page Types in Admin mode, but that doesn't necessarily mean that only Administrators handle Properties. On the contrary, more often than not Property handling will be the domain of developers.

EPiServer has a number of built-in data types for Properties and also a whole infrastructure in place to make it easy for Editors to work with the properties and their values as they create and maintain Web Pages from Page Types. Properties are hinted at in figure 3-1 by the names MainBody, MainIntro and so on found on the Page Type in mid-picture. EPiServer Properties will be discussed below.

It's also in Admin mode that the workings of the database are more obvious, as Page Types are stored in the database along with Property definitions for the different Page Types. In other words if you'd like to know what Page Types have been defined for an EPiServer Web site, the answer can always be found by querying the database. In fact, when we discuss EPiServer Admin mode in more detail below we'll show you a SQL query to retrieve Page Type definitions. Please

keep all direct database accesses out of any EPiServer solution you create, use the built-in functions for that and use direct access as a learning tool only.

Web Pages Are Created in EPiServer Edit Mode by EPiServer Editors

Until Web Page creation only developers were involved. Developers create Framework Definition Files, Page Templates and Page Types. Web Pages are created by designated EPiServer Editors, at least when the Web site is up and running. During development and maintenance of the Web site developers will have to get their hands dirty by creating Web Pages themselves. 'Starting' the EPiServer Edit mode is very similar to starting Admin mode, load the Web site start page, log on and then enter Edit mode (see page 18).

The bulk of the data in the database is the result of Editors' labour. The Web Pages they create are nothing more than records in database tables connecting properties and their values with Page Types. Moreover, EPiServer saves as many versions of Web Pages as desired.

Remember, only the values of Properties are handled in Edit mode; there is no provision for removing or deleting Properties in Edit mode. In fact, one aspect of Properties is that they can be marked 'Value must be entered' in Admin mode, thus requiring Editors to supply a value for these properties.

The Web Pages Live in a Web Page Tree

New Web Pages created by Editors are added to the collection of Web Pages called the 'Web Page Tree' or simply 'Page Tree' (sometimes even 'Edit Tree'). The Page Tree is hierarchical and just like any other tree in the computer world its root points upwards. The site's start page is a daughter of the Root Page.

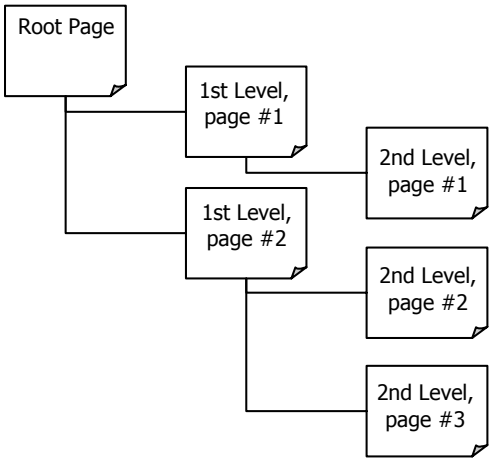


Figure 3-2: Web Page Tree.

As you see, the Web Page Tree has a lot in common with folder trees, if you consider the Root Page to be equivalent to the disk itself. The order in the Page Tree

is quite important as it controls, for example, the Web Page display order and menu order for some of the Web User Controls that ship with EPiServer, such as TopMenu.ascx and Menu.ascx.

As the Web Page Tree is of paramount interest to EPiServer solution developers, there are two properties that are used to point out two of the most important pages. The Properties EPiServer.Global.EPConfig.RootPage and EPiServer .Global.EPConfig.StartPage point to the root page of the Web Page Tree and the Start Page of the Web site, inside the Web Page Tree, respectively (more about these on page 163 under the heading *EPiServer.ApplicationConfiguration.RootPage and StartPage*).

Editors have full control of the Page Tree when in EPiServer Edit mode. Web Pages can be moved around, deleted and new Pages may be created anywhere in the Page Tree.

EPiServer 4 Architecture

As it is based on ASP.NET, it not surprising that the architecture for EPiServer 4 comprises many .NET elements.

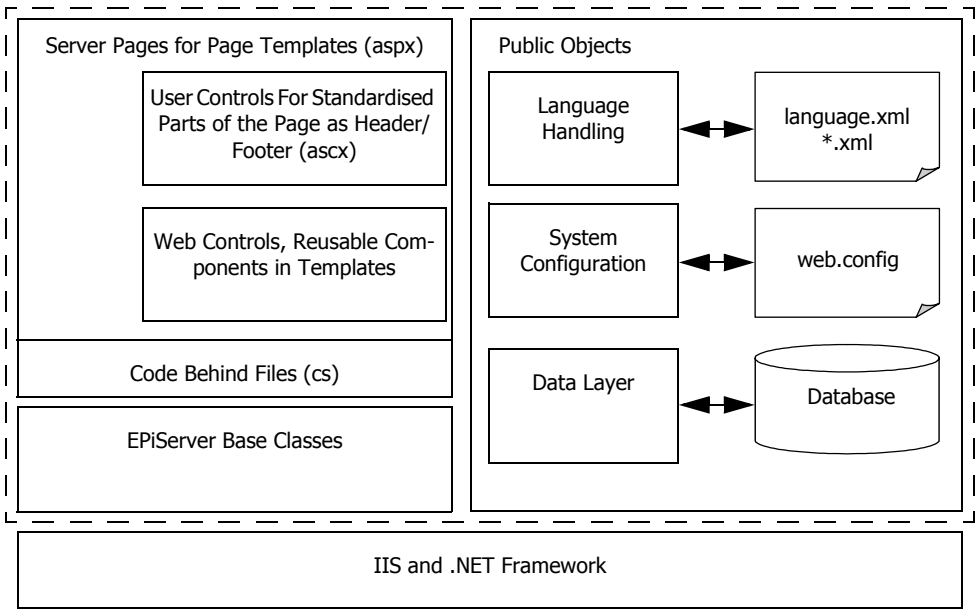


Figure 3-3: EPiServer 4 Architecture.

Figure 3-3 conveys the overall structure of EPiServer. EPiServer comprises all the elements within the dashed rectangle. Certainly when you install EPiServer and configure it to use the example Web site, all the functionality within the rectangle would come into play. It should be evident from figure 3-3 that the graphical part and code in EPiServer are separated, just as ASP.NET makes possible. What is not apparent in the figure is that page contents (head lines, references and

other text) are stored in the database. Thus we have the promise and possibility of complete separation of page layout, page content and EPiServer code; there doesn't have to be any overlap between the three. How to go about this will be clarified below.

A very important aspect of the database is that it allows EPiServer to handle several versions for Web Pages.

EPiServer Architecture and Folder Contents

It's always interesting to look at images depicting shoe-boxes, sorry, architecture. Even more interesting is to find out what the boxes and other pictures correspond to in real developer life.

Let's try to mentally connect the architecture to the folders and files created when installing EPiServer and letting it configure the example Web site. Here are the 'root' sub-folders and the files in the root folder:

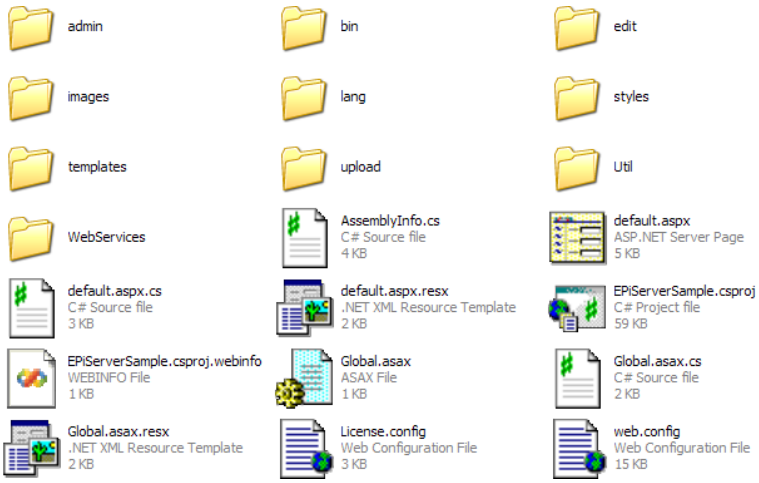


Figure 3-4: EPiServer folder tree and files in root folder.

To connect the folders and the architecture we need a table:

Table 3-1: EPiServer folder contents.

Folder Name	*Contents*
admin	ASP.NET Server Pages (aspx) and ASP.NET Web User Controls (ascx) for EPiServer administration mode.
bin	EPiServer executables and Dynamic Link Libraries, DLLs. The Microsoft .NET DLLs constitute EPiServer Base Classes.
edit	ASP.NET Server Pages and ASP.NET Web User Controls for EPiServer Edit mode.

Table 3-1: EPiServer folder contents.

Folder Name	*Contents*
images	Well, images (for the example web site).
lang	Language files for EPiServer, e.g. the English language files are languageEN.xml and templateLanguageEN.xml, respectively. All language files are formatted in standard XML format.
styles	In the case of EPiServer 4.2 there's only one file here: episerver.css. EPiServer.css is the Cascading Style Sheet document controlling all text on the Web site. For EPiServer 4.3, the number of style sheet files has increased to three; Editor.css, Structure.css and Units.css.
templates	An important folder. Here, and in its subfolders Framework and Units, we find all ASP.NET Server Pages and ASP.NET User Controls that are used for a particular Web site.
upload	Folder used by Editors when creating new content.
Util	Utilities for Administration and Edit modes, mostly ASP.NET Web User Controls. Also folders for style sheets, JavaScript files and images.
WebServices	Home to three XML Web Services files provided by ElektroPost: Authenticate.asmx, CacheService.asmx and DataFactoryService.asmx.

The Very Important templates Folder

To a developer, not all EPiServer folders are created equal. In particular you'd want to familiarise yourself with the contents of the templates folder and its subfolders, Frameworks and Units.

In the templates folder there are about forty ASP.NET Server Pages (aspx) and their corresponding C# code-behind files. In templates\Frameworks we find a handful of ASP.NET User Controls (ascx). Lastly, in the folder templates\Units there are again some forty ASP.NET User Controls (ascx).

All in all, the contents of these folders provide a rich source both of information and examples to be used when developing solutions with EPiServer 4.

EPiServer Content Framework: Combining Pages and Contents to Make a Web Site

How, exactly, is it that EPiServer 'does the trick' of putting together ASP.NET Server Pages, User Controls, Web Custom Controls and contents to make a Web site? To understand this, we must first look at the 'EPiServer Content Framework'.

The EPiServer Content Framework has been designed with these objectives in mind (among others):

- Separate presentation from content: keep visual layout and dynamic content in separate files
- Use ASP.NET standard elements and classes for every aspect of the Content Framework

A Visual Layout Divided

We now know that the Framework Definition File and Page Template File co-operate to make it possible to have a skeleton layout, which is static in nature, and dynamic content separated. Not surprisingly, the static layout elements go into the Framework Definition File and the Page Template File defines the dynamic content. This separation is used to define the overall visual layout in the Framework Definition File, ultimately resulting in a consistent look and feel for the Web site, and putting dynamic content in Page Template Files. This should result in the use of very few Framework Definition Files for a Web site and as many Page Template Files as there is need for. Framework Definition File are all well-formed HTML files; one of them is actually an XHTML 1.1 file.

Page Template Files are not obliged to use Framework Definition Files; quite a few of the ones shipped with EPiServer do not. Consistently, Page Definition Files that forego using Framework Definition Files are themselves well-formed HTML files.

Giving yourself a little click-around tour on the example Web site should be convincing. It's evident that some parts of the Web pages do not seem to change at all, whilst others reflect your actions. In particular there's a lot going on in the left-hand menu and in the centre of the page.

As shipped, EPiServer 4 contains four Framework Definition Files and about forty Page Template Files (including Default.aspx in the root folder).

Separate Presentation from Content: EPiServer Page Type Properties

Adding EPiServer Properties to an, almost plain vanilla, ASP.NET Web Form turns it into an EPiServer Page Type. These are managed by the EPiServer infrastructure and thus have access to all of its capabilities.

EPiServer Page Type Properties come in two flavours:

- Built-in Properties, always present, supplied by the EPiServer Run-Time System
- User-defined Properties, handled by developers in EPiServer Admin mode

The User-defined Properties, the user being a developer, can also be viewed as two groups.

- Static Properties, defined for and fixed to a single Page Type

- Dynamic Properties, defined as stand-alone, subsequently linked to a Web Page in the Web Page Tree. Inherited by all subordinate Web Pages

Properties Always Have a Data Type

No matter which kind of EPiServer Property we're looking at, they all share a common set of data types. There are currently ten Property data types:

Table 3-2: EPiServer Property Data Types.

Data Type	*Description*
Boolean	Traditional boolean type. PropertyBoolean
Category	List of categories. PropertyCategory
Date	Time and Date. PropertyDate
FloatNumber	Number with decimals, e.g. 3.14. PropertyFloatNumber
Form	HTML Form. PropertyForm
LongString	Used for text that can exceed 255 characters in length. Property-LongString
Number	Integers. PropertyNumber
PageReference	Link to Web Page. PropertyPageReference
PageType	Page Type, as in EPiServer Page Type defined in Admin mode. PropertyPageType
String	Short text, fewer than 256 characters in length. PropertyString

All data types inherit from EPiServer.Core.PropertyData which inherits from System.Object. Please note that Property Data Type Form, PropertyForm, is not a basic data type, as it inherits from PropertyNumber. It is included with the basic data types for convenience and for practical reasons.

In addition to the built-in Property types, it's always possible to create new types by inheriting from the existing ones. Using the example Web site as an illustration, it defines eleven custom Property types.

Built-in Properties

The built-in Properties for Page Types in EPiServer are very important; they hold such information as when to first publish a page, when to stop publishing a page, the position of the Web Page in the Page Tree and a lot more. Some of these Properties can be changed directly in Edit mode; others are changed indirectly, e.g. when a Web Page is moved in the Page Tree, and yet others are completely

under the control of the EPiServer Run-Time system. This table summarises some of the built-in properties:

Table 3-3: Built-in Properties for Page Types and thus Web Pages.

Property Name	*Description, Data Type*
PageDeleted	True if page is in the waste basket. PropertyBoolean
PageStartPublish	Time and date to start publishing the page. PropertyDate
PageWorkStatus	Editing status for page. PropertyNumber
PageParentLink	PageLink to parent page. PropertyPageReference
PageTypeID	Page type ID. PropertyPageType
PageName	Name if the Web Page. PropertyString
PageTargetFrame	The frame that this page should be displayed in. PropertyFrame

One of the built-in properties in table 3-3, PageTargetFrame, has a custom Data Type, PropertyFrame. It is defined in the class EPiServer.SpecializedProperties.PropertyFrame which inherits from PropertyNumber.

User-defined Properties: Static Properties

Static Properties are created, named, given a data type and other settings all in Admin mode and for a particular Page Type. The only thing left for Editors to do in Edit mode is to provide a value for the Property. Some common static properties are shown in table 3-4.

Table 3-4: Common static user-defined properties.

Property Name	*Description*
MainBody	The body text of the page. LongString
MainIntro	The introductory text, preamble, for the page. String
WriterName	The name of the person who authored the Web Page. String

User-defined Properties: Dynamic Properties

Dynamic Properties are treated in much the same way as as static Properties with the important exception that they are not tied to a particular page. Dynamic Properties allows you to do smart things. Take for example a dynamic property MetaKeywords of type String. Assign this to your start page and your Web site will have these keywords for global keywords. Now, let's say that you'd like to replace the global keywords on some pages. You then simply equip that Page Type with

its own static attribute MetaKeywords, then in Edit mode, with keywords pertinent to the Page Type.

Dealing with Properties in Code

Dealing with the value, contents, of Properties is easy both in HTML code and in C#, as is evident from these examples (the identifier Container is explained on page 170):

Example 3-1: Displaying the value of built-in Property PageName in HTML code.

```
<%# Container.CurrentPage.PageName %> // Or...
<%# CurrentPage.Property[ "PageName" ] %>
```

Example 3-2: Displaying the value of user-defined Property MainIntro in HTML code.

```
<%# Container.CurrentPage[ "MainIntro" ] %>
```

Example 3-3: Displaying the value of built-in Property PageName in C#.

```
Response.Write( CurrentPage.PageName ); // Or...
Response.Write( CurrentPage.Property[ "PageName" ].Value.ToString() );
```

Example 3-4: Displaying the value of user-defined Property MainIntro in C#.

```
Response.Write( CurrentPage.Property[ "MainIntro" ].Value.ToString() );
```

Whether or not a Property is dynamic is answered by the field IsDynamicProperty:

Example 3-5: Using IsDynamicProperty for user-defined Property MainIntro in C#.

```
if ( CurrentPage.Property[ "MainIntro" ].IsDynamicProperty ) {
    Response.Write( "Dynamic Property" );
}
```

Boolean Properties

Be aware that the value returned from a boolean Property is either 'true' or 'null'.

Interaction between Web Pages and the Database

Separating presentation from content is achieved in EPiServer by having all presentation elements in ASP.NET Server Pages and all content in database tables. Simple, don't you think? Actually, the architecture behind this is quite straightforward.

It all revolves around properties as defined by EPiServer. These properties appear in several shapes and forms. Properties can be added to a Framework Definition File or a Page Template File by User Controls or added by administrators. In the latter case, administrators add properties selecting from one of the twenty different kinds of properties defined by EPiServer. Most content will be preserved in properties of either the String type or the Long string type. (String is for

text having a total length of 255 characters, or less, Long string for all longer texts.)

When Editors go about their jobs, it is the contents of properties that they deal with; they are never allowed to affect which properties are available for certain Page Types.

The physical separation of content and presentation structure is maintained by keeping all Web Forms and their code at the Web server and content in the database. In particular the table tblProperty contains all properties for a particular page.

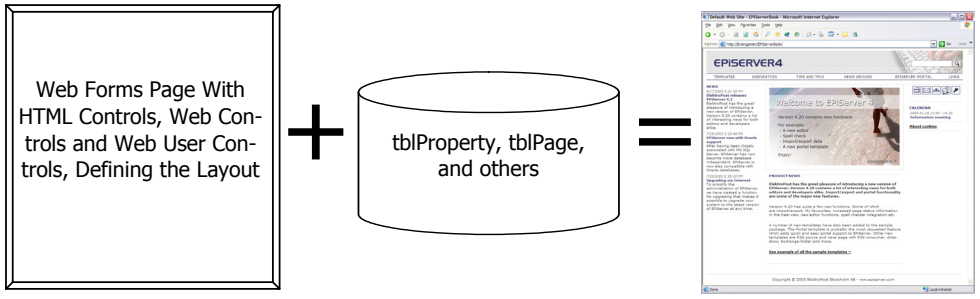


Figure 3-5: Cooperation between structure and content.

If figure 3-5 the object on the left comprises both the Framework Definition File and the Page Template File.

Using Web User Controls in Framework Definition Files and Page Template Files

ASP.NET Web User Controls can be described as components with a visual HTML component and a code component (in C# or any other Microsoft .NET compatible language). In EPiServer development these are typically used to encapsulate visual elements and code into one package which can be re-used in many places.

At the core of every EPiServer-based Web site we find quite simple structures, at least from an ASP.NET perspective. Typically there are a few master Framework Definition Files, presumably a slightly greater number of Page Template Files and, to complete the story, some Web User Controls.

The relation between a Framework Definition File and Page Template Files is depicted in figure 3-6.

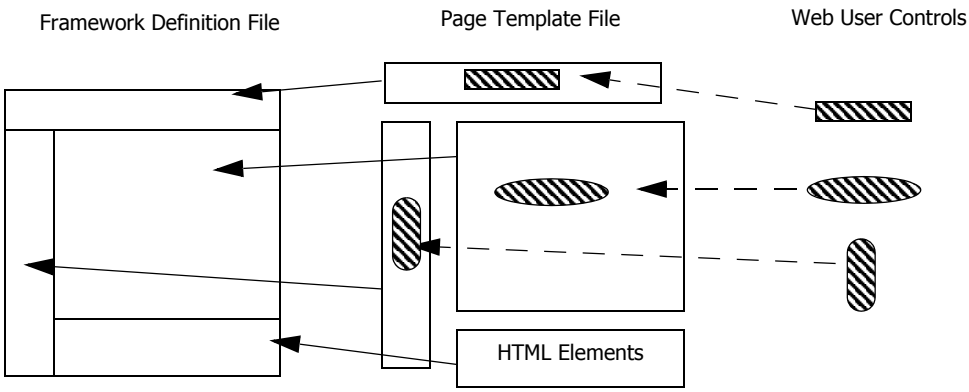


Figure 3-6: Framework Definition Files 'declare' regions that are used by Page Template Files.

Now, this relationship is by no means controlled by the Framework Definition File; actually it's the other way around: Page Template Files can choose to adhere to any Framework Definition File they so please (i.e. the developer creating the Page Template File decides).

Framework Definition Files are charged with the task of visual design, they are often constructed as standard HTML tables (using the HTML elements table, tr and td), but can also be implemented using HTML Div areas. In EPiServer parlance, we say that the Framework Definition Files defines visual regions, the contents of which are determined by the Page Template Files. When a Page Template File specifies the use of a certain Framework Definition File it is expected to also specify which regions in the Framework Definition file it will utilise. Figure 3-7 attempts to show the relationship between the visual regions of a Framework Definition File, Page Templates using the regions on this particular Framework Definition File and the various objects that can be used on the final Web site. Please note that one of the regions on the Framework Definition File, the bottom one, will be filled with 'pure' HTML; it will not host any Web User Control. This is typical of EPiServer Solution Development; for some jobs Web User Controls will be needed, others require only the use of HTML text.

Purpose of EPiServer Framework Definition Files

The mission for Framework Definition Files is to define the visual layout and visual 'regions'. These regions are later referenced in the Page Template Files and used at their discretion (or not used at all). As with most current Web sites, the layout is made up up HTML tables and is not frame-based. Using HTML tables is by no means a prerequisite, you can also use HTML div elements for the same purpose. Below we'll take a look at two ways to create a Framework Definition File with the same visual layout as in figure 3-6. First we'll use HTML tables and then HTML div elements.

Take a look at this figure:

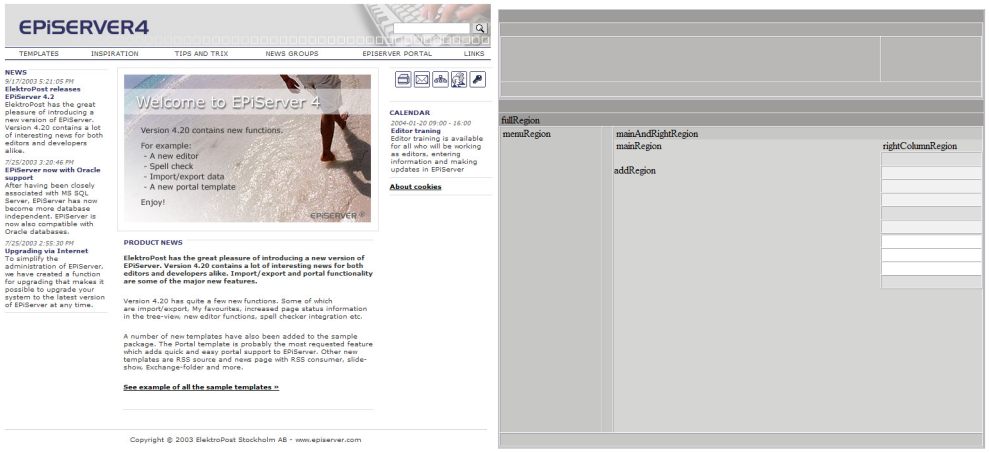


Figure 3-7: Start page of the example Web site to the left and the tables from its Framework Definition File to the right.

Figure 3-7 show the start page of the example Web site in the left half and the Framework Definition File, DefaultFramework.ascx from the example Web site, to the right in the figure. To make the HTML tables in DefaultFramework more visible, all borders attributes were changed from 0 to 1 (border="1"), bgcolor was added in varying shades of grey, and all table columns were given a content of a least one non-breaking space ( ) and the regions' names were included. The impact of these non-breaking spaces and region names is small, but they do make the rows somewhat higher. (The file was also copied in its entirety and the new file given an extension of html and subsequently displayed in our favourite Web browser.)

As you can see in figure 3-7 there is a close correspondence between the lay-out in the start page of the example Web site and the layout of the Framework Definition File. (Please make some allowance for the non-breaking spaces that were added.)

EPiServer Regions

Regions are visual rectangles defined in Framework Definition Files and used in Page Template Files. The purpose of Regions is two-fold: to allow the creator of Framework Definition Files to supply default visual HTML elements and User Controls and to allow the maker of Page Template Files to benefit from these default elements or replace them with other as she pleases.

An EPiServer Region can span one table column (<td>), one row (<tr>) or even a whole table or more. Regions can be nested to any level desired. They are implemented as ASP.NET Custom Controls.

Regions are very easy to define in Framework Definition Files. First you state your intention to use the name space EPiServer.WebControls and then define Regions just like HTML/XML tags.

Example 3-6: Defining an EPiServer Region in a Framework Definition File.

```
<%@ Register TagPrefix="EPiServer" Namespace="EPiServer.WebControls" Assembly="EPiServer" %>
...
<EPiServer:Region id="topRegion" runat="server">
...
</EPiServer:Region>
```

EPiServer Regions Are Used by Means of the Content Class

Whenever a Page Template File intends to use a region in a Framework Definition File it does so by using another class from EPiServer.WebControls by the name of Content. It looks like this:

Example 3-7: Regions are used by a Content class object.

```
<%@ Register TagPrefix="EPiServer" Namespace="EPiServer.WebControls" Assembly="EPiServer" %>
...
<EPiServer:Content ID="TopMenu" Region="topRegion" runat="server">
...
</EPiServer:Content>
```

Both the Region and Content Classes Are Part of EPiServer.WebControls

Both the Region and the Content classes are part of the important name space EPiServer.WebControls. Classes in this name space are used in almost every Framework Definition File and Page Template File. We will certainly be using classes from this name space in every chapter of this book.

Page Template Files

The ultimate goal of Page Template Files is to provide dynamic content for a Web site. Most commonly they do so by harbouring Properties for Editors to handle. Page Template Files can use Framework Definition Files but, as mentioned above, this is not mandatory. Remember, as we're striving to produce HTML, a Page Template File, either by itself or in combination with a Framework Definition File, must constitute well-formed HTML.

Using HTML Tables for Layout

For those familiar with ASP.NET, neither Framework Definition Files nor Page Template Files hold any real secrets. Of course, since we're working in a team with EPiServer, a lot of EPiServer pre-built functionality is used in both types of files.

A Very Simple Framework Definition File Using HTML Tables for Layout

A Framework Definition File implementing the visual layout of figure 3-6 could look like this:

Example 3-8: Simple Framework Definition File using HTML tables.

```
<%@ Register TagPrefix="EPiServer" Namespace="EPiServer.WebControls" Assembly="EPiServer" %>
<%@ Control Language="c#" AutoEventWireup="false" Codebehind="TableFramework.ascx.cs"
Inherits="development.Frameworks.TableFramework" TargetSchema="http://schemas.microsoft.com/
intellisense/ie5" %>
<html>
<head>
</head>
<body class="NormalPage" xmlns:EPiServer="http://schemas.episerver.com/webcontrols">
    <form runat="server" id="Default">
        <table width="750" border="1" cellspacing="0" cellpadding="0" align="center">
            <tr height="80" >
                <td colspan="2" align="center">
                    <EPiServer:Region id="topRegion" runat="server">
                    </EPiServer:Region>
                </td>
            </tr>
            <tr height="520">
                <td rowspan="2" width="20%">
                    <EPiServer:Region id="leftColumnRegion" runat="server">
                    </EPiServer:Region>
                </td>
                <td>
                    <EPiServer:Region id="centreRegion" runat="server">
                    </EPiServer:Region>
                </td>
            </tr>
            <tr height="100">
                <td>
                    <EPiServer:Region id="footerRegion" runat="server">
                    </EPiServer:Region>
                </td>
            </tr>
        </table>
    </form>
</body>
</html>
```

As you can see, there are no Web User Controls in this Framework Definition File, but the visual layout is just as we want it. A single HTML table is used comprising only three rows and a total of four table cells.

Since all the regions are defined inside table cells, and thus inside an HTML table, whatever we choose to place in those regions will also be inside table cells.

A Very Simple Page Template Using the Very Simple Framework Definition File

Remember, when a Page Template File uses a Framework Definition File, it can replace regions or leave them alone as the Page Template Developer pleases. A simple Page Template File using the Framework Definition File in listing 3-8 could look like this:

Example 3-9: Page Template File using the HTML Table based Simple Framework Definition File.

```
<%@ Register TagPrefix="development" TagName="TableFramework"Src="~/templates/Frameworks/
TableFramework.ascx"%>
<%@ Register TagPrefix="EPiServer" Namespace="EPiServer.WebControls" Assembly="EPiServer" %>
<%@ Page language="c#" Codebehind="default.aspx.cs" AutoEventWireup="false"
Inherits="development.Default" %>
<development:TableFramework ID="TableFramework" runat="server">
    <EPiServer:Content region="leftColumnRegion" id="Menu" runat="server">
        Stuff in the left column.
    </EPiServer:Content>
    <EPiServer:Content region="centreRegion" id="CentralMessage" runat="server">
        Important message that goes in the centre region.
    </EPiServer:Content>
    <EPiServer:Content region="footerRegion" id="Footer" runat="server">
        <table cellpadding="10" cellspacing="0" width="100%" border="0">
            <tr>
                <td align="center">Last changed 15 April 2004<br/>&copy; 2004 ElektroPost AB<br/>
                    <a href="mailto:webmaster@ep.se">Contact Web Master</a>.
                </td>
            </tr>
        </table>
    </EPiServer:Content>
</development:TableFramework>
```

This Page Template File uses three of the four regions in the Framework Definition File: leftColumnRegion, centreRegion and footerRegion. Although simple, this Page Template shows the effortlessness of using EPiServer Regions.

Using Visual Studio .NET to test run the result looks like this:

Figure 3-8: Effect of using HTML Tables based Framework with simple Page Template.

Accessibility Considerations Starting with EPiServer 4.3

In this version, we provide a rewritten set of page templates that has a higher accessibility according to WAI (Web Accessibility Initiative). The basic framework is now based on HTML Div tags rather than Table tags, as previously used.

Support for formatting with H tags (H1, H2, etc.) in the editor has been added. Simply set your style rules as H1 or H1.Heading1 depending on your needs.

Using HTML Div Elements for Layout

As stated above, HTML tables are not the only way to create visual layout. In fact it's not that much more involved to create a Framework Definition File based on HTML Div elements instead:

Example 3-10: Simple Framework Definition file based on HTML Div elements.

```
<%@ Control Language="c#" AutoEventWireup="false" Codebehind="DivFramework.ascx.cs"
Inherits="development.Frameworks.DivFramework" TargetSchema="http://schemas.microsoft.com/
intellisense/ie5" %>
<%@ Register TagPrefix="EPiServer" Namespace="EPiServer.WebControls" Assembly="EPiServer" %>
<html>
<head>
    <style>
        #ContainerArea {
            position:    absolute;
            left:        50%;
```

```
            width:      750px;
            height:     700px;
            margin-left: -375px;/* Half of width. */
            margin-top: 10px;
        }
        #HeaderArea {
            border:     solid 1px #747170;
            text-align: left;
            width:      100%;
            height:     80px;
        }
        #LeftMenuArea {
            border:     solid 1px #747170;
            width:      20%;
            height:     520px;/* 700 - Header - Footer. */
            float:      left;
        }
        #ContentArea {
            border:     solid 1px #747170;
            width:      80%;
            height:     520px;/* See LeftMenu. */
            float:      right;
        }
        #FooterArea {
            border:     solid 1px #747170;
            text-align: center;
            float:      right;
            width:      80%;
            height:     100px;
        }
    </style>
</head>
    <body class="NormalPage" xmlns:EPiServer="http://schemas.episerver.com/webcontrols">
        <form runat="server" id="Div">
            <div id="ContainerArea">
                <div id="HeaderArea">
                    <EPiServer:Region id="topRegion" runat="server">
                    </EPiServer:Region>
                </div>
                <div id="LeftMenuArea">
                    <EPiServer:Region ID="leftColumnRegion" runat="server">
                        EPiServer:Region ID="leftColumn"
                    </EPiServer:Region>
```

```
                </div>
                <div id="ContentArea">
                    <EPiServer:Region id="centreRegion" runat="server">
                        EPiServer:Region id="centreRegion"
                    </EPiServer:Region>
                </div>
                <div id="FooterArea">
                    <EPiServer:Region id="footerRegion" runat="server">
                        EPiServer:Region id="footerRegion"
                    </EPiServer:Region>
                </div>
            </div>
        </form>
    </body>
</html>
```

Depending on your personal persuasion, the Framework Definition File in example 3-10 is either table-free or table-less. Irrespective of which it uses HTML Div elements to create a visual layout. There are five HTML Div areas used, as opposed to four table cells in the table based Framework. Key to using HTML Div elements are the positioning capabilities found in Cascading Style Sheets, CSS, version 2. The reason for the fifth Div element is that it acts as an invisible enclosure for the four main Div areas and allows us to position all Div areas as one group using style sheet elements.

Changing three lines in the Page Template File listed in example 3-9 lets us use the Div based Framework Definition File.

Example 3-11: Original Framework usage lines in Page Template File.

```
<%@ Register TagPrefix="development" TagName="TableFramework"Src="~/templates/Frameworks/
TableFramework.ascx"%>
...
<development:TableFramework ID="TableFramework" runat="server">
...
</development:TableFramework>
```

Example 3-12: Changed Framework usage lines in Page Template File.

```
<%@ Register TagPrefix="development" TagName="DivFramework "Src="~/templates/Frameworks/
DivFramework.ascx"%>
...
<development:DivFramework ID="TableFramework" runat="server">
...
</development:DivFramework>
```

Running the changed Page Template File in Visual Studio .NET produces this output:

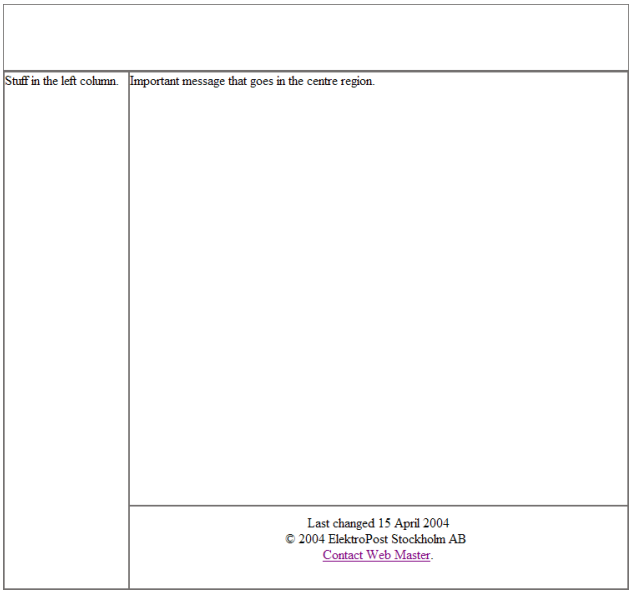


Figure 3-9: Effect of using HTML Div elements based Framework with simple Page Template.

Figure 3-9 is very similar to figure 3-8, apart from the vertical alignment of text in some regions.

Inner Make-Up of a Page Template File which Doesn't Use a Framework Definition File

For the sake of completeness, we should also take a look at a Page Template File that doesn't benefit from the use of a Framework Definition File. There are several to choose from in the templates folder; we've settled on Login.aspx which is used on the example Web site.

Example 3-13: Parts of the Page Template File Login.aspx

```
<%@ Register TagPrefix="EPiServer" Namespace="EPiServer.WebControls" Assembly="EPiServer" %>
<%@ Page language="c#" Codebehind="Login.aspx.cs" AutoEventWireup="false"
    Inherits="development.Templates.Login" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<html>
    <head>
        <title>Login</title>
        <meta name="GENERATOR" Content="Microsoft Visual Studio 7.0">
        <meta name="CODE_LANGUAGE" Content="C#">
        <meta name="vs_defaultClientScript" content="JavaScript">
        <meta name="vs_targetSchema" content="http://schemas.microsoft.com/intellisense/ie5">
        <link rel="stylesheet" type="text/css" href="<%= EPiServer.Global.EPConfig.RootDir %>
```

```
            util/styles/login.css">
    </head>
    <body>
        <table width="100%" height="100%">
            <tr>
                <td valign="middle" align="center">
                    <form id="Login" method="post" runat="server">
...
                                        <td class="login" valign=middle align=left>
                                            <EPiServer:Translate Text="/login/username"
                                                runat="server" ID="UsernameLabel" />
                                        </td>
...
                    </form>
                </td>
            </tr>
        </table>
    </body>
</html>
```

There are at least two interesting things to note about this Page Template File. It uses EPiServer.Global.EPConfig.RootDir to retrieve the root folder for the Web site and it uses the class EPiServer.WebControls.Translate to make sure that any text presented to the viewer is in the preferred language. Both represent much recommended ways of doing things. You should always strive to make your solutions language agnostic.

EPiServer Content Framework Is Not Unlike ASP.NET 2.0 Master Pages and Content Pages

One important addition to ASP.NET will be Master Pages and Content Pages. You can find quite a lot of information about ASP.NET 2.0, project name Whidbey, on the Microsoft Developer Network Web site (http://msdn.microsoft.com). Here's what one article found at that site had to say about Master Pages and Content Pages:

'You'll create a Master Page for the site layout and design and create Content Pages for each content resource, somehow connecting them to the Master Page. As users navigate to *.aspx resources in your site ASP.NET will serve up the requested page displaying it within the layout of its associated master.'

This sounds a lot like and will work a lot like EPiServer Content Framework and we hope that this gives you an extra incentive to start using Framework Definition Files and Page Template Files.

EPiServer Name Spaces

Those of you familiar with EPiServer 3 will appreciate that EPiServer 4 preserves a lot of the vocabulary, in spite of all other differences. We're hoping that this will

make the transition easier for both application developers and page template creators.

A .NET Framework name space is simply a way to group classes, delegates, structures, constants, enumerations and other such information. It is up to the developers to make this grouping as logical and consistent as possible.

These are some of the name spaces found in EPiServer 4.2:

- EPiServer
- EPiServer.Core
- EPiServer.Core.Html
- EPiServer.Filters
- EPiServer.Personalization
- EPiServer.PlugIn
- EPiServer.Security
- EPiServer.SpecializedProperties
- EPiServer.WebControls

We will take a closer look at name spaces in later chapters, but suffice to say that it is unlikely you'll be using all of them in a single project, and perhaps some will never be touched by a developer's hand.

EPiServer

The EPiServer root name space contains some of the most important and central classes, such as DataFactory, PageBase, TemplatePage, ApplicationConfiguration and the Global class.

EPiServer.Core

The EPiServer.Core name space contains classes for handling page data, properties and templates. This name space also contains the important PageReference structure which functions as a pointer to other pages in the system.

EPiServer.Core.Html

The EPiServer.Core.Html name space contains classes for text searching, Index Server support and content parsing. This name space includes the HtmlParser class that can parse HTML and return valid XHTML as a string or as an XmlDocument (System.Xml.XmlDataDocument in the .NET Framework).

EPiServer.Filters

Filters are a new feature in EPiServer 4, they are used to control visual presentation of listings.

EPiServer.Personalization

The EPiServer.Personalization name space supplies classes for personalization and subscription. The PersonalizedData class can be used to retrieve and save information about a user, as well as to store global personalized values or page specific values.

EPiServer.PlugIn

The EPiServer.PlugIn name space is used to extend the Edit and Admin interfaces with custom functionality.

EPiServer.Security

The classes, enumerations and delegates in the EPiServer.Security name space allow you to identify the current user, make your own authentication handlers and make a highly secure site. If you want to create your own authentication scheme, see the AuthenticationProvider class.

EPiServer.SpecializedProperties

The EPiServer.SpecializedProperties name space contains property classes for various special edit mode requirements like the PropertyWeekDay class which renders a list of checkboxes, for each weekday, for the user to select.

EPiServer.WebControls

EPiServer.WebControls is home to all the ASP.NET Web Custom Control classes that ship with EPiServer. It contains 60+ classes, from Calendar to XmlNameValidator, for rendering page information, tree structures, page lists, etc. to a browser. The controls have rich data binding support, the lists can be filtered and most controls are templated.

The classes in EPiServer.WebControls are ASP.NET Web Custom Controls, which are like ASP.NET Web User Controls but they have only code representation, there is no corresponding ASCX file. Do not be confused by the four User Controls in templates\Units which have the same name as some Custom Controls: Calendar, ChangedPages, PortalRegion and SiteMap.

Permissions and User Identities Are Handled By EPiServer (and You)

Usually when you set up an IIS Web site, you take great care in getting all folder and file permissions set up the way you want them. When working with EPiServer, handling permissions is a lot easier as handling them is part of EPiServer Ad-

min mode, permissions for folders and files are set once for the root folder of the application and then inherited through the entire sub-tree.

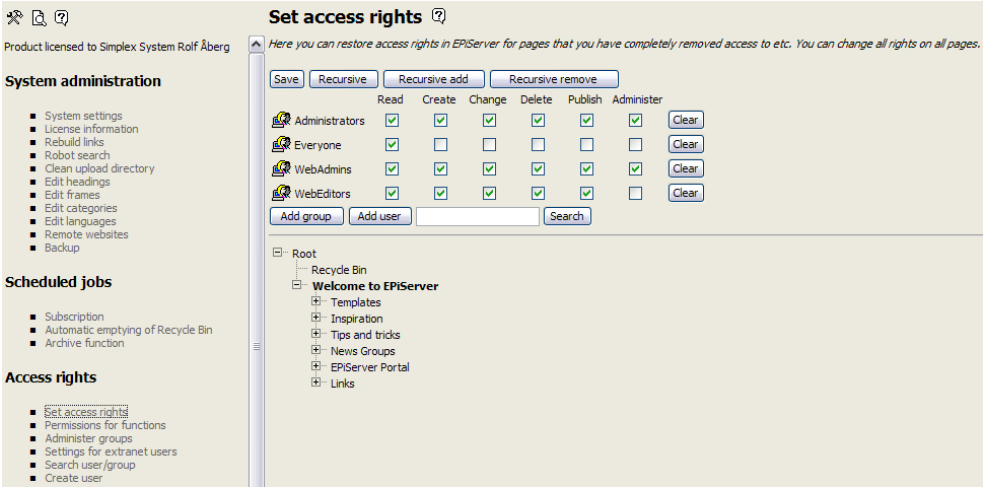


Figure 3-10: Handling access permissions in EPiServer Admin mode.

In figure 3-10, you can see that when handling access permissions you have immediate access to the EPiServer Web Page tree and group accounts (both EPiServer groups and, e.g. Windows groups). (EPiServer creates two groups at installation: WebAdmins and WebEditors.)

Should your Web site be one that's serving the Internet, you probably won't have much need to change the access permissions from their default settings. Access to the Web site for the Windows group Everyone (on the Internet) is set to Read. It's only when we're discussing more interesting scenarios, from an access permissions perspective, that account and group handling along with access permissions become important.

User Identities and Permissions Are Easy to Handle In Code

With groups set up and access permissions in place, you can handle both in code for users who log on. EPiServer.PageBase is the mother class for all Page Templates. Being itself a descendant of System.Web.UI.Page, there's a public property User which holds information about the user who is currently logged on. PageBase.User is simply a pointer to an interface, IPrincipal, which is implemented by GenericPrincipal and WindowsPrincipal. More important is the property PageBase.CurrentUser which is an instance of EPiServer.Security.UnifiedPrincipal. UnifiedPrincipal gives you access to a lot more information than IPrincipal, e.g. The Security Identifier, SID, for the account that was used to log on and IP settings along with per-user cache settings.

Interface IPrincipal declares the interesting property Identity and method IsInRole. From Identity, the name of the user account is retrieved by calling Identity.Name; group membership is tested with the boolean function IsInRole.

For testing permissions, there is a method called CheckAccess. Also, if you want to deny the current user access for some reason you can call the method AccessDenied.

Example 3-14: Using PageBase.AccessDenied.

```
if ( CurrentUser.UserData == null ) {
    AccessDenied();
}
```

The code in example 3-14 denies further processing if the current user is the Anonymous user, i.e. not logged in with an acknowledged user account.

EPiServer System Settings in the web.config File

Just like other ASP.NET applications, EPiServer stores settings in the web.config file in the root folder for the solution. Some of the interesting settings are found in system.web section and deal with such matters as authentication method, debug and trace settings.

In the section appSettings we find more specific EPiServer settings, some of which are described in table 3-5.

Table 3-5: EPiServer settings in web.config (not comprehensive).

Setting Name	*Purpose*
EPsConnection	Database connection string.
EPnStartPage	ID of the Web Page which serves as the start page for the site.
EPsRootDir	Root folder for the Web site, when installed to a virtual folder.
EPsHostUrl	Uniform Resource Locator, URL, for the host.
EPnUserCacheTimeout	Timeout limit, in minutes, for user cache.
EPsLanguage	EPiServer language settings, current versions allow one of 'DK', 'EN', 'FI', 'NO' or 'SV'; meaning Danish, English, Finnish, Norwegian or Swedish.
EPnLocale	Locale ID, following the Windows setting: 1030 (0x0406) for Denmark; 2057 (0x0809) for United Kingdom; 1035 (0x040B) or 2077 (0x081D) for Finland; 1044 (0x0414) or 2068 (0x0814) for Norway and 1053 (0x041D) for Sweden. Please note that the locale information is loaded with the Web page.

Table 3-5: EPiServer settings in web.config (not comprehensive).

Setting Name	*Purpose*
EPsSiteName	Web site name.

Many of the system settings are under administrator control in the EPiServer Admin mode as shown in this picture:

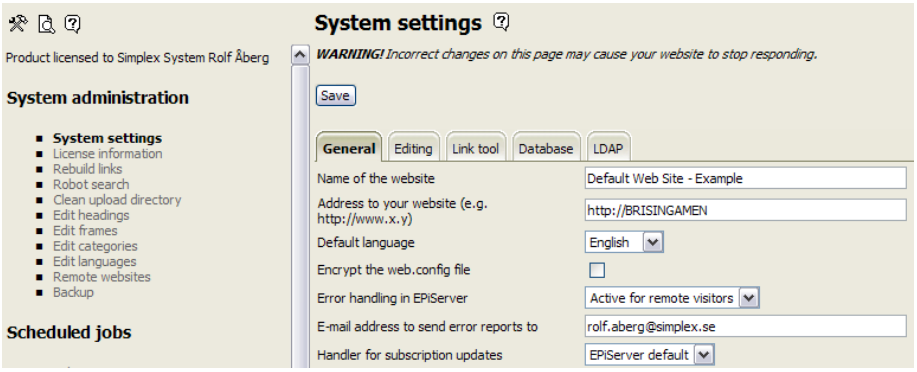


Figure 3-11: EPiServer System Settings (from the Example Web site).

Accessing System Settings from Code

There is more than one way to access EPiServer system settings from code. Your code will always have access to EPiServer.Global which is instantiated by Global.asax. Its property EPConfig allows access to the system settings. EPConfig is of the type EPiServer.ApplicationConfiguration.

Web Pages created from EPiServer Page Template Files are descendants of the class EPiServer.PageBase. Among the attributes for PageBase is Configuration, which is of the same type as EPConfig.

In other contexts, you might want to utilise the Page property (common to all Web pages descending from System.Web.UI.Control), which is a reference to the page that contains the server control (ASP.NET Web Controls are server controls). (EPiServer.PageBase inherits System.Web.UI.Page which is a descendant of System.Web.UI.Control.)

Several of the system settings are also implemented with their own identifiers, e.g. RootDir and StartPage. In other words, Configuration.RootDir is equivalent to Configuration["EPsRootDir"] and Configuration.StartPage is akin to Configuration["StartPage"] (the former is of type PageReference, the latter of type int).

Some examples might help to clarify things:

Example 3-15: Using EPConfig in the code-behind file for a Web User Control (ascx.cs file).

```
private void Page_Load( object sender, System.EventArgs e ) {
    IPrincipal user = Context.User;
    if ( user != null && user.Identity.IsAuthenticated ) {
```

```
        Logout.Visible= ( EPiServer.Global.EPConfig.Authentication ==
                System.Web.Configuration.AuthenticationMode.Forms );
    } else {
        DisplayLoginLink();
    }
}
```

Example 3-16: Using EPConfig in the code-behind file for a Web Form (aspx.cs file).

```
XmlNode title= doc.CreateElement( "title" );
title.InnerText= EPiServer.Global.EPConfig.SiteName;
channel.AppendChild( title );
XmlNode link= doc.CreateElement( "link" );
link.InnerText= EPiServer.Global.EPConfig.HostUrl + EPiServer.Global.EPConfig.RootDir;
```

Example 3-17: Using EPConfig in HTML for a Web User Control (ascx file).

```
<IFrame src="<%= EPiServer.Global.EPConfig.RootDir %>Edit/NewPage.aspx<%= "?mode=" +
Request["mode"] + "&id=" + Request["id"] %>" Width="100%" Height="100%" Name="EditPanel"">
</IFrame>
```

Example 3-18: Using EPConfig in the code-behind file of a Web User Control (ascx.cs file).

```
private string GetSystemLanguage() {
    if ( IsValue( "PageLanguageID" ) ) {
        return (string) CurrentPage[ "PageLanguageID" ];
    } else {
        if ( EPiServer.Global.EPConfig[ "EPsLanguage" ] != null ) {
            return (string) EPiServer.Global.EPConfig[ "EPsLanguage" ];
        }
    }
    return "EN";
}
```

Example 3-19: Using Configuration in the code-behind file of a Web User Control.

```
private void Page_Load( object sender, System.EventArgs e ) {
    if ( Configuration.Authentication == System.Web.Configuration.AuthenticationMode.Forms ) {
        CookieLogin.Text = "/cookie/usageloginform";
    } else {
        CookieLogin.Text = "/cookie/usageloginwindows";
    }
}
```

Example 3-20: Using Configuration in the code-behind file of a Web User Control.

```
protected string GetTitleString()
    try {
        if ( this.Title.Length > 0 ) {
            return this.Title + this.TitleSeparator + Configuration.SiteName;
        } else {
            return CurrentPage.PageName + this.TitleSeparator + Configuration.SiteName;
    }
    catch {
        return String.Empty;
    }
}
```

Example 3-21: Using Configuration in JavaScript in an ASP.NET Web Form (for EPiServer Edit mode).

```
window.StartPage = <%= Configuration.StartPage.ID %>;
window.RootPage = <%= Configuration.RootPage.ID %>;
window.WastebasketPage = <%= Configuration.Wastebasket.ID %>;
```

Example 3-22: Using Configuration in an ASP.NET Web Form (for EPiServer Edit mode).

```
<EPiServer:ExplorerTree PublishedStatus="Ignore" ShowIcons="False" EnableVisibleInMenu="False"
    PageLink=<%# Configuration.RootPage %> ShowRootPage="True"
    ClickScript= "SetLocalValues( '{PageLink}', '{PageName}' );" id="Explorertree1" runat="server" />
```

Example 3-23: Using Configuration in a code-behind file that's a descendant of EPiServer.PageBase

```
…
mail.Body   += "\n\n" + Configuration.HostUrl + CurrentPage.LinkURL;
…
Configuration.InitSmtpServer();
```

Example 3-24: Using Configuration in HTML.

```
<mobile:link runat=server NavigateUrl=
    '<%#Configuration.RootDir+"templates/mobile.aspx?id="+Container.CurrentPage.PageLink.ID %>'
    ID="Link1"><%#Container.CurrentPage.PageName%>
</mobile:link><br/>
```

Example 3-25: Using the Page property in HTML.

```
<a href="<%= ( (EPiServer.PageBase) Page ).Configuration.RootDir %>" target="_parent">
```

Example 3-26: Using the Page attribute in a code-behind file.

```
protected string HeaderImage {
```

```
    get {
        EPiServer.PageBase PageBasePage = (EPiServer.PageBase) Page;
        if ( PageBasePage.CurrentPage.Property.Exists( "HeaderImage" ) ) {
            return (string) PageBasePage.CurrentPage[ "HeaderImage" ];
        }
        return PageBasePage.Configuration.RootDir + "images/header.gif";
    }
}
```

You Can Add Your Own Settings

Using the Item method of class ApplicationConfiguration makes it possible to add your own settings to web.config. How you do this is described on page page 156.

EPiServer Development

Developing EPiServer Solutions is a Little Different to Developing ASP.NET Solutions

As EPiServer 4 is based on and uses ASP.NET, developing with EPiServer also is based on and uses ASP.NET. However, there are important differences as we shall see. Although ASP.NET comprises both Web Services and Web Forms, we'll be concentrating on Web Forms, i.e. building Web sites using EPiServer, ASP.NET Web Forms and Controls. (EPiServer 4 is also implemented as a Web Service; more on this in a later chapter.)

Tools Needed

Before we can start developing solutions with EPiServer 4, we need several items of software:

- EPiServer 4.20, or any later version
- EPiServer Software Development Kit, SDK
- Microsoft .NET Framework, version 1.0 or 1.1 (EPiServer 4.3 and later requires version 1.1 of the Framework)
- ASP.NET (included in Microsoft .NET Framework)
- Visual Studio .NET (or Visual Studio .NET 2003)
- Microsoft SQL Server 2000 or Oracle Server 9i

The EPiServer products needed can be acquired from ElektroPost at http://www.episerver.com.

Developing ASP.NET Solutions

When developing ASP.NET Web Forms solutions, developers typically only seem to use Visual Studio .NET and Internet Information Services, IIS. Less visible, but very active indeed are both Microsoft .NET Framework and the underlying operating system.

Developing Solutions with EPiServer

So, by association, developing solutions with EPiServer must have a great deal in common with ASP.NET Web Forms application development. This is also the case.

EPiServer's Built-In Web User Controls

The folder templates\Units contains more than thirty ASP.NET Web User Controls which help with such chores as displaying and handling menus, Menu.ascx, MenuItem.ascx and TopMenu.ascx, and also printing, PrintFunctions.ascx, and much more.

You will also find a handful of Web User Controls in the admin and edit folders. These controls are used by EPiServer Admin and Edit modes, respectively.

EPiServer Base Classes and Interfaces

EPiServer ships with a number of useful classes, ready for you to benefit from. Paramount among them is of course the Region, its counterpart Content, and the Property class. The Property class is found in the name space EPiServer.WebControls. This is, as should be evident by now, a versatile class whose objects can provide a home for many different types of data.

Obviously the EPiServer Base Classes are important to an EPiServer developer. Among the Base Classes, perhaps the two most important are PageBase and PageData

The PageBase class is an abstract class used as a base class for one of the more specialised derived classes like SimplePage or TemplatePage. The latter is the most used base class for templates in EPiServer. All Web forms that are to be used as Page Templates in EPiServer must inherit from SimplePage or a descendant class.

Template logic often uses the PageBase class members to retrieve information about other aspects of the EPiServer solution, such as system configuration through the Configuration property, the user who is currently logged on (through the CurrentUser property), etc.

The PageData class contains information about a specific page. This includes the name of the page (PageName), reference (PageLink), URL (LinkURL) and more.

Other useful classes are PropertySearch and Content.

When using classes in EPiServer.WebControls, a reference to the EPiServer WebControls schema must be added in order for Visual Studio .NET IntelliSense

to work properly. The reference is often made part of an HTML Body tag or a Table tag.

Example 3-27: A reference to EPiServer WebControls schema added to an HTML Body tag.

```
<body class="NormalPage" xmlns:EPiServer="http://schemas.episerver.com/WebControls">
```

Example 3-28: A reference to EPiServer WebControls schema added to an HTML Table tag.

```
<table cellspacing="0" width="100%" xmlns:EPiServer="http://schemas.episerver.com/WebControls">
```

Please note: 'schemas.episerver.com' should be all lower case (following Internet tradition).

Extending EPiServer 4 Is a Lot Easier

In order to extend EPiServer 3, developers had to familiarise themselves with the source code for EPiServer, its include files and source files. Now, with EPiServer 4 all this has changed as EPiServer 4 has been developed in accordance with Microsoft's guidelines for .NET Framework. This means that you'll find everything you need in EPiServer 4 Software Development Kit, SDK, e.g. EPiServer's name spaces that you'll be using. Extending EPiServer 4 is as easy as inheriting from any base class and extending its functionality using new methods.

It should be evident from figure 3-3 that EPiServer itself is an ASP.NET application comprised of ASP.NET Web Forms pages (aspx files) with their corresponding code behind files. Used on the Web Forms pages are ASP.NET User Control (ascx files) and standard ASP.NET Web Controls. In addition we also find EPiServer Base Classes, which make up the bulk and core of EPiServer.

One, somewhat over-simplified, way of describing EPiServer 4 would be to say that it is a .NET Framework Assembly written in C# having a graphical user interface implemented in ASP.NET and using a database back end (either Microsoft SQL Server or Oracle 9i). To an application developer, the Assembly, comprising name spaces, classes and even objects, would attract the most interest.

Performance Considerations

One of the highest priorities in developing EPiServer 4 was its ultimate performance on customers' Web sites. When developing solutions with EPiServer there are only a few rules to abide by:

- When working with Page data, always use the static property EPiServer.Global.EPDataFactory
- Beware of shared data
- Be careful with personalization. It's a great feature but may result in heavy demand on the database

We will present a few tools and methods to enhance performance in a later chapter, *5. Avoiding Errors, Testing and Debugging.*

4
Mimicking the Example Web Site

Let's Create a Web Site by Mimicking the Example Site

In this chapter we'll recreate the example Web site that ships with EPiServer 4 and can be installed as the last step in the installation process. In doing this we'll guide you through the process step by step, just as if this were the first EPiServer Web site you've ever created.

During the site creation process, you'll be exposed to, and guided through, the Admin and Edit parts of EPiServer. In fact, creating your first site you will get a real feel for many of the possibilities and inherent power in EPiServer. The fact is: we won't be using Visual Studio .NET at all in this chapter!

Our two Web sites will be called Example and Mimic respectively. We'll start by creating and inventorying the Example Web site. We then create the Mimic Web site and make it look and act much like Example.

To conclude the chapter, we'll take a closer look at the Example Web site and see just how Framework Definition Files, Page Template Files, Web Controls and HTML cooperate.

Game Plan

First we'll install EPiServer and have the installation process create the example Web site as a last step. This Web site will be used as a template for our first Web site. We'll create an identical Web site using the Admin and Edit parts of EPiServer.

As EPiServer Web sites revolve around page templates, page types, with their properties, and displayed pages, we'll start with taking an inventory of all page types used in the example Web site.

Install EPiServer 4 and Let It Create the Example Web Site

We used the Web site name 'Example' for the first copy of EPiServer. You will see this in the figures below.

It doesn't matter whether you elect to create a new Web site or simply a new virtual folder. We installed to http://localhost/Example.

So, please install your first copy of EPiServer. When EPiServer is installed, a Web page will be presented, headlined 'Install basic content', and you should then

select 'English sample templates with all templates installed' and English for system language. Clicking on the button labelled 'Complete installation' will have the desired effect of completing the installation.

Next we'll be using EPiServer Admin and Edit modes to inventory the Example Web site.

Inventory Example Web Site

The Example Web site provided shows off some of EPiServer's capabilities.

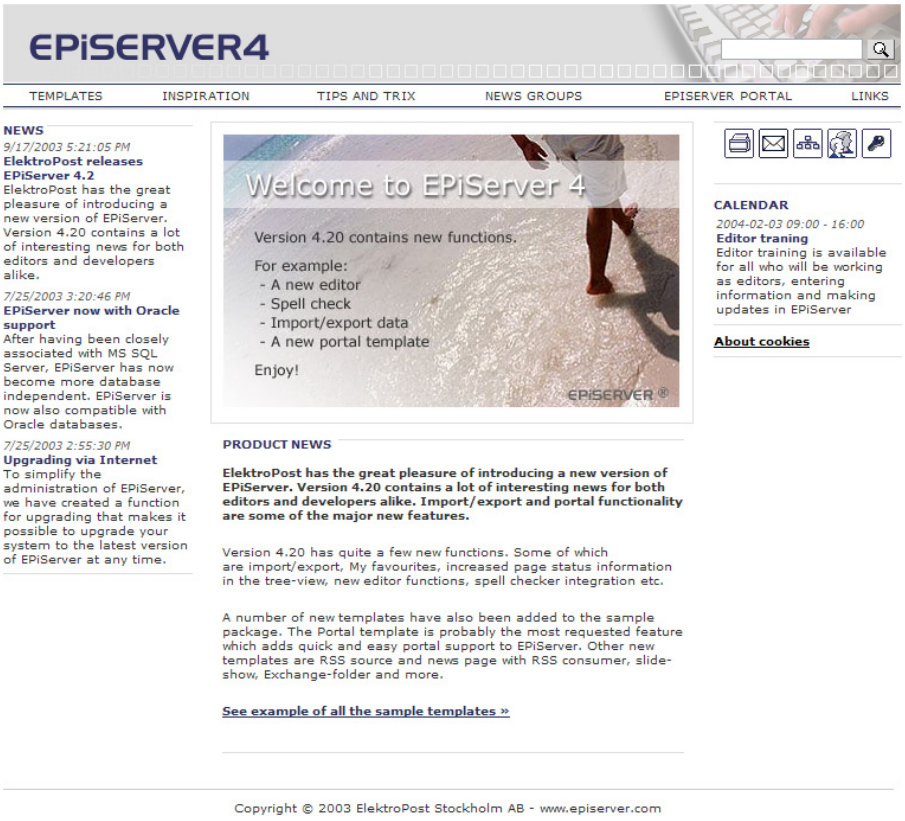


Figure 4-1: Example Web site in Internet Explorer.

Figure 4-1 shows the first page of the example Web site. Among the components are:

- Clickable invisible GIF picture, go to start page function (upper left corner)
- Main menu (horizontally across, below the top picture)
- News list (left margin area)
- Main editor area (centre)

- Tool-bar with print, e-mail, site map, register user and log-on buttons (to the right, below the main menu)
- Calendar with events (below tool-bar)
- Search function (upper right corner)

This example Web site is not a dummy. It is fully operational and can be added to and changed at will.

Page Templates, Page Types and Their Properties, Web Pages, Folders and the Database

Being an example Web site, it's only natural that it uses a lot of Page Templates (aspx files), but most of these are actually only included for show; they're not used to the extent they could be.

Page Templates which Really are Used in the Example Web Site

If we distinguish between the Page Templates which really are used in their own right and Page Templates that are included just to to demonstrate their capabilities, we find that the example Web site uses these Page Templates:

Table 4-1: Essential Page Templates in the Example Web site.

Page Template Name	*Page Template Name*
AlphaNumericListing.aspx	Calendar.aspx
ChangedPages.aspx	Default.aspx
Exchange.aspx	ExternalWebPage.aspx
News.aspx	NewsGroup.aspx
Page.aspx	PageRoller.aspx
Portal.aspx	Profile.aspx
ProfileFinder.aspx	Search.aspx
SiteMap.aspx	Subscribe.aspx

Depending on your previous experience with Web site management, this may seem like a lot of Page Templates, but let's take a look at how many Page Types are created from each Page Template. Notice that in the case of the example Web sites, we've been fortunate enough to be able to base several Page Types on a single Page Template. This is probably not going to be your experience, as one Page Type created from on Page Template File is more the norm.

Page Types Are Created from Page Templates–Page Types Own Page Templates

When you're new to EPiServer, it seems natural to think that Page Templates beget Page Types, which in turn beget Web Pages. However, as you become more familiar with EPiServer, you might start thinking of Page Types as above Page Templates in a kind of hierarchy.

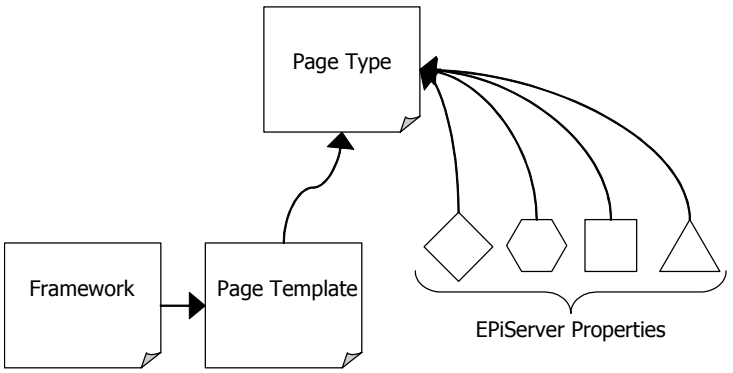


Figure 4-2: EPiServer Page Type viewed as 'Queen of the hill'.

An EPiServer developer spends most of her time using Visual Studio .NET, but a fair amount of time is also spent in EPiServer Admin mode creating Page Types from Page Templates. Table 4-2 lists the Page Types in the example Web site and the Page Templates from which they are created.

Table 4-2: Page Types in the example Web site.

Page Type	*Page Template Used to Created Page Type*
Alphabetical table of contents	AlphanumericListing.aspx
Calendar	Calendar.aspx
Calendar event	Page.aspx
Changed recently	ChangedPages.aspx
Discussion forum	Conference.aspx
EPiServer portal	Portal.aspx
Exchange folder	Exchange.aspx
External link	ExternalWebPage.aspx
File listing	FileListing.aspx
Flash page	FlashPage.aspx
Form page	Form.aspx

Table 4-2: Page Types in the example Web site.

Page Type	*Page Template Used to Created Page Type*
Index	SiteMap.aspx
Mobile page	MobileInfo.aspx
My Settings	PersonalSettings.aspx
News groups, category	NewsGroup.aspx
News groups, main page	NewsGroup.aspx
News groups, news group	NewsGroup.aspx
News groups, news item	NewsGroupItem.aspx
News list	News.aspx
News page	Page.aspx
Ordinary web page	Page.aspx
Personal profile	Profile.aspx
Profile search	ProfileFinder.aspx
Register users	Register.aspx
RSS source	RssProvider.aspx
Search	Search.aspx
Slide show	PageRoller.aspx
Start page	Default.aspx
Subscription	Subscribe.aspx
Vote	Form.aspx

The Page Types SysRecycleBin and SysRoot are only used in Edit mode; in fact they're part of Edit mode. Page Type Start page is the only Page Type created from a Page Template outside the templates folder. It is created from Default.aspx in the root folder of the example Web site. This makes a lot of sense,

since the Start page is under the control of Internet Information Services, IIS, initially.

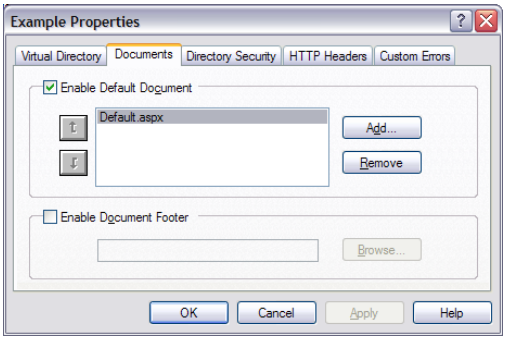


Figure 4-3: Properties for example Web site showing Default.aspx to be the default Web page.

Some Page Types Have Common Page Templates

Some of the Page Templates are used to create more than one Page Type, as shown in table 4-3.

Table 4-3: Page Templates used to create more than one Page Type in the example Web site.

Page Template Name	*Page Types created from Page Template*
Form.aspx	Form page; Vote; (2 Page Types)
NewsGroup.aspx	News groups, category; News groups, main page; News groups, news group; (3 Page Types)
Page.aspx	Calendar event; News page; Ordinary web page; (3 Page Types)

The leverage for the example Web site may not be the most impressive one: 26 Page Templates are used to create 32 Page Types, but when we turn our attention to the actual Web Pages, the story takes a more interesting turn. But first let's take a look at the properties used in the example Web site.

Properties Used for Page Types

The fastest way to get a list of all properties is to use the tables directly. There are two tables of interest for the properties: tblPageDefinitionType and tblPageDefinition. Table tblPageDefinitionType contains basic property type and tblPageDefinition user-defined properties. There are 101 distinct properties found in tblPageDefinition and thus 101 different properties are used to create the 32 Page

Types in the example Web site, averaging 3+ properties to every Page Type. Listing only those properties that appear more than twice we get this table:

Table 4-4: The most commonly used properties on Page Types.

Property Name	*Number of Times Used on Page Types*	*Base Property Type*
MainBody	22	LongString
MainIntro	15	String
WriterName	12	String
ListingContainer	9	PageReference
EPSUBSCRIBE	9	Boolean
ListingCount	7	Number
AllowInPortal	6	Boolean
PersonalizableProperties	6	Selector
ListingType	5	PageType
MetaKeywords	5	String
EPSUBSCRIBEHIDDEN	4	Boolean
MetaDescription	4	String

Dynamic Properties Used

The example Web site uses the dynamic properties listed in table 4-5.

Table 4-5: Dynamic properties used in the example Web site.

Property Name	*Data Type*	*Comment*
HeaderImage	URL to image	Header image on the Web site (at the top of the Web site)
MainMenuContainer	Page	Menu tree place from where menus will be displayed on the selected page
MainSearchPage	Page	Global search page
MetaKeywords	String	Comma-separated list of search words
RegisterPage	Page	Personal profile page in quick links

Table 4-5: Dynamic properties used in the example Web site.

Property Name	*Data Type*	*Comment*
RightListingContainer	Page	Starting point for the help list to the right.
Sitemap	Page	Site map page in quick links

Of the dynamic properties used on the example Web site, we'll re-create only MainSearchPage on our mimicking Web site. As we'll see later, there will be a highly visual 'scar' on the Web site as long as MainSearchPage is not defined and used.

Properties Used on Start Page

The properties used on the Start page determine what properties we have to add to the Mimic Web site. They are summarised in table 4-6.

Table 4-6: Properties used on Page Type Start page (start page for the Web site Example).

Property Name	*Purpose*
MainImage	Image to display on the start page (in MainRegion).
MainBodyHeading	Heading for the main body text of the start page (in MainRegion).
MainBody	Text, images, documents, tables, etc. for MainRegion.
NewsContainer	News listing page (in MenuRegion).
NewsCount	Number of news items to display (in MenuRegion).
EventsContainer	Specify the page you want to download calendar events from (in RightColumnRegion).
EventsCount	Using an integer, specify how many events are to be displayed.
ListingContainer	Specify the page you want to fetch the listing from.
ListingCount	Using an integer, specify the number of pages to be displayed in the listing.
MetaDescription	Describe the content of the page.
MetaKeywords	Specify a comma-separated list of search words that match this page.

Of these properties, EventsContainer is going to need some extra attention due to the fact that any Page Type pointed to by EventsContainer must have the properties EventStartDate and EventStopDate (this is an EPiServer requisite for calendar type information handling and we'll be discussing such matters in later chapters in more detail). This criterion is met by creating a Calendar event Page Type and adding those very properties to that Page Type. In Edit mode, a chain will then be created in which EventsContainer on the Start page will be set to point to a Calendar 'mother' page, which in turn has Calendar event pages for children.

Web Pages Created from Page Types

To see what Web Pages are created from the available Page Types, we enter EPiServer Edit mode for the example Web site.

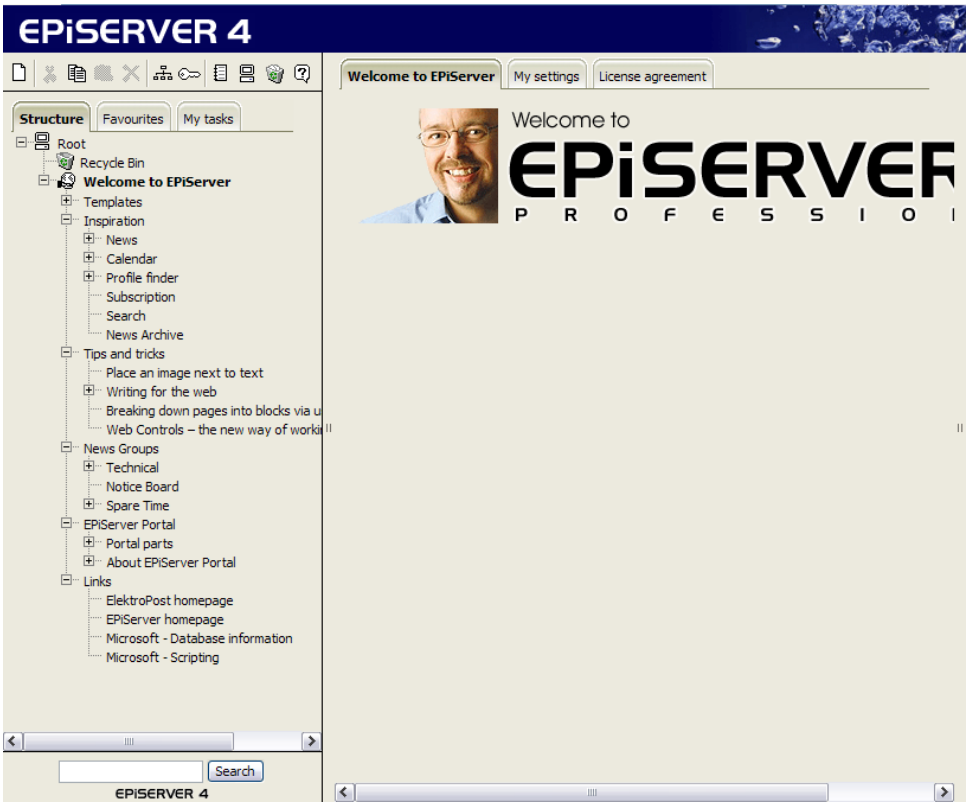


Figure 4-4: Example Web site in Edit mode, first level of page tree mostly expanded.

In all there are 102 Web pages in the example Web site. Of these, 86 are created from only 14 different Page Types, averaging 6+ Web Pages for each of the 14 Page Types. Taking this a step futher, we notice that the Page Template Page.aspx

is ultimately used to create 48 Web pages (as it's used to create three of the Page Types that are used to create multiple Web pages).

Table 4-7: Page Types used to create more than one Web page for the example Web site.

Page Type Name	*Number of Web Pages Created from Page Type*	*Page Template Used to Create Page Type*
Ordinary web page	34	Page.aspx
News page	8	Page.aspx
Calendar event	6	Page.aspx
Personal profile	6	Profile.aspx
News groups, news group	5	NewsGroup.aspx
News list	5	News.aspx
Discussion forum	4	Conference.aspx
Exchange folder	3	Exchange.aspx
External link	3	ExternalWebPage.aspx
Mobile page	3	MobileInfo.aspx
News groups, category	3	NewsGroup.aspx
Calendar	2	Calendar.aspx
Index	2	SiteMap.aspx
Slide show	2	PageRoller.aspx

Folders and Database Tables Used

Of the three Page Templates, Page Types and Web Pages, only Page Templates have a physical representation as files in folders. We already know which folder holds the most Page Templates: templates. Only the start page, Default.aspx, resides in another folder, the root folder for the Web site. One obvious deduction is that Page Types and Web Pages aren't stand-alone ASP.NET Web Forms; they cannot be handled in Visual Studio .NET.

So if Page Types and Web Pages don't live in folders, presumably they live in the database? This is certainly true; Page Types live in the table tblPageType and Web Pages live in the table tblPage. These tables are supplemented by two additional tables: tblPageDefinition and tblProperty, which hold the properties de-

fined for a Page Type and content of the properties for a Web Page, respectively. Let's summarise this in a table:

Table 4-8: Page Templates, Page Types and Web Pages in folders and the database.

	Folder	*Definition Table*	*Properties Table*
Page Templates	templates	—	—
Page Types	—	tblPageType	tblPageDefinition
Web Pages	—	tblPage	tblProperty

Create the Mimic Web Site: Install a New Version of EPiServer, or Re-Install

We are now done with the Example web site, although you might want to keep it around for comparative purposes.

Please install a second copy of, or re-install, EPiServer. Name this second Web site/virtual folder Mimic. Instead of installing the example Web site, 'English sample templates with all templates installed', this time select 'Only start page

without any Page Types or content'. Actually there will be some content, we won't have to re-create the Example Web site from scratch.

Figure 4-5: Visual appearance of Web site when selecting 'Only start page without any Page Types or content'.

As can be seen in figure 4-5, there is quite a lot of content in a Web site even when selecting 'Only start page without any Page Types or content' at the end of installation. However, most functionality is purely visual. You cannot use the mail functionality and there's no site map and no user registration. And, the visual 'scar' that was mentioned before is evident in the upper right corner. Near the search tool button, the text 'Quicksearch isn't properly configured. (Dynamic property "Global search page" must have been defined)' is displayed. Before we fix this, let's inventory the supposedly empty Web site.

Inventory 'Empty' Web Site

Figure 4-5 shows the first page of the 'empty' Web site. Among the components are:

- Clickable invisible GIF picture, go to start page function (upper left corner)
- Main editor area (centre)

- Tool-bar with print, e-mail, site map, register user and log-on buttons (to the right, below the main menu). Only some of the tool buttons are working
- Search function (upper right corner), displayed, but not working

Missing from this Web site compared to the example Web site are:

- Main menu (horizontally across, below the top picture)
- News list (left margin area)
- Calendar with events (below tool-bar)
- Search function (upper right corner), operational

Page Templates, Page Types and Their Properties, Web Pages, Folders and the Database

The only Page Type currently available for Mimic is Start page, so we're not surprised to find that the Page Template used for Start page is /Mimic/default.aspx.

Properties Used on Page Type Start Page

We find eleven static properties on the Start page:

Table 4-9: Properties used on Page Type Start page.

Property Name	*EPiServer Property Type*
EventsContainer	Page
EventsCount	Integer
ListingContainer	Page
ListingCount	Integer
MainBody	LongString
MainBodyHeading	String
MainImage	URL to image
MetaDescription	String
MetaKeywords	String
NewsContainer	Page
NewsCount	Integer

Dynamic Properties Used

As you might have guessed already, there are no defined Dynamic Properties for this Web site.

Web Pages Created from the Single Page Type

To see what Web Pages are created from the available Page Types, we enter EPiServer Edit mode for the Mimic Web site. No surprise here either: there's a single Web Page, appropriately named 'Start page'.

Order of Business

When enhancing the Mimic Web site to look and behave like the Example Web site, there's really no preferred way or order to go about the tasks. This is the proposed order of business:

1. Create Dynamic Property 'Global search page', assign this to Start page and give it a value.
2. Create the same Page Types that exist in the Example Web site
3. Create the same Web Pages that exist in the Example Web site.

We happen to know that we'll have to create a new data type, but let's deal with that as we get to that particular Page Type.

Create and Use Dynamic Property 'Global search page'

The name of the 'Global search page' Dynamic Property is MainSearchPage and its EPiServer data type is Page (see table 4-5 on page 63).

Open up Mimic Admin mode, click on 'Dynamic properties' in the left pane and then on 'Add property' in the right pane. Assign Name, Heading and Data type as appropriate, then save. Figure 4-6 shows the list of Dynamic Properties after adding MainSearchPage.

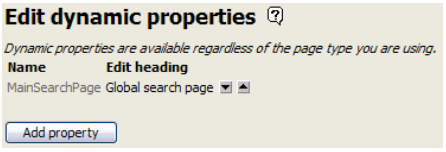


Figure 4-6: Dynamic Property MainSearchPage added.

Switch to Edit mode, click on the tool button (Edit mode), and assign MainSearchPage to Start page by first clicking on Start Page and then on the tool button (Edit dynamic properties). Let the Global search page be the Start page (this is the only possible choice at this time). This will get rid of the annoying error text on the start page and also, coincidentally, of the Search function altogether. Don't worry, we'll bring it back when we have a proper Search page.

Create the Same Page Types That Exist in the Example Web Site

Actually, we won't create all the Page Types that exist in the Example Web site, as most of them are only there to serve as examples themselves.

We'll Cheat a Little

Our objective in mimicking the Example Web site is to show some of the power of EPiServer and not to have you perform lots of tedious chores. To this end we'll cheat in two areas:

- We are going to create a Page Type with the most commonly used Properties and then copy from this Page Type, leaving some extraneous Properties
- We are not going to create all of the Page Types that exist in the Example Web site, and more specifically we won't bother with portals just now

Create a Page Type to Hold Most Common Properties

On the Example Web site, the most commonly used Properties are found in table 4-4 (on page 63). We are going to make a Page Type whose sole purpose is to hold all of these properties and then allow us to copy new Page Types using this as a template, thereby eliminating some work for ourselves.

Open Mimic in Admin mode, click on 'Create new page type' in the left pane and then fill in the fields as follows::

Table 4-10: Template Type information.

Field Name	*Value*
Name	Template Type
Description	Page Type to hold most commonly used Properties (not available in Edit mode).
File Name	/Mimic/templates/Page.aspx
Available in Edit mode	No (deselect)
Sort index	100 (leave as is)

Filling in 'Description' isn't absolutely necessary, but we like to do it as it forms part of the ever-important documentation.

We won't add Properties in the same order as in table 4-4, but rather with an eye on how Editors use the properties, as the properties we add will be displayed to Editors in the order we choose. Furthermore, the property MetaKeywords will not be added to Template Type, as we will let the Dynamic Property MetaKeywords be used instead. Also, the properties AllowInPortal and PersonalizableProperties will be left out since portals are beyond the current scope.

When you're ready, add the properties found in the list below in the same order to Template Type. Leave all properties under heading Information. Click the check box 'Searchable property' for all properties that are either Strings or Long strings, i.e. MainIntro, MainBody and WriterName.

- MainIntro
  Heading: Introduction
  Help text: Introductory text often shown in listings
  Data Type: String
- MainBody
  Heading: Editor
  Help text: Content of your page. Here you can enter text, insert images, documents, tables, etc.
  Data Type: Long string
- WriterName
  Heading: Writer
  Help text: Name of the person who is responsible for the content
  Data Type: String
- ListingContainer
  Heading: Fetch listing from
  Help text: Specify the page you want to fetch the listing from
  Data Type: Page
- ListingType
  Heading: Type of page in the listing
  Help text: Specify the page type that is to be shown in the listing
  Data Type: Page type
- ListingCount
  Heading: Display number of pages in the list
  Help text: Using an integer, specify the number of pages to be displayed in the listing
  Data Type: Integer

Create Page Type 'Ordinary web page'

As with the Example web site, we'll also be using 'Ordinary web page' for most Web Pages in Mimic. Create the Page Type by copying Template Type. In Admin mode, click on Copy page type in the left pane, copy from Template Type. Name the new Page Type 'Ordinary web page' and enter 'Multi-purpose Page Type using DefaultFramework' for description. The properties we need on Ordinary web page are already present, so we won't change anything.

Create Page Types 'Calendar' and 'Calender event'

As noted earlier, we need a Page Type created from a Page Template that includes properties such as Calendar.aspx. Create a new Page Type, this time by selecting 'Create new page type' in the left pane, with the following information:

- Name: Calendar
  Description: Mother page for calendar events.
  File Name: /Mimic/templates/Calendar.aspx
  Available in Edit mode: Yes
  Sort index: 100 (leave as is)

We chose not to copy Template Type, as none of the properties needed for the Calendar Page Type are present in Template Type. Add these properties to the Calendar Page Type:

- nDaysToRender
  Heading: Number of days to show in calendar
  Help text: Specify an integer for the number of days you want to show in the calendar.
  Data Type: Integer

- CalendarType
  Heading: Type of pages to show in calendar
  Help text: Specify the Page Type to use for calender event
  Data Type: Page type

- CalendarContainer
  Heading: Show calendar information from
  Help text: Specify a place in the menu tree from where calendar will be displayed
  Data Type: Page

Now, Calendar must point to a Page Type that includes the properties EventStartDate and EventStopDate. For this Page Type, we'll be using the Page Template Page.aspx and simply adding the proper properties to it. We wish to be able to describe the event using MainIntro and Mainbody, so we'll use Template Type as base again. Create a new Page Type by copying Template Type:

- Name: Calendar event
  Description: Single calendar event.
  File Name: /Mimic/templates/Page.aspx
  Available in Edit mode: Yes
  Sort index: 100 (leave as is)

Add these properties to Calendar event:

- EventStartDate
  Heading: Start date and time for this event.
  Help text: Specify date and time when this event begins.
  Data Type: Time/Date

- EventStopDate
  Heading: Stop date and time for this event.

Help text: Specify date and time when this event is over.
Data Type: Time/Date

To underline the importance of these two properties, use the tool button Move up,  , to place them at the top of the Properties list.

(In the Example Web site, this Page Type also has the ability to cater for re-curring events; this possibility will be left as an exercise for the reader.)

We will come back to Admin mode shortly, after a short interlude to create a few top-level Web Pages.

Interlude: Create the Top-Level Web Pages

Just to break up the Admin mode session, we'll direct our attention to Edit mode and create the top-level Web Pages.

Switch to Edit mode and create the following Web Pages, base all on Ordinary web page, in the same order as in the list. Right-click on 'Start page' and select Create new in the shortcut menu. Select Ordinary web page, name the new Web Page and then click on 'Save and publish'.

- Links
- EPiServer Portal
- News Groups
- Tips & Tricks
- Inspiration
- Templates

If you try to view the Start page, you'd most likely see an error page stating that 'PageType must include properties EventStartDate and EventStopDate'. This is due to the property EventsContainer (Download calendar events) not being handled correctly. To remedy this we'll create a new child Web Page for Inspiration of Page Type Calendar and set its pertinent attributes correctly and, eventually, set EventsContainer on the Start Page to point to the new Calendar Web Page.

Create a new child Web Page for Inspiration by right-clicking on Inspiration and then selecting Create new Web page in the shortcut menu. Make the new Web Page from Page Type calendar and name it 'Calendar'. Edit the Calendar Web Page and enter the following information:

Table 4-11: Important property settings for Calendar Web Page.

Property Caption	*Value*
Number of days to show in calendar	7
Types of pages to show in calendar	Calendar event

Table 4-11: Important property settings for Calendar Web Page.

Property Caption	*Value*
Show calendar information from	This page

Save and publish this information and then edit the Start page and set the property 'Download calendar events' to point to the Calendar Web Page.

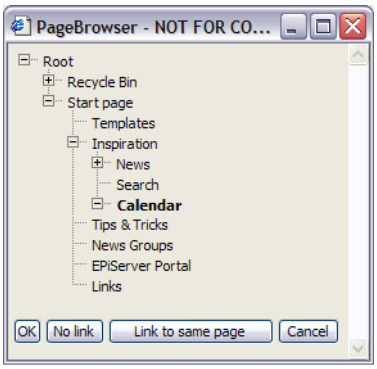


Figure 4-7: Pointing the 'Calendar' Web page.

Switch the Web site to view mode and make sure that everything works as expected. Note for example what happens to the QuickSearch function whenever you navigate away from the Start page: it appears in the upper right corner and when the tool button 'Quick search', , is pressed the Start page is loaded. This is exactly as we specified: we set the Dynamic Property 'Global search page' to the Start page.

Now that we've got peace of mind, let's create some more Page Types.

Create the Last Page Types

To make Mimic look really snazzy like the Example Web site, we need a new list function. To this end we'll create two new Page Types: News list and News page. The former is based on Page Template News.aspx.

Create a new Page Type based on News.aspx by copying Template Page:

- Name: News list
  Description: Mother page for news items (pages).
  File Name: /Mimic/templates/News.aspx
  Available in Edit mode: Yes
  Sort index: 100 (leave as is)

We won't need all properties; if you like you can remove all but ListingContainer and ListingCount.

The next Page Type we'll create is News page. This is based on Page.aspx.

Create a new Page Type based on News.aspx by copying Template Page:

- Name: News page
  Description: Single news item (page).
  File Name: /Mimic/templates/Page.aspx
  Available in Edit mode: Yes
  Sort index: 100 (leave as is)

Again we only need a few of the Properties; remove all but MainBody, MainIntro and WriterName.

One last Page Type to go. This Page Type we'll be used for the Search function, so we can have the QuickSearch bar back on the start page. The Search Page Type will be based on the Page Template Search.aspx.

Create a new Page Type based on News.aspx by copying Template Page:

- Name: Search
  Description: Main search page.
  File Name: /Mimic/templates/Search.aspx
  Available in Edit mode: Yes
  Sort index: 100 (leave as is)

This time leave only two Properties: MainBody and MainContainer. But, we should also add a few more. Add these Properties to Search.

- MainCatalog
  Heading: Catalog for Index Server.
  Help text: Specify the Index Server catalog to search.
  Data Type: String

- MainContainer
  Heading: Start page for the search.
  Help text: If this field is left empty, the search will start from the start page.
  Data Type: Page

- MainScope
  Heading: Scope for Index Server search.
  Help text: Specify the folder in which the search will take place.
  Data Type: String

That means we're done. Now we have all the Page Types we need to make Mimic look and behave a lot like Example. The Mimic Web site now comprises the following Page Types:

Table 4-12: Page Types for the Mimic Web site.

Page Type Name	*Description*
Calendar	Mother page for calendar events.
Calendar event	Single calendar event.

Table 4-12: Page Types for the Mimic Web site.

Page Type Name	*Description*
News list	Mother page for news items (pages).
News page	Single news item (page).
Ordinary web page	Multi-purpose Page Type using DefaultFramework,
Start page	Start page for Web site.
Template Page	Page Type to hold most commonly used Properties (not available in Edit mode).

It's time to create more Web Pages.

Create the Same Web Pages That Exist in the Example Web Site

There are already some Web Pages in the Mimic Web site. We are now going to create the remaining to make the Mimic Web site almost perfectly mimic the Example Web site.

Putting the QuickSearch Bar Back on the Start Page

Enter Mimic's Edit mode and right-click on the Web Page Inspiration in the left pane. Select 'Create new' and then click on 'Create' in the right pane to create a Web Page from the Search Page Type. Name the new Web Page 'Search'. Let the Property 'Start page for the search' point to the Start page and then click on 'Save and publish'. The QuickSearch bar will be displayed on the Start page as soon as the Dynamic Property 'Global search page' has been set to point to the new Search page.

To set the Dynamic Property, first select the Start page and then click on the Dynamic Properties tool button  . Set 'Global search page' to point to the new Search Page. Click Save and make sure that the QuickSearch bar is now back on the Start page.

Spreading The News

Mimic's news list is currently flawed, as it is using the Web Page Tree for news items. This won't do. The remedy is to create a proper news list from Page Type News list and then produce a couple of news items for it to display.

Create a new Web Page of type News list and place it under Inspiration (select Inspiration, right-click and select 'Create new', click on Create in the right pane for Page Type News list.). Set the Properties to these values:

- Fetch listing from
  Setting: Click the ellipsis button, then click on 'Link to same page' which lists as 'This page'.

- Display number of pages in the list
  Setting: 3

Let's spread some news. Select News in the left pane and create a couple of news items from Page Type News page. Remember: they will be displayed in reverse creation order. Here is some example information for you to use:

- Name: Mimic Up And Coming
  Introduction: Web site Mimic is a strong contender says Reuters.
  Editor: Today Reuters news agency report that Mimic...

- Name: To Be Reckoned with
  Introduction: Creators of Web sites should pay close attention to Mimic.
  Editor: Web site Mimic has been an underdog for too long...

- Name: Mimic Almost as Good as Any Example
  Introduction: The Mimic Web site has been found to equal any example Web site in appearance.
  Editor: So there!

Now click on Start page, activate the Edit tab in the right pane and let the Property 'News list' point to the Web Page News. After saving and publishing, the Start Page has a clickable news list.

Set up a Few Dates

The penultimate thing we'll add to the Mimic Web site is a few appointments, calendar events. There is already a container for calendar events; it only needs some actual events.

Create a couple of calendar events and place them all under Calendar in the Web Page Tree. As for dates, make sure that at least one of them is for the current month, since the Web User Control Calendar.aspx only displays events for the current month.

- Name: Mimic Editor Training
  Start date: 1 July 2004 8 AM
  Stop date: 2 July 2004 4 PM (16.00)
  Introduction: Inaugural Mimic Editor Training Course
  Editor: This is the first Mimic Editor training course ever presented.

- Name: Mimic Admin Training
  Start date: 1 June 2004 8 AM
  Stop date: 4 June 2004 4 PM (16.00)
  Introduction: Inaugural Mimic Admin Training Course
  Editor: This is the first Mimic Admin training course ever presented.

- Name: Mimic Developer Training
  Start date: 3 May 2004 8 AM

Stop date: 5 May 2004 4 PM (16.00)
Introduction: Inaugural Mimic Developer Training Course
Editor: This is the first Mimic Developer training course ever presented.

Cleaning up the Start Page

The last thing to do is to clean up the Start page. It lacks an image, a heading and some text. Click on Start page in the left pane and then select the Edit tab in the right pane. Enter settings for the Properties of your own choice or use these:

- Main image on start page: /Mimic/images/startpage_image.jpg
  Main body heading:
  Editor: Mimic is here to stay!

Final Result

Comparing the present Web site with figure 4-5, it is apparent that Mimic has come a long way.

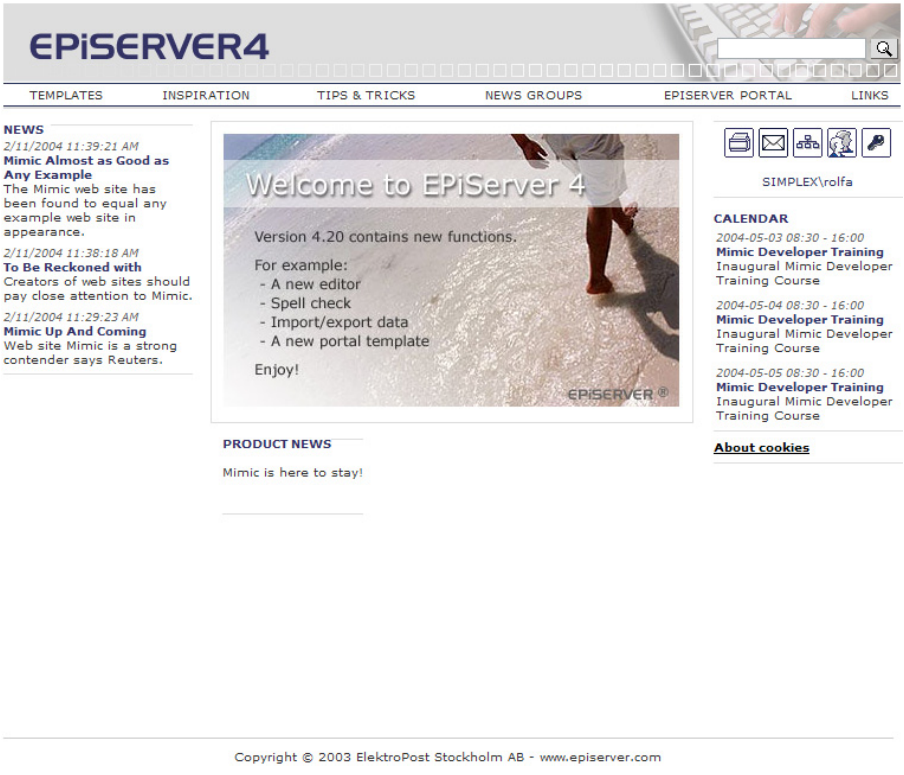


Figure 4-8: Final appearance of the Mimic Web site.

Using a lot of built-in and supplied EPiServer functionality for the Mimic Web site, all of the following now work:

- Homing device (the ability to 'go home' when you click on the EPiServer 4 logo in the upper left corner)
- Top menu
- News list with news items
- Calendar list with calendar items
- Printing
- Logging on
- Searching the Web site

Of course, the vital Admin and Edit modes have been working all along.

Hopefully this first example has been successful in demonstrating EPiServer's capabilities and familiarising you with Admin and Edit modes. There's one thing more we can look at: how to clean up after ourselves.

Removing Superfluous Web Page Versions

Every time a Web Page is saved, the old settings are stored away safely letting Editors handle as many versions of individual Web Pages as they like. During the course of creating the Mimic Web site, we have created several versions of some Web Pages, in particular the Start Page. We'll remove all but the last version now.

In Edit mode, select the Start page in the Web Page Tree, left pane, and then select the Version list tab in the right pane. Delete all but the current version, the last, of the Start page. (Simply click the Delete button on the first line as many times as there are extraneous versions.)

A Closer Look at Frameworks and Page Templates in the Example Web Site

Anatomy of an EPiServer Framework Definition File

Web User Controls and EPiServer Base Classes in DefaultFramework

We have discussed them already, but let's take another look at the Web User Controls and Base Classes in the DefaultFramework.

Thirteen of the Web User Controls found in templates\Units are registered for use in the Framework Definition File DefaultFramework.ascx.

Example 4-1: User Controls registered for use in DefaultFramework.ascx.

```
<%@ Register TagPrefix="development" TagName="WriterInfo" Src="~/templates/Units/
WriterInfo.ascx"%>
<%@ Register TagPrefix="development" TagName="SiteFooter" Src="~/templates/Units/
SiteFooter.ascx"%>
<%@ Register TagPrefix="development" TagName="RightListing" Src="~/templates/Units/
RightListing.ascx"%>
```

```
<%@ Register TagPrefix="development" TagName="QuickBar" Src="~/templates/Units/
QuickBar.ascx"%>
<%@ Register TagPrefix="development" TagName="Quicksearch" Src="~/templates/Units/
QuickSearch.ascx"%>
<%@ Register TagPrefix="development" TagName="Print" Src="~/templates/Units/
PrintFunctions.ascx"%>
<%@ Register TagPrefix="development" TagName="PageHeader" Src="~/templates/Units/
PageHeader.ascx"%>
<%@ Register TagPrefix="development" TagName="PageBody" Src="~/templates/Units/
PageBody.ascx"%>
<%@ Register TagPrefix="development" TagName="LeftMenu" Src="~/templates/Units/Menu.ascx"%>
<%@ Register TagPrefix="development" TagName="TopMenu" Src="~/templates/Units/
TopMenu.ascx"%>
<%@ Register TagPrefix="development" TagName="LoginStatus" Src="~/templates/Units/
LoginStatus.ascx"%>
<%@ Register TagPrefix="development" TagName="Listing" Src="~/templates/Units/Listing.ascx"%>
<%@ Register TagPrefix="development" TagName="Header" Src="~/templates/Units/Header.ascx"%>
```

In order to use classes found in the name space EPiServer.WebControls, this is also registered in DefaultFramework.ascx.

Example 4-2: Name space EPiServer.WebControls registered for use in DefaultFramework.ascx.

```
<%@ Register TagPrefix="EPiServer" Namespace="EPiServer.WebControls" Assembly="EPiServer" %>
```

The HTML Tables in DefaultFramework

The HTML body in DefaultFramework comprises just one form, that's all. In this form we find one table, but with nested tables inside it. The tabular skeleton looks like this:

Example 4-3: HTML table skeleton in DefaultFramework.

```
<table width="760" border="0" cellspacing="0" cellpadding="0" align="center">
    <tr>
        <td colspan="2" align="center"> 
            <table width="100%" border="1" cellspacing="0" cellpadding="0" background="<%=
HeaderImage %>" style="background-repeat:no-repeat">
            <tr>
                <td>
                    <table width="100%" border="1" cellspacing="0" cellpadding="0" height="96">
                    <tr>
                        <td width="79%" valign="top"> 
                            <a href='<%=EPiServer.Global.EPConfig.RootDir%>'><EPiServer:Clear
width="300" height="70" runat="server" /></a>
                        </td>
                        <td width="21%" align="center"> 
```

```
                        <EPiServer:Clear height="18" runat="server" /><br />
                        <development:quicksearch ID="QuickSearch" runat="server" />
                    </td>
                </tr>
                <tr>
                    <td colspan="3" height="23"> 
                        <development:TopMenu runat="server" id="TopMenu" />
                    </td>
                </tr>
                </table>
            </td>
        </tr>
        </table>
    </td>
</tr>
<tr>
    <td colspan="2"><EPiServer:Clear height="10" runat="server" /></td>
</tr>
<tr>
...
</tr>
</table>
```

Perhaps we need to dissect the table layout a bit. The first table has only three rows in it. All three rows have a single column, but, as a preparation for other files using this Framework Definition File, this single column is set to span two columns (colspan="2").

In turn, the first line of the first table has two nested tables inside it.

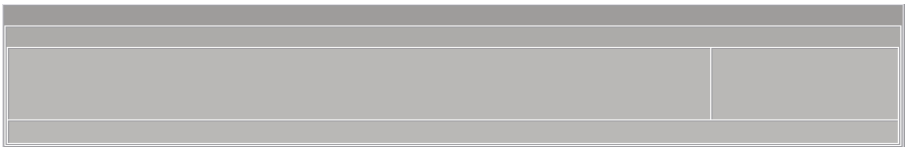


Figure 4-9: The nested tables in the first row of the outer table.

Row two doesn't have much in it, so most of the action is found in the last row, row number three. This row is home to no fewer than four nested tables.

Looking at only the nested tables in the third row of the outer table we notice that the outermost of those tables has two rows, the last of which is simply used as a footer (EPiServer:SiteFooter).

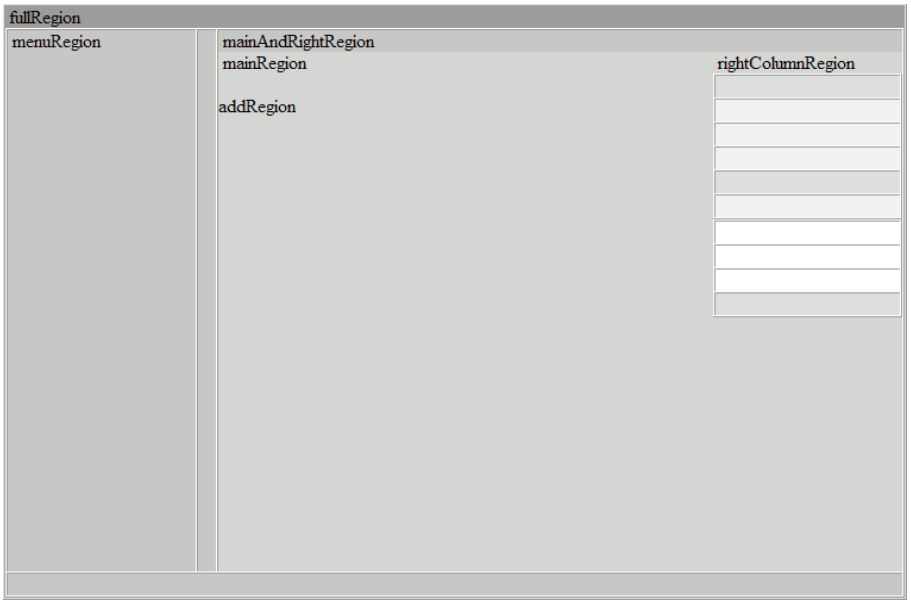


Figure 4-10: The nested tables in row two of the outer table (region names included).

That's it for the layout by HTML tables in the Framework Definition Files, now let's turn our attention to Regions as used by EPiServer.

EPiServer Regions

Regions are visual rectangles defined in Framework Definition Files and used in Page Template Files. The purpose of Regions is two-fold: to allow the creator of Framework Definition Files to supply default visual HTML elements and User Controls and to allow the maker of Page Template Files to benefit from these default elements or replace them with others at she pleases.

No fewer than six regions are defined in DefaultFramework.ascx. No regions are defined to cover any of the two first rows in the first table so these are immutable.

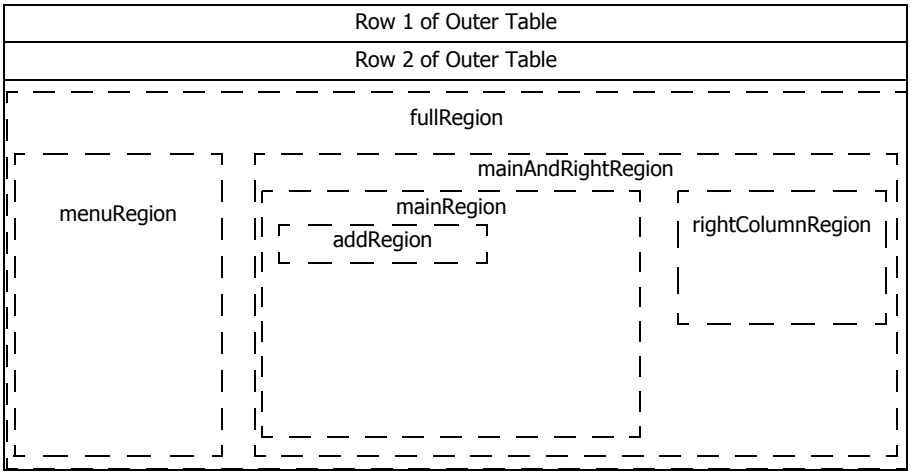


Figure 4-11: The six regions defined in Default Framework.ascx.

In the figure, all six regions are depicted. One region, fullRegion, encompasses all the others so an easy way to replace all default content in DefaultFramework.ascx would be to only define content for fullRegion in a Page Template File. Besides fullRegion, mainAndRightRegion is also nested. There are three regions inside it. So there you have it: using DefaultFramework.ascx gives a Page Template File a form and an HTML table, the first two rows of which are static and the third row may be altered at will using any or all six regions.

Inside an EPiServer Page Template File

One of several users of DefaultFramework.ascx is default.aspx, the start page for the example Web site.

Example 4-4: Part of default.aspx, the start page for the example Web site.

```
...
<EPiServer:Content ID="NewsListing" Region="menuRegion" runat="server">
...
</EPiServer:Content>
<EPiServer:Content Region="mainRegion" runat="server">
...
</EPiServer:Content>
<EPiServer:Content ID="Events" Region="rightListingRegion" runat="server">
...
</EPiServer:Content>
```

As you can see, default.aspx uses three of the six regions in DefaultFramework.ascx: menuRegion, mainRegion and rightColumnRegion.

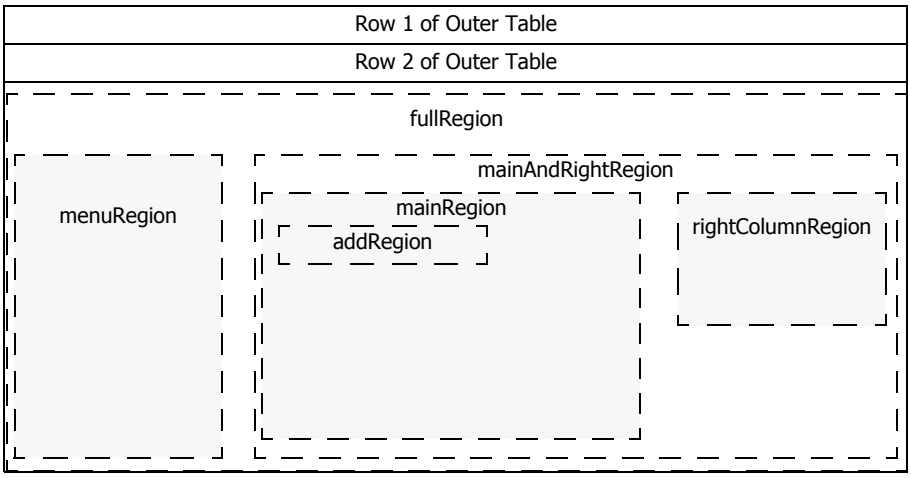


Figure 4-12: Regions in DefaultFramework.ascx used by default.aspx.

Figure 4-12 shows the regions in DefaultFramework.ascx which are used by the Page Template File Default.aspx.

Framework Definition Files

The Example Web site uses four Framework Definition Files: AdvancedPortalFramework.ascx, DefaultFramework.ascx, PortalUnitFramework.ascx and SimplePortalFramework.ascx. Saving portals for later we'll focus on DefaultFramework.ascx.

Page Template Files

Of the Page Template Files listed in table 4-1, only Exchange.aspx, ExternalWebPage.aspx and NewsGroup.aspx do not use a Framework Definition File at all. All the others use DefaultFramework.ascx and some also use one or more portal frameworks.

Looking into the Start Page, Default.aspx and DefaultFramework.ascx

The visual layout of the Start page and the functions on it are the responsibility of Default.aspx and its Framework Definition File, DefaultFramework.ascx. So, here we go (again).

Looking into the Immutable Part of DefaultFramework.ascx

To understand how Default.aspx and DefaultFramework.ascx cooperate to form the Example Web site's first page, and how other Page Templates use DefaultFramework.ascx we need to look at pertinent parts of DefaultFramework.ascx. As stated in the last chapter, the visual layout of DefaultFramework.ascx is real-

ised by HTML Table elements, but using tables is simply a matter of choice, Div elements will work just as well.

Example 4-5: First row of outer table in DefaultFramework.ascx contains two nested tables.

```
<td colspan="2" align="center">
    <table width="100%" border="0" cellspacing="0" cellpadding="0"
        background="<%=HeaderImage%>" style="background-repeat:no-repeat">
        <tr>
            <td>
                <table width="100%" border="0" cellspacing="0" cellpadding="0" height="96">
                    <tr>
                        <td width="79%" valign="top">
                            <a href='<%=EPiServer.Global.EPConfig.RootDir%>'>
                                <EPiServer:Clear width="300" height="70" runat="server" /></a>
                        </td>
                        <td width="21%" align="center">
                            <EPiServer:Clear height="18" runat="server" /><br />
                            <development:QuickSearch ID="QuickSearch" runat="server" />
                        </td>
                    </tr>
                    <tr>
                        <td colspan="3" height="23">
                            <development:TopMenu runat="server" id="TopMenu" />
                        </td>
                    </tr>
                </table>
            </td>
        </tr>
    </table>
</td>
```

The listing in example 4-5 shows the first row of the outer table in DefaultFramework.ascx. (See figure 4-9 on page 82.)

The most interesting elements have been marked with bold face.

Remember, every Page Template using DefaultFramework.ascx will have the same functions and appearance in the first two rows of the three-row outer table. The only thing that can change is the background image used. There is a Dynamic Property with the name HeaderImage used for this very purpose.

Setting the Table Background 'background="<%=**HeaderImage**%>"'

The HTML code '<table … background="<%=**HeaderImage**%>" … >' sets the background of the first table in the first row of the outer table in DefaultFramework.ascx to this image, which is found in /Example/images/Header.gif:

Figure 4-13: Picture used for background in the first table in the first row of the outer table in DefaultFramework.ascx.

The picture in figure 4-13 is readily recognised from the Example Web site. It is present on all pages that use DefaultFramework.ascx for Framework Definition File. As the first two rows of the outer table in DefaultFramework.ascx never change, there are no Regions defined.

HeaderImage is a Dynamic Property (see table 4-5), but if you were to look at its value for the Start Page, 'Welcome to EPiServer', you'd notice that it hasn't been set. There must be some default mechanism at play since its easy to see that the file Header.gif is displayed when the start page is loaded. And indeed there is a default mechanism, this time in the code-behind file DefaultFramework.ascx.cs. Loading this file into Visual NotePad or Visual Studio .NET, we find the appropriately named property function HeaderImage:

Example 4-6: Property function HeaderImage in the code-behind file DefaultFramework.ascx.cs.

```
protected string HeaderImage {
    get {
        PageBase page = (PageBase) Page;
        if ( page.CurrentPage.Property.Exists( "HeaderImage" ) ) {
            return (string) page.CurrentPage[ "HeaderImage" ];
        }
        return page.Configuration.RootDir + "images/header.gif";
    }
}
```

The logic of the HeaderImage function is quite obvious: first test for the presence of a property, static or dynamic, called HeaderImage on the current page, i.e. the start page Default.aspx. If there is no such property, return a string consisting of the concatenation of the contents of page.Configuration.RootDir and 'images/header.gif'. The value of page.Configuration.RootDir is the installation folder path for the EPiServer Web site with a slash appended, such as '/Example/'.

Anchor '<a href='<%=EPiServer.Global.EPConfig.RootDir%>'>' with an Image

Again we're dealing with the installation root folder path for the Example Web site, only this time we use the EPiServer.Global class and its static (Shared in Vis-

ual Basic .NET) method EPConfig. EPConfig returns an instance of the class ApplicationConfiguration which among its method has RootDir. The explanation for RootDir is 'Root folder [directory] for the site', which would be '/Example/' for the Example Web site (it's a virtual folder).

The visual part of the HTML anchor is: '<EPiServer:Clear width="300" height="70" runat="server" />'. Clear is an EPiServer class that creates a transparent GIF picture with the given dimensions, defaulting to 1 if either height or width is missing. The HTML code resulting from the code above contains an image, img, tag and looks like this:

Example 4-7: Actual HTML code resulting from DefaultFramework.ascx.

```
<a href='/Example/'><img src="/Example/util/images/clear.gif" alt="" width="300" height="70"
border="0" /></a>
```

In fact, the file Clear.gif contains a 1x1 transparent GIF picture.

The end result is a transparent area covering the text 'EPiSERVER4' in the upper left corner on the pages acting as a 'homing device'.

Using QuickSearch '<development:QuickSearch ID="QuickSearch" runat="server" />'

All you have to do to equip your Web site with search capabilities is to include the User Control QuickSearch.ascx and give it a Web page to use.

In order to use the User Control QuickSearch.ascx, it must first be registered. This is done by including the line '<%@ Register TagPrefix="development" TagName="Quicksearch" Src="~/templates/Units/QuickSearch.ascx"%>' at the top of DefaultFramework.ascx. The effect of including the line '<development:QuickSearch ID="QuickSearch" runat="server"/>' in the HTML code is this:

Example 4-8: Resulting HTML code when using QuickSearch.ascx.

```
<span id="defaultframework_QuickSearch_QuickSearchSpan">
    <table border="0" cellpadding="0" cellspacing="0">
        <tr>
            <td>
                <input name="defaultframework:QuickSearch:SearchText" type="text"
                    id="defaultframework_QuickSearch_SearchText" style="width:120px;" />
            </td>
            <td style="padding-left: 4px;padding-top: 2px">
                <a id="defaultframework_QuickSearch_QuickSearchButton"
                    href="javascript:__doPostBack(
                    'defaultframework$QuickSearch$QuickSearchButton','')">
                <img src="images/ico_find.gif" id="defaultframework_QuickSearch_Img1" border="0"
                    alt="Quick search" /></a>
            </td>
        </tr>
```

```
    </table>
</span>
```

We get what we see: an input box and a tool button. Clicking on the tool button calls the JavaScript function '__doPostBack' with two arguments.

Web Pages As Menu: <development:TopMenu runat="server" id="TopMenu" />

A very powerful and yet simple construct in EPiServer is that every Web Page in the Edit mode Page Tree is a potential menu entry. The Page Tree and menu hierarchy has a direct correspondence, a first-level Web Page is a first-level menu, and so on. The choice of presenting this menu horizontally across or vertically in the left margin is almost as easy as choosing between the two User Controls TopMenu.ascx and Menu.ascx. Only Menu.ascx, often used as menu in the left-most area, is actually hierarchical, TopMenu.ascx always displays only the first-level Web Pages as menu items.

In this case, TopMenu.ascx is used in the second row of the inner-most table. This corresponds to the 'empty' row in the background picture in figure 4-13. This is the resulting HTML code:

Example 4-9: HTML code when using TopMenu.ascx.

```
<table width="100%" border="0" cellspacing="0" cellpadding="0" XmlNs:EPiServer="http://
schemas.episerver.com/WebControls">
    <tr valign="middle" align="center">
        <td height="23">
            <a class="MenuHead" href="/Example/templates/Page.aspx?id=4">Templates</a>
        </td>
...
        <td height="23">
            <a class="MenuHead" href="/Example/templates/Page.aspx?id=98">Links</a>
        </td>
    </tr>
</table>
```

This code is created from the template found in TopMenu.ascx and EPiServer built-in functions, which have full access to the Edit mode Page Tree. As you navigate around using the menus, you'll notice that the 'active' menu is always marked using bold face. This is also built-in functionality.

That concludes our scrutiny of the first row of the outer table. The second row is processed in a jiffy, it looks like this: '<td colspan="2"><EPiServer: Clear height="10" runat="server" /></td>'. It is no more than a single table cell spanning two columns containing a transparent GIF picture ten pixels high and one pixel wide. So, let's take a look at the third and last row of the outer table, the one that Page Template can insert its own functions in, since it contains defined Region.

Use of Regions in DefaultFramework.ascx by Default.aspx Page Template

Page Template Default.aspx uses the regions menuRegion, mainRegion and rightListingRegion from DefaultFramework.ascx; see figure 4-11 on page 84.

News Items Go in the Left-Most Area, Region menuRegion

The menuRegion is used to house a news items list by exchanging a NewsList for the default contents of menuRegion. NewsList is a class found in EPiServer .WebControls which is declared near the top of Default.aspx.

Example 4-10: Declaration of EPiServer.WebControls in Default.aspx.

```
<%@ Register TagPrefix="EPiServer" Namespace="EPiServer.WebControls" Assembly="EPiServer" %>
```

With this declaration in place, all the developer has to do to use any of the classes in EPiServer.WebControls is to insert a new tag, such as this for NewsList found in Default.aspx:

Example 4-11: Declaration of NewsList in Default.aspx.

```
<EPiServer:Newslist ID=Newslistnew Pagelinkproperty="NewsContainer" runat="Server"
    MaxCount="<%# GetNewsCount() %>">
```

To use the NewsList class you look up its documentation in the EPiServer SDK Help File, where you'll find that NewsList uses several pre-defined templates names, HeaderTemplate, FirstNewsTemplate, NewsTemplate, FooterTemplate, and more. This enables you to use separate visual elements together with each of the first four news items, news following the first four, the header and a footer. If you'd like to handle all news items the same, then specify NewsTemplate. Here's the NewsTemplate from Default.aspx:

Example 4-12: NewsList.NewsTemplate specification in Default.aspx.

```
<Newstemplate>
    <tr>
        <td class="DateListingText"><%# Container.CurrentPage[ "PageStartPublish" ] %></td>
    </tr>
    <tr>
        <td><a href="<%# Container.CurrentPage.LinkURL %>" class="StartPageHeading">
            <%# Container.CurrentPage.PageName %></a> <br />
            <span class="Normal"><%# Container.CurrentPage["MainIntro"] %></td>
        </tr>
        <tr>
            <td><EPiServer:Clear height="6" runat="server" /></td>
        </tr>
</Newstemplate>
```

This NewsTemplate lives inside a table and every news item results in three news rows in the table. The first news item row contains the value of the property

PageStartPublish for the news item (a single news item is defined on a single Web Page, there's one Web Page for every news item). Row number two is an HTML anchor where the link, href, is a link to the full page on which the news item is described and the text part is comprised of two lines: the first line being the name of the Web Page as entered in Edit mode and the second any introduction entered for the news item Web Page. Finally, row number three contains a spacer to make the news item table aesthetically pleasing.

Region mainRegion Gets a Picture, a Heading and Some Text

Moving to the next Region used by Default.aspx, mainRegion, we find that it is the destination for the contents of three values: StartPageImage, MainBodyHeading and PageBody.

Example 4-13: Specification for mainRegion in Default.aspx.

```
<EPiServer:Content Region="mainRegion" runat="server">
<table border="0" cellpadding="0" cellspacing="0"
        xmlns:EPiServer="http://schemas.episerver.com/WebControls">
    <tr>
        <td>
            <table border="0" cellpadding="8" cellspacing="1" bgcolor="#DEDEDE">
                <tr>
                    <td bgcolor="#ffffff">
                        <img src="<%= StartPageImage %>" border="0" />
                    </td>
                </tr>
            </table>
        </td>
    </tr>
    <tr>
        <td>
            <table border="0" cellpadding="0" cellspacing="10">
                <tr>
                    <td background="images/L_triangleBG.gif" height="15">
                        <EPiServer:Property PropertyName="MainBodyHeading"
                            DisplayMissingMessage="false" class="ListHeading"
                            runat="server" /> 
                    </td>
                </tr>
                <tr><td><development:PageBody id=pagebody runat="server" /></td></tr>
                <tr><td bgcolor="#dedede"><EPiServer:Clear runat="server" /></td></tr>
            </table>
        </td>
    </tr>
```

```
</table>
</EPiServer:Content>
```

The way these values are used is quite interesting, as we'll see. Somewhere there's a connection for all three with Properties defined in EPiServer Admin mode and given values in EPiServer Edit mode, but the way to get at their values differs.

The first Property, StartPageImage, is not found in the Page Template Start page. As can be seen in listing 4-14, it is used as a property and we're dealing with an property function from the code-behind file, Default.aspx.cs.

Example 4-14: Extract from Default.aspx.cs, the code-behind file.

```
private readonly string _startPageImage = "images/startpage_image.jpg";
...
protected string StartPageImage {
    get {
        if ( CurrentPage[ "MainImage" ] != null ) {
            return (string) CurrentPage[ "MainImage" ];
        }
        return _startPageImage;
    }
}
```

The code for StartPageImage reveals what's happening here: if the static Property MainImage has been given a value for Start page this will be used, otherwise the image path in _startPageImage will be returned.

Next we look at the Property MainBodyHeading, the second found in Default.aspx. It is dealt with in short order. The class EPiServer.WebControls.Property is used to simply insert the contents of MainBodyHeading at the current location in the HTML page.

Third and last is PageBody, which is a User Control (PageBody.ascx).

Example 4-15: Registration of User Control PageBody in Default.aspx.

```
<%@ Register TagPrefix="development" TagName="PageBody "Src="…/Units/PageBody.ascx"%>
```

PageBody.ascx is anything but complicated. Its code-behind file has no real contents and its HTML component looks like this:

Example 4-16: The HTML part of User Control PageBody.ascx.

```
<p><EPiServer:Property id="PageBody" runat="server" PropertyName="MainBody" /></p>
```

As you can see, the static Property MainBody is used in PageBody.ascx.

Summarising the three Properties and the various retrieval methods used gives us table 4-13:

Table 4-13: Retrieval methods used for three static Properties in Page Template Start.

Property Name	*Value Retrieval Method*
MainImage	Property function in code-behind file to make it easy and clean to use a default image.
MainBodyHeading	Simplest possible method used: EPiServer.WebControls.Property.
MainBody	User Control wraps the Property. Common and very efficient method to standardise and re-use functionality.

Page Template Default.aspx uses only these two Regions in DefaultFramework .ascx.

5
Avoiding Errors, Testing and Debugging

Many human languages have a saying along the lines of the English 'An ounce of prevention is worth a pound of cure'. Funnily enough, not many computer language constructors have given much thought to this. Some alleviance comes from computer languages that force developers to clearly state their intentions and declare variables before use and have constructs such as exceptions to help provide some structure to the eternal quest for structured error handling.

In this regard, C#, in its Visual C# .NET incarnation, is a comprehensive third-generation computer language that offers a sound development platform with good constructs and that firstly helps minimise problems and secondly assists in finding them. Its primary development environment, Visual Studio .NET, has a powerful built-in source code debugger whose use we'll be exploring to some extent in this chapter.

Before looking at debugging as such, we'll provide you with some insights into both ASP.NET and EPiServer solution development that we have accumulated.

Separate Presentation and Data

The now rather old adage of separating presentation and data also holds true for ASP.NET and EPiServer solution development. If the litmus test for separation of presentation and data in the n-tier world is 'no SQL on the client', the equivalent in the ASP.NET world would be 'no Response.Write on the server'. Those familiar with ASP development recognise Response.Write as the method to write text, including HTML text, to the client.

Instead, what we propose is the EPiServer Content Framework. It provides for and encourages separation of presentation and data. We believe you should start your development effort in designing the Web site visual layout and then add the functionality needed to achieve that layout. One excellent piece of this puzzle is ASP.NET templated controls in general and the EPiServer templated controls in particular.

ASP.NET Templated Controls Has Built-In Separation of Presentation and Data

ASP.NET Templated Controls represent an elegant way to separate presentation and data, whilst keeping some connection between the two. In short, templated controls offer HTML template tags with given names, each control class may de-

fine their own names, which are handled by code in the control's class. ASP.NET has its own set of templated controls, e.g. DataList and Repeater.

Most of the classes in EPiServer.WebControls are templated controls. Among the templated control classes in EPiServer WebControls we find:

- Calendar, for displaying calendar events
- Newslist, for displaying news items
- PageList, for displaying a list of pages
- PageTree, for displaying a list of pages as a tree

Express Your Intent Clearly in Code, Comment When You Must

This is not a book on coding style but we have found that most of us find that the statement 'the code is the comment' holds true. The way we interpret the statement is that the code should be so clear and obvious that the need for comments isn't very great. Many are the times that we have improved the readability of our own code simply by taking a group of source code lines and making them into a function whose name conveys the intent of the code better than the source code lines themselves.

There are a couple of good ways to know if you need to improve your skills in this area: ask a colleague to read your source code and relate its purpose to you and also try reading some of your own source code which you haven't touched for a few months.

Testing Equals Module Testing

Make Tests Easy, Easy to Interpret and Self-Documenting

A very nice spin-off effect of separating presentation and data is that it enables separation of tests for data handling code and tests for presentation code. Here we'll concentrate on tests for data handling code.

Have Your Code Write Data to Files, Compare Files between Versions of the Code

In his excellent book *Refactoring: Improving the Design of Existing Software*, Martin Fowler suggests an elegant yet powerful testing method: have the code write data to files and compare new files with historical ones. Mr. Fowler himself uses a text comparison tool whose output is restricted to 'OK' if the contents of two text files are the same. We would like to add a few suggestions to this method, as we feel that a direct comparison of file contents is too restrictive and precludes self-documenting output files. Instead, we suggest that all output be XML formatted and a tool used which allows you to exclude XML elements, and child elements, from the comparison.

We have such a tool, XMLComp.Exe, freely available at our Web site, www.episerver.com. In honour of Martin Fowler, the tool outputs 'OK!' and '!OK', respectively (interpretation obvious).

When Bugs Are Reported Start by Expanding the Test Suite

It's very important that you keep your test suites, such as they are, in sync with the solutions you develop. An important part of this is to treat every bug report in the same way and always start by expanding the existing test suite so it will also capture the reported bug. Only when the test suite itself is debugged and the newly reported bug incorporated into it is it time to correct the erring code.

Common Problems in ASP.NET Development

The fact that EPiServer itself and the solutions you develop for it are ASP.NET applications gives common ground as regards avoiding and handling errors. The most common problem for EPiServer developers has nothing to do with either ASP.NET or EPiServer: it concerns file and folder access permissions. Most application developers shy away from access permissions and do not wish to have any dealings with them whatsoever.

Nevertheless, access permissions are almost always a fact of life for the applications that are developed. For ASP.NET applications, many access permissions problems can be traced back to incorrect, excessively restrictive access permissions for the ASPNET account, the user account used by ASP.NET.

As a part of an effort to track down a problem, there a few things you can do regarding the ASPNET account:

- Make the ASPNET account member of the local Administrators group account
- Change access permissions for the \INetPub folder tree to include Change, or even Full Control, for ASPNET

It is important that you find the problem and then restore the group memberships and access permissions – you don't want to deploy your solution with too broad a privilege set for the ASPNET account.

Useful Tools

No doubt you have your own set of useful tools. We just want to make sure that you are aware of some excellent tools from the same source: the SysInternals Web site hosted by Mark Russinovich and Bryce Cogswell. Mark Russinovich has earned a reputation for knowing just about everything about the Windows family of operating systems. He has co-authored 'Inside Windows 2000' together with Dave Solomon.

There are three tools at SysInternals that you might want to take a look at:

- DebugView, shows output from DbgPrint and OutputDebugString
- FileMon, shows file access activity
- RegMon, show Registry access activity

All three tools function like a 'tape' recorder recording debug output, file and folder access and Registry key and value access, respectively. You have the option to start and stop the recording, filter the output and even save it for further study.

DebugView

DebugView shows debug output in a window, in the words of Mark and Bryce themselves: 'This program intercepts calls made to DbgPrint by device drivers and OutputDebugString made by Win32 programs. It allows for viewing and recording of debug session output on your local machine or across the Internet without an active debugger.'

Calls to System.Diagnostics.Debug.WriteLine are displayed by DebugView, as they end up as calls to OutputDebugString.

We do not recommend that you leave Debug output calls in your code when you've reached the production phase, as there is a small performance penalty, but in the testing and pre-production phase you will probably find DebugView quite useful.

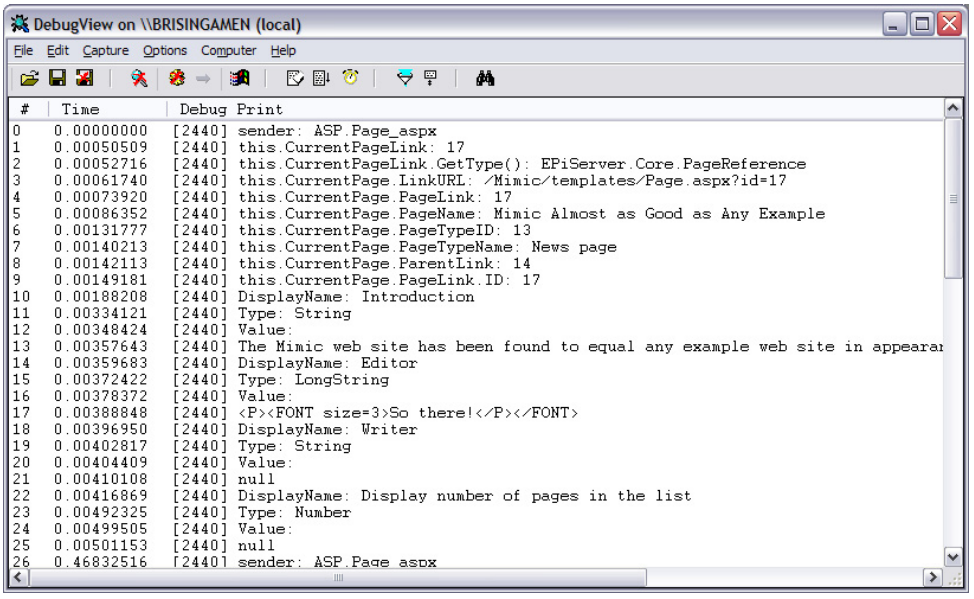
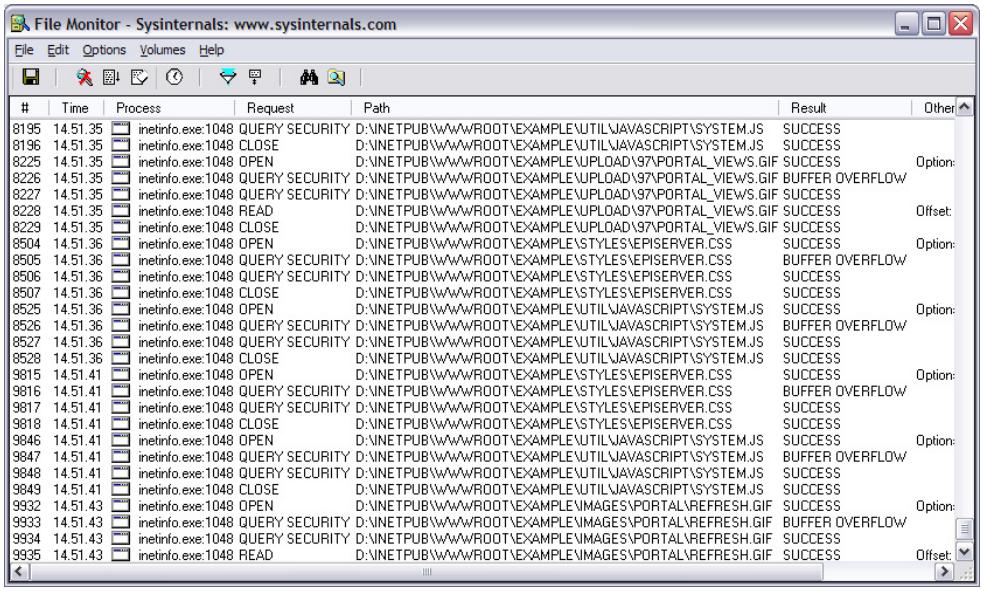


Figure 5-1: DebugView at work.

FileMon

File Monitor, FileMon, records and lets you analyse file and folder access. This introduction to FileMon is taken from the SysInternals Web site:

'FileMon monitors and displays file system activity on a system in real-time. Its advanced capabilities make it a powerful tool for exploring the way Windows works, seeing how applications use the files and DLLs, or tracking down problems in system or application file configurations. FileMon's timestamping feature will show you precisely when every open, read, write or delete occurs, and its status column tells you the outcome. FileMon is so easy to use that you'll be an expert within minutes. It begins monitoring when you start it, and its output window can be saved to a file for off-line viewing. It has full search capability, and if you find that you're getting information overload, simply set up one or more filters'.



#	Time	Process	Request	Path	Result	Other
8195	14.51.35	inetinfo.exe:1048	QUERY SECURITY	D:\INETPUB\WWWROOT\EXAMPLE\UTIL\JAVASCRIPT\SYSTEM.JS	SUCCESS	
8196	14.51.35	inetinfo.exe:1048	CLOSE	D:\INETPUB\WWWROOT\EXAMPLE\UTIL\JAVASCRIPT\SYSTEM.JS	SUCCESS	
8225	14.51.35	inetinfo.exe:1048	OPEN	D:\INETPUB\WWWROOT\EXAMPLE\UPLOAD\97\PORTAL_VIEWS.GIF	SUCCESS	Option:
8226	14.51.35	inetinfo.exe:1048	QUERY SECURITY	D:\INETPUB\WWWROOT\EXAMPLE\UPLOAD\97\PORTAL_VIEWS.GIF	BUFFER OVERFLOW	
8227	14.51.35	inetinfo.exe:1048	QUERY SECURITY	D:\INETPUB\WWWROOT\EXAMPLE\UPLOAD\97\PORTAL_VIEWS.GIF	SUCCESS	
8228	14.51.35	inetinfo.exe:1048	READ	D:\INETPUB\WWWROOT\EXAMPLE\UPLOAD\97\PORTAL_VIEWS.GIF	SUCCESS	Offset:
8229	14.51.35	inetinfo.exe:1048	CLOSE	D:\INETPUB\WWWROOT\EXAMPLE\UPLOAD\97\PORTAL_VIEWS.GIF	SUCCESS	
8504	14.51.36	inetinfo.exe:1048	OPEN	D:\INETPUB\WWWROOT\EXAMPLE\STYLES\EPISERVER.CSS	SUCCESS	Option:
8505	14.51.36	inetinfo.exe:1048	QUERY SECURITY	D:\INETPUB\WWWROOT\EXAMPLE\STYLES\EPISERVER.CSS	BUFFER OVERFLOW	
8506	14.51.36	inetinfo.exe:1048	QUERY SECURITY	D:\INETPUB\WWWROOT\EXAMPLE\STYLES\EPISERVER.CSS	SUCCESS	
8507	14.51.36	inetinfo.exe:1048	CLOSE	D:\INETPUB\WWWROOT\EXAMPLE\STYLES\EPISERVER.CSS	SUCCESS	
8525	14.51.36	inetinfo.exe:1048	OPEN	D:\INETPUB\WWWROOT\EXAMPLE\UTIL\JAVASCRIPT\SYSTEM.JS	SUCCESS	Option:
8526	14.51.36	inetinfo.exe:1048	QUERY SECURITY	D:\INETPUB\WWWROOT\EXAMPLE\UTIL\JAVASCRIPT\SYSTEM.JS	BUFFER OVERFLOW	
8527	14.51.36	inetinfo.exe:1048	QUERY SECURITY	D:\INETPUB\WWWROOT\EXAMPLE\UTIL\JAVASCRIPT\SYSTEM.JS	SUCCESS	
8528	14.51.36	inetinfo.exe:1048	CLOSE	D:\INETPUB\WWWROOT\EXAMPLE\UTIL\JAVASCRIPT\SYSTEM.JS	SUCCESS	
9815	14.51.41	inetinfo.exe:1048	OPEN	D:\INETPUB\WWWROOT\EXAMPLE\STYLES\EPISERVER.CSS	SUCCESS	Option:
9816	14.51.41	inetinfo.exe:1048	QUERY SECURITY	D:\INETPUB\WWWROOT\EXAMPLE\STYLES\EPISERVER.CSS	BUFFER OVERFLOW	
9817	14.51.41	inetinfo.exe:1048	QUERY SECURITY	D:\INETPUB\WWWROOT\EXAMPLE\STYLES\EPISERVER.CSS	SUCCESS	
9818	14.51.41	inetinfo.exe:1048	CLOSE	D:\INETPUB\WWWROOT\EXAMPLE\STYLES\EPISERVER.CSS	SUCCESS	
9846	14.51.41	inetinfo.exe:1048	OPEN	D:\INETPUB\WWWROOT\EXAMPLE\UTIL\JAVASCRIPT\SYSTEM.JS	SUCCESS	Option:
9847	14.51.41	inetinfo.exe:1048	QUERY SECURITY	D:\INETPUB\WWWROOT\EXAMPLE\UTIL\JAVASCRIPT\SYSTEM.JS	BUFFER OVERFLOW	
9848	14.51.41	inetinfo.exe:1048	QUERY SECURITY	D:\INETPUB\WWWROOT\EXAMPLE\UTIL\JAVASCRIPT\SYSTEM.JS	SUCCESS	
9849	14.51.41	inetinfo.exe:1048	CLOSE	D:\INETPUB\WWWROOT\EXAMPLE\UTIL\JAVASCRIPT\SYSTEM.JS	SUCCESS	
9932	14.51.43	inetinfo.exe:1048	OPEN	D:\INETPUB\WWWROOT\EXAMPLE\IMAGES\PORTAL\REFRESH.GIF	SUCCESS	Option:
9933	14.51.43	inetinfo.exe:1048	QUERY SECURITY	D:\INETPUB\WWWROOT\EXAMPLE\IMAGES\PORTAL\REFRESH.GIF	BUFFER OVERFLOW	
9934	14.51.43	inetinfo.exe:1048	QUERY SECURITY	D:\INETPUB\WWWROOT\EXAMPLE\IMAGES\PORTAL\REFRESH.GIF	SUCCESS	
9935	14.51.43	inetinfo.exe:1048	READ	D:\INETPUB\WWWROOT\EXAMPLE\IMAGES\PORTAL\REFRESH.GIF	SUCCESS	Offset:

Figure 5-2: FileMon.Exe at work.

The session recorded by FileMon when capturing the picture in figure 5-2 was of course an EPiServer solution. As the file paths show this was recorded during a session with the example Web site. The requests labelled Query Security reveal a pattern which you'll encounter a lot when using FileMon and RegMon. This pattern consists of calling some Win32 functions twice; the first call is just made to determine how much data we can expect back, it will always be expected to fail; then a big enough buffer is allocated and lastly the second call is made. You might wonder why Query Security requests are made in the first place? Well, apparently all the file access calls are made by INetInfo.Exe using its own logon account, often the System (a.k.a. 'LocalSystem') account. The Query Security requests are made to establish whether the actual Web viewer has access to this particular page. If she doesn't have access, the Web Server may return HTTP error 403 (Forbidden: Access is denied) or a similar diagnostic code and a corresponding Web page.

RegMon

Being the most 'secret' and enigmatic part of Windows, any tool that helps unravel the mysteries of the Registry is precious. This certainly applies to RegMon, not only for its comprehensive feature set but also for the fact that the source code used to be published on the SysInternals Web site. Unfortunately, some of the source code for the SysInternals tools found the way into malware, which has led Mark and Bryce to stop publishing the source code for most tools.

RegMon works like a Registry access recorder. It records information about every access to the Registry including time, process name, process ID, type of request, Registry path accessed, result and 'other' data. When you're sure that RegMon has recorded the pertinent information, you stop the recording and start analysing.

You use RegMon to find out which Registry keys your EPiServer solution accesses and whether that access was successful. For each unsuccessful access, you use RegEdit to inspect, and perhaps change, the access permissions for the Registry key. In the words of its creators:

'Regmon is a Registry monitoring utility that will show you which applications are accessing your Registry, which keys they are accessing, and the Registry data that they are reading and writing – all in real-time. This advanced utility takes you one step beyond what static Registry tools can do, to let you see and understand exactly how programs use the Registry. With static tools you might be able to see which Registry values and keys have changed. With Regmon you'll see how the values and keys changed.'

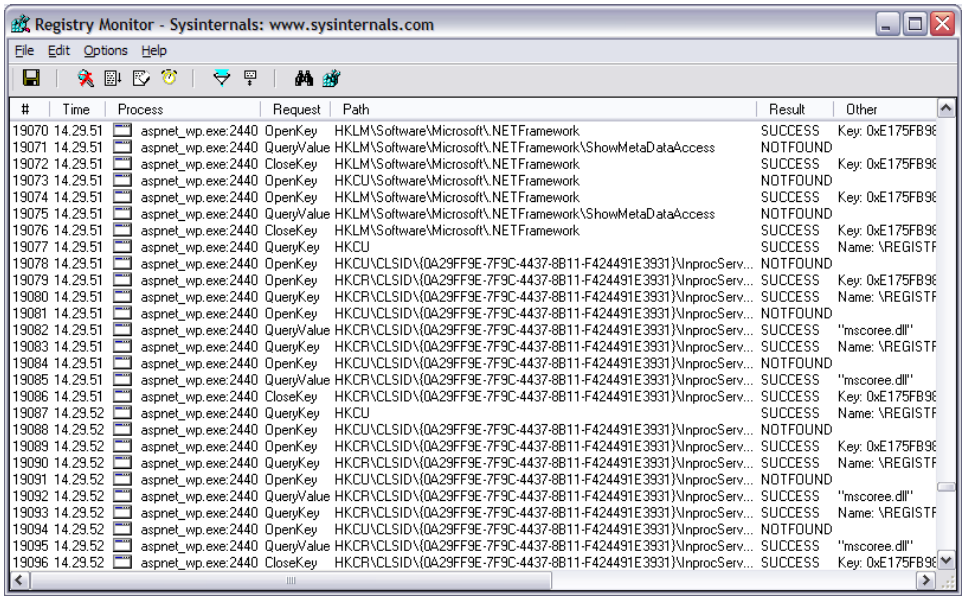


Figure 5-3: RegMon.Exe at work.

Figure 5-3 shows a screen dump from a RegMon session. This particular session was recorded during the start of an EPiServer application. The extract shown in the picture reveals that the process AspNet_WP.Exe performs some Registry accesses. This is to be expected for any ASP.NET application running on Windows 2000 and Windows XP. On Windows Server 2003, the ASP.NET worker process is instead called W3WP.Exe. AspNet_WP.Exe and W3WP.Exe are the ASP .NET worker processes for their respective Internet Information Services, IIS, versions.

The Importance of Knowledge and Experience

The importance of choosing the right tools, methods and algorithms for software development cannot be over-emphasized. Whether this ability comes from formal studies or extensive experience is not really relevant as long as you keep abreast of new insights and knowledge. For example, a relatively easy coding technique as recursion can save both time and effort and help you produce succinct and readable code, yet there are developers who do not feel comfortable using recursion.

As for Microsoft .NET, ASP.NET and C# .NET, there seem to be more books published than there are days in a year, so a lack of material is not a problem for ASP.NET developers. We'll provide a short list of books that have been important to us at the end of this one (see page 323).

Microsoft .NET Framework and Visual C# .NET

Boxing Is Very Popular

Everything in Microsoft .NET Framework is an object, or can be treated as an object. This means we can write code that looks downright strange, such as the one in example 5-1.

Example 5-1: Effects of treating a simple variable as an object in Microsoft .NET.

```
Console.WriteLine( 1.ToString() );                // Ouputs '1' to the console.
Console.WriteLine( ( 1 + 1 ).ToString() );        // Ouputs '2' to the console.
Console.WriteLine( "Hello world".ToString() );    // Outputs 'Hello world' to the console.
```

The reason that we can use the ToString method in the sample above is that this method is declared for System.Object and System.Object is the mother (root) class for every other class in Microsoft .NET Framework. Even simple data types such as integers and booleans are classes in Microsoft .NET Framework and thus inherit from System.Object. Calling the method ToString for what is obviously a string constant, "Hello world", is indeed overkill, but nonetheless perfectly legal in Microsoft .NET Framework.

For performance reasons, variables of simple data types aren't actually instantiated objects until they are treated as such. In this example:

Example 5-2: Variables of simple data types behave as objects when treated as such.

```
int IntVar1  = 1;
int IntVar2  = 2;
int IntVar3  = IntVar1 + IntVar2;
Console.WriteLine( IntVar3.ToString() );
```

All three variables are simple data types, allocated on the stack, until the last line of code, where IntVar3 is treated as an object and space for its mirror object is allocated on the heap. This way of handling variables of simple data types like objects is called 'boxing' in the Microsoft .NET Framework context. Be aware that boxing is time consuming and you might need to keep an eye on it.

Use StringBuilder Instead of String, But Not Always

Microsoft has equipped Microsoft .NET Framework with quite an array of capable string handling classes – they're almost as easy to use as in BASIC or even Simula.

There's one big caveat though which has to do with boxing in Microsoft .NET Framework. It's extremely time-consuming to concatenate two String objects! When there's a need to 'fiddle around' with the content of string objects, StringBuilder objects should always be used instead. Of course, if the fiddling around happens on very few occasions during the execution of the code, there will be no noticable performance gain. Let's look at some code samples.

Example 5-3: String handling using String objects.

```
string StringAppendedTo= string.Empty;
for ( int Revs = 0; Revs < 100000; Revs++ ) {
    StringAppendedTo += "a";
}
```

(Data type 'string' in Visual C# .NET and 'String' in Visual Basic .NET are the same as System.String.)

Running the code in example 5-3 takes 95 seconds, i.e. 0.95 milliseconds per lap. Using StringBuilder we see quite different results:

Example 5-4: String handling using StringBuilder objects.

```
System.Text.StringBuilder StringBldrAppendedTo= new System.Text.StringBuilder();
for ( int Revs = 0; Revs < 100000000; Revs++ ) {
    StringBldrAppendedTo.Append( "a" );
}
```

The duration for the code in example 5-4 is 19 seconds, but this code executes 1000 times more laps than the previous code sample! Each lap in the StringBuilder example uses 0.19 **micro**seconds.

So, in this comparison we find that StringBuilder is about 5000 times faster than String in append operations!

It Is Faster to Use Implicit Concatenation than More Calls to Append

If you're really looking for ways to save time with StringBuilder objects, there's an interesting fact to consider. It so happens that if you do an implicit concatenation in the call to StringBuilder.Append, is is faster than using two calls to Append. Look at the code samples below.

Example 5-5: Implicit concatenation in the call to StringBuilder.Append.

```
StringBldrAppendedTo.Append( "a" + "a" );
```

Example 5-6: Two calls to StringBuilder.Append.

```
StringBldrAppendedTo.Append( "a" );
StringBldrAppendedTo.Append( "a" );
```

The code in example 5-5 is about 20% slower than a call to Append without concatenation, but much faster than using two calls such as in example 5-6. Naturally, the code in example 5-6 takes about twice the amount of time compared to a single call to Append.

Debugging Is a Three-Pronged Choice

Debugging ASP.NET applications offer a little bit more than debugging Windows Forms applications in that you can add trace output to your ASP.NET Web Forms. When it comes to debugging C# code, you have a choice of either adding print-out of debug messages and assertions to your code or simply using the built-in debugger in Visual Studio .NET. In many cases, your preferred route is likely to result in a combination of the two.

Tracing in HTML

By adding the Trace directive to an EPiServer Page Template File and setting its contents to true, you enable tracing information to be output at the end of the rendered page.

There are seven categories of information output when using the Trace directive:

- Request Details
- Trace Information
- Control Tree

- Cookies Collection
- Headers Collection
- Querystring Collection
- Server Variables

Example 5-7: Trace directive added to ASP.NET Web Form, i.e. EPiServer Page Template.

```
<%@ Page language="c#" Codebehind="Page.aspx.cs" AutoEventWireup="false"
Inherits="development.Templates.StandardPage" Trace = "true" %>
```

Setting the attribute Trace to true for a single page results in trace information being displayed at the bottom of the HTML page every time it is 'called', but only for this page.

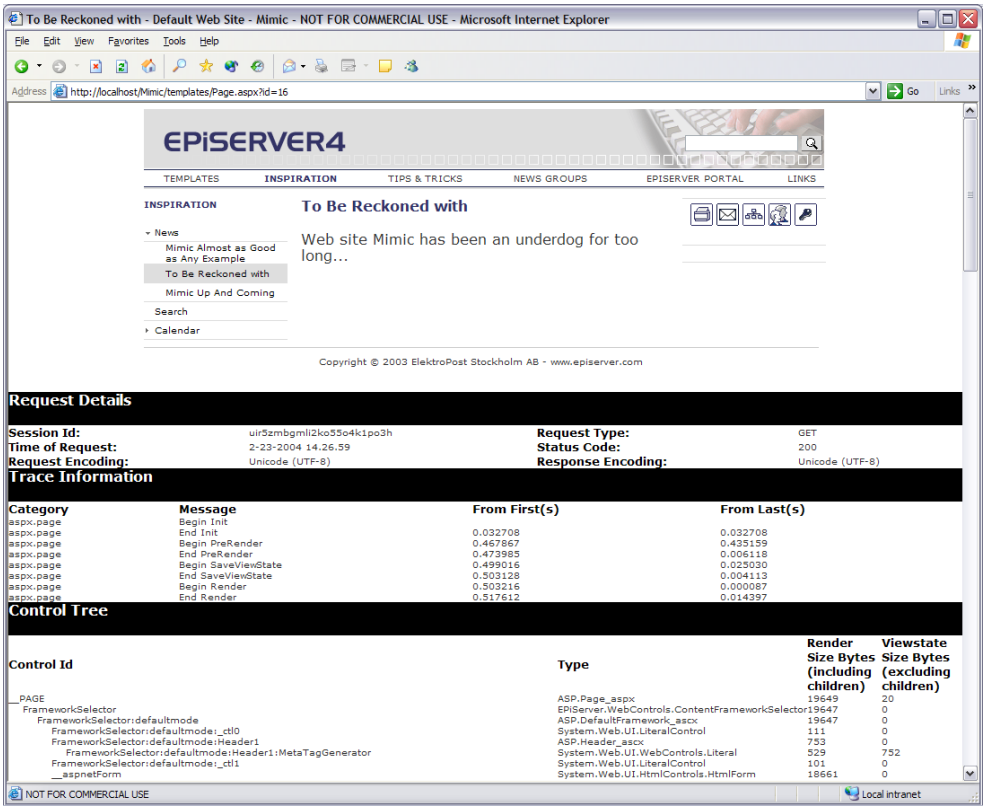


Figure 5-4: Effect of setting page attribute Trace to true.

Among the oceans of information displayed, you will find such information as how the current page was called upon, date and time of call, session ID and a lot more.

Switching on Tracing for the Whole Application

At first when you need tracing, you might want to switch it on for the whole application and not for just a few EPiServer Page Templates. This is done by enabling trace in the application's web.config file.

Example 5-8: Trace directive added to web.config enables tracing for the whole application.

```
<!-- APPLICATION-LEVEL TRACE LOGGING
    Application-level tracing enables trace log output for every page within an application.
    Set trace enabled="true" to enable application trace logging. If pageOutput="true", the
    trace information will be displayed at the bottom of each page. Otherwise, you can view the
    application trace log by browsing the "trace.axd" page from your web application
    root. -->
    <trace enabled="true" requestLimit="10" pageOutput="false" traceMode="SortByTime"
        localOnly="true" />
```

Example 5-8 is taken from an EPiServer application web.config file, only the 'enabled' attribute value has been changed.

Keeping the value for attribute 'localOnly' at 'true' means that only Web browsers started on the same computer that the Web server is executing will we able to use trace.axd to view the tracing output. Letting 'pageOutput' keep its setting of 'false' means that no tracing information is displayed in the Web browser window; it can only be viewed with trace.axd.

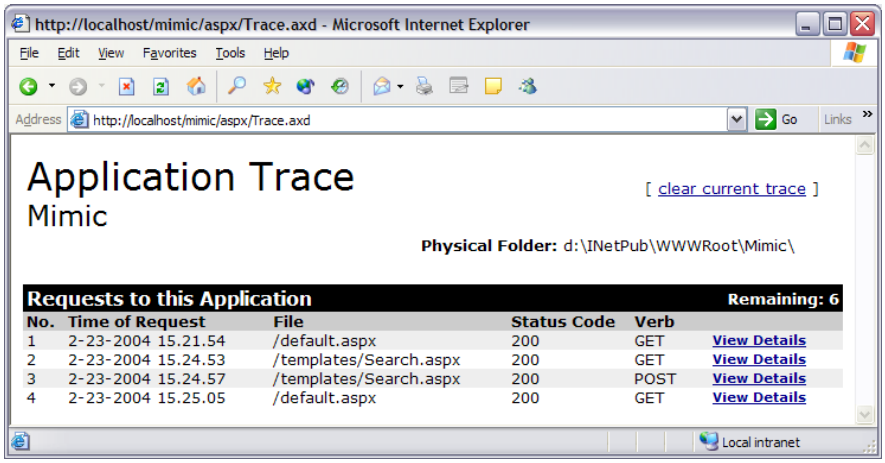


Figure 5-5: Viewing trace information for application Mimic.

Trace.axd is a series of normal HTML files placed in sub-folders to the Temporary Internet Files folder.

Adding Debug Code to Your Code

There is one big advantage and one great drawback to adding debug code to your code. The advantage is that you control every aspect of debugging yourself and

the disadvantage is that you have to add more code, which itself is prone to errors.

All of C#'s debug functions are part of the System.Diagnostics name space meaning that they aren't particular to C# but available in all .NET Framework compatible languages.

System.Diagnostics.Debug

There are primarily five methods that are of interest in the System.Diagnostics.Debug class:

- Assert
- Write and WriteLine
- WriteIf and WriteLineIf

All the properties and methods in this class are shared (Static in Visual Basic .NET), so you never need to instantiate an object to use them.

The Assert function should be an old friend of C developers. It's quite easy to understand and use. It has three overloaded versions, two of which are seen here:

- public static void Assert(*bool*): if the boolean expression *bool* is false, the Call Stack is printed to the Output window
- public static void Assert(*bool*, string): if the boolean expression *bool* is false, the contents of string is printed to the Output window

Now, some would argue that the need for calls to Assert in your code is not as pronounced in C# and Visual Basic .NET compared to C, as all languages that produce managed code have access to the excellent exception handling. You will probably find that both exception handling and calls to Assert have a place in your code.

System.Diagnostics.Debug.Assert

Example 5-9: Use of Assert method to test the IsNull property of variable prop.

```
System.Diagnostics.Debug.Assert( ! prop.IsNull, "Prop is null" );
```

Adding the code from example 5-9 would probably be done in code that handles some aspect of EPiServer Property objects, as IsNull is a standard attribute for the PropertyData (EPiServer.Core.PropertyData) class. When the code in example 5-9 is executed, the message 'Prop is null' will be displayed depending on the outcome of the test '! prop.IsNull'. If it evaluates to true, no message is displayed; otherwise the message is output.

System.Diagnostics.Debug.Write and WriteLine; WriteIf and WriteLineIf

The two methods System.Diagnostics.Debug.Write and WriteLine both output text to the Output window in Visual Studio .NET, the only difference being that method WriteLine appends System.Environment.NewLine to the output which Write doesn't. There's also a variant of both of these: WriteIf and WriteLineIf, which both work a lot like the Assert method.

Example 5-10: Using System.Diagnostics.Debug.Write and WriteLine in code.

```
System.Diagnostics.Debug.Write( "Value: " );
if ( ! prop.IsNull ) {
    System.Diagnostics.Debug.WriteLine( prop.Value.ToString() );
} else {
    System.Diagnostics.Debug.WriteLine( "null" );
}
```

The code in example 5-10 is rather typical of Debug code and also points to the problem of adding extra code to your source code: this code also has to be bug-free and some of it will be active even when we change configuration settings to Release. The only way to 'fix' the latter problem is to add more code and surround the code with '#if (DEBUG)' and '#endif'.

Debug Output Can Be Effortlessly Passed to a File

The output of both Assert, WriteLine and WriteLineIf is directed to the Output window of Visual Studio .NET. This is quite handy, since they won't clutter your Web pages nor will there be a series of error popups to handle. It's very easy to have debug output saved in a file, should you want to. Functions to accomplish this are readily available in Visual Studio .NET as all debug output functions write to a collection of TraceListeners which monitor all debug output. The code sample below adds a new TextWriterTraceListener to the Debug output listeners.

Example 5-11: Adding a Debug output TraceListener.

```
using System.Diagnostics;
...
TextWriterTraceListener DbgOut = new TextWriterTraceListener( "d:\\MimicDebug.Txt" );
Debug.Listeners.Add( DbgOut );
...
Debug.WriteLine( this.CurrentPageLink.ToString() );
...
Debug.Assert( ! prop.IsNull, "Prop is null" );
...
DbgOut.Close();
```

The code in example 5-11 will result in debug output being written both to the Output window in Visual Studio .NET and to the file d:\MimicDebug.Txt.

Conditional Compilation

C and Visual Basic developers alike have had access to conditional compilation, albeit not for very many Visual Basic versions. For Visual Studio .NET compatible languages there are additional improvements to conditional compilations which make it easy to use.

By convention, all Visual Studio .NET languages utilise two conditional compilation constants: Debug and Trace. Using conditional compilation constants is a little bit different in Visual Basic .NET and C#. For the former, the constants are Boolean, which determines their use. In the case of C#, it's true to its C inheritance so here we look at whether or not the constant exists and don't bother at all with its content (strictly speaking conditional compilation constants in C and C# do not have any contents).

For C# the default Debug settings are that both Debug and Trace are defined and in the Release settings only Trace is defined. Another C tradition that C# is true to is the rather non-sensical one of spelling all constants using only upper case, so in code be sure to write 'DEBUG'.

Example 5-12: Using conditional compilation constant Debug.

```
#if ( DEBUG )
    System.Diagnostics.Debug.WriteLine( "DEBUG is defined!" );
#endif
```

The code in example 5-12 doesn't represent the most intelligent use of conditional compiling, of more interest are the possibilities allowed by the Conditional attribute for functions.

The Conditional Attribute for Functions

For managed code, there's a very elegant way to utilise conditional compilation. The function attribute Conditional (System.Diagnostics.Conditional) acts as if a function declaration, and all calls to it, were surrounded by '#if (DEBUG)' and '#endif'.

Example 5-13: Using the function attribute Conditional.

```
[ System.Diagnostics.Conditional( "DEBUG" ) ]
private void Verbose_Page_Load( object sender, System.EventArgs e ) {
...
}
...
Verbose_Page_Load( sender, e );
...
Verbose_Page_Load( Sender, Evts );
```

Suppose that your code features the lines from example 5-13. As long as the conditional compilation constant DEBUG is defined, the function Verbose_Page

_Load is defined and all calls to it executed. Switching to Release settings would usually un-define DEBUG and thus the declaration of Verbose_Page_Load is removed along all calls to it, of course without doing anything with, or to, your source code.

The obvious benefit for developers is that we don't have to litter our code with '#if (DEBUG)' and '#endif' around every function declaration and each call to them. Please note the small inconsistency here; for the Conditional attribute you must use double quotes around the name of the constant, something you should not do inside '#if' statements.

Debugging with Visual Studio .NET

The debugging capabilities of Visual Studio .NET (and later versions) are pretty much what you'd expect from a modern software construction tool. With the built-in debugger you can:

- Set static Break points in source code
- Set dynamic, conditional, Break points in source code
- Single-step through code using one of three ways:
  - Step Into next line of code
  - Step Over (execute current function call and stop on the next line of code)
  - Step Out (execute rest of current function's code and stop on the first line after function call)
- Move the point of execution
- Add Watch expressions
- Let the debugger handle Exceptions instead of code or handle Exceptions that are not handled in code
- Access the Call Stack (lets you see which execution route led to the present state)
- Access the Command Window

Debugging ASP.NET applications is somewhat different to debugging Windows Forms applications, since ASP.NET applications are executed by the process ASPNET_WP.Exe (W3WP.Exe in Windows Server 2003). When you start debugging an ASP.NET session in Visual Studio .NET, it silently attaches to the ASPNET_WP.Exe/W3WP.Exe process.

Example 5-14: Message from Visual Studio .NET when attaching to ASPNET_WP.Exe for debugging purposes.

```
Auto-attach to process '[4052] aspnet_wp.exe' on machine 'BRISINGAMEN' succeeded.
```

Debugging a Live EPiServer Application

It's possible to debug an EPiServer application that is already running by attaching to it. Ideally, you would have access to the application's source code and have it loaded into Visual Studio .NET, basically prepared for an ordinary debugging session.

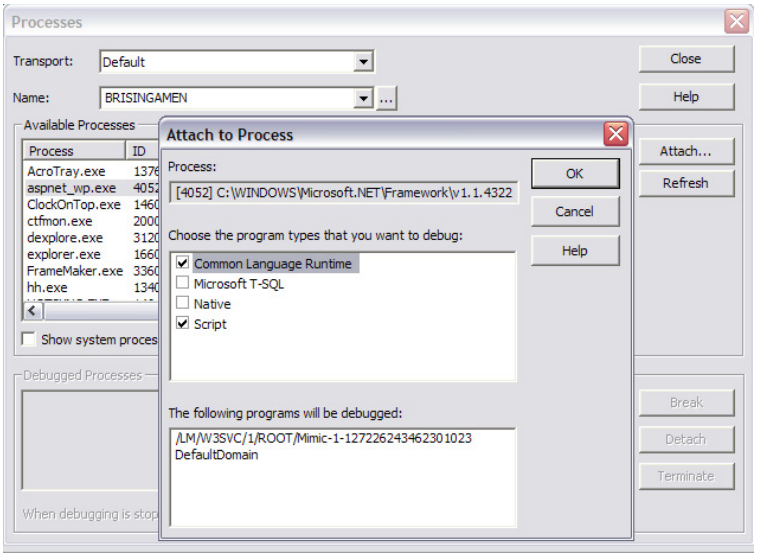


Figure 5-6: Attaching to a running process from the Visual Studio .NET debugger.

To attach to ASPNET_WP.Exe while it's running, open the Debug menu in Visual Studio .NET

Break Points

Break points are a line of source code with a special marking so that when executing the application in the Visual Studio .NET debugger execution will halt when it reaches a line of source code with a break point on it.

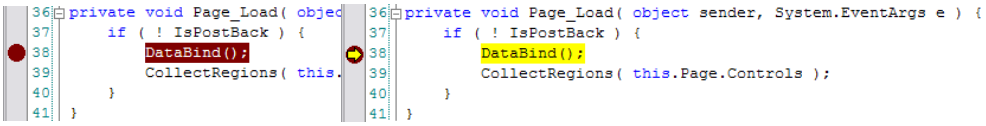


Figure 5-7: A source code line with a break point set, on the left, and appearance when execution halts at the line, on the right.

In figure 5-7, you can see the same source code lines, on the left when a break point has been set on it and on the right when execution has been first started and then halted at the break point.

Break points have several advantages, but one of them is very appealing: you do not have to touch the source code to use them. We all know that source code should be kept to a minimum since there is a relation between lines of source code and number of bugs introduced. So, anything that helps us avoiding expanding the source code should be welcome.

Also, Visual Studio .NET remembers break point settings between sessions, so when you finally have the ideal set of static and dynamic break points at four in the morning, you can save your work, go to bed, and return later and have the same settings still being current.

There are two types of break points: static and dynamic break points.

Static Break Points

Using static break points is as easy as clicking on a line of code, starting the application and then waiting for the execution to reach the line of code with the break point and subsequently stop and allow the debugger's other facilities to be used, such as single-stepping, QuickWatch and Watch points. With static break points, execution is halted every time it reaches the source code line with the break point on it.

To set a break point, either click to the left of the line number (the line number display setting is located at Tool.Options.Text Editor.C#.General or Basic.General) in the left margin or press F9 (when profile set to 'Visual Studio Developer') with the cursor on the line in question. Since this actually toggles a break point on and off, do the same again to unset it.

Dynamic, Conditional, Break Points

The other group of break points allows more control than simply breaking into the debugger every time a particular line of source code is about to be executed. Dynamic break points only halt execution at a break point if an additional condition is met. When developing with EPiServer, you typically use an ASP.NET Web Form to create a few EPiServer Page Types, which in turn are used to create several, or even a lot of, EPiServer Web Pages. Now, break points are set in source code, meaning either the code-behind file for the ASP.NET Web Form or the code-behind file for an ASP.NET User Control, be it a Framework Definition File or another type of control. How do we know which Web Page was 'executing' when the break point was hit? Simple, don't use static break points but conditional ones and use the ubiquitous property PageLinkURL to decide when to actually stop execution. The only problem with this scenario is that Visual Studio .NET 2003 and earlier do not permit what is called 'Data Breakpoints'. But the next best thing is available: Conditional Break Points.

To add a condition to a break point, you can create a break point by clicking on a line of source code and then adding the condition by clicking on the line with the break point and choosing Breakpoint Properties from the shortcut menu. It is also possible to first select the line of source code and then create the break point by choosing Debug.New Breakpoint (Ctrl+B). Create a File location break point. Next click on Condition and enter a condition of your choice. An excellent choice if you wish to stop only when the code is executed for a certain Web Page is 'CurrentPage.Page.ID'. This is an integer attribute so you could enter a condition such as this:

Example 5-15: Condition for break point to stop execution when CurrentPage.Link.ID equals 17.

```
CurrentPage.PageLink.ID == 17
```

Remember to use C# syntax where '==' means equals and '=' means assignment. Should the condition you wish to use include non-integer data, you might be able to convert them to strings and use a condition like this one:

Example 5-16: Condition for break point to stop execution when CurrentPageLink equals "17".

```
string.Equals( this.CurrentPageLink.ToString(), "17" )
```

The condition in example 5-16 stops execution for the current break point only when the property this.CurrentPageLink equals "17". According the EPiServer 4 Software Development Kit Help File, EPiServer4SDK.chm, CurrentPageLink has a data type of PageReference but converting it to a string is OK. You might be able to do more interesting conversions but that hinges on whether or not the class in question implements the System.IConvertible interface. As this interface is not compatible with Microsoft's Common Language Specification, we suggest you try to make do with ToString.

It's also possible to use Hit Count with dynamic break points, not the most useful in EPiServer context, but there when we need it.

Single-Stepping in Code

Single-stepping through source code in Visual Studio .NET is very useful. It lets you use your own eyes to see every line of code executed and inspect the contents of every variable along the execution path.

Step Into

Step Into is classic single-stepping; you follow the execution path wherever it takes you. This means into functions, methods and property functions (for objects) as they are called, provided their source code is available. If Visual Studio .NET does not have access to the source code for a function, the call will be executed without stepping; for this particular call Step Into will behave exactly like Step Over.

If you chose the profile 'Visual Studio Developer' the keyboard shortcut for Step Into i F11.

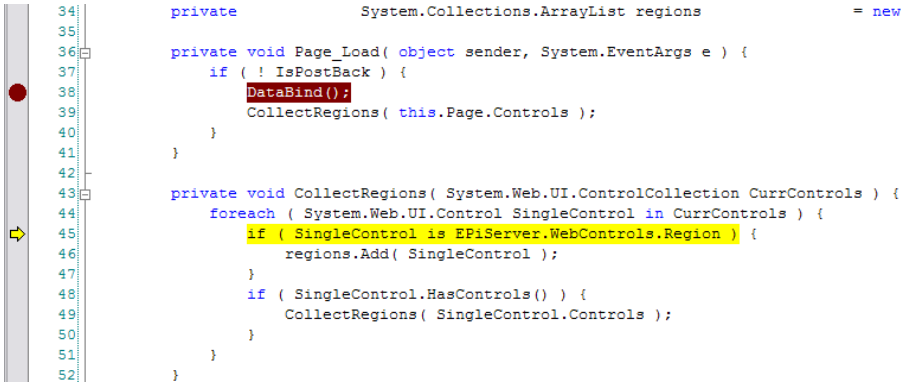


Figure 5-8: After selecting Step Into from the break point at line 38, execution is currently inside the function CollectRegions.

Single-stepping from a break point is shown in figure 5-8. Execution was first halted at line 38, 'DataBind();', and then Step Into was selected a number of times which resulted in single-stepping into the function CollectRegions, which is called at line 39 in the source code. Notice that since VS .NET doesn't have access to the source code for the function DataBind we could have selected either Step Into or Step Over for the first step after execution halted at line 38.

Step Over (Step/Execute Call)

Naming this function 'Step Over' cannot have been a natural choice since it means single-stepping except when the line to be executed is a function call (in the broad sense) which is executed with no stepping, and the execution is again halted on the source code line after the function call (strictly speaking it's not the first line of source code after the current line, rather it's the first line of source

code that is in turn to be executed). Another choice for its name might have been 'Step/Execute Call'.

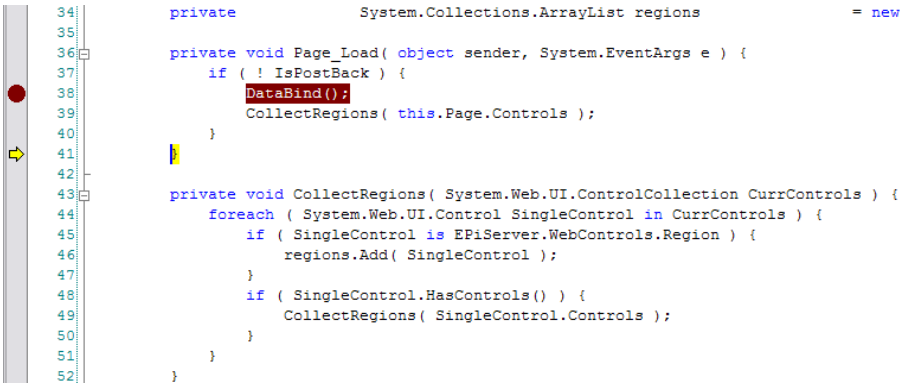


Figure 5-9: Effect of using Step Over twice after execution halted on line 38.

The picture in figure 5-9 was taken after execution was halted at line 38 and the Step Over was selected twice. The current line, to be executed, is line 41, which is the closing brace for function Page_Load.

Key F10 would be the keyboard shortcut for Step Over with profile 'Visual Studio Developer' chosen.

Step Out (Finish up Here and Return to Caller)

Step Out is very handy since it means 'Finish up here and return to caller'. In other words, if you've single-stepped into a function and wish you hadn't simply choose Step Out and you're returned to the source code line after the call to the function. To envisage this, we can use both figures 5-8 and 5-9. Imagine that you've single-stepped into the function CollectRegions and the current execution state looks like the one in figure 5-8. Now instead of single-stepping through every recursive call to CollectRegions you choose Step Out. This will immediately return to the caller of CollectRegions and the execution state will look like that in figure 5-9.

Recursive calls require a comment, as each call to Step Out only applies to the current call to the function. For every recursive call that has been executed you need to select Step Out once.

The key combination Shift+F11 is the way to go with Step Out, provided your current profile is that of 'Visual Studio Developer'.

Move the Point of Execution

The ability to move the point of execution is old news to Visual Basic developers but has not been available for other Microsoft development environments before Visual Studio .NET.

You can only use this function when execution has been halted and you see the yellow execution pointer in the left margin of VS .NET:  . When you

move the mouse cursor on top of the execution marker, the mouse pointer changes into an icon which looks like a big right-pointing arrow with two vertical arrows on its flank: ⇨. A balloon text is also displayed stating that you can drag the execution pointer to another line of code at your own risk!

There's at least one instance where this can be very useful: to make sure that every code path is executed. Naturally, the testing phase should execute every code path in your application but moving the point of execution manually allows you to make sure your code will pass the tests with flying colours.

Watch Expressions, QuickWatch and QuickestWatch

Watch expressions are used when there's a certain variable or certain variables whose contents you want to keep an eye on during execution.

QuickWatch

QuickWatch can be called for any entity in source code, variables, functions and methods alike (in fact it can be called for any token in the source code, but the results may be of no use, as when calling it for the token 'void': "error: identifier 'void' out of scope").

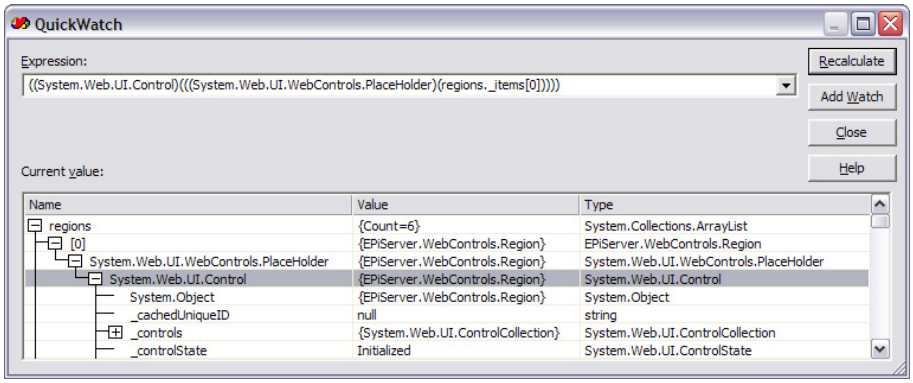


Figure 5-10: QuickWatch window displaying information for a variable named regions.

QuickWatch lets you inspect the contents of compound variables such as the one shown in figure 5-10.

There are at least three ways to call upon QuickWatch: type Ctrl+Alt+Q, select a token and then select QuickWatch from the shortcut menu (right click) or open the Debug menu and then select QuickWatch.

Notice that when the QuickWatch window is displayed, you can turn the QuickWatch into a regular watch by clicking on the button Add Watch.

QuickestWatch

There is no function called QuickestWatch, we were simply at a loss for a name to assign to the function that's called whenever you let the mouse pointer hover over a variable name – the function that uses a balloon window to display the cur-

rent contents of the variable, like this: regions = {Count=6}. As regions is a compound variable, it is not practical to display its contents in the balloon window; instead the value of one its significant attributes is displayed, in this instance Count.

Handling Exceptions

You have the option of letting the debugger handle exceptions either before exceptional handling code or in lieu of it.

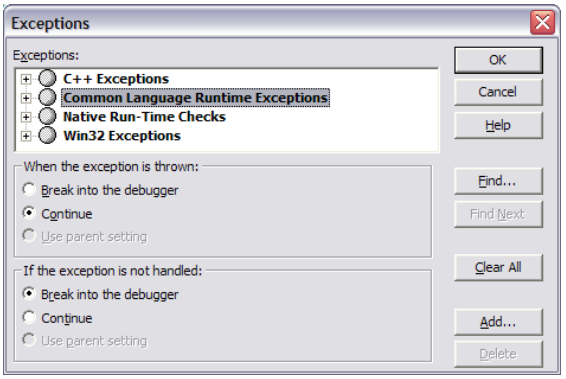


Figure 5-11: Settings for handling, or not, exceptions in the VS.NET debugger.

Figure 5-11 shows that you can control the exception handling or individual exceptions or whole groups, such as Common Language Run-time, CLR, exceptions.

Even if you employ your own exception handling in code, which you should, these exception handling capabilities allow you more freedom during debugging.

Call Stack

Becoming 'friends', or at least on speaking terms, with the Call Stack is important when you develop solutions with EPiServer, as the Call Stack will help answering the all-important question: 'how did my code end up here?' As Visual Studio .NET is a versatile development environment, the Call Stack window can be displayed anywhere you wish. You will always be able to get to it by opening the Debug menu, then Windows and finally selecting Call Stack (or press Ctrl+Alt+C on the keyboard).

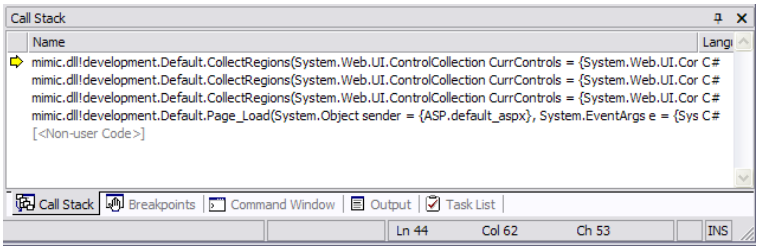


Figure 5-12: Call Stack window.

Take a look at the screen dump in figure 5-12. The Call Stack really is presented like a stack; the latest item is at the top of the stack, nearest the top of the figure. Since there are three calls to CollectRegions at the top of the stack, we can safely assume that out of the three calls two are recursive calls. At the bottom of the stack, meaning the oldest call, we see a call to Default.Page_Load, which is hardly surprising since Page_Load is usually called first for any ASP.NET Web Form. Before that call, being the first, we only see the text '[<Non-user Code>]', implying that we did not create that code ourselves, again an accurate statement.

Command Window: Command Mode and Immediate Mode

The Command Window is quite handy and something that should be familiar to Visual Basic developers as such developers had access to similar functionality in the Immediate window of Visual Basic 6.0 and earlier. It features two different modes of operation:

- Command Mode lets you execute Visual Studio commands, including those that aren't available in menus
- Immediate Mode allows you to display the contents of variables, call function, and more

The same pane is used for both the Command Mode and Immediate Mode. You'll probably find that Immediate Mode is the one more frequently used.

With the Command Window - Immediate Mode you may:

- Display contents of variables including compound variables
- Change the content of variables
- Evaluate expressions

The syntax in Immediate Mode is very similar to that of C# itself, meaning that a statement such as 'Index = 3' assigns the number 3 to the variable Index, provided there is a variable Index declared within the current scope.

To print the contents of a variable, you start the line with a question mark: '?Index'. Remember that the case-sensitivity of Immediate Mode follows that of the current development language; when using Visual C# .NET Immediate Mode is case-sensitive ('index' is not the same as 'Index').

The Immediate Mode of the Command Window is invoked by pressing Ctrl+Alt+I or by opening the Debug menu, choosing Window and finally choosing Immediate.

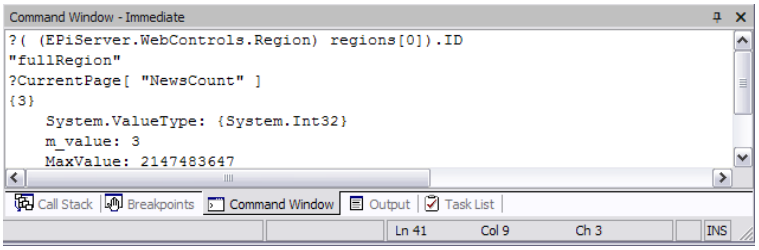


Figure 5-13: Command Window in Immediate Mode.

The screen dump in figure 5-13 is the result of displaying the contents of two variables during the debugging session of an EPiServer application.

The variable *regions* is an array variable declared as System.Collections.ArrayList, but it only holds objects of type EPiServer.WebControls.Region. The Region class has a string property ID (inherited from System.Web.UI.Control). We wish to display ID, since it's the name of a region set up in a Framework Definition File and used by the Page Template File which was used to create the EPiServer Page Type we're currently debugging. Hence this command line in Immediate Mode:

Example 5-17: Command line in Immediate Mode to display the contents of the ID property.

```
?( (EPiServer.WebControls.Region) regions[ 0 ] ).ID
```

The other command line prints the contents of the property NewsCount. This property is one added to the EPiServer Page Type Start page, from which the site's start page was later created in EPiServer Edit mode.

Example 5-18: Command line in Immediate Mode to display the contents of CurrentPage["NewsCount"].

```
?CurrentPage[ "NewsCount" ]
```

As an interesting aside, if instead of 'CurrentPage["NewsCount"]' you use the equivalent 'CurrentPage.Property["NewsCount"]' the result is the same but you will see a different 'trail' displayed in the Command Window, one which also tells you that this particular property has a data type of EPiServer.Core.PropertyNumber.

Use the Logging Capabilities Introduced with EPiServer 4.3

As of EPiServer 4.3, you as a developer now have access to an array of built-in logging capabilities. The logging system in EPiServer uses the log4net framework and we therefore recommend that you read the introduction on the log4net Web site (http: //logging.apache.org/log4net).

Logging is controlled from a configuration file named EPiServerlog.config and should be placed in the same folder as the application's web.config file. The logging configuration settings were deliberately kept out of the web.config file for a number of reasons – one being that if you want to enable logging when the application has entered a bad state, you have to change and save the configuration file. If the information is in web.config, the Web application would be restarted when you save web.config, possibly clearing the cause of the problem which you wanted to log. Keeping the log configuration separate from web.config eliminates this problem.

Statistics logging

EPiServer has one specific logger which generates output which can be used for statistics analysis similar to traditional Web log analysis. This is the EPiServer.Diagnostics.StatisticsLog. The reason for its existence is that a dynamic Web site may exhibit behaviour that is different from a static site. For example when a person views the start page of an intranet site, she may be presented with a headline news item and two smaller product related news items. The Web site owner may want to have information about the specific news items being read, so this single 'start page hit' should actually be counted as three separate 'content' hits.

The statistics logging function in EPiServer is designed to support such behaviour. It should not usually be considered as a replacement for the traditional IIS log file since the EPiServer log will not register information concerning image files (e.g. GIF) or other static file accesses.

Optimising Performance

A truly great book which should be a part of every serious developer's library is *Code Complete* by Steve McConnell. In the chapter 'Code-Tuning Strategies' he cites Michael Jackson's (remember Jackson Structured Programming, JSP?) 'Rules of Optimisation':

1. Don't do it.
2. [*For experts only*] Don't do it yet.

Apparently the corollary to the second rule should be: 'Don't do it until you have a perfectly clear and unoptimised solution'. Or, in other words, first solve the problem, then (and only then) make the solution faster.

More Rules

If we are allowed some presumptuousness, we would add two more corollaries to Jackson's rules:

3. Never optimise until you've profiled.
4. Changing algorithms can make your solution a lot faster than tweaking code.

What Is Taking So Long?

There are over a zillion stories about optimising the wrong part of an application. One contender for the top spot must be the one where a group of programmers optimised the idle loop of an operating system to make it 50% faster!

But seriously, you need to find the areas in your application which are using up the most time. This can be done in several ways. An easy way is to let your application simply print the current time to a log file and then analyse the results. Another way is to use a profiling tool. You will find some low-cost profilers for Microsoft .NET Framework applications at SourceForge (http://sourceforge.net), for example:

- NProf, '.NET profiler and generic profiling API'
- NProfiler, 'An application profiler for .NET'

There are also commercial profilers, these are but a few examples:

- Red-Gate ANTS Profiler
- AutomatedQA AQTime .NET Edition
- DevPartner Profiler
- Rational Quantify for Windows
- Borland OptimizeIt Profiler for the Microsoft .NET Framework

Of course, using a profiler makes your application very sluggish, but that's a non-problem since you're not interested in absolute speed, you simply want to find out which parts of your code use up the most execution time.

Figure 5-14 is from a session using NProf to profile the EPiServer example Web site.

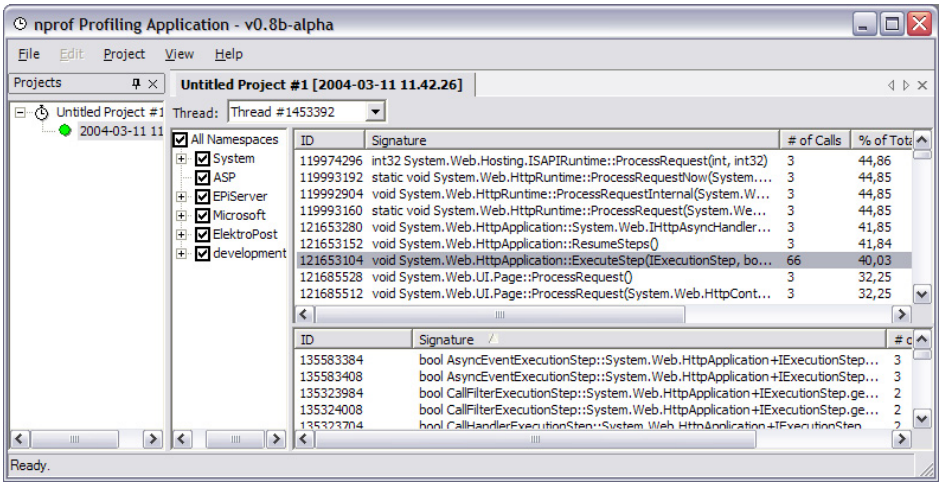


Figure 5-14: Results from an ASP.NET application profiling in NProf.

Change Algorithms

People are great for many things, e.g. if you find yourself stuck and cannot think of an alternative way to solve a particular problem, you should try asking colleagues or even Internet forums. You will soon notice that 'she who asks, receives many a good answer'.

Using the Debug Switch for EPiServer Scheduler Service

The service responsible for scheduled jobs, EPiServer Scheduler Service, has a Debug switch which makes the services print diagnostic messages. To use this switch, start the service as a regular program, from the command prompt:

Example 5-19: Command line to start EPiServer Scheduler service with its debug switch.

```
"C:\Program Files\EPiServer.Scheduler\EPiServer.SchedulerSvc.exe" DEBUG
```

The command line in example 5-19 is what to expect if you've installed EPiServer on a computer with Windows 2000, XP or Windows Server 2003 installed on drive C:. Please note that the Debug parameter should be entered exactly as shown in the example command line: the word 'DEBUG', in all capital letters and not preceded by a dash nor a slash.

When the EPiServer Scheduler service is running in debug mode, it outputs a message at least once a minute. If you have several Web sites on the same computer, the output strings are colour coded so each Web site gets its own 'output colour'.

Example 5-20: Example output from EPiServer Scheduler service in debug mode.

```
[10.56.32 UTC] #INF# ************************************************
[10.56.32 UTC] #INF# * EPiServer Scheduler Service running in DEBUG mode
[10.56.32 UTC] #INF# ************************************************
[10.56.32 UTC] #INF# Loaded queue with 2 sites:
[10.56.32 UTC] #INF# [_LM_W3SVC_1_ROOT_Mimic] http://localhost/Mimic/
[10.56.32 UTC] #INF# [_LM_W3SVC_1_ROOT_Example] http://localhost/Example/
[10.56.32 UTC] #INF# Service started - begin accepting connections
[10.56.32 UTC] #INF# [_LM_W3SVC_1_ROOT_Mimic] Site listener started
[10.56.32 UTC] #INF# [_LM_W3SVC_1_ROOT_Example] Site listener started
[10.56.32 UTC] #INF# [_LM_W3SVC_1_ROOT_Mimic] New scheduled job 2004-03-01 11.31.14
                     (7d8fad39-2682-43b6-b49c-d7c8199d5195)
[10.56.32 UTC] #INF# [_LM_W3SVC_1_ROOT_Example] New scheduled job 2004-03-01 11.31.14
                     (7d8fad39-2682-43b6-b49c-d7c8199d5195)
[10.57.32 UTC] #INF# [_LM_W3SVC_1_ROOT_Mimic] No new scheduled jobs, currently waiting for
                     2004-03-01 11.31.14 (7d8fad39-2682-43b6-b49c-d7c8199d5195)
[10.57.32 UTC] #INF# [_LM_W3SVC_1_ROOT_Example] No new scheduled jobs, currently waiting for
                     2004-03-01 11.31.14 (7d8fad39-2682-43b6-b49c-d7c8199d5195)
...
```

```
[11.31.13 UTC] #INF# [_LM_W3SVC_1_ROOT_Mimic] Execute job 7d8fad39-2682-43b6-
                     b49c-d7c8199d5195
[11.31.13 UTC] #INF# [_LM_W3SVC_1_ROOT_Example] Execute job 7d8fad39-2682-43b6-
                     b49c-d7c8199d5195
[11.31.14 UTC] #INF# [_LM_W3SVC_1_ROOT_Example] New scheduled job 2004-03-01 12.31.14
                     (7d8fad39-2682-43b6-b49c-d7c8199d5195)
[11.31.15 UTC] #INF# [_LM_W3SVC_1_ROOT_Mimic] New scheduled job 2004-03-01 12.31.14
                     (7d8fad39-2682-43b6-b49c-d7c8199d5195)
...
```

All output from EPiServer Scheduler in debug mode is stamped with time in UTC (Universal Time Coordinated, a.k.a. Greenwich Mean Time).

If you've been adding and deleting EPiServer Web sites, the sites data file for EPiServer.SchedulerSvc.exe, EPiServer.SchedulerService.Sites.xml, might contain more EPiServer Web sites than are current installed on your Web server. In this case you can manually remove those Web sites from EPiServer.SchedulerService .Sites.xml. You need to change this in three sections of the XML file. First remove the obsolete Web sites from the first SOAP-ENC:Array section. Having done that, change the number in brackets for xsd:anyType so it reflects the new number of Web sites.

Example 5-21: Example of SOAP-ENC:Array section listing Web sites in EPiServer.SchedulerService.Sites .xml.

```
<SOAP-ENC:Array id="ref-2" SOAP-ENC:arrayType="xsd:anyType[2]">
<item id="ref-6" xsi:type="SOAP-ENC:string">_LM_W3SVC_1_ROOT_Mimic</item>
<item id="ref-9" xsi:type="SOAP-ENC:string">_LM_W3SVC_1_ROOT_Example</item>
</SOAP-ENC:Array>
```

Do the same for the second SOAP-ENC:Array section.

Example 5-22: Example of SOAP-ENC:Array section listing Web sites in EPiServer.SchedulerService.Sites .xml.

```
<SOAP-ENC:Array id="ref-3" SOAP-ENC:arrayType="xsd:anyType[2]">
    <item href="#ref-12"/>
    <item href="#ref-15"/>
</SOAP-ENC:Array>
```

Lastly, remove all superfluous a3:SiteConnect sections so only the relevant ones remain.

Example 5-23: Example of an a3:SiteConnect section.

```
<a3:SiteConnect id="ref-12" xmlns:a3="http://schemas.microsoft.com/clr/nsassem/EPiServer
    .SchedulerService/EPiServer.SchedulerSvc%2C%20Version%3D4.22.0.98%2C%20Culture
    %3Dneutral%2C%20PublicKeyToken%3Dnull">
    <sSiteUnique href="#ref-6"/>
```

```
    <sSiteURL id="ref-21">http://localhost/Mimic/</sSiteURL>
    <sRemoteObjectURL id="ref-22">pipe://_LM_W3SVC_1_ROOT_Mimic/Callback.rem
    </sRemoteObjectURL>
</a3:SiteConnect>
```

You don't have to change any IDs, so you can keep all 'ref-*n*' and '#ref-*n*' intact. What's important is that the numbers in brackets for xsd:anyType are correct and that there are as many a3:SiteConnect sections as there are Web sites.

6
EPiServer Base Classes and Interfaces

Developing solutions with EPiServer 4, you will be working with both EPiServer code libraries and Microsoft .NET Framework. Knowing what's available will save many hours of unnecessary coding for you and will even be able to solve problems you had previously thought to be unsolvable.

In this chapter we'll take a closer look at these base classes and interfaces:

- EPiServer.PageBase (ancestral class of these:)
  - EPiServer.SimplePage
  - EPiServer.EditPage
  - EPiServer.TemplatePage
  - EPiServer.SystemPage
  - EPiServer.Util.LoginBase
- EPiServer.UserControlBase
- EPiServer.Core.PageData
- EPiServer.Core.IPageSource
- EPiServer.Global
- EPiServer.ApplicationConfiguration

The Ever-Present Web Page Tree

When developing solutions with EPiServer, you will notice that you're not simply dealing with a set of Web pages without any inter-relationships. As you'll see, some of the control class types in EPiServer.WebControls handle the Web pages as part of a hierarchical tree structure (more on these in chapter 7, *EPiServer Web Controls*), and you'll soon realise that this page tree is everywhere.

This tree structure is realised in the EPiServer database table tblPage by using two attributes (columns): pkID and fkParentID. The attribute pkID is simply the integer page ID (used as primary key for tblPage). The other attribute, fkParentID, is an internal foreign key as it 'points' to pkID in the same table. (For steps

to create a Microsoft Transact-SQL stored procedure to visualise the tree, see page 320.)

There's a third attribute that's important to the page tree. Its name is VisibleInMenu and its value controls whether or not the Web page should be presented in menus and lists. That's fair enough you may ask, but how does that affect you? Well, that too will be addressed in the chapter entitled *EPiServer Web Controls*, as the effects are noticed when working with some of the EPiServer Web Custom Controls. As a general rule, VisibleInMenu is set to 1 for the pages that constitute the Web site or are used in EPiServer and to 0 for pages such as saved user profiles, pages that do not have a role in presenting the Web site to the average user.

PageBase, UserControlBase and PageData: When to Use What

The overall EPiServer object model is easy and small. When you need to handle other Web Pages as objects from inside a Web Page, but outside any User Controls, the choice is the PageBase class. From inside User Controls, you go with UserControlBase (the mother class for EPiServer User Controls). Notice that both PageBase and UserControlBase have a method called GetChildren, which returns all child pages to a specified page, using the already mentioned page tree (also notice that GetChildren is specified in the interface IPageSource, see page 153).

To access attributes on Web Pages, you use PageData irrespective of whether the current context is that of a Web Page or a User Control.

EPiServer.PageBase

Being the mother class for all ASP.NET Web Forms to be used as EPiServer Page Templates, PageBase is very important indeed. Suffice it to say that should you try to use anything but Page Templates that inherit from EPiServer.PageBase to create Page Types and eventually EPiServer Web Pages, no dynamic content will ever be presented on the page.

Be aware that it is not at all unlikely for this to happen and the only clue to what's going on is that nothing that's been added to the Web Page in Edit mode is visible in View mode. Beware of any Web Forms inheriting from 'System.Web .UI.Page'. Please note that it is perfectly legal, and sometimes a good choice, to use non-EPiServer Web Forms in an EPiServer solution.

PageBase is an abstract class (MustInherit in Visual Basic .NET), which is very advantageous when you're dealing with objects since every Web Page you'll be handling is, or at least should be, a descendant of PageBase. In other words, object variables declared as EPiServer.PageBase can handle any Web Page and casting with '(EPiServer.PageBase)' will always enable access to all attributes and methods defined for objects of PageBase descendant classes.

EPiServer.PageBase itself inherits from System.Web.UI.Page, which has the core functionality for setting up and rendering a standard Web form (aspx file). The PageBase class extends the functionality of the Page class with EPiServer specific features such as access to configuration settings (Configuration property), user information (CurrentUser property) and information about the current EPiServer page (CurrentPage property.)

PageBase also implements the interface IPageSource interface, which enables retrieval of other EPiServer pages.

The most important part of the object model for PageBase looks like this:

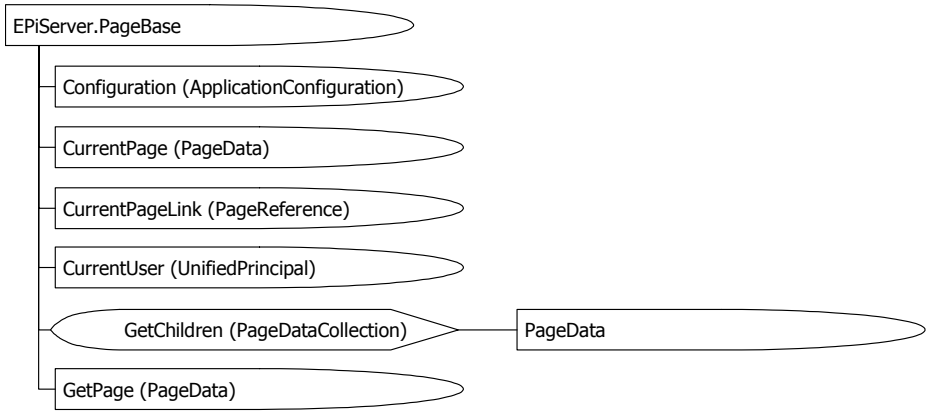


Figure 6-1: Object model for EPiServer.PageBase.

The class tree for PageBase is quite shallow, but still covers most needs.

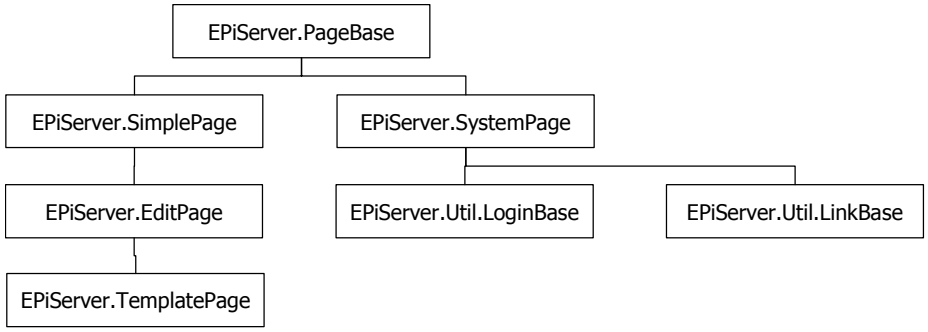


Figure 6-2: Inheritance tree for class EPiServer.PageBase.

Example 6-1: Declaration for class PageBase in Visual Basic .NET, C# and JScript .NET.

```
Visual Basic:   Public MustInherit Class PageBase
                Inherits Page
                Implements IPageSource
C#:             public abstract class PageBase : Page, IPageSource
JScript:        public abstract class PageBase extends Page implements IPageSource
```

When you write code for a User Control, you'll often want to cast the control's Page property (which is a System.Web.UI.Page type) to a PageBase class in order to get to the EPiServer page information.

Example 6-2: Cast using EPiServer.PageBase.

```
// From inside a user control
string pageName = ( (EPiServer.PageBase) Page ).CurrentPage.PageName;
```

Public Properties for EPiServer.PageBase

EPiServer.PageBase has many public properties, public methods, public events and also protected properties and methods. The following tables describe the most important of these.

Table 6-1: Public properties for EPiServer.PageBase.

Automatic-Translation	Select the type of automatic translation of controls that will take place.
Configuration	Get the configuration object.
CurrentPage	Currently loaded page on templates [EPiServer.PageData] (from EPiServer.IPageSource).
CurrentPageLink	Get the reference to the current page [EPiServer.Core .PageReference]. (You can safely assume that EPiServer.Core .PageReference is an integer and the same as the pkID column in tblPage.)
CurrentUser	Get current EPiServer user [EPiServer.Security.UnifiedPrincipal].
EPCharset	The character set encoding to use for displaying the current page [string].
EPLanguage	Determine the user interface language that the page uses [string].
EPLocale	The locale ID to use for formatting date, time, numbers, etc. [int]

Public Methods for EPiServer.PageBase

The public methods for EPiServer.PageBase comprise the tow implemented for IPageSource, a couple dealing with access permissions, one used to check wheth-

er a named property (PropertyData) exists and has been assigned a value, and one to help in translation.

Table 6-2: Public methods for EPiServer.PageBase.

AccessDenied	Invoked when the current user's access permissions are not sufficient for the operation she is trying to perform.
CheckAccess	Verify that access control requirements are met for the current Web request.
GetChildren	Retrieve a PageData listing based on supplied EPiServer .Core.PageReference argument (from EPiServer.IPage-Source).
GetPage	Retrieve a PageData object based on supplied EPiServer .Core.PageReference argument (from EPiServer.IPage-Source).
IsValue	Determine if the named property exists and contains a non-null value (same functionality as in EPiServer.User-ControlBase).
QueryAccess	The access level that the current user holds to the current page.
QueryDistinctAccess	Verify that access control requirements are met for the current Web request.
RequiredAccess	Determine the access level that is required for the current Web request.
Translate	Translate the given string to the current language (same functionality as in EPiServer.UserControlBase).
ValidatePageTemplate	Make sure that the right page template is used to present a page

More Information on Using the Public Properties and Methods in EPiServer.PageBase

PageBase.Configuration

PageBase.Configuration holds no surprises; it deals exclusively with configuration settings. The attribute proper is of the type EPiServer.ApplicationConfiguration (see page 156).

Example 6-3: Using PageBase.Configuration and PageBase.CurrentPage.

```
protected void SendEmailButton_Click( object sender, System.EventArgs e ) {
    if ( ! IsValid ) {
```

```
        return;
    }
    System.Web.Mail.MailMessage mail= new System.Web.Mail.MailMessage();
    mail.BodyFormat= System.Web.Mail.MailFormat.Text;
    mail.From       = From.Text;
    mail.To         = To.Text;
    mail.Subject    = Subject.Text;
    mail.Body       = Other.Text;
    mail.Body       += "\n\n" + Configuration.HostUrl + CurrentPage.LinkURL;
    mail.Body       += "\n\nMessage was sent from IP address " + Request.UserHostAddress + " at " +
        System.DateTime.Now.ToString();

    Configuration.InitSmtpServer();
    System.Web.Mail.SmtpMail.Send( mail );

    Page.RegisterStartupScript( "CloseWindow", "<script type='text/javascript'>
        window.returnValue = true;window.parent.close();</script>" );
}
```

The code in example 6-3 is used in the example Web site to send e-mail. Both Configuration and CurrentPage are used. Configuration is used to initialise the mail server and also, together with CurrentPage, to send information to the recipient of the e-mail (current HostUrl and LinkUrl).

Example 6-4: Using Configuration to display information about the start page for the Web site.

```
<tr><td>ID of Start Page: <%= Configuration.StartPage.ID %></td></tr>
```

In example 6-4, Configuration is used simply to display information about the Web site's start page (compare this example with 6-40 on page 155).

Example 6-5: Using Controls property.

```
private void ShrinkFramework() {
    if ( FrameworkSelector.ActiveFramework == null ) {
        return;
    }
    Table minatureTable = PreviewLoader.GetFrameworkMap( FrameworkSelector.ActiveFramework, 2 );
    if ( minatureTable != null ) {
            FrameworkSelector.Controls.Add( minatureTable );
    }
    FrameworkSelector.Controls.Remove( FrameworkSelector.ActiveFramework );
}
```

PageBase.Controls

Although Controls is inherited from System.Web.UI.Control, it might still be of interest since Controls not only comprises ASP.NET and HTML controls but also EPiServer Web Custom Controls (from the EPiServer.WebControls name space). The code in example 6-6 uses Controls to enumerate all Region controls on the current page. This would make sense if the current page is an EPiServer Page Template File which includes a Framework Definition File.

Example 6-6: Using System.Web.UI.ControlCollection.Controls to find all Region control objects on the current page (included with the Content Framework Definition File).

```
private          System.Collections.ArrayList regions= new System.Collections.ArrayList();

private void Page_Load( object sender, System.EventArgs e ) {
    if ( ! IsPostBack ) {
        DataBind();
        CollectRegions( this.Page.Controls );
    }
}

private void CollectRegions( System.Web.UI.ControlCollection CurrControls ) {
    foreach ( System.Web.UI.Control SingleControl in CurrControls ) {
        if ( SingleControl is EPiServer.WebControls.Region ) {
            regions.Add( SingleControl );
        }
        if ( SingleControl.HasControls() ) {
            CollectRegions( SingleControl.Controls );
        }
    }
}
```

A reference to every Region control object is stored in the ArrayList variable regions. CollectRegions is a recursive function which is executed when the page is loaded, but only on first load, not during post-backs.

PageBase.CurrentPage

A little caution is prudent when using CurrentPage in the HTML part of Web Forms, since CurrentPage is implemented from IPageSource and EPiServer.UserControlBase also implements IPageSource. This means that CurrentPage will have different meanings in the same Web Form depending on whether it's used outside any included EPiServer Web Custom Control in general and outside any templated EPiServer Web Custom Control in particular.

Example 6-7: Using PageBase.CurrentPage on a Page Template File, outside of any included controls.

```
CurrentPage.PageName on a Page Template File, outside of any included controls:
<%= CurrentPage.PageName %>
```

The HTML code in example 6-7 results in the PageName property for CurrentPage being displayed, just as expected. In the next example, 6-8, CurrentPage.PageName is used inside both a ContentFramework and a Content control.

Example 6-8: Using CurrentPage.PageName on a Page Template File, inside ContentFramework and Content control.

```
<development:DefaultFramework id="DefaultFramework" runat="server">
    <EPiServer:Content ID="NewsListing" Region="menuRegion" runat="server">
        CurrentPage.PageName on a Page Template File, inside ContentFramework and Content controls:
            <%= CurrentPage.PageName %>
...
```

Now let's see what happens if we do the same thing inside a templated control.

Example 6-9: Using PageBase.CurrentPage inside an EPiServer templated control.

```
<EPiServer:Newslist ID=Newslistnew Pagelinkproperty="NewsContainer" runat="Server"
    MaxCount='<%# GetNewsCount() %>'>
    <NewsTemplate>
        CurrentPage.PageName on a Page Template File, inside Newslist control (in NewsTemplate):
            <%= CurrentPage.PageName %>
    </NewsTemplate>
</EPiServer:Newslist>
```

The HTML code in example 6-9 will produce the same results for all the news items: the name of the page on which the control is included. But say we wanted to have the name of the page in the page collection that was the result of using the NewsList control. Well, then we would have to use the Data Binding Expression Syntax available in ASP.NET.

Remember that '<%= *expression* %>' in the HTML part of an ASP.NET file is shorthand for 'Response.Write(*expression*)'. In ASP.NET, there is also the new Data Binding Expression Syntax which allows you to enter expressions such as '<%# *property name* %>' as an HTML tag. The success of this hinges on the DataBind method being executed for the page in question binding any controls, even Server controls, to their respective data source. For our NewsList example, we must use the *mysterious* templated controls Container property as shown in example 6-10. Please note that you must use both Container and '#', otherwise it won't work.

Example 6-10: User PageBase.CurrentPage in conjunction with Container in a templated control.

```
<EPiServer:Newslist ID=Newslistnew Pagelinkproperty="NewsContainer" runat="Server"
    MaxCount='<%# GetNewsCount() %>'>
    <NewsTemplate>
        Container.CurrentPage.PageName on a Page Template File, inside Newslist control (in
        NewsTemplate): <%# Container.CurrentPage.PageName %>
```

```
    </NewsTemplate>
</EPiServer:Newslist>
```

CurrentPage is useful in code-behind files, too.

Example 6-11: Using PageBase.CurrentPage and PageBase.IsValue.

```
protected int GetNewsCount() {
    return IsValue( "NewsCount" ) ? (int) CurrentPage[ "NewsCount" ] : -1;
}

protected string StartPageImage {
    get {
        return IsValue( "MainImage" ) ? (string) CurrentPage[ "MainImage" ] : _startPageImage;
    }
}
```

In example 6-11, CurrentPage is used to read the contents of two properties; NewsCount and MainImage. To safeguard the property access, the presence and possible contents of both properties are first tested using the method Page.IsValue.

PageBase.CurrentPageLink

Having the type EPiServer.Core.PageReference makes CurrentPageLink an excellent candidate for an argument in calls to GetChildren and GetPage. This is also how it's often used.

Example 6-12: Using CurrentPageLink to retrieve the PageData object.

```
PageData tmpPageData = GetPage( CurrentPageLink );
```

CurrentPageLink has no place inside templated controls, since the Container property is of the type PageTemplateContainer which doesn't implement CurrentPageLink.

PageBase.CurrentUser

CurrentUser has a type of EPiServer.Security.UnifiedPrincipal, which is used to facilitate handling the current Web site user and her access permissions. As part of the user identity, UnifiedPrincipal lets you handle the account's Security Iden-

tified, SID, through a mapping. UnifiedPrincipal has several useful properties and methods, some of which are presented in the following table.

Table 6-3: Some of UnifiedPrincipal's properties and methods.

Attribute or Method Name	*Description*
AnonymousUser	Static field representing the anonymous user. Any user who is not logged-on can be represented by AnonymousUser.
Current	Static property returning the UnifiedPrincipal object for the current HTTP request.
CurrentSid	Static property returning the security identity of user for the current HTTP request.
Identitity	Implementation of System.Security.Principal.IIdentity interface.
Sid	EPiServer user account security identifier.
SidList	List of EPiServer group account security identifiers. One item for each group the user belongs to.
ClearCache	Static method clearing the cache from this user.
IsInRole	Used to check membership in specified group.
IsPermitted	Used to check if specified permissions has been assigned to the user.
UserData	An EPiServer.Personalization.PersonalizedData object.

Example 6-13: Using PageBase.CurrentUser in HTML.

```
...
<%= CurrentUser.Sid %></td><td><%= CurrentUser.Identity.Name %>
...
```

The HTML code in example 6-13 displays the EPiServer Security Identifier and Name property of the Identity property. In the case of the Anonymous user, the latter would result in the text 'Anonymous' being displayed.

(See also example 3-14 on page 48.)

PageBase.EPCharset, PageBase.EPLanguage and PageBase.EPLocale

PageBase.EPCharset, PageBase.EPLanguage and PageBase.EPLocale all deal with locale and language aspects.

PageBase.AccessDenied

The method AccessDenied can be called in order to display a message to the user that she, for some reason, has been denied access.

Example 6-14: Using AccessDenied to deny access to a user not logged-on.

```
public abstract class LoginStatus : EPiServer.UserControlBase {
    protected System.Web.UI.WebControls.LinkButtonLogin;

    protected void Login_Click( object sender, System.EventArgs e ) {
        System.Security.Principal.IPrincipal user = Context.User;
        if ( ( user != null ) && user.Identity.IsAuthenticated ) {
            return;
        }
        PageBase.AccessDenied();
    }
}
```

The code in 6-14 is taken from the Web User Control LoginStatus.ascx which ships with EPiServer. It must be understood that the click event for ASP.NET Server control Login has been linked to the function Login_Click, so when the user clicks the Login button this function is executed. Now, the call to Access-Denied doesn't necessarily mean that the user is denied access, it can also be used to present the user with a log-on dialogue window. This happens in Web User Control QuickBar.ascx.

PageBase.GetChildren

Using the Web Page Tree as a basis, GetChildren returns possible child objects as an EPiServer.Core.PageDataCollection (a collection of EPiServer.Core.Page-Data objects).

Example 6-15: Using PageBase.GetChildren in a code-behind file for a Web Form.

```
private void WriteRssDocumentFromCurrentPage() {
    EPiServer.Core.PageDataCollection ChildPages = GetChildren( CurrentPageLink );
...
    foreach ( EPiServer.Core.PageData ChildPageData in ChildPages ) {
        System.Xml.XmlNode item= doc.CreateNode( System.Xml.XmlNodeType.Element, "item", null );

        System.Xml.XmlNode itemTitle= doc.CreateElement( "title" );
        itemTitle.InnerText= ChildPageData.PageName;
        item.AppendChild( itemTitle );

        System.Xml.XmlNode itemLink= doc.CreateElement( "link" );
        itemLink.InnerText= EPiServer.Global.EPConfig.HostUrl + ChildPageData.LinkURL;
        item.AppendChild( itemLink );
...
```

```
    }
    doc.Save( Response.OutputStream );
}
```

The code in example 6-15 shows how to use PageBase.GetChildren to retrieve a list of PageDataCollection items containing the child page's PageData objects. This collection is then enumerated in order to build an RSS (Really Simple Syndication) object which is finally dispatched to the user's Web browser.

PageBase.GetPage

GetPage returns the PageData object for the specified EPiServer.Core.PageReference argument.

Example 6-16: Using PageBase.GetPage and PageBase.CurrentPage to retrieve PageData information for the page specified in the attribute EventsContainer.

```
protected EPiServer.Core.PageData EventRootPage {
    get {
        if ( _eventRootPage == null ) {
            if ( CurrentPage[ "EventsContainer" ] != null ) {
                _eventRootPage =
                    GetPage( (EPiServer.Core.PageReference) CurrentPage[ "EventsContainer" ] );
            }
        }
        return _eventRootPage;
    }
}
```

Example 6-17: Using GetPage to retrieve the PageData object for the current page.

```
EPiServer PageData tmpPageData = GetPage( CurrentPageLink );
```

PageBase.IsValue

You have already seen IsValue at work in example 6-11. It's the safe way to check whether a property exists and has been assigned contents.

PageBase.QueryDistinctAccess

QueryDistinctAccess uses the combination of the CurrentUser property and the current PageBase object to decide whether or not the user has Read access.

Example 6-18: Using PageBase.QueryDistinctAccess.

```
protected void Logout_Click( object sender, System.EventArgs e ) {
    EPiServer.Security.UnifiedPrincipal user = EPiServer.Security.UnifiedPrincipal.Current;
    if ( user.Identity.IsAuthenticated ) {
        if ( Configuration.Authentication != System.Web.Configuration.AuthenticationMode.Forms ) {
```

```
            return;
        }
        System.Web.Security.FormsAuthentication.SignOut();
        EPiServer.Security.UnifiedPrincipal.RemoveFromCache( user.Identity );

        PageBase.CurrentUser = EPiServer.Security.UnifiedPrincipal.AnonymousUser;

        if ( ! PageBase.QueryDistinctAccess() ) {
            Response.Redirect( Configuration.RootDir );
        }
        HandleLoginStatusButton();
    }
}
```

The code in example 6-18 is from the Web User Control QuickBar.ascx. It's executed each time a logged-on user clicks on the log-off tool-bar button,  . If the anonymous user (who has been made current, 'PageBase.CurrentUser = EPiServer.Security.UnifiedPrincipal.AnonymousUser;') does not have access to the current page, a call to Response.Redirect is made which takes the user to the Web site's default page; otherwise no new page is loaded.

PageBase.RequiredAccess

CheckAccess, QueryAccess, QueryDistinctAccess and RequiredAccess all deal with access permissions. Using RequiredAccess, your code can check the current user's access to another page before actually trying to load the page. When called without any arguments, RequiredAccess checks for Read access.

PageBase.Translate

The method Translate can be used instead of EPLang.Translate (see page 155). They perform exactly the same duty and take the same arguments.

EPiServer.SimplePage

Base class for EPiServer pages which do not need access to the EPiServer shortcut menu (right-click menu) or Direct On-Page Editing (see below).

Pages that inherit from SimplePage have all the other capabilities that TemplatePage descendants have, meaning you can still add EPiServer Properties and edit their contents. However, when a Web Page descending from SimplePage is displayed you will not have access to Admin mode or Edit mode from the shortcut menu.

Example 6-19: Web Form which inherits from EPiServer.SimplePage.

```
public class Default : EPiServer.SimplePage {
...
```

```
}
```

EPiServer.EditPage

Base class for EPiServer pages which need to save page data or create new pages. As TemplatePage is a child (the only child) of EditPage, any attributes or methods defined for EditPage will be available for TemplatePage.

Public Properties

Table 6-4: Public properties for EPiServer.EditPage.

Property Name	*Description*
IgnoreModifiedOnSave	Allow a page to be saved without modifying any content
IsNewPage	Check if the query string parameters indicate that this is a request to create a new page.

Public Methods

Table 6-5: Public methods for EPiServer.EditPage.

Method Name	*Description*
AllowCheckInVersion	Determine if the current user is allowed to check in the version currently being edited
AllowDeleteVersion	Determine if the user is allowed to delete the currently selected version
AllowPublishVersion	Determine if the user is allowed to publish the currently selected version of the page
AllowRejectVersion	Determine if the user is allowed to reject the selected version of the current page

Protected Properties

Table 6-6: Protected properties for EPiServer.EditPage.

Property Name	*Description*
IsPageSaved	Indicates if SavePage has been called
IsSave	Check if the posted form indicates that this is a request to save the page

Table 6-6: Protected properties for EPiServer.EditPage.

Property Name	*Description*
NewPageParent	The newly created page will be placed below this page
NewPageType	Page type of the newly created page
RequestedSaveAction	The type of 'Save' action that should be performed

Protected Methods

Table 6-7: Protected methods for EPiServer.EditPage.

Method Name	*Description*
ParseParameters	Get parameters used by the page class
SavePage	Save the page data posted in this request

One such example is the attribute IsNewPage, which is used in the Web User Control Profile.ascx that is shipped with EPiServer.

Example 6-20: Using EditPage.IsNewPage in Profile.ascx.cs.

```
private void SavePage() {
    if ( ( (EPiServer.EditPage) Page).IsNewPage ) {
        CurrentPage[ "Sid" ] = PageBase.CurrentUser.Sid;
        CurrentPage.PageName = PageBase.CurrentUser.Identity.Name;
        CurrentPage.VisibleInMenu = false;
    }
    EPiServer.Global.EPDataFactory.Save( CurrentPage, EPiServer.DataAccess.SaveAction.Publish );
}
```

As it is defined for EPiServer.EditPage, it is possible to use IsNewPage both for classes of the type EditPage, TemplatePage or any other class derived from any of them. The code in example 6-20 uses IsNewPage to give any new page the same name as the currently logged-on user (we have to assume that 'Anonymous' is not allowed access to Profile.ascx). It also sets the page property Sid to the Sid of the user and lastly sets VisibleInMenu to false so the page won't suddenly appear in menus and page lists (see table 6-11 on page 145).

EPiServer.TemplatePage

EPiServer.TemplatePage is the base class for EPiServer Page Templates that should support Direct-on-Page-Editing (DOPE) and right-click menu access to Admin and Edit modes.

TemplatePage has but one public attribute which is not inherited:

- RightClickMenu (of the type EPiServer.RightClickMenu)

Creating an EPiServer Page Template File

Most EPiServer Page Templates inherit from EPiServer.TemplatePage. They start out as ordinary ASP.NET Web Forms and are subsequently tailored to the EPiServer Content Framework.

We'll create an EPiServer Page Template File from scratch, but for now we'll use the example Web site infrastructure.

Open the Example Web Site Solution in Visual Studio .NET

1. Open the EPiServer example Web site in Visual Studio .NET and then open the templates folder in Solution Explorer.
2. Right-click on 'templates' and choose Add–Add New Item. Add a new Web Form, name it 'PageTemplateFile.aspx'.
3. Open PageTemplateFile.aspx.cs (select the file name in Solution Explorer and either click on the tool-bar button  , type F7 or use the File menu).
4. Change the inheritance for class PageTemplateFile so it inherits from EPiServer.TemplatePage instead of System.Web.UI.Page. Notice that the name space is 'development.templates' (you can change the first 't' in templates to to 'T' if you wish), which is exactly what we want.

Add a Framework File, Change Prefixes

5. Switch to Design view by clicking on the toolbar button  .
6. Add a Framework by dragging the file DefaultFramework.ascx to the design window.
7. Open the HTML window (still in Design mode) by clicking on the button labelled 'HTML' in the lower left corner of the Design window.
8. Change TagPrefix for DefaultFramework from 'uc1' to 'DefaultFramework'. Also change the tag to reflect this change:

Example 6-21: Tag for DefaultFramework in Page Template File.

```
<development:DefaultFramework id="DefaultFramework" runat="server">
</development:DefaultFramework>
```

EPiServer.SystemPage

Base class for EPiServer system pages such as the Edit and Admin modes.

EPiServer.Util.LoginBase

LoginBase is the base class used to handle forms-based authentication in EPiServer. Use it, for example, to implement your own login form, such as redesigning the input box. In doing so, there is no need to re-implement the entire forms authentication module. Instead, you could simply let your code-behind class inherit from this class and re-use the HandleFormsLogin method after you have collected user name and password. Another reason to use the HandleFormsLogin class might be to implement login from a URL in a low-security/low-threat environment. Then you could once again collect the user name and password and call this method to set the login cookie which will be used to authenticate future requests.

HandleFormsLogin Method

Performs the actual login handling for forms/cookie based authentication

Example 6-22: Using HandleFormsLogin.

```
public static string HandleFormsLogin( string userName, string password, bool persistCookie )
```

Table 6-8: Parameters for method EPiServer.Util.LoginBase.HandleFormsLogin.

Parameter Name	*Comment*
userName	Name of the user
password	Password for the user
persistCookie	True if the authentication cookie should be a persistent cookie

HandleFormsLogin returns a string with the url to redirect to if the authentication was successful, null if the authentication failed.

EPiServer.UserControlBase

UserControlBase is to EPiServer Web User Controls what PageBase is to EPiServer Page Template Files: it provides access to the EPiServer infrastructure. In essence, letting a Web User Control inherit from EPiServer.UserControlBase instead of the default System.Web.UI.UserControl turns the plain vanilla User

Control class into a full-fledged EPiServer Universe member, having access to EPiServer Web Pages and current configuration settings, among other things.

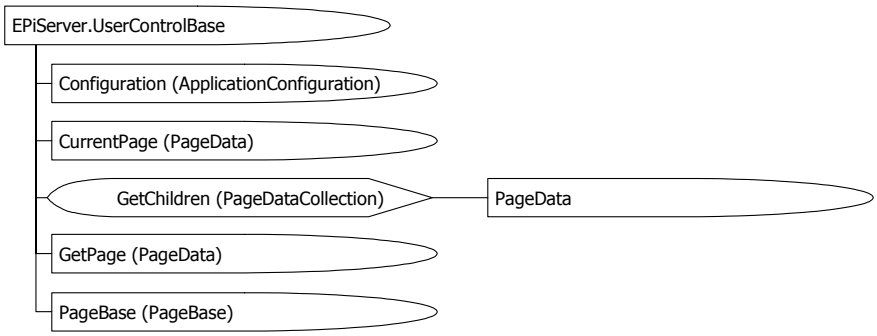


Figure 6-3: Object model for EPiServer.UserControlBase.

Table 6-9 lists the three attributes that are defined by class UserControlBase.

Table 6-9: UserControlBase's own public attributes.

Attribute Name	*Description*
Configuration	Gets the configuration object (same functionality as in EPiServer .PageBase).
CurrentPage	Gets the page data for the current page (from EPiServer.IPage-Source).
PageBase	Returns the executing Page cast to an EPiServer.PageBase class.

All of the public methods in UserControlBase have equal counterparts in EPi-Server.PageBase.

Table 6-10: UserControlBase's own public methods.

Method Name	*Description*
GetChildren	Returns an EPiServer.Core.PageDataCollection of child Pages to specified EPiServer.Core.PageReference (from EPiServer.IPage-Source).
GetPage	Returns a the corresponding EPiServer.Core.PageData object to the supplied EPiServer.Core.PageReference reference (from EPiS-erver.IPageSource).
IsValue	Determine whether the named property exists and holds a non-null value (same functionality as in EPiServer.PageBase).
Translate	Translates the key into a language specific text (same functionality as in EPiServer.PageBase).

More Information on Using the Public Properties and Methods in UserControlBase

Only UserControlBase.PageBase differ between EPiServer.UserControlBase and EPiServer.PageBase; all the other properties and methods have the same purpose in both PageBase and UserControlBase.

UserControlBase.PageBase

You have already seen UserControlBase.PageBase in action: in the examples 6-14 (page 135), 6-18 (page 136) and 6-20 (page 139), UserControl.PageBase is used. Here are some more.

Example 6-23: Using PageBase.CurrentLink (part of a function in the code-behind file for a User Control).

```
if ( (EPiServer.Core.PageReference) CurrentPage[ "MainSearchPage" ] ==
    PageBase.CurrentPageLink ) {
    QuickSearchSpan.Visible= false;
    return;
}
```

In example 6-23, UserControlBase.PageBase.CurrentLink is used to test whether the current page (in the form of CurrentPageLink) is equal to the contents of the property 'MainSearchPage'.

Example 6-24: Using PageBase.CurrentUser (part of the code-behind file for a Web User Control).

```
public abstract class Subscribe : EPiServer.UserControlBase {
    protected System.Web.UI.WebControls.DropDownList   Interval;
    protected System.Web.UI.WebControls.Panel          SubscribeArea;
    protected EPiServer.WebControls.SubscriptionList   SubList;
    protected System.Web.UI.WebControls.TextBox        Email;

    private void Page_Load( object sender, System.EventArgs e ) {
        if ( ! IsPostBack ) {
            foreach ( System.Web.UI.WebControls.ListItem item in Interval.Items ) {
                item.Selected = System.Int32.Parse( item.Value ) ==
                    EPiServer.Personalization.Subscription.Interval;
                item.Text = Translate( item.Text );
            }
            if ( ! Page.User.Identity.IsAuthenticated ) {
                SubscribeArea.Visible = false;
            } else {
                Email.Text = PageBase.CurrentUser.UserData.Email;
            }
        }
    }

    protected void Save_Click( object sender, System.EventArgs e ) {
        EPiServer.Personalization.Subscription.Interval =
```

```
                System.Int32.Parse( Interval.SelectedItem.Value );
            if ( PageBase.CurrentUser.UserData.Email != Email.Text ) {
                PageBase.CurrentUser.UserData.Email = Email.Text;
            }
        }
    }
```

Example 6-24 shows how PageBase.CurrentUser can be used. In Page_Load is first used to assign the currently logged-in user's e-mail address to the Text property of the TextBox control Email (provided the user is logged on). Later, in Save_Click, there's a check to make sure that the logged-on user's e-mail address is the same as the text stored in Email.Text. If they are not the same, the user's data is updated.

EPiServer.Core.PageData

The PageData class contains information about a specific page. This includes the name of the page (PageName), its reference (PageLink), Url (LinkURL) and much more. All built-in and custom properties defined for the Page Type are available through the Property property.

You will notice that the PageData object is in many respects synonymous with the Web Page it's holding properties for. One example of this is the Changed property which holds last change date and time for the PageData object, i.e. the Web Page.

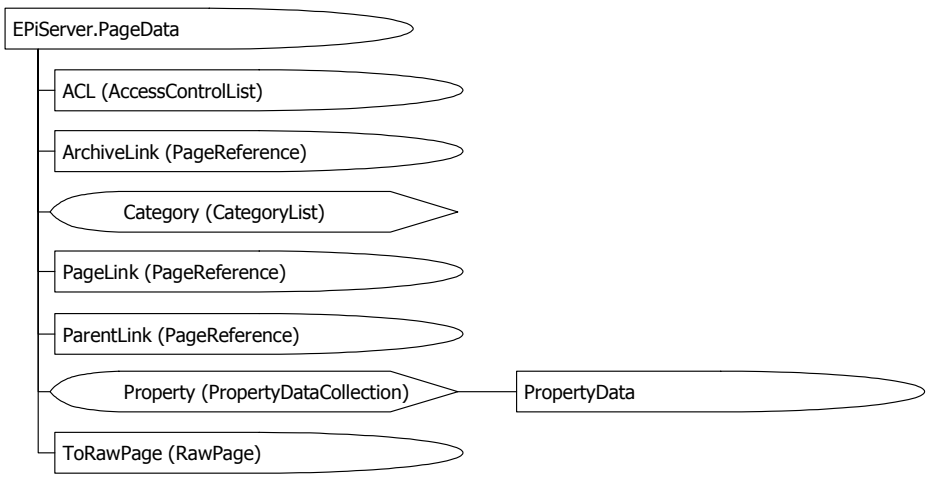


Figure 6-4: Object model for EPiServer.PageData.

There are many public properties defined for EPiServer.Core.PageData.

Table 6-11: Public properties for EPiServer.Core.PageData.

Property Name	*Description*
ACL	Access Control List, ACL (List of Access Permissions), of the type EPiServer.Security.AccessControlList
ArchiveLink	Reference to archive page.
Category	The categories that this page belongs to
Changed	The date on which the page was marked as changed; type is System .DateTime.
ChangedBy	The user name of the user who most recently marked the page as modified. For instance, if a Windows account was used to create the page, expect a string being returned looking like 'DOMAIN\User Name'.
Created	The date on which the page was created.
CreatedBy	The user name of the user who created the page (see ChangedBy).
Indent	Indent level if part of a tree structure.
IsDeleted	If page has been deleted.
IsModified	Check if page has been modified since load.
Item	Access the PropertyData.Value object of properties in the page object.
LinkURL	URL for this page.
PageLink	Reference to this page.
PageName	Display name of the page.
PageTypeID	Page type identifier.
PageTypeName	Name of the page's page type
ParentLink	Reference to parent page.
PendingArchive	Indicate whether the page should be moved to its archive folder
Property	Access property value.
Saved	The date on which the page was last saved.

Table 6-11: Public properties for EPiServer.Core.PageData.

Property Name	*Description*
StartPublish	The date on which the page was published.
StopPublish	The date on which the page will stop being published. It will be set to System.DateTime.MaxValue if no stop publish date has been set.
VisibleInMenu	Indicates whether this page should be visible in menus and tree structures. This setting is important for several of the control classes in EPiServer.WebControls.
WorkPageID	Page version identifier.

EPiServer.Core.PageData doesn't define very many public methods.

Table 6-12: Public methods for EPiServer.Core.PageData.

Method Name	*Description*
CheckPublishedStatus	Indicates whether the page is to be displayed based on publish date
Copy	Create a deep copy of the current object (as opposed to a shallow copy), in the Microsoft .NET sense
InitializeData	Give properties on the page a chance to initialize themselves with additional data
QueryAccess	Return the access level that the current user has to this page.
ToRawPage	Return the current instance converted to a RawPage object

More Information on Using the Public Properties and Methods in EPiServer.Core.PageData

EPiServer.Core.PageData.ACL

ACL holds the Access Control List for a PageData object. Since PageData has the attribute Property which is a PropertyDataCollection, ACL effectively controls access to the Web Page. Keep in mind that the Access Control List applies to all of the PageData object and its attributes. It is not possible to have different access permissions for different properties.

The Access Control List is comprised of an Access Control Entry, ACE, array and is accessed by calling the method ACL.ToRawACEArray. Don't let the word

'Raw' in the name intimidate you; if you've worked with Access Control Entries in Windows NT, these are very high-level indeed.

Table 6-13: Public fields for EPiServer.Core.PageData.ACL.

Public Fields	*Description*
AnonymousSidList	Static field, returning default SID list used for anonymous users, i.e. it contains Anonymous and Everyone.
FullAccess	Static field returning shortcut for Full access permission.
NoAccess	Static field returning shortcut for No access permission.

Table 6-14: Public properties for EPiServer.Core.PageData.ACL.

Public Properties	*Description*
Creator	Set/get the owner of the object that the ACL is attached to.

Public Methods

Table 6-15: Public methods for EPiServer.Core.PageData.ACL.

Public Methods	*Description*
Add	Add a SID / permission pair to the access control list.
Copy	Create a copy of the current instance
QueryAccess	Determine the access level for the current user.
QueryDistinctAccess	Checks specified access for CurrentUser, specified SID value or specified list of SID values.
ToRawACEArray	Returns an array of EPiServer.Security.RawACE objects that form the Access Control List.
TranslatableString	Returns a string that can be used in calls to Translate to get a language-specific representation of the access level.

Example 6-25: Using ACL.ToRawACEArray to enumerate the Access Control Entries.

```
foreach ( EPiServer.Security.RawACE Ace in CurrentPage.ACL.ToRawACEArray() ) {
    if ( ( Ace.Access & EPiServer.Security.AccessLevel.Create ) ==
        EPiServer.Security.AccessLevel.Create ) {
...
    }
}
```

The code in example 6-25 enumerates the EPiServer.Security.RawAce objects which together form the Access Control List and checks if one of them is the Create permission (compare this with the code in examples 6-26, 6-27 and 6-28).

Example 6-26: Using ACL.QueryDistinctAccess to check specific access for the current user.

```
if ( CurrentPage.ACL.QueryDistinctAccess( EPiServer.Security.AccessLevel.Create ) ) {
                                    // Checks whether the currently logged-on user has
                                    // Create permission for the current page.
...
}
```

Example 6-27: Using ACL.QueryDistinctAccess with CurrentUser.Sid.

```
if ( CurrentPage.ACL.QueryDistinctAccess( EPiServer.Security.AccessLevel.Create, CurrentUser.Sid ) ) {
                                    // CurrentUser has Create permission.
...
}
```

Example 6-28: Using ACL.QueryDistinctAccess with CurrentUser.SidList.

```
if ( CurrentPage.ACL.QueryDistinctAccess( EPiServer.Security.AccessLevel.Create, CurrentUser.SidList ) ){
                                    // CurrentUser is member of a group that has Create permission.
...
}
```

Example 6-29: Using EPiServer.Security.AccessControlList.AnonymousSidList.

```
foreach ( EPiServer.Security.RawACE Ace in CurrentPage.ACL.ToRawACEArray() ) {
    foreach ( int Sid in EPiServer.Security.AccessControlList.AnonymousSidList ) {
        if ( Ace.SID == Sid ) {
                                    // The SID found in the Access Contol Entry belongs
                                    // to the AnonymousSidList.
...
        }
    }
}
```

In example 6-29, the SID found in the Access Control List is compared against the contents of EPiServer.Security.AccessControlList.AnonymousSidList to determine whether the current user SID is one of the less fortunate.

EPiServer.Core.PageData.Changed

Date/time when the page was last changed.

Example 6-30: Using PageData.Changed.

```
string ChangedDateTime = CurrentPage.Changed.ToString( "r" );
```

EPiServer.Core.PageData.ChangedBy

User name of the user who last changed the page.

Example 6-31: Using PageData.ChangedBy.

```
Page changed by: <%# CurrentPage.ChangedBy %>
```

EPiServer.Core.PageData.Created, Saved and Changed

Date/time when the page was created. This happens mostly in EPiServer Edit mode, but pages can be saved when users create profile pages, fill in forms, etc.

PageData.Saved is the sibling of Created. It contains the date/time of last save operation.

Both Created and Saved are maintained by the EPiServer infrastructure – you have no control over them. The Changed, on the other hand, is only updated when a page is changed and CurrentPage.Property["PageChangedOnPublish"] .Value is True.

EPiServer.Core.PageData.CreatedBy

User name of the user who created the page and saved it for the first time.

EPiServer.Core.PageData.Indent

At first sight, Indent may seem to be of little value. But it is quite useful when displaying hierarchical tree structures. You will find that several of the Web User Controls that ship with EPiServer use Indent to visually enhance the hierarchical quality of tree structures.

Example 6-32: Using PageData.Indent with EPiServer.WebControl.Clear.

```
<EPiServer:PageTree runat="server" id="Pagedatatree1" PageLink=<%# CurrentNewsGroup %>
    EnableVisibleInMenu="false" ExpandAll="True" >
    <ItemTemplate>
...
        <EPiServer:Clear width='<%# ( Container.CurrentPage.Indent - 1 )*10 %>' runat="server" />
...
    </ItemTemplate>
</EPiServer:PageTree>
```

The listing in example 6-32 is an example of how Indent can be used in a visual context. For every new item in the PageTree control listing, the PageData.Indent property is used to create a transparent GIF picture that is 0, 10, 20 and so on pixels wide for the first, second and third level pages.

EPiServer.Core.PageData.Item and EPiServer.Core.PageData.Property

PageData Item is the indexer for the PropertyDataCollection PageData.Property. In C#, the indexers are never used literally; instead the name is replaced by the indexer parentheses pair. In the case of C#, this means the square brackets, '[' and ']'.

Example 6-33: Using the indexer PageData.Item and PageData.Property.

```
foreach ( string PropName in CurrentPage.Property ) {
    EPiServer.Core.PropertyData Prop = CurrentPage.Property[ PropName ];
    if ( Prop.Value != null ) {
                                    // Property exists and has been given a value.
...
    }
}
```

The code in example 6-33 is a good example of how to enumerate the PageData.Property collection. Due to a quirk in one of the Microsoft .NET base classes used to implement PageData.Property, it returns a string array consisting of the collections keys.

The easiest way to access properties in HTML is to use the EPiServer Web Custom Control Property (EPiServer.WebControls.Property). Read more about Property on page 191 and following pages.

EPiServer.Core.PageData.LinkURL

The LinkURL string property contains the page part of the URL for the page, i.e. to get a complete URL prepend LinkURL with EPiServer.Global.EPConfig.HostUrl.

Usage of LinkURL can be seen in example 6-3 on page 129).

EPiServer.Core.PageData.PageLink

Being of the type EPiServer.Core.PageReference, PageLink is the unique page ID. Its use is derived from the way in which EPiServer 4 is structured. Pages are nothing more than an instance of an EPiServer Page Type and a collection of Property settings stored in the database. Individual pages are identified by the pkID column in the database table tblPage and transforming this into a URL we get something looking like 'templates/emailpagecontainer.aspx?id=*n*?', where '*n*' is the already mentioned pkID and thus the unique page identifier.

Example 6-34: Using PageData.PageLink in HTML.

```
<EPiServer:PageList SortBy="PageName" DataSource=<%#PropertySearchControl%> runat="server"
    ID="PageListControl">
    <ItemTemplate>
...
                <input type="checkbox" id="ViewUser<%#Container.CurrentPage.PageLink%>"
```

```
                name="ViewUser<%# Container.CurrentPage.PageLink %>" />
...
    </ItemTemplate>
</EPiServer:PageList>
```

As PageLink is guaranteed to be unique among all pages in a certain EPiServer instance, it is used in 6-34 to create unique id's and names for HTML check boxes. The code is used to present the viewer with a selection of pages and allows her to check one or more for further processing. (The PageList control is explained in more detail on page 182 and following pages.)

EPiServer.Core.PageData.PageName

Not many page settings are under absolute editor control, but PageName is certainly one of them. Contrary to many a developer's gut feelings, PageName is not a 128 bit Globally Unique Identifier, GUID, but rather a descriptive name given the page in EPiServer Edit mode.

Having realised this, PageName can be used to great benefit in many cases, such as in templated controls.

Example 6-35: Using PageData.Item, PageData.LinkURL and PageData.PageName.

```
...
<EPiServer:Newslist ID=Newslistnew Pagelinkproperty="NewsContainer" runat="Server"
    MaxCount='<%# GetNewsCount() %>'>
    <NewsTemplate>
        <tr>
            <td class="DateListingText"><%# Container.CurrentPage[ "PageStartPublish" ] %></td>
        </tr>
        <tr>
            <td>
                <a href="<%# Container.CurrentPage.LinkURL %>" class="StartPageHeading">
                    <%# Container.CurrentPage.PageName %></a> <br />
                <span class="Normal">
                        <%# Container.CurrentPage[ "MainIntro" ] %>
                </span>
            </td>
        </tr>
    </NewsTemplate>
</EPiServer:Newslist>
...
```

The HTML code in example 6-35 produces two HTML table rows for every news item page displayed. The first row displays the content of the Property PageStartPublish and the second row is an HTML anchor tag where PageData.LinkURL is the relative URL for the page and PageData.PageName is used as

a new head line. Please note that PageData.Item is used twice in this HTML code, as the two attributes PageStartPublish and MainIntro are accessed using the indexer for PageData.Property.

EPiServer.Core.PageData.PageTypeID

PageTypeID is the same as column fkPageTypeID in the database table tblPage.

EPiServer.Core.PageData.PageTypeName

Name of the Page Type used to create this page. Same as the column Name in the database table tblPageType.

EPiServer.Core.PageData.ParentLink

EPiServer.Core.PageReference type which points to the parent page for this page. Parent- and childhood is determined by placement in the Web Page Tree.

Example 6-36: Using PageData.ParentLink and PageBase.GetChildren.

```
if ( CurrentPage.ParentLink != EPiServer.Core.PageReference.EmptyReference ) {
    EPiServer.Core.PageDataCollection siblings = GetChildren( CurrentPage.ParentLink );
...
}
```

PageData.ParentLink is used twice in example 6-36, first to make sure it contains a non-empty PageReference and secondly to retrieve the sibling of the current page.

Example 6-37: Using PageData.ParentLink.

```
Parent Page Name: <%= GetPage( CurrentPage.ParentLink ).PageName %>
```

The HTML code in example 6-37 results in the name of parent page being displayed. For the start page of a Web site, you expect 'Root' to be displayed.

EPiServer.Core.PageData.StartPublish and EPiServer.Core.PageData.StopPublish

StartPublish and StopPublish are again settings that are under editor control. Ordinarily you need never bother with them, as the EPiServer run-time system takes care not to publish pages before their StartPublish date/time and stops publishing pages after their StopPublish date/time. (See also PageData.CheckPublishStatus).

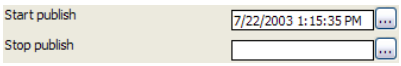


Figure 6-5: StartPublish and StopPublish are controlled from EPiServer Edit mode.

EPiServer.Core.PageData.VisibleInMenu

The setting PageData.VisibleInMenu controls visibility of this page when used with some Web Custom Controls in EPiServer.WebControls. The rule of thumb

is that menu lists and page lists do not display pages whose VisibleInMenu attribute is set to false. VisibleInMenu is stored in the column VisibleInMenu in database table tblPage.

Example 6-38: Using PageData.VisibleInMenu.

```
CurrentPage.VisibleInMenu = false;
```

The code in example 6-38 is taken from the Web User Control Profile.ascx, or rather its code-behind file, which is shipped with EPiServer. Setting PageData.VisibleInMenu to false is a preparation for storing the user profile page and making sure it won't be displayed with the other Web pages in the site's Web Page Tree, which would otherwise happen automatically.

EPiServer.Core.PageData.QueryAccess

QueryAccess returns an EPiServer.Security.AccessLevel object detailing the access level that the current user has to the PageData object. Be aware of the difference between PageData.ACL.QueryAccess and PageData.QueryAccess

EPiServer.Core.IPageSource

IPageSource is quite a lightweight interface. It contains only one public property and two public methods. The methods defined by this interface are used to retrieve page information about other pages in the page tree hierarchy of a site.

The IPageSource interface is implemented by many classes – such as EPiServer.PageBase (and its descendants), EPiServer.DataFactory, EPiServer.WebControls.PageControlBase (which many EPiServer Web Custom Control classes inherit from) to name a few. This means that there are ample opportunities for you to get to other pages or page information working with templates and controls.

IPageSource was created as an interface due to the fact that the GetPage and GetChildren methods normally have the same implementation independently of the class that implements IPageSource. The CurrentPage property has different meanings for different implementations. As an example, CurrentPage on the PageBase class refers to the currently loaded page (based on the id in the query string). CurrentPage on Web Custom Control PageList (see page 182) when it's used inside a template refers to the currently iterated page in the internal PageData collection.

Table 6-16: Public properties for EPiServer.IPageSource.

CurrentPage	Returns information about the currently loaded page, or a page in a collection when used inside a (templated) control.

Table 6-17: Public methods for EPiServer.IPageSource.

GetChildren	Returns a collection of pages directly below the page referenced by the EPiServer.Core.PageReference parameter.
GetPage	Retrieves a PageData object with information about a page, based on the EPiServer.Core.PageReference parameter.

More Information on Using the Public Properties and Methods in EPiServer.IPageSource

As IPageSource is an interface, it doesn't implement any of its own attributes or methods itself. It's a blueprint, not a construction. This means we must always be aware of which class implementing IPageSource we're dealing with. For example, PageBase.CurrentPage returns an EPiServer.Core.PageData object for the current page, the same for UserControlBase. However, in a templated control such as EPiServer.WebControls.PageList CurrentPage instead means the current page in the iteration.

EPiServer.Global

The Global class is another of the most important classes in EPiServer. This class is the one used by global.asax for initialisation of the site. The Global class reads the configuration from disk, sets up the EPiServer.Core.LanguageManager and registers the authentication modules that are to be used for this site.

As the Global class is globally available, you can use it by just referring to it like this:

Example 6-39: Using EPiServer.Global.

```
string siteName;
siteName = EPiServer.Global.EPConfig.SiteName;
```

If you check the global.asax.cs file on an EPiServer site, you'll find that the local Global class inside it inherits from EPiServer.Global. This is a requirement for the site to work.

Table 6-18: Public Properties

BaseDirectory	The physical root folder where the site is installed, e.g. 'd:/INetPub/WWWRoot/Example/', when using a virtual Internet Information Services, IIS, folder. Please note that slashes are used instead of back slashes.
EPConfig	Global instance of configuration information.
EPDataFactory	Global instance of data factory, i.e. EPiServer.DataFactory.
EPLang	Global instance of language information

Table 6-18: Public Properties

InstanceName	IIS Metabase key path for the application, e.g. '_LM_W3SVC_1_ROOT_Example'. Notice that underscore is used in place of slash.

The Static Properties BaseDirectory, EPConfig, EPDataFactory, EPLang and InstanceName

EPiServer.Global's five static properties are the preferred way to access certain application-wide settings. Two of these properties, BaseDirectory and InstanceName, are merely string variables containing the local path (on the Web server) for the installation folder and the IIS Metabase key path, respectively.

EPiServer.Global.EPConfig

EPConfig is an instance of class EPiServer.ApplicationConfiguration, see below. It's instantiated by Global.asax. A very important use for EPiServer.EPConfig is derived from its ability to read, and write, settings in the application's web.config file.

Example 6-40: Using EPConfig to display information about the start page for the Web site.

```
<tr><td>ID of Start Page: <%= EPiServer.Global.EPConfig.StartPage.ID %></td></tr>
```

EPiServer.Global.EPDataFactory

The EPDataFactory, of the type EPiServer.DataFactory, member is the preferred way to access the DataFactory class from your code. Do not instantiate your own DataFactory classes if you are not certain that you have to. A DataFactory object not created by EPiServer will not use the cache, and the **performance** will be significantly **impeded**. EPiServer.DataFactory is discussed at length in chapter 9, *Data Modelling*.

EPiServer.Global.EPLang

Perhaps the main use for the EPLang property is to translate EPiServer strings into the current language. EPLang is an instantiation of EPiServer.Core.LanguageManager, which has one property, Directory, and seven methods. It is the Translate method that is primarily used.

Example 6-41: Using EPLang.Translate in a Web User Control (the ascx file).

```
<title><%=EPiServer.Global.EPLang.Translate( "/editor/tools/font/toolheading" )%></title>
```

Example 6-42: Using EPLang.Translate in JavaScript.

```
restartConfirmed = confirm( '<%= EPiServer.Global.EPLang.Translate( "/editor/tools/search/
reachedend" ) %>');
```

Translate has two overloaded variants. The first one is used in both example 6-41 and example 6-42. It takes a string argument, key, in a simplified XML XPath

form and returns the proper language string. The key is typically something like '/admin/settings/heading', where the expression closely follows the folder/file/usage pattern. You can also enter a '#' to automatically construct a path to the current file. For example, calling the method Translate in the file /templates/mypage.aspx with key set to '#heading', Translate("#heading") is equivalent to Translate("/templates/mypage/heading").

Notice that if the key does not begin with a '/' or '#', the key itself is simply returned as the result. The reason for this behaviour is to be able to use EPiServer Web Controls that use Translate for visible strings but you might not have translations in place or you prefer not to translate the text.

The second form of Translate takes two string arguments, the first again being a key used in the same manner as earlier, and the second argument being a language identifier.

Example 6-43: Using EPLang.Translate in code.

```
string LangName = EPiServer.Global.EPLang.Translate( "#invalidsettings", GetSystemLanguage() );
```

The string function GetSystemLanguage is a local function that returns either the local language setting for the page, if any exists, or the language setting for the site, in EPiServer.Global.EPConfig["EPsLanguage"].

EPiServer.ApplicationConfiguration

This class is used to save/load information from the applications configuration file. You can access this object directly from a template page using the Configuration property (defined in PageBase).

If you add your own values to the <appSettings> section in web.config, you can read those through the Item property (the indexer).

You can also get the configuration settings from the global object EPConfig, defined as a static member (Shared in VB.NET) in the Global class.

When you make any changes to the configuration settings, remember to call Persist to save the changes.

Almost all of the settings are available through the Item method / indexer. Some of the common settings are also exposed with typed properties.

The interfaces support generic objects. However, the internal implementation only supports data types string, integer and boolean. Be aware of the fact that accessing a setting that doesn't exist is not an error but will simply return a default value. This could lead to a situation where you think a setting has changed but cannot be read back.

Adding Your Own Settings

If you look in the appSettings sections of an EPiServer solution web.config file, you will no doubt notice that there's a prefix scheme at work.

The configuration object names have the syntax 'EP*x*Nomen', where 'EP' is a required string literal, the '*x*' is one of 'f', 'n' or 's' which indicates data type and '*Nomen*' is the name of the object. The following table lists the data types and their default values.

Table 6-19: Prefixes used for settings in the appSettings section of web.config file.

Key Prefix	*Data Type*
EPf	Boolean (flag). Default value is false.
EPn	Numeric value. Default value is 0 (zero).
EPs	System.String. Default value is System.String.Empty.

This prefix scheme is not simply a mnemonic for the developer, it has actual bearing on the data type for the setting.

Example 6-44: Testing the prefix scheme in web.config.

```
EPnRootID: <%= EPiServer.Global.EPConfig[ "EPnRootID" ].GetType().ToString() %>
EPsRootDir: <%= EPiServer.Global.EPConfig[ "EPsRootDir" ].GetType().ToString() %>
EPfEnableUnicode: <%= EPiServer.Global.EPConfig[ "EPfEnableUnicode" ].GetType().ToString() %>
LicensedCompany: <%= EPiServer.Global.EPConfig[ "LicensedCompany" ].GetType().ToString() %>
```

Putting the code from example 6-44 in the HTML part of a page yields the following results:

Example 6-45: Output from HTML code in example 6-44.

```
EPnRootID: System.Int32
EPsRootDir: System.String
EPfEnableUnicode: System.Boolean
LicensedCompany: System.String
```

In other words: prefixing settings in web.config with 'EPn' turns the setting into a true integer value ('int' in Visual C# .NET is shorthand for System.Int32), and so on. Thus, when adding your own settings to web.config, be sure to use the proper prefix. Without the prefix schemes, EPiServer doesn't meddle so those settings retain the System.String data type.

Use EPiServer.ConfigFileSettings to Create New Settings

In order to add a new setting to the web.config file from code, you use the SetAppSetting method defined for the EPiServer.ConfigFileSettings class.

Example 6-46: Adding the setting EPnImportantValue to the web.config file.

```
int ImportantValue= 17;
...
EPiServer.Global.EPConfig.ConfigFile.SetAppSetting( "EPnImportantValue", ImportantValue.ToString() );
```

```
EPiServer.Global.EPConfig.ConfigFile.Persist();
```

Settings May Be Encrypted

It's easy to protect individual settings in web.config by encryption. When the setting is created simply prepend its value with 'encrypt:' and the value will be stored in encrypted form. Encryption is then transparent to your code; decryption will be taken care of by the EPiServer infrastructure.

Example 6-47: Adding the setting EPsInnocuousValue to the web.config file.

```
string SecretKey= "!#¤%/";
...
EPiServer.Global.EPConfig.ConfigFile.SetAppSetting( "EPsInnocuousValue", "encrypt:" + SecretKey );
EPiServer.Global.EPConfig.ConfigFile.Persist();
```

Public Properties and Methods for EPiServer.ApplicationConfiguration

Table 6-20: Public properties for EPiServer.ApplicationConfiguration.

Property Name	*Description*
Authentication	Gets the authentication mode that is in use for the application.
BackendSite	Name of this site when communicating with other servers
CacheListeners	Site names that listens to cache changes
CachePolicyTimeout	The timeout policy for content
CacheSpinlockCount	The number of attempts that the page cache will make to wait for pending read operations.
CacheSpinlockTimeout	The timeout in milliseconds for each pending read.
CacheVaryByCustom	Vary by custom values resolved in global.asax.
CacheVaryByParams	Vary by query string parameters.
ConfigFile	Get the ConfigFile used to save / load the configuration settings.
ConnectionString	The connection string to use for accessing the local database. Stored in the EPsConnection setting in appSettings in web.config.
CookieRequireSSL	Get information about the RequireSSL setting

Table 6-20: Public properties for EPiServer.ApplicationConfiguration.

Property Name	*Description*
CssPath	Path to CSS style file.
DisableKeepUserLoggedOn	If users should be kept logged on.
EPiServer3Compability	Returns true if the system is running in EPiServer 3 compatibility mode.
HasAdminAccess	Tests whether the current user has access to EPiServer admin mode
HasEditAccess	Tests whether the current user has access to edit mode
HostUrl	Returns a URL which points to this host.
IsDirty	Checks whether changes have been made to the configuration settings.
IsSensitive-InformationEncrypted	Checks whether all sensitive system settings are encrypted.
Item	Access individual configuration settings. Item is the indexer for ApplicationConfiguration.
LocalSite	Name of this site when communicating with other servers.
LoginUrl	Returns the url to the file used for form-based authentication.
LogoffTimeout	Gets the inactivity timeout (in minutes) used for forms authentication.
PageCacheTimeout	Number of hours to store pages in cache
PhysicalUploadDir	Physical folder for uploaded files. To get the virtual path, use the UploadDir property.
QueryTimeout	The time (in seconds) to wait for database queries to execute.
RemotePageCacheTimeout	Number of hours to store remote pages in cache
RootDir	Root folder for site.
RootPage	Root page for this instance.

Table 6-20: Public properties for EPiServer.ApplicationConfiguration.

Property Name	*Description*
SafeHtmlTags	An array of HTML tags considered safe for public use.
SiteName	Name of site.
StartPage	Start page for this instance.
UploadDir	Virtual folder for uploaded files. To get the physical path, use the PhysicalUploadDir property.
ValidatePageTemplate	Returns true if the system should validate that the page template matches the specified template.
Version	Gets the assembly name and version.
Wastebasket	Wastebasket for system.

Table 6-21: Public methods for EPiServer.ApplicationConfiguration.

Method Name	*Description*
Exists	Checks whether the setting has an explicit value set.
InitSmtpServer	Initializes the SMTP server which will be used by all mail functions. Call this function before sending mail using the System.Web.Mail.SmtpMail.Send() function to be certain that the correct SMTP server is used. The default SMTP server name can be defined in the EPsSmtpServer key in web.config. If not defined, 'localhost' is used.
IsEncrypted	Tests whether the configuration value is stored in encrypted form
Persist	Save any changes to the configuration file.
SwitchToTest	Change configuration entries to their test counterparts.

More Information on Using the Public Properties and Methods in ApplicationConfiguration

All the examples below use either EPiServer.EPConfig, PageBase.Configuration or UserControlBase.Configuration. Don't be alarmed, there's only one set of configuration settings. The reason for having multiple access paths is simply one of convenience, to allow you to choose the most convenient and appropriate path depending on the context.

EPiServer.ApplicationConfiguration.Authentication

Authentication is read-only and of the type System.Web.Configuration.AuthenticationMode and can thus be one of Forms, None, Passport or Windows.

Example 6-48: Using read-only attribute EPiServer.Global.EPConfig.Authentication.

```
...
protected System.Web.UI.WebControls.LinkButton   Logout;
...
Logout.Visible = ( EPiServer.Global.EPConfig.Authentication == AuthenticationMode.Forms );
```

In example 6-48, the read-only attribute Authentication is used to control whether the LinkButton Logout should be visible or not. (If Forms authentication is active, there has been an explicit log-on to the Web site and therefore a log-off is conceivable.)

Since the code from example 6-48 was taken from a Web User Control inheriting from EPiServer.UserControlBase, it could as well have been written like this:

Example 6-49: Using read-only attribute UserControlBase.Configuration.Authentication.

```
...
protected System.Web.UI.WebControls.LinkButton   Logout;
...
Logout.Visible = ( Configuration.Authentication == AuthenticationMode.Forms );
```

EPiServer.ApplicationConfiguration.ConfigFile

The ConfigFile attribute is the structured way of getting access to the contents of the web.config file.

ConfigFile has a type of EPiServer.ConfigFileSettings, which in turn has a few attributes and methods itself. One of these attributes is AllAppSettings, which is System.Collections.Specialized.NameValueCollection holding all settings in the appSettings sections.

Example 6-50: Displaying the name of all settings in the appSettings section–code-behind file.

```
protected   System.Web.UI.WebControls.DataListConfigFileSettings;

    private void Page_Load( object sender, System.EventArgs e ) {
        if ( ! IsPostBack ) {
            DataBind();
            ConfigFileSettings.DataSource = EPiServer.Global.EPConfig.ConfigFile.AllAppSettings;
            ConfigFileSettings.DataBind();
        }
    }
```

Example 6-51: Displaying the name of all settings in the appSettings section–HTML part.

```
<asp:DataList id="ConfigFileSettings" runat="server" >
    <HeaderTemplate>
            Configfile settings.
    </HeaderTemplate>
    <ItemTemplate>
        <asp:label Text='<%# DataBinder.Eval( Container, "DataItem" ) %>'
            id="Label1" runat="server" />
    </ItemTemplate>
</asp:DataList>
```

The code in example 6-50 and 6-51 used together in a Web Form produce a listing of all the names of settings in the appSettings section in the web.config file.

EPiServer.ApplicationConfiguration.HostUrl

HostUrl is used in example 6-3 on page 129 and 6-15 on page 135. Both those examples could be rewritten to use the 'other' route to access HostUrl.

Example 6-52: Using EPiServer.Global.EPConfig instead of Configuration in the same code as 6-3.

```
    mail.Body       += "\n\n" + EPiServer.Global.EPConfig.HostUrl + CurrentPage.LinkURL;
...
    EPiServer.Global.EPConfig.InitSmtpServer();
```

Example 6-53: Using Configuration instead of EPiServer.Global.EPConfig in the same code as 6-15.

```
...
        itemLink.InnerText= Configuration.HostUrl + ChildPageData.LinkURL;
...
```

EPiServer.ApplicationConfiguration.RootDir

RootDir is one of the configuration settings you'll probably use all the time. It saves you from having to hard-code folder paths, which in itself justifies the presence of this setting.

Example 6-54: Using RootDir to specify location of style sheet.

```
<link rel="stylesheet" type="text/css" href="<%= EPiServer.Global.EPConfig.RootDir %>util/styles/
login.css">
```

Example 6-55: Using RootDir to specify location of image.

```
<img src="<%#EPiServer.Global.EPConfig.RootDir%>images/portal/img.gif" />
```

Example 6-56: Using RootDir to specify location of image.

```
link.InnerText = EPiServer.Global.EPConfig.HostUrl + EPiServer.Global.EPConfig.RootDir;
```

EPiServer.ApplicationConfiguration.RootPage and StartPage

RootPage and StartPage are probably used even more than RootDir. They are very important, as they provide shortcuts for the root page of the Web Page Tree and the start page of the Web Page Tree, respectively. These two shortcuts are used when listing pages in templated control, searching for information on Web pages and in many other instances.

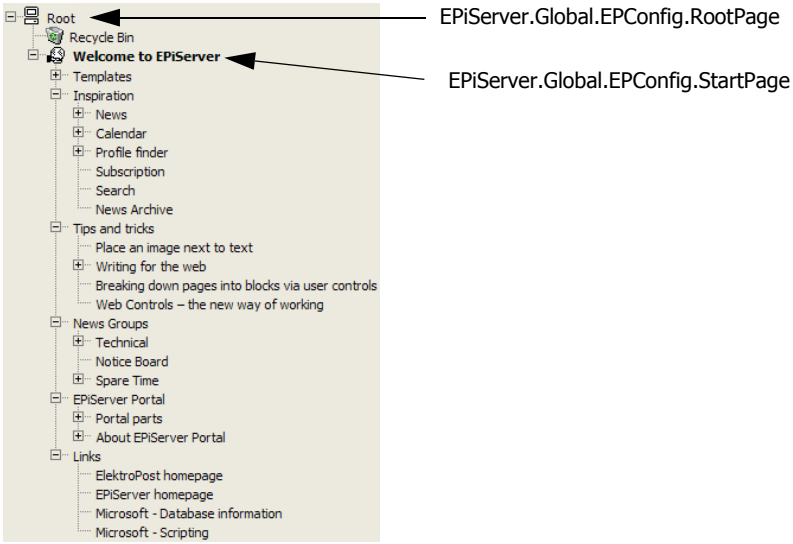


Figure 6-6: RootPage and StartPage for the example Web site Web Page Tree.

Figure 6-6 shows EPiServer.Global.EPConfig.RootPage and StartPage for the example Web site. You can always assume that the RootPage will point to the uppermost page in the Web Page Tree and that StartPage will point to the page which is the default start page for the Web site.

EPiServer.ApplicationConfiguration.Exists

When you add configuration settings of your own to web.config, it's probably a good idea to include a call to Exists before reading its current value. In the unlucky event that the expected settings are in fact not present in web.config, Exists will help you catch this information, as simply trying to read it will only result in the default value being returned (see page 156 for default values for settings).

Example 6-57: Using EPiServer.EPConfig.Exists before reading a setting in web.config.

```
if ( EPiServer.Global.EPConfig.Exists( "EPnImportantValue" ) ) {
    ImportantValue = EPiServer.Global.EPConfig[ "EPnImportantValue" ];
} else {
    // Do something appropriate.
}
```

EPiServer.ApplicationConfiguration.InitSmtpServer

See example 6-3 on page 129 for use of InitSmtpServer.

EPiServer.ApplicationConfiguration.IsEncrypted

Use IsEncrypted to test whether configuration setting has been encrypted. As such, your code doesn't have to take the possibility of encrypted values into account, as they are silently decrypted before your code tries to access them.

7
EPiServer Web Controls

EPiServer Web Controls are a number of ASP.NET Web Custom Controls collected in the name space EPiServer.WebControls. ASP.NET Web Customs Controls differ from ASP.NET Web User Controls in several ways. One is the fact that Web Custom Controls are created solely in code; there's no visual part (no ascx file).

It's a safe bet to state that there is a definite pattern to the control types found in the name space EPiServer.WebControls. Very many of them deal with the Web Page Tree that's created in the EPiServer Edit mode and indeed make up your Web site. Thus, you'll find control classes that deal with creating menus from part of or all of the Web Page Tree, as well as control classes used to create various kinds of lists. Most of these controls are templated controls giving you excellent control of their visual behaviour without the need for code in the code-behind file. (ASP.NET templated controls are discussed below.)

Table 7-1: EPiServer ASP.NET Web Custom Controls

Class	*Description*
Calendar	Displays a template-based calendar where the calendar events are pages in EPiServer.
CalendarEventTemplate-Container	Template for a Calender Event.
CategoryTemplate-Container	Used for template context at category level.
CategoryTree	Renders a tree of categories from a given root category.
ChangedPages	Lists changed pages for the specified root page and all child pages.
Clear	Simple control to put an HTML Img tag with a transparent GIF picture of the given dimensions.
Content	Content in ContentFramework on a Web Form that will populate controls to Region.

Table 7-1: EPiServer ASP.NET Web Custom Controls

Class	*Description*
ContentFramework	Provides support for visual inheritance. A skeleton user control populated with Region controls, the contents of which can be replaced by Content controls contents in a ContentFramework on a Web Form.
ContentFramework-Preview	Used to create a miniature model of a table structure.
ContentFramework-Selector	Select framework to use when multiple frameworks are defined
DayTemplateContainer	Template for rendering information about a single day.
DirectoryTemplate-Container	Template class used by FileTree to display directories
DynamicCell	A TableCell whose width can be changed.
DynamicResizeCell	A TableCell that can be used to change width on DynamicCells.
DynamicResizeRow	A TableRow that can be used to change width on DynamicRows.
DynamicRow	A TableRow that can change height.
DynamicTable	A Table that contains resizable Rows and Cells. These can be personalized and saved by authenticated users.
ExplorerTree	Displays a tree Explorer style.
FileTemplateContainer	Template class used by FileTree to display files
FileTree	Displays a file tree from the server.
FormFieldStatistic	Helper class to keep track of information for a field.
FormPostings	Gets a list of form postings as created by the PropertyForm control
FormStatistics	Used to create a statistics view of data posted with a PropertyForm control
InputBase	Base class for simple input types.

Table 7-1: EPiServer ASP.NET Web Custom Controls

Class	*Description*
InputCategoryTree	Simple input type that displays a page selector.
InputEditorOptions	Simple input type that displays editor options.
InputFrame	Simple input type that displays a frame drop-down list.
InputLanguage	Simple input control used to select a language
InputPageDefinitionType	Simple input type that displays a page definition type drop-down list.
InputPageReference	Simple input type that displays a page selector.
InputPassword	Simple input control that allows a user to enter / change passwords
InputSortOrder	Simple input type that displays available sort orders.
InputTab	Simple input type that displays a tabbed drop-down list.
InputTimeSpan	Simple input type that displays a time span.
MenuList	Renders a menu list of top level items; useful for navigations that contain a top level menu displaying subtrees as clicked.
NewsList	Control for rendering news list with specialised template for top level news.
PageControlBase	Serves as a base class for all controls that generate any type of PageData collection.
PageList	Control for rendering page list; extends PageListData with templates.
PageListData	Base data control for accessing page list.
PageSearch	WebControl that handles text searches against EPiServer pages in the database and files indexed by Microsoft Index Server.
PageTemplateContainer	Used for template context at page level.

Table 7-1: EPiServer ASP.NET Web Custom Controls

Class	*Description*
PageTree	Control for rendering page trees; extends PageTreeData with templates.
PageTreeData	Base data control for accessing page trees.
PortalFramework	A Web control that renders a table structure containing IFrames. This control will load an XML file which has to be located in the /Util/PortalFramework folder.
PortalRegion	A region in ContentFramework which can be populated with controls from a Content.
Property	WebControl for rendering page properties.
PropertyCriteriaControl	Holder of criteria information used by property searching.
PropertyCriteriaControlBuilder	Builds child controls of the type PropertyCriteriaControl.
PropertySearch	Advanced property search in the complete database.
PropertyTemplateContainer	Used for template context at property level.
Region	A region in ContentFramework that can be populated with controls from a Content.
ResetDynamicTableButton	A LinkButton that will remove any personalized values from all DynamicTables on the page.
SaveDynamicTableButton	A LinkButton that will collect and save changed values to height and width for all DynamicTables on the page.
SiteMap	Displays a site map using templates and predefined design options.
SubscriptionList	Displays a list of subscription options.
Translate	WebControl for language specific strings.
XmlNameValidator	Checks that the validated controls value conforms with naming rules for an XML identifier

Inheritance Tree for EPiServer.WebControls

The inheritance tree for EPiServer.WebControls is quite extensive.

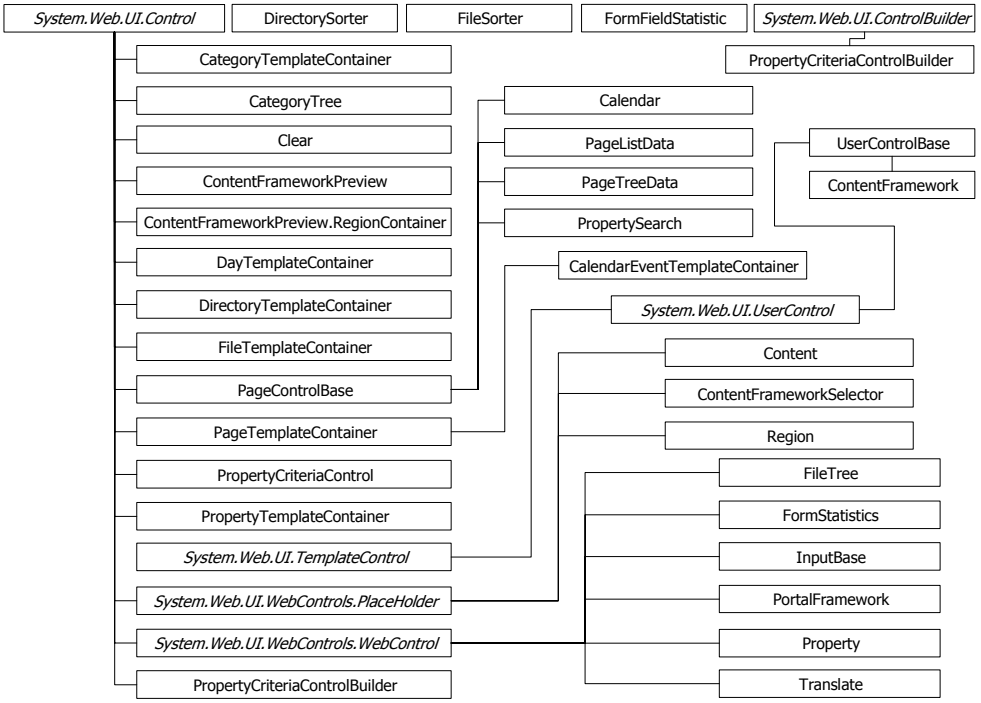


Figure 7-1: Inheritance Tree for EPiServer.WebControls name space, first three levels shown for all, more for UserControlBase and ContentFramework.

All of the five top level classes inherit directly from System.Object. The inheritance tree for EPiServer.WebControls is at most seven levels deep (including System.Object). Only the classes EPiServerValidator and XmlNameValidator have a seven level deep inheritance chain: (not shown in figure 7-1).

ASP.NET Templated Controls

ASP.NET Templated Controls represent an elegant way to separate presentation and data whilst keeping some connection between the two. In short, templated controls offer HTML template tags with given names. Each control class can define their own names, which are handled by code in the control's class.

For example, the built-in DataList server control (System.Web.UI.WebControls.DataList) defines these template names:

- HeaderTemplate
- FooterTemplate

- ItemTemplate
- AlternatingItemTemplate
- SeparatorTemplate
- SelectedItemTemplate
- EditItemTemplate

Visual Studio .NET has support for the visual layout of templated controls, but there's no problem adding them directly to a Web Forms page. This example comes from the help file:

Example 7-1: Using templated control DataList.

```
<asp:datalist ID="DataList1" runat="server">
    <HeaderTemplate>
        Employee List
    </HeaderTemplate>
    <ItemTemplate>
        <asp:label id=Label1 runat="server" Text='<%# DataBinder.Eval( Container,
            "DataItem.EmployeeName") %>' />
        <asp:label id=Label2 runat="server" Text='<%# DataBinder.Eval( Container,
            "DataItem.PhoneNumber") %>' />
        <asp:Hyperlink id=Hyperlink1 runat="server"
            Text='<%# DataBinder.Eval( Container, "DataItem.Email") %>'
            NavigateURL='<%# DataBinder.Eval( Container, "DataItem.Link") %>' />
    </ItemTemplate>
</asp:datalist>
```

Templated Controls Have an Imaginary Foreach Statement

It is possible to envisage a C# language foreach-statement enclosing parts of the templated control. Rewriting the example above results in this:

Example 7-2: Pseudo code for templated control with an imaginary foreach statement.

```
<DataList control>
    <!-- ItemTemplate -->
    foreach ( object DataItem in Container.DataItems ) {
        use DataBinder.Eval, Container and Container.DataItem
    }
    <!-- ItemTemplate -->
</DataList>
```

The Container Property

ASP.NET templated controls seem to have this mysterious property Container which is used inside the control (you can see it in both example 7-1 and 7-2).

'Container' is actually a code-generated property which refers to the individual row/item in the outside control. As we are using DataList in the example, Container is a DataListItem. Container only exists for autogenerated templates, and you must implement it yourself in the ITemplate class. Generally, it's the NamingContainer of the control you get sent to instantiate into. Luckily, this has been done in all of EPiServer templated controls.

EPiServer Templated Controls

You can make good use of Templated Controls when developing solutions with EPiServer. Most of the controls in EPiServer.WebControls are templated controls. We'll take a look at NewsList (EPiServer.WebControls.NewsList). This control class defines these templates:

- FirstNewsTemplate, template for the first news item
- SecondNewsTemplate, template for the second news item
- ThirdNewsTemplate, template for the third news item
- FourthNewsTemplate, template for the fourth news item
- NewsTemplate, default template for news items

In addition to the listed templates, there are also FooterTemplate and HeaderTemplate, with obvious uses.

An example of how to use this control in a Web Form or a User Control might look like this:

Example 7-3: Using EPiServer templated control NewsList.

```
<EPiServer:Newslist ID=Newslistnew Pagelinkproperty="NewsContainer" runat="Server"
    MaxCount='<%# GetNewsCount() %>'>
...
    <NewsTemplate>
        <tr>
            <td class="DateListingText"><%# Container.CurrentPage[ "PageStartPublish" ] %></td>
        </tr>
        <tr>
            <td>
                <a href="<%# Container.CurrentPage.LinkURL %>" class="StartPageHeading">
                    <%# Container.CurrentPage.PageName %></a> <br />
                <span class="Normal"><%# Container.CurrentPage[ "MainIntro" ] %></span>
            </td>
        </tr>
        <tr><td><EPiServer:Clear height="6" runat="server" /></td></tr>
    </NewsTemplate>
</EPiServer:Newslist>
```

The code in example 7-3 uses just NewsTemplate, meaning that all news items will look the same in the list. Each news item will be displayed inside an HTML table using three rows each. On the first row of the table, the contents of the PageStartPublish property for the page are displayed. The second row will show the contents of the MainIntro property and the third row is simply a spacer to offset the news items vertically from each other. The use of classes in the Table Data tag and the Span tag is linked to the Cascading Style Sheet used. Looking at class DateListingText in the style sheet actually used, we find that it is set using a different foreground colour (#606060) and font (7 pt, italic).

Figure 7-2: Actual news items presented using the templated control NewsList.

Comparing the code in 7-3 with the news items in figure 7-2 reveals a high degree of correspondence between the two.

In the resulting HTML code, this news list will be distinguished by the first news item being the only one which is displayed.

The Container Property in EPiServer Templated Controls

For the EPiServer templated controls in EPiServer.WebControls, the natural naming container is hosted on a Web page most of the time. This means that the Container property has a type of EPiServer.WebControls.PageTemplateContainer. For Calendar events, EPiServer.WebControls.Calendar, the Container property type is EPiServer.WebControls.CalendarEventTemplateContainer.

EPiServer.WebControls.Clear

Not many EPiServer controls are designed to produce invisible results, but that's exactly what Clear does. Including an object of the Clear class in HTML produces an invisible, transparent, area of specified dimensions using a transparent GIF image. Its primary use is to avoid cluttering, to make the visual impression of the Web page lighter.

Example 7-4: Clear control object using in HTML.

```
<%@ Register TagPrefix="EPiServer" Namespace="EPiServer.WebControls" Assembly="EPiServer" %>
...
<EPiServer:Clear Width="150" Height="10" runat="server" />
```

The Clear control object included in example 7-4 would result in the final rendering of a transparent GIF image with the specified dimensions.

Example 7-5: HTML IMG tag equivalent to Clear control object.

```
<img src="/util/images/clear.gif" alt="" width="150" height="10" border="0">
```

The HTML IMG tag in example 7-5 and the EPiServer.WebControls.Clear control object in example 7-4 are equivalent.

If you don't specify either or both of Height or Width, they will assume their default values of '1'.

Example 7-6: Clear control object using in HTML.

```
<%@ Register TagPrefix="EPiServer" Namespace="EPiServer.WebControls" Assembly="EPiServer" %>
...
<EPiServer:Clear Height="10" runat="server" />
```

The Clear control object in example 7-11 produces a GIF image which is 10 pixels high and 1 pixel wide.

As can be seen in figure 7-1, Clear is a direct descendant of System.Web.UI .Control; it only adds the Height and Width attributes.

The Visible attribute can be used to decide whether or not to actually render an image.

Example 7-7: Using HasChildren attribute to control attribute Clear.Visible.

```
<%@ Register TagPrefix="EPiServer" Namespace="EPiServer.WebControls" Assembly="EPiServer" %>
...
<EPiServer:Clear Visible=<%# Container.HasChildren %> Width="7" Height="7" runat="server" />
```

EPiServer.WebControls.Content

Objects instantiated from the Content control class are always used in conjunction with Region control object (read more about the Region class on page 197, onwards). Content objects define and name an area in the HTML landscape which can be claimed by Region objects simply by stating the name of the pertinent area. There's a clear distinction between Content and Region objects though: Region objects are used in Framework Definition Files and Content objects are found in Page Template Files. Together, they form the building blocks of the EPiServer Content Framework.

Two of the attributes in the Content class are important:

Table 7-2: The two important attributes in EPiServer.WebControls.Content.

Attribute Name	*Purpose*
ID	Unique ID for this object (this particular instance of Content) [string].
Region	Name of the Region whose contents will be replaced by the contents of Content [string].

Using Content objects is easy:

Example 7-8: Using Content control in a Page Template File.

```
<%@ Page … Inherits="development.Default" %>
<%@ Register TagPrefix="EPiServer" Namespace="EPiServer.WebControls" Assembly="EPiServer" %>
<%@ Register TagPrefix="development" TagName="DefaultFramework … %>
…
<development:DefaultFramework id="DefaultFramework" runat="server">
    <EPiServer:Content ID="NewsListing" Region="menuRegion" runat="server">
        <table>
                <tr>
                    <td>
                        <EPiServer:Newslist ID=Newslistnew Pagelinkproperty="NewsContainer"
                                runat="Server" MaxCount='<%# GetNewsCount() %>'>
                            <Newstemplate>
                                <tr><td><%# Container.CurrentPage[ "MainIntro" ] %></td></tr>
                            </Newstemplate>
                        </EPiServer:Newslist>
                </td>
            </tr>
        </table>
    </EPiServer:Content>
…
</development:DefaultFramework>
```

Comparing the HTML code in example 7-8 with that in example 7-30 on page page 197, we see that the Web User Control Menu.ascx that is used in example 7-30 is replaced by a very simple news listing in example 7-8.

Even if you're working with nested Region controls, there's no need to nest the Content controls, as they will replace the proper Region control contents anyway.

Remember, in the C# world many strings are case-sensitive. To improve portability always treat them as such, no matter which programming language you are using.

EPiServer.WebControls.ContentFramework

The only purpose for EPiServer.WebControls.ContentFramework is to act as a base class for classes in Framework Definition Files. Remember that Framework Definition Files are ASP.NET Web User Controls (they're just not used like that in the EPiServer Content Framework), a point which is made even clearer by the fact that ContentFramework inherits EPiServer.UserControlBase.

Example 7-9: Code-behind file from a Framework Definition File.

```
namespace development.Frameworks {
                                /// <summary>SimpleFramework is a simple Framework Definition
                                /// File.</summary>
    public abstract class SimpleFramework : EPiServer.WebControls.ContentFramework {
        protected EPiServer.WebControls.RegiontopRegion;
        protected EPiServer.WebControls.Region leftColumn;
        protected EPiServer.WebControls.Region centreRegion;
        protected EPiServer.WebControls.Region footerRegion;

        private void Page_Load( object sender, System.EventArgs e ) {
                                // Put user code to initialize the page here
        }

        // Web Form Designer generated code
    }
}
```

EPiServer.WebControls.ContentFrameworkSelector

Its name is a dead give-away: ContentFrameworkSelector enables you to adapt to different Framework Definition Files, e.g. ordinary Framework Definition Files and those which have portal functionality.

In the example Web site, the Page Template File Calendar.aspx uses Content-FrameworkSelector to render either of two Web User Controls: Calendar.ascx or PortalCalendar.ascx.

Example 7-10: ContentFrameworkSelector in Calendar.aspx.

```
<%@ Page language="c#" Codebehind="Calendar.aspx.cs" AutoEventWireup="false"
Inherits="development.Templates.calendar" %>
<%@ Register TagPrefix="EPiServer" Namespace="EPiServer.WebControls" Assembly="EPiServer" %>
<%@ Register TagPrefix="development" TagName="DefaultFramework"
    Src="~/templates/Frameworks/DefaultFramework.ascx"%>
<%@ Register TagPrefix="development" TagName="PortalFramework"
    Src="~/templates/Frameworks/PortalUnitFramework.ascx"%>
```

```
<%@ Register TagPrefix="development" TagName="Calendar"
    Src="~/templates/Units/Calendar.ascx"%>
<%@ Register TagPrefix="development" TagName="PortalCalendar"
    Src="~/templates/Units/PortalCalendar.ascx"%>

<EPiServer:ContentFrameworkSelector runat="server" FrameworkKeyName="MyMode"
    ID="FrameworkSelector" DefaultFramework="DefaultFramework">
    <development:DefaultFramework id="DefaultFramework" runat="server">
        <EPiServer:Content Region="mainRegion" ID="calendarContent" runat="server">
            <development:Calendar id="Calendar" runat="server"></development:Calendar>
        </EPiServer:Content>
        <EPiServer:Content Region="rightColumnRegion" ID="emptyContent" runat="server" />
    </development:DefaultFramework>
    <development:PortalFramework id="portalmode" runat="server">
        <EPiServer:Content Region="mainRegion" ID="calendarContent2" runat="server">
            <development:PortalCalendar id="PortalCalendar" runat="server" />
        </EPiServer:Content>
    </development:PortalFramework>
</EPiServer:ContentFrameworkSelector>
```

When DefaultFramework is active, Calendar.aspx (watch out for the small difference between Web Form file Calendar.**aspx** and Web User Control file Calendar.**ascx**) replaces the contents of Region 'mainRegion' with Web User Control Calendar and also replaces the contents of Region 'rightColumnRegion' with nothing (an empty Content control). On the other hand, when the Framework PortalFramework is active, another Web User Control, PortalCalendar.ascx is used.

EPiServer.WebControls.ExplorerTree

Windows Explorer is the visual ideal for ExplorerTree. It displays a tree, such as the Web Page tree, just like the Windows Explorer (in its left pane). ExplorerTree control objects are used in both Admin and Edit modes of EPiServer.

ExplorerTree is a direct descendant of EPiServer.WebControls.PageTreeData (see figure 7-1).

Figure 7-3 below shows the visual appearance of ExplorerTree. (You should recognise the Web Page Tree from the example Web site; it's the tree shown in the left pane of Edit mode.)

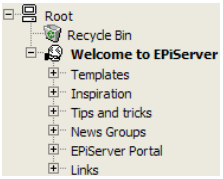


Figure 7-3: Using ExplorerTree in EPiServer Edit mode.

As with most other EPiServer Web Custom Controls, ExplorerTree is very easy to use. You only have to decide on a root page for the control. In the code-behind file, you call the DataBind method for ExplorerTree which connects the control with the EPiServer Web Page Tree.

Example 7-11: ExplorerTree in the code-behind file.

```
protected    EPiServer.WebControls.ExplorerTreeExplorerTreeObj;

private void Page_Load( object sender, System.EventArgs e ) {
    if ( ! IsPostBack ) {
        ExplorerTreeObj.DataBind();
    }
}
```

In the HTML file:

Example 7-12: HTML code needed for ExplorerTree.

```
<EPiServer:ExplorerTree EnableVisibleInMenu="False" ShowRootPage="True"
    PageLink='<%# ( (EPiServer.PageBase) Page ).Configuration.StartPage %>' ShowIcons="True"
    ClickScript="window.location.href = '{PageLinkURL}'" id="ExplorerTreeObj" runat="server"
/>
```

This results in the following rendition:

Figure 7-4: Using ExplorerTree control on a Web page.

Some of ExplorerTree's attributes are more important than others.

Table 7-3: ExplorerTree attributes.

Attribute	*Description*
BranchUrl	The URL that will be called to load child pages; default is 'Util/ExplorerTreeBranch.aspx'.
ClickScript	Set script code to be executed when the page name is clicked.
EnableVisibleInMenu	(Inherited from PageTreeData) If tree should check that page should be 'visible in menus' to appear in tree.
ExpandAll	(Inherited from PageTreeData) Expand all tree nodes.
ImageDirectory	Path to image folder.
PageLink	(Inherited from PageControlBase) The root page to read data from
ShowIcons	Show icons on system pages
ShowRootPage	(Inherited from PageTreeData) If root page should be loaded in the page list
ShowStatusIcons	Show status icons, indicating access rights and published status

EPiServer.WebControls.MenuList

Having a name like MenuList destines a class for handling menus. Of course, this is what EPiServer.WebControls.MenuList does. MenuList is a direct descendant of EPiServer.WebControls.PageTreeData and thus a sibling to EPiServer.WebControls.ExplorerTree, EPiServer.WebControls.PageTree and EPiServer.WebControls.SiteMap.

In the User Control TopMenu that ships with EPiServer, MenuList is used to create a horizontal top-level menu.

TopMenu.ascx itself is used in the Framework Definition File DefaultFramework outside of any Regions, meaning that all Page Template Files that use DefaultFramework get TopMenu.ascx.

Figure 7-5: Result of using TopMenu.ascx in the example Web site.

TopMenu.ascx.cs is not terribly complex:

Example 7-13: TopMenu.ascx.

```
namespace development.Templates.Units {
                                /// <summary>TopMenu implements a horizontal top menu.
                                /// </summary>
    public abstract class TopMenu : EPiServer.UserControlBase {
        protected   EPiServer.WebControls.PropertyProperty1;
        protected   EPiServer.WebControls.PropertyProperty2;
        public      EPiServer.WebControls.MenuListMenuListControl;

        private void Page_Load( object sender, System.EventArgs e ) {
            if ( ! IsPostBack ) {
                MenuListControl.DataBind();
            }
        }

        protected EPiServer.Core.PageReference MenuRoot {
            get {
                if ( CurrentPage[ "MainMenuContainer" ] != null ) {
                    return (EPiServer.Core.PageReference) CurrentPage[ "MainMenuContainer" ];
                } else {
                    return Configuration.StartPage;
                }
            }
        }
    }
}
```

Nor is TopMenu.ascx very complex:

Example 7-14: TopMenu.ascx.

```
<%@ Register TagPrefix="EPiServer" Namespace="EPiServer.WebControls" Assembly="EPiServer" %>
<%@ Control Language="c#" AutoEventWireup="false" Codebehind="TopMenu.ascx.cs"
    Inherits="development.Templates.Units.TopMenu"
    TargetSchema="http://schemas.microsoft.com/intellisense/ie5"%>
<table width="100%" border="0" cellspacing="0" cellpadding="0"
    xmlns:EPiServer="http://schemas.episerver.com/WebControls">
    <tr valign="middle" align="center">
        <EPiServer:MenuList runat="server" ID="MenuListControl" PageLink="<%#MenuRoot%>">
            <ItemTemplate>
                <td height="23">
                    <EPiServer:Property id="Property1" runat="server" PropertyName="PageLink"
                        CssClass="MenuHead"></EPiServer:Property>
                </td>
            </ItemTemplate>
            <SelectedTemplate>
```

```
                <td height="23">
                    <EPiServer:Property id="Property2" runat="server" PropertyName="PageLink"
                        CssClass="ActiveMenuHead"></EPiServer:Property>
                </td>
            </SelectedTemplate>
        </EPiServer:MenuList>
    </tr>
</table>
```

As you can see in example 7-14, all TopMenu.ascx does is create one row in an assumed outer table containing one table data cell for every top level record. Menu items are displayed according to the style sheet style MenuHead (which among other settings specifies 'text-transform : uppercase'). Selected menu items take on the style ActiveMenuHead, which is the same as MenuHead except for using a bold font. 'font-weight:bold'.

EPiServer.WebControls.NewsList

The rationale for the NewsList class is to provide your Web site with an easy, yet powerful way to display news items.

Again looking at the Mimic Web site, we see that a NewsList control object is used on the start page.

Figure 7-6: Using NewsList on Mimic's start page.

As NewsList is a templated control, the corresponding HTML code might seem a bit large.

Example 7-15: HTML code to render NewsList control (from the Mimic start page).

```
<table cellspacing="0" cellpadding="0" border="0" width="100%">
    <tr>
        <td>
            <EPiServer:Newslist ID=Newslistnew Pagelinkproperty="NewsContainer" runat="Server"
                    MaxCount='<%# GetNewsCount() %>'>
                <HeaderTemplate>
                    <tr>
                        <td height="15" background="images/L_triangleBG.gif">
```

```
                        <a class="ListHeading" href="<%#Container.CurrentPage.LinkURL%>">
                            <%# Container.CurrentPage.PageName.ToUpper() %> 
                        </a>
                    </td>
                </tr>
            </HeaderTemplate>
            <NewsTemplate>
                <tr>
                    <td class="DateListingText">
                        <%# Container.CurrentPage[ "PageStartPublish" ] %>
                    </td>
                </tr>
                <tr>
                    <td>
                        <a href="<%# Container.CurrentPage.LinkURL %>"
                            class="StartPageHeading">
                            <%# Container.CurrentPage.PageName %>
                        </a> <br />
                        <span class="Normal">
                            <%# Container.CurrentPage[ "MainIntro" ] %>
                        </span>
                    </td>
                </tr>
                <tr>
                    <td><EPiServer:Clear height="6" runat="server" /></td>
                </tr>
            </NewsTemplate>
            <FooterTemplate>
                <tr>
                    <td bgcolor="#DEDEDE"><EPiServer:Clear height="1" runat="server" /></td>
                </tr>
            </FooterTemplate>
        </EPiServer:Newslist>
    </td>
</tr>
</table>
```

A discussion of the templates used in NewsList is presented on page 171 and following pages. Apart from these templates, the NewsList class only offers inherited attributes, methods and events.

EPiServer.WebControls.PageList

EPiServer is currently geared towards presentation information in a Web site context, which makes it only natural that so many classes and controls deal with handling Web pages. EPiServer.WebControls.PageList is one such page handling control. It is used to present a list of the Web pages that form the Web site. It's an easy way to render a list of references (HTML Anchor tags) to all the Web pages that belong to the site.

Figure 7-7: Example of PageList on Web page.

The PageList control object as seen in figure 7-7 was placed on a Web Form using the HTML code, as seen in example 7-16.

Example 7-16: EPiServer.WebControls.PageList used on a Web Form.

```
<EPiServer:PageList PageLink="<%# ( (EPiServer.PageBase) Page ).Configuration.StartPage %>"
        runat="server" ID="PageList1">
    <ItemTemplate>
        <tr>
            <td>
                <EPiServer:Property PropertyName="PageLink" runat="server" ID="Property2"/>
            </td>
        </tr>
    </ItemTemplate>
</EPiServer:PageList>
```

PageList has three templates: HeaderTemplate, ItemTemplate and FooterTemplate.

The PageLink property of PageList is used to link the control object to the proper place in the Web Page Tree. To link it to the current page, and its child pages, you would use 'PageLink=<%# CurrentPage.PageLink %>' instead.

EPiServer.WebControls.PageSearch

It is not searching for Pages but for information on Pages that PageSearch is used for. It also cooperates with Microsoft Index Server and thus allows searching outside of the Web site proper.

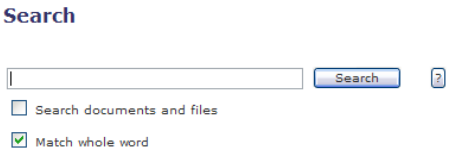


Figure 7-8: Using PageSearch on a Web Form Search.aspx.

A good example of using PageSearch is the Search.aspx Web Form that is shipped with EPiServer. Actually Search.aspx is simply a host for the Web User Control Search.ascx. As always, the reason for using both a User Control and a Web Form is the EPiServer Content Framework; the Web Form is an EPiServer Page which is using a Framework Definition File.

This is what it looks like when searching the Mimic Web site for the word 'mimic'.

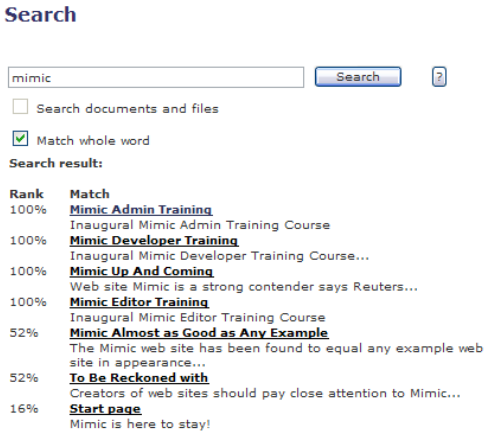


Figure 7-9: Result presented when searching the Mimic Web site for the word 'mimic'.

Looking at the screen dump in figure 7-9, you'd be quite right in assuming that the Search Web Custom Control Class is a templated control.

All the action takes place in the Web User Control, Search.ascx, and its code-behind file, Search.ascx.cs.

Example 7-17: Using EPiServer.WebControls.PageList in Web User Control Search.ascx.

```
<%@ Register TagPrefix="EPiServer" Namespace="EPiServer.WebControls" Assembly="EPiServer" %>
<%@ Control Language="c#" AutoEventWireup="false" Codebehind="Search.ascx.cs"
    Inherits="development.Templates.Units.Search"
    TargetSchema="http://schemas.microsoft.com/intellisense/ie5" %>
```

```
<table cellspacing="2" border="0" xmlns:EPiServer="http://schemas.episerver.com/WebControls">
…
    <!-- Note: PageLink is the default start page for search, used if MainContainer is empty -->
    <EPiServer:PageSearch
        Runat="server"
        ID="SearchResults"
        SearchQuery='<%# SearchQuery.Text %>'
        SearchFiles='<%# SearchFiles.Checked %>'
        OnlyWholeWords='<%# OnlyWholeWords.Checked %>'
        MainScope='<%# CurrentPage[ "MainScope" ] %>'
        MainCatalog='<%# CurrentPage[ "MainCatalog" ] %>'
        PageLink='<%# Configuration.StartPage %>'
        PageLinkProperty="MainContainer"
    >
        <HeaderTemplate>
            <tr>
                <td colspan="2">
                    <b>
                    <EPiServer:Translate id="Translate6" runat="server"
                        Text="/templates/search/searchresult" CssClass="EP-tableHeading" />
                    </b>
                </td>
            </tr>
            <tr>
                <td colspan="2"> </td>
            </tr>
            <tr>
                <td align="left" width="50">
                    <b><EPiServer:Translate id="Translate4" runat="server"
                        Text="/templates/search/rank" /></b>
                </td>
                <td>
                    <b><EPiServer:Translate id="Translate5" runat="server"
                        Text="/templates/search/match" /></b>
                </td>
            </tr>
        </HeaderTemplate>
        <ItemTemplate>
            <tr>
                <td align="left" width="50">
                    <%# (int) Container.CurrentPage[ "PageRank" ] / 10 %>%
                </td>
                <td align="left">
```

```
                    <b>
                    <EPiServer:Property id="Property1" runat="server" PropertyName="PageLink" />
                    </b>
                </td>
            </tr>
            <tr>
                <td> </td>
                <td><%# Container.PreviewText %></td>
            </tr>
        </ItemTemplate>
        <FileTemplate>
            <tr>
                <td align="left" width="50">
                    <%# (int) Container.CurrentPage[ "PageRank" ] / 10 %>%</td>
                <td align="left">
                    <img src='<%# Container.CurrentPage[ "IconPath" ] %>' >
                    <a href='<%# Container.CurrentPage[ "PageLinkURL" ]%>'>
                        <b><%# Container.CurrentPage.PageName %></b></a>
                </td>
            </tr>
        </FileTemplate>
        <NoMatchesTemplate>
            <tr>
                <td colspan="2">
                    <br/>
                    <br/>
                    <EPiServer:Translate CssClass="EP-tableHeading"
                        Text="/templates/search/nomatches" runat="server" ID="Translate3" />
                </td>
            </tr>
        </NoMatchesTemplate>
    </EPiServer:PageSearch>
</table>
```

Of course, there are many opportunities for those who like more control. These are some of the pertinent attributes in the PageList class:

Table 7-4: PageList attributes.

Attribute Name	*Description*
FileTemplate	Template for files
ItemTemplate	(Inherited from PageList) The default template for pages
MainCatalog	The index server catalog to search in.

Table 7-4: PageList attributes.

Attribute Name	*Description*
MainScope	The scope parameter for Index Server searches. The default is deep traversal under the catalog root.
MaxCount	(Inherited from PageListData) Restrict listing to a maximum number of pages.
NoMatchesTemplate	The template for no search matches
OnlyWholeWords	A flag to control if the search will match only whole words. If False, the search will match all words that begin with the words in the query (word* match).
SearchFiles	A flag to indicate whether or not files should be included in the search.
SearchQuery	Search query string, i.e. whatever the user types.

EPiServer.WebControls.PageTree

EPiServer.WebControls.PageTree is a direct descendant of EPiServer.WebControls.PageTreeData (see figure 7-1). PageTree is a templated control; templates are its only extension to PageTreeData. The names of the templates are self-explanatory:

- ExpandedItemTemplate
- ExpandedTopTemplate
- FooterTemplate
- HeaderTemplate
- ItemTemplate
- SelectedExpandedItemTemplate
- SelectedExpandedTopTemplate
- SelectedItemTemplate
- SelectedTopTemplate
- TopTemplate

FooterTemplate and HeaderTemplate are simply used to define the areas immediately above and below the PageTree control area.

PageTree is often used in conjunction with EPiServer.MenuList, where it uses the MenuList as its data source to create hierarchical menus.

One example of using PageTree, and Menu, is the DefaultFramework file shipped with EPiServer. The menu displayed along the left side of the Web page is created by using EPiServer.WebControls.PageTree and EPiServer.WebControls.MenuList in an ASP.NET User Control, Menu.ascx.

This is what the final result looks like:

Figure 7-10: Menu created by Web Control Menu.ascx using Custom Controls PageTree and MenuList. The right-hand picture is the result of expanding the item News.

Looking into Menu.ascx

Example 7-18: Menu.ascx.cs

```
namespace development.Templates.Units {
                                /// <summary>Menu uses EPiServer.WebControls.MenuList
                                /// and EPiServer.WebControls.PageTree to implement a
                                /// vertical menu.</summary>
    public abstract class Menu : EPiServer.UserControlBase {
        protectedEPiServer.WebControls.PageTree   PageTreeControl;
        private  EPiServer.WebControls.MenuList     _menuListControl;

        private void Page_Load( object sender, System.EventArgs e ) {
            if ( MenuListControl != null ) {
                PageTreeControl.DataSource = MenuListControl;
            }
            if ( ! IsPostBack ) {
                PageTreeControl.DataBind();
            }
        }

        public EPiServer.WebControls.MenuList MenuListControl {
            set { _menuListControl = value; }
            get { return (EPiServer.WebControls.MenuList) _menuListControl; }
        }
...
    }
}
```

The HTML code looks like this (just browse it, a more readable version follows):

Example 7-19: Menu.ascx.

```
<%@ Control Language="c#" AutoEventWireup="false" Codebehind="Menu.ascx.cs"
    Inherits="development.Templates.Units.Menu"
    TargetSchema="http://schemas.microsoft.com/intellisense/ie5" %>
<%@ Register TagPrefix="EPiServer" Namespace="EPiServer.WebControls" Assembly="EPiServer" %>
<table cellpadding="0" cellspacing="0" border="0" width="160"
    xmlns:EPiServer="http://schemas.episerver.com/WebControls">
    <EPiServer:PageTree runat="server" id="PageTreeControl">
        <HeaderTemplate>
        <tr>
            <td valign="middle" height="15" colspan="6">
                <a class="ListHeading" href="<%# Container.CurrentPage.LinkURL %>">
                    <%#Container.CurrentPage.PageName.ToUpper()%>
                      </a>
            </td>
        </tr>
        <tr>
            <td colspan="6"><EPiServer:Clear height="5" runat="server" /></td>
        </tr>
        <tr>
            <td width="12" height="1"></td>
            <td width="12" height="1"></td>
            <td width="12" height="1"></td>
            <td width="12" height="1"></td>
            <td width="12" height="1"></td>
            <td width="100" height="1"></td>
        </tr>
        </HeaderTemplate>
        <ExpandedItemTemplate>
            <tr>
                <td height="20" colspan='<%#Container.CurrentPage.Indent%>' align="right">
                    <asp:Image Visible="<%# Container.HasChildren %>"
                        ImageUrl="~/images/openMenuArrow.gif" Width="7" Height="7"
                        AlternateText="" runat="server" />
                    <EPiServer:Clear Visible="<%# ! Container.HasChildren %>" Width="7" Height="7"
                        runat="server" />
                     
                </td>
                <td height="20" colspan='<%#6-Container.CurrentPage.Indent%>'>
                    <EPiServer:Property CssClass="MenuLink" runat="server"
                        PropertyName="PageLink" id="Property1" name="Property1" />
                </td>
            </tr>
```

```
    <tr>
        <td bgcolor="#dedede" colspan="6"><EPiServer:Clear runat="server" /></td>
    </tr>
</ExpandedItemTemplate>
<ItemTemplate>
    <tr>
        <td height="20" colspan="<%#Container.CurrentPage.Indent%>" align="right">
            <asp:Image Visible="<%# Container.HasChildren %>"
                ImageUrl="~/images/closedMenuArrow.gif" Width="7" Height="7"
                AlternateText="" runat="server" />
                <EPiServer:Clear Visible="<%# ! Container.HasChildren %>" Width="7"
                    Height="7" runat="server" />
             
        </td>
        <td height="20"
            colspan="<%#6-Container.CurrentPage.Indent%>">
                <EPiServer:Property CssClass="MenuLink" runat="server"
                    PropertyName="PageLink" id="Property2" name="Property2" />
        </td>
    </tr>
    <tr>
        <td bgcolor="#dedede" colspan="6"><EPiServer:Clear runat="server" /></td>
    </tr>
</ItemTemplate>
<SelectedItemTemplate>
    <tr bgcolor="#dedede">
        <td height="20" colspan="<%#Container.CurrentPage.Indent%>"
            align="right">
            <asp:Image Visible="<%# Container.HasChildren %>"
                ImageUrl="~/images/openMenuArrow.gif" Width="7" Height="7"
                    AlternateText="" runat="server" />
                <EPiServer:Clear Visible="<%# ! Container.HasChildren %>" Width="7"
                Height="7" runat="server" />
             
        </td>
        <td height="20" colspan="<%#6-Container.CurrentPage.Indent%>">
            <EPiServer:Property CssClass="MenuLink" runat="server"
                PropertyName="PageLink" ID="Property3" NAME="Property3" />
        </td>
    </tr>
    <tr>
        <td bgcolor="#dedede" colspan="6"><EPiServer:Clear runat="server" /></td>
    </tr>
```

```
        </SelectedItemTemplate>
    </EPiServer:PageTree>
</table>
```

Now, that's a mouthful. Let's simplify it:

Example 7-20: Menu.ascx in much simplified form.

```
<%@ Control Language="c#" AutoEventWireup="false" Codebehind="Menu.ascx.cs"
    Inherits="development.Templates.Units.Menu"
    TargetSchema="http://schemas.microsoft.com/intellisense/ie5" %>
<%@ Register TagPrefix="EPiServer" Namespace="EPiServer.WebControls" Assembly="EPiServer" %>
<table cellpadding="0" cellspacing="0" border="0" width="160"
    xmlns:EPiServer="http://schemas.episerver.com/WebControls">
    <EPiServer:PageTree runat="server" id="PageTreeControl">
        <HeaderTemplate>
        <tr>
            <td>
                <a href="<%# Container.CurrentPage.LinkURL %>">
                    <%# Container.CurrentPage.PageName.ToUpper() %>
                </a>
            </td>
        </tr>
        </HeaderTemplate>
        <ItemTemplate>
            <tr>
                <td>
                    <a href="<%# Container.CurrentPage.LinkURL %>">
                        <%# Container.CurrentPage.PageName %>
                    </a>
                </td>
            </tr>
        </ItemTemplate>
    </EPiServer:PageTree>
</table>
```

You probably agree that the latter is much easier to read. It still retains the basic functionality of the code in example 7-19.

Figure 7-11: Menu created by the simplified Web Control Menu.ascx using Custom Controls PageTree and MenuList. The right-hand picture is the result of expanding the item News.

Underlining is controlled by the style sheet, episerver.css, which by default underlines all links (anchor tags).

The original Menu.ascx (see example 7-19) uses four out of PageTree's ten templates: HeaderTemplate, to create a header for the menu; ItemTemplate, for all ordinary items whether they are mother or children items; ExpandedItemTemplate for selected items that are expanded, again whether or not they have any children items; and SelectedItemTemplate for the currently selected item.

There are several examples of elegant coding in example 7-19. One is the use of EPiServer.WebControls.Property instead of the HTML anchor tag.

Example 7-21: Using EPiServer.WebControls.Property instead of HTML anchor tag.

```
<EPiServer:Property runat="server" PropertyName="PageLink" ID="Property1" />
<!-- Equivalent to: -->
<a href="<%# Container.CurrentPage.LinkURL %>"><%# Container.CurrentPage.PageName %></a>
```

EPiServer.WebControls.Property

Property is probably the most frequently used of all control types in the EPiServer.WebControls name space. It provides access to all page properties, whether they are built-in, changes automatically or added in EPiServer Admin mode. The class name, 'Property', must be understood in an historical context. Had EPiServer been written in and for an object-oriented environment, such as Microsoft .NET, today, this class would most likely have been called 'Value' or perhaps 'Setting'. It will be interesting to see whether a name change is feasible for future versions of EPiServer.

Table 7-5: Attributes for EPiServer.WebControls.Property.

Attribute Name	*Description*
DisplayMissingMessage	Hide or show error message when property is missing
Editable	Determines whether the property is editable with DOPE
EditMode	Determines whether the property should render its edit mode
InnerProperty	Set or get the inner property used by this control
PageLink	The root page to read data from, if different from current
PageLinkProperty	The property that contains the root page to read data from, if different from current
PageSource	Returns the IPageSource implementation that this property control uses to read page data

Table 7-5: Attributes for EPiServer.WebControls.Property.

Attribute Name	*Description*
PropertyName	The name of the property to load automatically
PropertyValue	The value of the loaded property

Property objects, like Content and Region objects, are most often found in HTML part of Web Forms and User Controls, but they also are used to a great extent in code-behind files.

Properties on pages are collected in an EPiServer.Core.PropertyDataCollection collection called Property, provided that the template used implements IPageSource (see page 153).

Property objects used in HTML only have to include the name of the Property:

Example 7-22: Using a Property object to display contents of Property MainBodyHeading.

```
<EPiServer:Property PropertyName="MainBodyHeading"
    DisplayMissingMessage="false" class="ListHeading" runat="server" />
```

Example 7-23: Display the name of the page.

```
<EPiServer:Property propertyname="PageName" runat="server" />
```

For the Property control to be able to read the properties from the page, it needs to be hosted on a Web form or a control that implements the IPageSource interface. The control will iterate through the control hierarchy looking for this interface, and when it finds one, it will use the CurrentPage property to read the information about the specific built-in or custom property.

If you put a Property control inside a templated control like the PageList control which implements IPageSource, the Property control will use the CurrentPage property of the template control instead. The PageList then points the Property control to the current PageData object in its internal PageDataCollection. This is why the two following examples will print the same:

Example 7-24: Using Property inside a templated control.

```
<EPiServer:PageList PageLink=<%# CurrentPage.Configuration.StartPage %> runat="server">
    <ItemTemplate>
        <EPiServer:Property PropertyName="PageName" runat="server"/>
    </ItemTemplate>
</EPiServer:PageList>
```

Example 7-25: Using Container inside a templated control.

```
<EPiServer:PageList PageLink=<%# CurrentPage.Configuration.StartPage %> runat="server">
```

```
    <ItemTemplate>
        <%# Container.CurrentPage.PageName %>
    </ItemTemplate>
</episerver:PageList>
```

The Property control will also give you DOPE (Direct On Page Editing) support if the underlying template page supports it (meaning any Web page that inherits directly or indirectly from TemplatePage). If you do not want DOPE support, you can either inherit from SimplePage or set the Editable attribute to false, like this:

Example 7-26: Switching off DOPE support for an EPiServer Property.

```
<EPiServer:Property PropertyName="PageName" runat="server" Editable='false' />
```

Note that DOPE support will only be available for properties on the current page, not properties rendered inside a PageList or any other templated control.

EPiServer.WebControls.PropertyCriteriaControl

The HTML code in example 7-27 uses a PropertySearch object as data source for a PageList object. Enclosed in the PropertySearch object are two PropertyCriteriaControl objects, specifying that the search proper concerns the Property PageName and that the contents of the PageName properties should start with one of the letters 'A', 'a', 'B' or 'b'. (Compare this with example 7-28, which puts the logic in the code-behind file.)

Example 7-27: Using PropertySearch, PropertyCriteriaControl and PageList to produce a list of pages meeting certain criteria.

```
<%@ Register TagPrefix="EPiServer" Namespace="EPiServer.WebControls" Assembly="EPiServer" %>
...
<EPiServer:PropertySearch PageLink=<%# Configuration.StartPage %> runat="server"
    ID="PropSearch" >
    <EPiServer:PropertyCriteriaControl Name="PageName" Value="a" Type="String"
        StringCondition="StartsWith" />
    <EPiServer:PropertyCriteriaControl Name="PageName" Value="b" Type="String"
        StringCondition="StartsWith" />
</EPiServer:PropertySearch>
<EPiServer:PageList SortBy="PageName" DataSource="<%# PropSearch %>" runat="server"
    ID="PL">
    <ItemTemplate>
        <tr>
            <td><EPiServer:Property PropertyName="PageName" runat="server" ID="Prop3" /></td>
        </tr>
    </ItemTemplate>
```

```
</EPiServer:PageList>
```

EPiServer.WebControls.PropertySearch

Having a lot of Web Pages in the Web Page and perhaps a lot of information on those pages makes is absolutely necessary to be able to search these pages using almost any attributes and any number of search criteria. PropertySearch is used when the capabilities of PageSearch (see page 183) just aren't enough. In addition, PropertySearch is not limited to searching for, or in, Web Pages in the Web Page Tree; it searches all of page database, in the database table tblProperty (see page 315). So pages such as user registrations and other kinds are part of the search. (For an illustration, open the example Web site and look at its site map, which shows only pages in the Web Page Tree. Then open Templates and select 'Alphabetical table of contents'. In the latter case, PropertySearch is used and thus all pages are returned, not just those in the Web Page Tree.)

Central to PropertySearch is its attribute Criterias (this is the only attribute which is not inherited). Criterias is of the type EPiServer.PropertyCriteriaCollection, i.e. a collection of EPiServer.PropertyCriteria. Proper use of the PropertyCriteria objects in Criterias is key to successfully using PropertySearch.

EPiServer.PropertyCriteria

PropertyCriteria inherits from System.Object and extends the mother class only by adding seven attributes. It is used for two different kinds of searches: string and non-string searches. When used for string searches, the search condition attribute StringCondition is employed whilst for non-string searches Condition is the attribute of choice.

Table 7-6: PropertyCriteria attributes.

Attribute Name	*Description*
Condition	The type of comparison condition. EPiServer.Filters.CompareCondition, specifies Equal, NotEqual, GreaterThan and LessThan.
IsNull	Test for value set to null. Boolean.
Name	Name of property. String.
Required	Determines whether this criterion is required for a match. Boolean. Default is false, meaning that this criterion is logically Or:ed with other criteria.
StringCondition	Comparison of strings when Equal is used. Type is EPiServer.Filters.StringCompareMethod (Identical, StartsWith, EndsWith and Contained). Comparison is case insensitive.

Table 7-6: PropertyCriteria attributes.

Attribute Name	*Description*
Type	Type of criterion, one of EPiServer.Core.PropertyDataType enumeration.
Value	Value of criterion. String. If you're using Microsoft SQL Server, its wild cards '%' and '?' may be used.

Using PropertySearch

Example 7-28: Using PropertyCriteria when searching for Web Pages by their name (from templates\Units\AlphanumericListing.ascx.cs).

```
...
protected EPiServer.WebControls.PropertySearchPropertySearchControl;
...
    protected void ChangeLetters( object sender, System.EventArgs e ) {
...
        AddLetter( "A" );
        AddLetter( "B" );
        AddLetter( "C" );
        AddLetter( "D" );
        AddLetter( "E" );
        AddLetter( "F" );
...
        DataBind();
    }

private void AddLetter( string letter ) {
    EPiServer.PropertyCriteria criterion = new EPiServer.PropertyCriteria();
    criterion.StringCondition = EPiServer.Filters.StringCompareMethod.StartsWith;
    criterion.Type = EPiServer.Core.PropertyDataType.String;
    criterion.Value = letter;
    criterion.Name = "PageName";
    PropertySearchControl.Criterias.Add( criterion );
}
```

The code in example 7-28 is extracted from the code-behind file of Web User Control AlphanumericListing, AlphanumericListing.ascx.cs.

Taking a look at the function AddLetter, we see that it's building up for a string search; criterion.StringCondition is given a value rather than criterion.Condition. Moreover, we're using a string value as a search criterion, 'criterion.Type = EPiServer.Core.PropertyDataType.String'; the string value proper is set to the formal argument 'letter' passed to AddLetter. The EPiServer Property which

we'll be comparing criterion.Value against is PageName. In other words, the search is performed against every Web Page comparing EPiServer Property PageName to the criteria in Criterias.

In this particular instance, you can also see that AddLetter is called multiple times. As criterion.Required is not altered, this means that PageName is matched against any of 'A' through 'F'. Setting criterion.Required to true for this kind of comparison would be non-sensical (criterion.StringCondition is set to EPiServer.Filters.StringCompareMethod.StartsWith), but for other kinds it is essential, e.g. when searching for Properties which should contain more than one letter.

The HTML code in AlphanumericListing.ascx is quite simple.

Example 7-29: AlphanumericListing.ascx.

```
<%@ Register TagPrefix="EPiServer" Namespace="EPiServer.WebControls" Assembly="EPiServer" %>
<%@ Control Language="c#" AutoEventWireup="false" Codebehind="AlphanumericListing.ascx.cs"
    Inherits="development.Templates.Units.AlphanumericListing"
    TargetSchema="http://schemas.microsoft.com/intellisense/ie5"%>
<table border="0" height="90%" cellpadding="0" cellspacing="0"
    xmlns:EPiServer="http://schemas.episerver.com/WebControls">
    <tr>
        <td valign="top">
            <table bgcolor="#cccccc" width="400" cellpadding="2" cellspacing="1">
                <tr>
                    <td bgcolor="#cccccc" Class="Heading2">
                        [<asp:LinkButton ID="Alphanumeric1" Runat="server"
                        OnClick="ChangeLetters" CssClass="Heading2">a-f</asp:LinkButton>] 
                        [<asp:LinkButton ID="Alphanumeric2" Runat="server"
                        OnClick="ChangeLetters" CssClass="Heading2">g-l</asp:LinkButton>] 
                        [<asp:LinkButton ID="Alphanumeric3" Runat="server"
                        OnClick="ChangeLetters" CssClass="Heading2">m-r</asp:LinkButton>] 
                        [<asp:LinkButton ID="Alphanumeric4" Runat="server"
                        OnClick="ChangeLetters" CssClass="Heading2">s-z</asp:LinkButton>]
                    </td>
                </tr>
                <EPiServer:PropertySearch PageLink="<%# EPiServer.Global.EPConfig.StartPage %>"
                    runat="server" ID="PropertySearchControl" />
                <EPiServer:PageList SortBy="PageName"
                    DataSource="<%# PropertySearchControl %>"runat="server"
                    ID="PageListControl">
                    <ItemTemplate>
                        <tr>
                            <td bgColor="#ffffff">
                                <EPiServer:Property id="Property1" runat="server"
                                    PropertyName="PageLink" />
```

```
                    </td>
                </tr>
            </ItemTemplate>
        </EPiServer:PageList>
    </table>
</td>
</tr>
</table>
```

EPiServer.WebControls.Region

Framework Definition Files is the only type of place where you'll find EPiServer.WebControls.Region objects. Region objects are used to define areas in a Framework Definition File which may have their contents exchanged by Content object in Page Template Files (yes, we're talking about the EPiServer Content Framework).

As such, Region objects are never instantiated in code-behind files, they simply have no purpose there.

Example 7-30: Use of EPiServer.WebControls.Region in a Framework Definition File.

```
<%@ Control Language="c#" … %>
<%@ Register TagPrefix="EPiServer" Namespace="EPiServer.WebControls" Assembly="EPiServer" %>
<%@ Register TagPrefix="development" TagName="LeftMenu" Src="~/templates/Units/Menu.ascx"%>
…
<EPiServer:Region id="menuRegion" runat="server">
    <development:LeftMenu id="LeftMenu" runat="server"/>
    <br/>
</EPiServer:Region>
```

Region control objects may be nested to any depth you need.

Example 7-31: Nested Region control objects.

```
<%@ Control Language="c#" … %>
<%@ Register TagPrefix="EPiServer" Namespace="EPiServer.WebControls" Assembly="EPiServer" %>
<%@ Register TagPrefix="development" TagName="LeftMenu" Src="~/templates/Units/Menu.ascx"%>
…
<EPiServer:Region id="fullRegion" runat="server">
    <table>
        <tr>
            <td>
                <EPiServer:Region id="menuRegion" runat="server">
                        <development:LeftMenu id="LeftMenu" runat="server" />
                        <br/>
```

```
                </EPiServer:Region>
            </td>
...
        </tr>
    </table>
</EPiServer:Region>
```

The only important attribute for Region controls is ID, the unique identifier. Only by using the proper ID string can a Content control exchange its own contents for that of a pre-defined Region.

EPiServer.WebControls.SiteMap

It's not surprising that EPiServer ships with such an easy way to produce a site map as using SiteMap control object.

SiteMap is used in just two ASP.NET objects in the example Web site: the Page Template File SiteMap.aspx and the Web User Control SiteMap.ascx (in the template\Units folder).

The Page Template File SiteMap.aspx is not very exciting; its code-behind file is all but empty. The HTML contents look like this:

Example 7-32: HTML contents of Page Template File SiteMap.aspx.

```
<%@ Page language="c#" Codebehind="SiteMap.aspx.cs" AutoEventWireup="false"
    Inherits="development.Templates.SiteMap" %>
<%@ Register TagPrefix="EPiServer" Namespace="EPiServer.WebControls" Assembly="EPiServer" %>
<%@ Register TagPrefix="development" TagName="DefaultFramework"
    Src="~/templates/Frameworks/DefaultFramework.ascx"%>
<%@ Register TagPrefix="development" TagName="Sitemap"
    Src="~/templates/Units/Sitemap.ascx"%>

<development:DefaultFramework ID="defaultframework" runat="server">
    <EPiServer:Content Region="fullRegion" ID="SiteMapContent" runat="server">
        <development:Sitemap id="Sitemap" runat="server"></development:Sitemap>
    </EPiServer:Content>
</development:DefaultFramework>
```

As can be seen in example 7-32, the Page Template File SiteMap.aspx replaces everything in the Region 'fullRegion' with the Web User Control SiteMap.ascx.

EPiServer.WebControls.Translate

In order to make your EPiServer solution as portable as possible, you should use the Translate class. With Translate and a set of language files, you can furnish groups of users with an EPiServer solution which literally speaks their own language.

Translate is a direct descendant of System.Web.UI.WebControls.WebControl (see figure 7-1).

There are only three attributes that Translate implements itself:

Table 7-7: Attributes implemented by EPiServer.Translate.

Attribute Name	*Description*
LocalizedText	Used to access the text to be translated
StringFormatObjects	Set object references if the localized text is a format string with '{n}' style instructions
Text	The property that holds the text to be translated.

Use the Translate control to translate your own strings or one of the built-in strings in the /lang folder of your site. The text displayed will vary according to the current user's chosen language (if chosen) or the system default language.

The control uses the LanguageManager to translate the text. You can access the LanguageManager through code using the Global.EPLang static property.

Example 7-33: Using EPiServer.Translate in HTML part of a form or control.

```
<EPiServer:Translate text="/button/search" runat="server"/>
```

This will show the translated value for the /button/search language resource.

Translating ASP Intrinsic controls

Translate can also be used to translate ASP.NET control objects such as the Label Web control. In the PreRender event, EPiServer will search the Controls hierarchy for controls with a 'Translate' attribute (case insensitive) and subsequently translate the textual value of the control.

Example 7-34: Using Translate attribute in an ASP.NET control.

```
<asp:button id="QuicksearchButton" runat="server" translate="/button/search" />
```

Automatic translation can be turned off by setting the AutomaticTranslation property in your template to false.

8
Custom Property Data Types and Filters

Customized Property Data Types (Customized Value Types)

Customized property data types (again Values or Settings would be a more appropriate name for EPiServer Properties) enable developers to build re-usable elements. They have been added to EPiServer 4 in order to make it easy to custom-build re-usable input fields. These input fields can be shared between any number of Web projects.

The customized properties are viewable in three modes:

- View Mode (Default)
- Direct On-Page Editing Mode, DOPE
- Edit Mode

On installation, EPiServer 4 contains fifteen standard property data types. These are all customizable using sub-classing.

Table 8-1: Standard property data types.

Property Data Type	*Name Space*	*Implementing Class*
Base class	EPiServer.Core	PropertyData
String (<= 255)	EPiServer.Core	PropertyString
Selected/Not selected	EPiServer.Core	PropertyBoolean
Category Selection	EPiServer.Core	PropertyCategory
Date–Time	EPiServer.Core	PropertyDate
Floating Point Number	EPiServer.Core	PropertyFloatNumber
Form	EPiServer.Core	PropertyForm
Frame	EPiServer.SpecializedProperties	PropertyFrame
Language	EPiServer.SpecializedProperties	PropertyLanguage

Table 8-1: Standard property data types.

Property Data Type	*Name Space*	*Implementing Class*
Long String (> 255)	EPiServer.Core	PropertyLongString
Integer	EPiServer.Core	PropertyNumber
Page Reference	EPiServer.Core	PropertyPageReference
Page Type	EPiServer.Core	PropertyPageType
Sort Order	EPiServer.SpecializedProperties	PropertySortOrder
URL	EPiServer.SpecializedProperties	PropertyUrl
WeekDay	EPiServer.SpecializedProperties	PropertyWeekDay

As with any sub-classing in C#, sub-classing of standard properties can be achieved by inheriting from either the base class or an existing sub-class. This way the developer decides herself how much of the implementation should be done in the customized property class and how much is deferred to the mother class.

Class EPiServer.PlugIn.PageDefinitionTypePlugIn

The PageDefinitionTypePlugIn attribute class signals to EPiServer that the class carrying the attribute extends some 'page definition type', such as new Property data types and registers them accordingly in EPiServer Admin mode. PageDefinitionTypePlugIn inherits EPiServer.PlugIn.PlugInAttribute (see figure 13-3 on page 269 and description on page 271), but doesn't extend it as regards properties. Most frequently used of the PageDefinitionTypePlugIn properties is DisplayName, with which you specify the new type's name as viewed in EPiServer Admin mode.

Creating New Property Data Type BackgroundColourType

In this example, we'll make a new customized property BackgroundColorType by extending the standard property PropertyString. The visual interface of BackgroundColorType is a drop-down menu allowing the user to chose between colours. The colour selected will then be used as the background colour for parts of the Web site (or even all of it). To accomplish the latter, the new customized property will be added as a dynamic property.

The new customized property BackgroundColorType will inherit from PropertyString and include a new implementation of CreateChildControls, overriding the implementation in the base class.

1. Open your project in Visual Studio .NET, add a new class file (Add Class) and call it BackgroundColourType.cs
2. Add code according to the listing below:

Example 8-1: Code for class BackgroundColourType class inheriting from PropertyString.

```
namespace development {
                                /// <summary>BackgroundColourType implements a background colour
                                /// type by sub-classing EPiServer.Core.PropertyString. User is presented
                                /// with colour names, storage is a hexadecimal string.</summary>
    [ EPiServer.PlugIn.PageDefinitionTypePlugIn( DisplayName = "BgColour" ) ]
    public class BackgroundColourType : EPiServer.Core.PropertyString {

        public override void CreateChildControls( string RenderType, System.Web.UI.Control Container ){
            switch ( RenderType.ToLower() ) {
                case "edit":
                    System.Web.UI.WebControls.DropDownList BgColDropDownList =
                        new System.Web.UI.WebControls.DropDownList();
                    BgColDropDownList.ID = Name;
                    CopyWebAttributes( Container, BgColDropDownList );
                    BgColDropDownList.Items.Add(
                        new System.Web.UI.WebControls.ListItem( "Blue", "#3333cc" ) );
                    BgColDropDownList.Items.Add(
                        new System.Web.UI.WebControls.ListItem( "Red", "#cc0000" ) );
                    BgColDropDownList.Items.Add(
                        new System.Web.UI.WebControls.ListItem( "White", "#ffffff" ) );
                    if ( base.Value != null ) {
                        BgColDropDownList.SelectedIndex = 0;// Default.
                        switch ( ( (string) base.Value ).ToLower() ) {
                            case "#cc0000" :
                                BgColDropDownList.SelectedIndex = 1;
                                break;
                            case "#ffffff" :
                                BgColDropDownList.SelectedIndex = 2;
                                break;
                        }
                    }
                    CopyWebAttributes( Container, BgColDropDownList );
                    Container.Controls.Add( BgColDropDownList );
                    Container.Controls.Add( CreateParseValidator( BgColDropDownList ) );
                    break;
                default:
                    base.CreateChildControls( RenderType, Container );
                    break;
            }
        }
    }
}
```

The class in example 8-1 inherits from EPiServer.Core.PropertyString and overrides the function CreateChildControls. In this function, base.Value (i.e. PropertyString.Value) is checked and if such a value exists it is used to set the SelectedIndex of the DropDownList used to input values (so the Editor won't think the setting wasn't stored).

The rationale for extending PropertyString, as opposed to another intrinsic property class, is that colours in HTML are often encoded as six hex numbers and in string format.

In this particular example, two new over-ridden function CreateChildControls are all that's needed. The first argument, string, for CreateChildControls, RenderType, contains the current view mode according to this list:

- edit
- dope
- default

As the envisaged use for this property data type is confined to settings under Editor control, only the 'edit' case is handled. For any other case, the mother class function, 'base.CreateChildControls', is called instead.

Make the New Property Type Part of the System

Adding the class attribute PageDefinitionTypePlugIn makes the registration of the new property type automatic. In its absence, putting the new EPiServer Property to work is a two-stage process: first compile the class, then enter the Web site's EPiServer Admin mode to create the new custom property type (section Page types, select Edit custom property types). Fill in the fields according to the table.

Table 8-2: Information when creating new property type.

Field Name	*Comment*
Base type	In this case String, in general the base type on which the new type is based.
Name	Your choice entirely. Choose an Editor-friendly name.
Class name	Fully qualified class name, in this case 'development.BackgroundColourType'
Assembly name	Probably 'EPiServerSample', but you might have changed it. It's the name of the ASP.NET project in Visual Studio .NET. Do not add any file extension: that's handled by the run-time system at load-time.

Having created the new custom property type, you can assign properties using this type to EPiServer Page Types.

In Edit mode, EPiServer takes care of positioning the property on the page.

Figure 8-1: Handling a property of the type BackgroundColourType in EPiServer Edit mode.

Creating New Restricted Property Data Type MailToUrl

Example 8-2: Code for class MailToUrl class inheriting EPiServer.Core.PropertyString.

```
namespace development {
                                    /// <summary>MailToUrl is a restricted string data type
                                    /// to hold only 'mailto:' URLs.</summary>
    [ EPiServer.PlugIn.PageDefinitionTypePlugIn( DisplayName = "MailTo-Url" ) ]
    public class MailToUrl : EPiServer.Core.PropertyString {

        System.Web.UI.WebControls.TextBox MailToUrlTextBox = null;

        public override void CreateChildControls( string RenderType, System.Web.UI.Control Container ){
            switch ( RenderType.ToLower() ) {
                case "edit":
                    if ( MailToUrlTextBox == null ) {
                        MailToUrlTextBox = new System.Web.UI.WebControls.TextBox();
                    }
                    if ( base.Value != null ) {
                        MailToUrlTextBox.Text = (string) base.Value;
                    }
                    MailToUrlTextBox.ID = Name;
                    CopyWebAttributes( Container, MailToUrlTextBox );
                    Container.Controls.Add( MailToUrlTextBox );
                    Container.Controls.Add( CreateCustomParseValidator(
                        new System.Web.UI.WebControls.ServerValidateEventHandler(
                        MailToUrlValidator ) ) );
                    break;
                default:
                    base.CreateChildControls( RenderType, Container );
                    break;
            }
        }

        private void MailToUrlValidator( object Source,
            System.Web.UI.WebControls.ServerValidateEventArgs ValArgs ) {
            string MailToRegExp = "^['_a-z0-9-]+(\\.['_a-z0-9-]+)";
            MailToRegExp += "*@[a-z0-9-]+(\\.[a-z0-9-]+)*\\.([a-z]{2,3})";
            MailToRegExp += "|(aero|coop|info|museum|name))$";
            ValArgs.IsValid = new System.Text.RegularExpressions.Regex( MailToRegExp ).Match(
                MailToUrlTextBox.Text.Trim() ).Success;
            if ( ValArgs.IsValid ) {
                base.Value = MailToUrlTextBox.Text.Trim();
            } else {
                ( (System.Web.UI.IValidator) Source ).ErrorMessage =
                    "Must be a well-formed e-mail address, e.g. editor01@episerver.com.";
```

```
            }
        }
    }
}
```

Example 8-2 represents a class of property data types which are slightly more complex than BackgroundColourType, as it includes restrictions on entries enforced in code. The idea behind MailToUrl is that entries of this type should be valid 'mailto:' URLs. The regular expression using the function MailToUrlValidator forms a template for a valid e-mail address. As it's a bit on the strict side, only top-level domains either two or three letters long or one of 'aero', 'coop', 'info', 'museem' or 'name' is allowed. (It was found on the Internet and seems to be used by many people.)

In order to be able to perform the tests, we introduce a new control, CustVal, of the type System.Web.UI.WebControls.ServerValidateEventHandler. This is linked to the function MailToUrlValidator.

Example 8-3: CustomValidatorControl CustVal introduced (compare example 8-2).

```
...
Container.Controls.Add( MailToUrlTextBox );
System.Web.UI.WebControls.CustomValidator CustVal =
    new System.Web.UI.WebControls.CustomValidator();
CustVal.ControlToValidate = MailToUrlTextBox.ID;
CustVal.ServerValidate +=
    new System.Web.UI.WebControls.ServerValidateEventHandler( MailToUrlValidator );
Container.Controls.Add( CustVal );
```

Should the user input not pass the test, two important things must both be true:

- ValArgs.IsValid must be set to false
- An error message to be displayed must be assigned to Source.ErrorMessage

When both these conditions are met, an error text is presented in EPiServer Edit mode and further processing is not possible (until the error has been corrected).

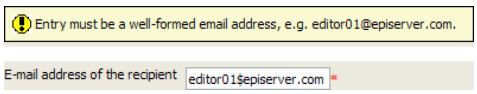


Figure 8-2: An error 'created' in custom validator is signalled in EPiServer Edit mode.

The error message assigned to Source.ErrorMessage is displayed in EPiServer Edit mode with a red asterisk displayed next to the failing input field.

Custom Filters

Custom filters, or filters for short, are connected to listings and menus. Filters aid in selecting what goes in the list and what doesn't. As we're talking EPiServer in this book, filters help getting the pertinent pages into listings or out of listings. One way of looking at them would be to see filters as an intermediary between the Web Page Tree and various lists and menus, often in the form of templated controls.

Filters perform three major tasks. They help in:

- Adding pages to listings
- Removing pages from listings
- Ordering listings

Filters are C# classes (or .NET classes, irrespective of what .NET language you happen to prefer).

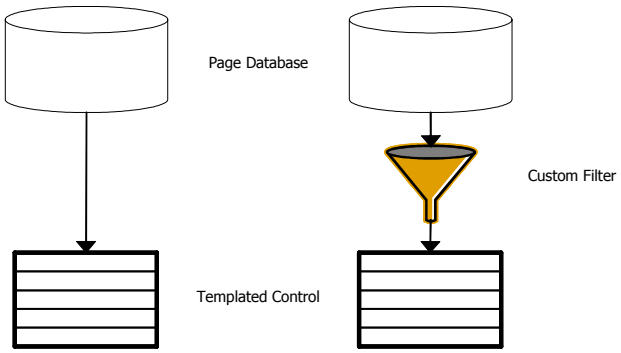


Figure 8-3: The role of Custom Filters. On the left a templated control fed from the Page database, and on the right a Custom Filter is used to select the data.

Figure 8-3 shows the place and role of Custom Filters. They act as go-betweens between data source and data recipient. The power of Custom Filters far exceed that of any other selection means available, e.g. FindPagesWithCriteria (EPiServer.DataFactory.FindPagesWithCriteria).

To fully appreciate filters, let's approach them in a roundabout manner. We will start by looking at a templated control which does not utilise any filtering, as in example 8-4.

Example 8-4: MenuList (EPiServer.WebControls.MenuList) using Start page as a starting point of a listing.

```
<EPiServer:PageList runat="server" id="CustomFilterPageList"
    PageLink="<%# EPiServer.Global.EPConfig.StartPage %>">
    <ItemTemplate>
        <tr>
            <td><EPiServer:Property PropertyName="PageLink" id="Prop1" runat="server" /></td>
```

```
        </tr>
    </ItemTemplate>
</EPiServer:PageList>
```

The code in example 8-4 produces a table of links comprising all the top-level pages (all pages that are daughter pages of the start page). But suppose we need this list to only display pages which in turn have their own daughter pages, what then? Well, this is a problem easily solved with filters (as you would've thought).

Let's break up the bond between the sender and receiver to insert a Filter. As a Filter is created in a .NET language, we can do anything we like in the Filter code.

Creating the Custom Filter Class

Example 8-5: Custom Filter Class CustomFilterOnlyMothers.

```
namespace development {
                                    /// <summary>CustomFilterOnlyMothers implements a filter for PageList
                                    /// controls to include only pages that have sub-pages.</summary>
    public class CustomFilterOnlyMothers {

        public CustomFilterOnlyMothers() {
        }

        public void OnFilter( object sender, EPiServer.Filters.FilterEventArgs e ) {
            EPiServer.Core.PageDataCollection PageColl = e.Pages;
            EPiServer.Core.PageData CurrPage;
            for ( int PageNum = 0; PageNum < PageColl.Count; PageNum++) {
                CurrPage = PageColl[ PageNum ];
                if ( EPiServer.Global.EPDataFactory.GetChildren( CurrPage.PageLink ).Count > 0 ) {
                    continue;
                }
                PageColl.RemoveAt( PageNum );
                PageNum--;
            }
        }

    }
}
```

The assignment to local variable PageColl is very important. As you can see in example 8-5, PageColl is not assigned a new PageDataCollection. It is simply set to point to the same collection as e.Pages. This is very important to remember, should you create a new collection and initialise this collection using e.Pages, your selection attempts will fail miserably. It will be as no filtering has taken place.

Please also notice that there's no change to the HTML part of the Web Form/ Web User Control, it's all happening in the code-behind file.

Connecting the Custom Filter to the Control

So, we have a control and we have a filter, now we need to connect them. Luckily for us, there's an event, Filter, implemented in the class PageControlBase (EPiServer.WebControls.PageControlBase). PageControlBase is an abstract class and mother class for very many of the templated controls in EPiServer including, e.g. PageList and MenuList.

The Filter event is always triggered before data is displayed in a control giving us the opportunity to insert our Custom Filter when this event is hooked.

To perform the connection between Custom Filter and control object we add one line of code to the OnInit function in the source-behind file of the Web Form or Web User Control onto which the templated control has been added.

Example 8-6: Code connecting the Custom Filter CustomFilterOnlyMothers to the PageList control CustomFilterPageList.

```
override protected void OnInit( System.EventArgs e) {
    InitializeComponent();
    base.OnInit( e );
    CustomFilterPageList.Filter += new EPiServer.WebControls.FilterEventHandler(
        ( new development.CustomFilterOnlyMothers() ).OnFilter );
}
```

In the code for example 8-6 you see 'development' used in class name for the first time in any example. You've probably already realised that this is the name space used for all EPiServer default installations ('namespace development {').

The Results of Using the Custom Filter

We used the Mimic Web site to create the filter, figure 8-4 shows the result from this site.

Figure 8-4: Before and after using the Custom Filter. On the left the templated control without the Filter and on the right the results after applying the Custom Filter.

More Information on the EPiServer Web Site

Filters may easily co-operate with properties added to pages. For more information about Custom Filters and an example on how to let Filters utilise page properties there's a document on the EPiServer Web site, http://www.episerver.com.

9
Data Modelling

This chapter is dedicated to data, we'll look more closely at EPiServer.DataFactory, the class responsible for all shuffling of Web pages between the database and the Web site. As the XML Web Services functions of EPiServer are dependent the database services we will discuss those here to and in the final part of the chapter we'll take a look at the export and import functions that were introduced in EPiServer 4.3, together with the exciting page synchronization functions.

EPiServer.DataFactory and EPiServer.Global.EPDataFactory

Both EPiServer.DataFactory and EPiServer.Global.EPDataFactory have been mentioned before, now the time has come to actually discuss them – they're very important to EPiServer Web site operation.

Take a look at figure 1-6 on page 10. If we blow up the little box labelled 'EPiServer DLLs' and its connection to the database we see that it's actually EPiServer.Global.EPDataFactory that handles the database access, i.e. it handles loading, saving and deleting pages as well as caching them.

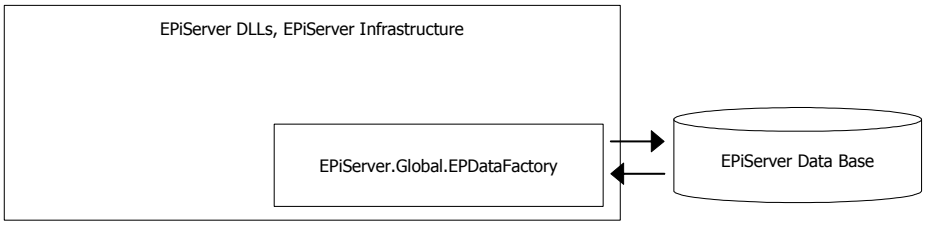


Figure 9-1: EPiServer.DataFactory object EPiServer.Global.EPDataFactory.

EPiServer.Global.EPDataFactory May Be Used in Non-Content Framework Web Forms

One use for EPiServer.Global.EPDataFactory is in Web Forms that for some reason do not adhere to the EPiServer Content Framework, meaning they do not inherit from EPiServer.PageBase or one of its descendants. In this case the developer may choose to implement EPiServer.Core.IPageSource directly in the Web Form. There's actually one example of this that ships with EPiServer. EPiServer has some provision for mobile units, such as WAP telephones and to assist in developing such solutions the Web Form Mobile.aspx is provided.

Example 9-1: Using EPDataFactory in non-Content Framework Web Form Mobile.aspx.

```
using EPiServer.Core;
...
public class Mobile : System.Web.UI.MobileControls.MobilePage, IPageSource {
...
    public virtual PageData GetPage( PageReference pageLink ) {
        return ( EPiServer.Global.EPDataFactory.GetPage( pageLink,
            EPiServer.Security.AccessLevel.Read ) );
    }

    public virtual PageDataCollection GetChildren( PageReference pageLink ) {
        return ( EPiServer.Global.EPDataFactory.GetChildren( pageLink,
            EPiServer.Security.AccessLevel.Read ) );
    }
...
}
```

As can be seen in example 9-1, the Web Form Mobile.aspx implements IPageSource.GetPage and IPageSource.GetChildren itself using EPiServer.Global.EPDataFactory.

It also implements IPageSource.CurrentPage. It is not shown in the example for space reasons. Mobile.aspx cannot rely on EPDataFactory for its CurrentPage implementation – EPDataFactory.CurrentPage always returns null.

Public Properties, Methods and Events for EPiServer.DataFactory

EPiServer.DataFactory implements the interface EPiServer.Core.IPageSource (see page 153), so we expect to find the property CurrentPage and the two methods GetPage and GetChildren.

Table 9-1: Public properties for EPiServer.DataFactory.

Property Name	*Description*
CurrentPage	Gets data for the current page from EPiServer.Core .IPageSource. DataFactory.CurrentPage always returns null – it was part of the IPageSource package.
DynPropTree	The global instance of DynamicPropertyTree that is used to access information about dynamic properties. Static (Shared in Visual Basic .NET).
ListingFetchCacheHits	The number of page listings that have been read from the cache.

Table 9-1: Public properties for EPiServer.DataFactory.

Property Name	*Description*
ListingFetchCount	How many page listings that has been returned from the DataFactory.
ListingFetchDatabaseReads	How many page listings has been read from the local database.
PageFetchCacheHits	How many pages has been read from the cache.
PageFetchCount	How many single pages that has been returned from the DataFactory.
PageFetchDatabaseReads	How many pages has been read from the local database.
StatisticsCollectedSince	Current statistics has been collected since this point in time.

Table 9-2: Public methods for EPiServer.DataFactory.

Method Name	*Description*
Copy	Copy a page to another container.
Delete	Delete a page from EPiServer
DeleteChildren	Delete children of a page from EPiServer
DeleteVersion	Delete a single version of a page.
FindPagesWithCriteria	Search for pages that match a specified set of criteria
GetChildren	Overloaded, from EPiServer.Core.IPageSource.
GetDefaultPageData	Get a PageData object with default values – the first step of creating a new page
GetPage	Overloaded, from EPiServer.Core.IPageSource.
LookupRemoteSite	Look-up a remote site by name.
Move	Move a page to another container
MoveToWastebasket	Move a page to the waste basket.
Ping	Calls a remote server for testing the connection.

Table 9-2: Public methods for EPiServer.DataFactory.

Method Name	*Description*
ReloadRemoteSites	Reload remote site cache.
ResetCounters	Reset all page statistics counters.
Save	Save page to database

Table 9-3: Public events for EPiServer.DataFactory.

Event Name	*Description*
CheckedInPage	Occurs when a version has been checked in.
CheckingInPage	Occurs when a version is requested to be checked in.
CreatedPage	Occurs when a new page is created.
CreatingPage	Occurs when a page is requested to be created.
DeletedPage	Occurs when a page has been deleted.
DeletingPage	Occurs when a page is requested to be deleted.
FailedLoading-Children	Occurs when a page list failed for some reason.
FailedLoadingPage	Occurs when a page load failed for some reason.
LoadedChildren	Occurs when a list has been loaded from GetChildren.
LoadedDefault-PageData	Occurs when a empty page is loaded through GetDefault-PageData.
LoadedPage	Occurs when a page is loaded through GetPage.
LoadingChildren	Occurs when a list is requested from GetChildren.
LoadingDefault-PageData	Occurs when an empty page is requested through GetDe-faultPageData.
LoadingPage	Occurs when a page is requested through GetPage.
MovedPage	Occurs when a page has been moved.
MovingPage	Occurs when a page is requested to be moved.
PublishedPage	Occurs when a page/version has been published.

Table 9-3: Public events for EPiServer.DataFactory.

Event Name	*Description*
PublishingPage	Occurs when a page/version is requested to be published.
SavedPage	Occurs when a existing page is modified and saved.
SavingPage	Occurs when a existing page is modified and saved.

More Information on Using the Properties, Methods and Events in EPiServer.DataFactory

EPiServer.DataFactory.DynPropTree

DynPropTree is an instance of the EPiServer.Core.DynamicPropertyTree class. It holds information about the dynamic properties that have been added to the EPiServer solution.

Page Cache Statistics Related Properties

Considering its role, it's only natural to find properties for EPiServer.DataFactory that tally pages as they move between the database, cache and Web server.

Example 9-2: Statistics example, from the SDK help file (EPiServer4SDK.chm).

```
<h2>Statistics</h2>
Since <%= EPiServer.Global.EPDataFactory.StatisticsCollectedSince.ToString( "r" ) %><br/>
<h3>Pages:</h3>
From Cache: <%= EPiServer.Global.EPDataFactory.PageFetchCacheHits %><br/>
From Database: <%= EPiServer.Global.EPDataFactory.PageFetchDatabaseReads %><br/>
Total: <%= EPiServer.Global.EPDataFactory.PageFetchCount %><br/>
Ratio: <%= Math.Round( (double) EPiServer.Global.EPDataFactory.PageFetchCacheHits / (double)
EPiServer.Global.EPDataFactory.PageFetchCount * 100, 2) %>% hit rate
<br/>
<h3>Listings:</h3>
From Cache: <%= EPiServer.Global.EPDataFactory.ListingFetchCacheHits %><br/>
From Database: <%= EPiServer.Global.EPDataFactory.ListingFetchDatabaseReads %><br/>
Total: <%= EPiServer.Global.EPDataFactory.ListingFetchCount %><br/>
Ratio: <%= Math.Round( (double) EPiServer.Global.EPDataFactory.ListingFetchCacheHits / (double)
EPiServer.Global.EPDataFactory.ListingFetchCount * 100, 2) %>% hit rate
```

To reset these counters and to set StatisticsCollectedSince call EPiServer.DataFactory.ResetCounters.

EPiServer.DataFactory.Delete

Example 9-3: Using DataFactory.Delete in an EPiServer Web User Control to delete the current page.

```
<!-- In the HTML part -->
```

```
<asp:LinkButton id="DeleteButton" onclick="DeletePage" runat="Server">[
<EPiServer:Translate Text="#delete" runat="server" id="Translate20" />]</asp:LinkButton>
...
// In the code-behind file.
protected void DeletePage( object sender, System.EventArgs e ) {
    EPiServer.Core.PageReference parentPage = CurrentPage.ParentLink;
    Global.EPDataFactory.Delete( PageBase.CurrentPageLink, true );
    EPiServer.Core.PageData parentData = GetPage( parentPage );
    Response.Redirect( parentData.LinkURL, true );
}
```

The code in example 9-3 is taken from an EPiServer Web User Control using EPiServer.DataFactory.Delete to delete the current page. It uses a combination of an ASP.NET LinkButton and a function in the code-behind file. The onclick event for the LinkButton is connected to the function DeletePage. This function is written to make the deletion smooth by first saving ParentLink property for the current page and later, after deletion of the current page using the saved ParentLink, redirecting the user's browser to that page.

EPiServer.DataFactory.DeleteChildren

DeleteChildren is the companion of Delete. It lets you delete all children of a page, e.g., the Recycle Bin.

EPiServer.DataFactory.FindPagesWithCriteria

Example 9-4: Using FindPagesWithCriteria to find pages that were published during the last seven days.

```
<!-- In the HTML part. -->
<EPiServer:PageList runat="server" ID="RecentlyPublishedPages">
    <HeaderTemplate>
        <tr><td>Recently Published Pages</td></tr>
    </HeaderTemplate>
    <ItemTemplate>
        <tr>
            <td><%# Container.CurrentPage.StartPublish.ToString( "r" ) %>
                 <%# Container.CurrentPage.PageName %>
            </td>
        </tr>
    </ItemTemplate>
</EPiServer:PageList>

// In the code-behind file.
EPiServer.PropertyCriteriaCollection RecentlyPublishedCriteria =
    new EPiServer.PropertyCriteriaCollection();

EPiServer.PropertyCriteria RecentlyPublishedCriterion = new EPiServer.PropertyCriteria ();
```

```
RecentlyPublishedCriterion.Condition = EPiServer.Filters.CompareCondition.GreaterThan;
RecentlyPublishedCriterion.Type = EPiServer.Core.PropertyDataType.Date;
RecentlyPublishedCriterion.Name = "PageStartPublish";
RecentlyPublishedCriterion.Value = System.DateTime.Now.AddDays( -7 ).ToString();
RecentlyPublishedCriterion.Required = true;
RecentlyPublishedCriteria.Add( RecentlyPublishedCriterion );

EPiServer.Core.PageDataCollection RecentPages;
RecentPages = EPiServer.Global.EPDataFactory.FindPagesWithCriteria(
    EPiServer.Global.EPConfig.RootPage, RecentlyPublishedCriteria );
RecentlyPublishedPages.DataSource = RecentPages;
RecentlyPublishedPages.DataBind();
```

One of the obvious uses for FindPagesWithCriteria is Web site maintenance, use it to find pages that were written by a certain person (e.g. using string property WriterName, or, as in example 9-4, to list all the pages that were published during the last seven days.

EPiServer.DataFactory.GetChildren and GetPage

GetChildren and GetPage are implementations of IPageSource.GetChildren and GetPage, respectively. This means that it's more natural to user either EPiServer.PageBase.GetChildren, or GetPage, or EPiServer.UserControlBase.GetChildren, or GetPage. However, not all Web Forms or Web User Controls in an EPiServer solution inherit from the EPiServer base classes and for them, as shown in example 9-1, EPiServer.DataFactory.GetChildren and GetPage are the best options.

EPiServer.DataFactory.GetDefaultPageData

It's not uncommon to have the need to create new Web pages in an EPiServer solution – remember: if it's a page it can be stored in the database and have the full support of the EPiServer infrastructure.

One such example is to provide a discussion forum. EPiServer ships with a template and Web User Control to provide this exact feature. If you open the example Web site and click on Templates in the top menu you'll see what it looks like.

Figure 9-2: Discussion forum template which ships with EPiServer.

When the link called 'Create new post' is clicked, another template is presented, one that allows creation of a new posting.

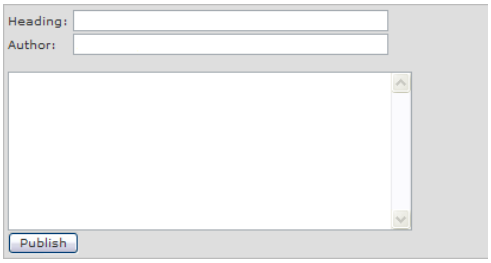


Figure 9-3: New discussion forum posting template.

The onclick event ASP.NET Button labelled Publish is connected to a function called SavePage.

Example 9-5: Function SavePage which saves newly created discussion forum postings.

```
protected void SavePage( object sender, System.EventArgs e ) {
    if ( _newPost ) {
        Page.Validate();
        if ( ! Page.IsValid ) {
            Page.DataBind();
            return;
        }
        EPiServer.Core.PageData newPage=
            Global.EPDataFactory.GetDefaultPageData( PageBase.CurrentPageLink, _pagetypeID );
        newPage.PageName= PageName.InnerProperty.Value.ToString();
        if ( ! MainBody.InnerProperty.IsNull ) {
            newPage[ "MainBody" ]= MainBody.InnerProperty.Value.ToString();
        }
        if ( ! WriterName.InnerProperty.IsNull ) {
            newPage[ "WriterName" ]= WriterName.InnerProperty.Value.ToString();
        }

        Global.EPDataFactory.Save( newPage, EPiServer.DataAccess.SaveAction.Publish );
        Response.Redirect( "Conference.aspx?id=" + newPage.PageLink.ID );
    } else {
        Global.EPDataFactory.Save( CurrentPage, EPiServer.DataAccess.SaveAction.Publish );
        Response.Redirect( "Conference.aspx?id=" + PageBase.CurrentPageLink.ID );
    }
}
```

If you look at the code in 9-5, you can see that after some initial checking a new EPiServer.Core.PageData is instantiated and given a default set of page data by calling GetDefaultPageData. Subsequently, the PageData.PageName property is

given the same contents and the header for the forum posting, in true EPiServer style. As the code is part of the example Web, site it's rather safe to assume that properties 'MainBody' and 'WriterName' both exists.

Lastly, the new page is saved and published and the viewer's Web browser re-directed to this new page.

EPiServer.DataFactory.Save

To save pages in the database, simply call EPiServer.Global.EPDataFactory.Save and you're done!

Example 9-6: Using EPiServer.DataFactory.Save.

```
private void SavePage() {
    if ( ( (EPiServer.EditPage) Page ).IsNewPage ) {
        CurrentPage[ "Sid" ] = PageBase.CurrentUser.Sid;
        CurrentPage.PageName = PageBase.CurrentUser.Identity.Name;
        CurrentPage.VisibleInMenu = false;
    }
    EPiServer.Global.EPDataFactory.Save( CurrentPage, EPiServer.DataAccess.SaveAction.Publish );
}
```

Example 9-6 shows code from an EPiServer Web User Control Profile.ascx. This control is used to handle user information, such as first and last name, e-mail address, etc. The function SavePage first tests whether the current page is a new page and if so sets a couple of attributes to the appropriate settings. Lastly EPiServer.DataFactory.Save is used to save the new, or changed, page in the database.

EPiServer.DataFactory Events

EPiServer.DataFactory.CreatingPage

As CreatingPage is triggered each time a new page is being created, you can use it for validation tasks which aren't as simplistic as just making sure a value has been entered (as this can be done by checking the pertinent check box in EPiServer Admin for any property/value).

The code in example 9-7 hooks the CreatingPage event to make sure that if there is a property/value called 'WriterName', it gets some content which links it back to the logged-on user.

Example 9-7: Using CreatingPage.

```
protected void Application_Start( object sender, System.EventArgs e ) {
    EPDataFactory.CreatingPage += new EPiServer.PageEventHandler( OnCreatingPage );
}

private void OnCreatingPage( object sender, EPiServer.PageEventArgs e ) {
    if ( e.Page.Property.Exists( "WriterName" ) ) {
        if ( e.Page.Property[ "WriterName" ].IsNull ) {
```

```
            e.Page[ "WriterName" ] =
                ( EPiServer.Security.UnifiedPrincipal.Current.UserData.Email != null ?
                    EPiServer.Security.UnifiedPrincipal.Current.UserData.Email :
                    EPiServer.Security.UnifiedPrincipal.Current.UserData.DisplayName );
        }
    }
}
```

EPiServer.DataFactory.PublishedPage

This event is used by the page synchronization example. See example 9-13 on page 231.

EPiServer.DataFactory.SavingPage

The following example was inspired by a question posed in the Developer Community forum on www.episerver.com. The issue was how to replace certain national letters with their corresponding HTML Entity tag, e.g. always replace 'æ', with 'æ'. EPiServer.DataFactory.SavingPage is an excellent candidate to help resolve this issue.

Example 9-8: Using EPiServer.EPDataFactory.SavingPage in Global.asax.cs.

```
namespace development {
                            /// <summary>EPiServer.Global is the focal point for all database access
                            /// and home to EPiServer.Global.EPDataFactory.</summary>
    public class Global : EPiServer.Global {

        protected void Application_Start( object sender, System.EventArgs e ) {
            EPDataFactory.SavingPage += new EPiServer.PageEventHandler( OnSavingPage );
        }

        string NationalLetterToHtmlEntity( string StringWithNationalLetters ) {
                            // Replace Nordic letter with HTML entity.
            StringWithNationalLetters = StringWithNationalLetters.Replace( "æ", "&aelig;");
            return StringWithNationalLetters;
        }

        private void OnSavingPage( object sender, EPiServer.PageEventArgs e ) {
            for ( int PropNum = 0; PropNum < e.Page.Property.Count; PropNum++ ) {
                EPiServer.Core.PropertyData     StringProp = e.Page.Property[ PropNum ];
                if ( ( StringProp.Type == EPiServer.Core.PropertyDataType.LongString ) &&
                    ( ! StringProp.IsNull ) ) {
                    StringProp.Value = NationalLetterToHtmlEntity( (string) StringProp.Value );
                }
            }
        }
```

```
    }
}
```

The code in example 9-8 connects the function OnSavingPage to the event SavingPage. In OnSavingPage, properties on the current page are enumerated and the Value property of all LongString properties is passed to the function NationalLetterToHtmlEntity.

Depending on your needs, it is easy to adapt the code to other scenarios, e.g. if you have a large number of LongString attributes, you may want to introduce a StringBuilder local variable in the function NationalLetterToHtmlEntity. Also, you might elect to only convert certain named properties as in the following example.

Example 9-9: Using EPiServer.EPDataFactory.SavingPage in Global.asax.cs to convert MainBody.

```
private void OnSavingPage( object sender, EPiServer.PageEventArgs e ) {
    const string PropName = "MainBody";
    if ( e.Page.Property.Exists( PropName ) ) {
        e.Page.Property[ PropName ].Value =
            NationalLetterToHtmlEntity( (string) e.Page.Property[ PropName ].Value );
    }
}
```

XML Web Services and EPiServer

Correctly naming Web Services as present in ASP.NET has been like trying to catch the wind – it keeps getting away. Whether you use the name 'Web Services', 'XML Web Services', 'SOAP Web Services' or any other name, EPiServer both acts as a Web Service and also plays nicely with others.

There are several possibilities involving EPiServer and Web Services.

- EPiServer can act as a server to any Web Services client
- One EPiServer Web site may access DataFactory on another EPiServer Web site
- EPiServer may use a Web Service as a data source in addition to DataFactory.

The Web Services possibilities in EPiServer stem from the fact that EPiServer exposes DataFactory as a Web Service.

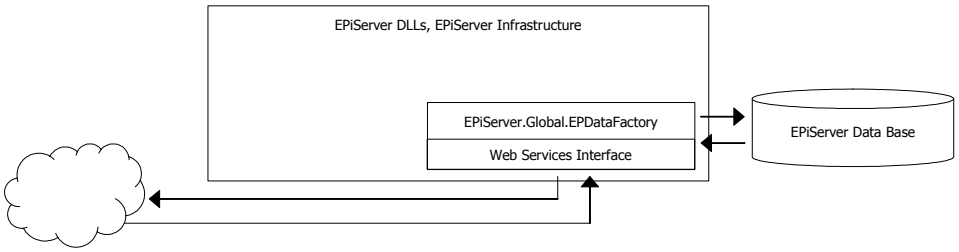


Figure 9-4: EPServer.Global.EPDataFactory is exposed as a Web Service.

In figure 9-4, which is an amendment to figure 9-1 on page 211, EPiServer's Web Services exposure layer is shown as an integral part of EPiServer.Global.EPDataFactory.

Consuming Data from an EPiServer Web Site – Web Services Client

Creating an EPiServer Web Services consumer client involves creating a DataFactoryService object to communicate with EPiServer.Global.EPDataFactory. Depending on the jobs in hand, we could also make the acquaintance of some rather obscure EPiServer data types. In our example, we will utilise the Web Services capabilities of EPiServer to retrieve a list of recently created Web pages (recent for the current EPiServer Web site, that is). In doing so, we'll be able to use several familiar methods and data types, but in completely new guises. More about that when we've had a chance to look at the source code.

Preparations

In order for the Web Services client to work, we need to be able to access the WebServices folder in the root folder of your EPiServer Web site. To do this we use a particular procedure outlined in the Technical note document *Authenticating Web services on forms authenticated site*. This document is available on the EPiServer home site, http://www .episerver.com. In the Web Services Client example, we assume that the account is called 'MyWebServiceAccount' and that its password is '7thHeaven'.

Create a Windows Application Web Services Client

The Web Services client we create is an ordinary Visual Studio .NET Windows Application. It is made into a Web Services client by including a Web Reference to a Web Services server application.

The client will retrieve the Web pages when a Button is clicked. Information on returned pages will be presented in a ListBox.

1. Create a new Windows Application project in Visual Studio .NET. You might want to name it 'WSFindRecentPages'.

2. Name the class 'FindRecentPages' and form file 'FindRecentPages.cs'.

3. Add a Web Reference to References (Solution Explorer). The URL to the Web server will look like this http://*server*/WebServices/DataFactoryService.asmx. You should see a page much like figure 9-5.

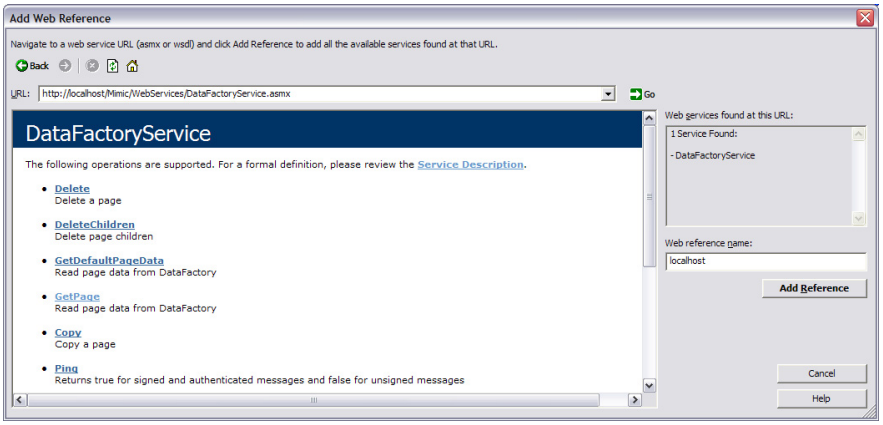


Figure 9-5: Page returned when logging on to the page http://.../WebServices/DataFactoryService.asmx.

4. Add a Button and a ListBox to the Form.

5. Everything that's supposed to happen will take place in the Click event for the Button. Click on the Button in Design mode and add this code.

Example 9-10: Code to retrieve Web pages from EPiServer via its Web Services interface.

```
private void butnGetRecentPages_Click( object sender, System.EventArgs e ) {
    WSFindRecentPages.localhost.DataFactoryService GetPagesFactory =
        new WSFindRecentPages.localhost.DataFactoryService();

    GetPagesFactory.Credentials = new System.Net.NetworkCredential(
        "MyWebServiceUser", "7thHeaven" );
    GetPagesFactory.PreAuthenticate = true;

    WSFindRecentPages.localhost.PropertyCriteria RecentlyCreatedCriterion =
        new WSFindRecentPages.localhost.PropertyCriteria();
    RecentlyCreatedCriterion.Condition=
        WSFindRecentPages.localhost.CompareCondition.GreaterThan;
    RecentlyCreatedCriterion.Type= WSFindRecentPages.localhost.PropertyDataType.Date;
    RecentlyCreatedCriterion.Name= "PageCreated";
    RecentlyCreatedCriterion.Value= System.DateTime.Now.Date.AddDays( -7 ).ToString();
    RecentlyCreatedCriterion.Required= true;
    WSFindRecentPages.localhost.PropertyCriteria[] RecentlyCreatedCriteria =
        new WSFindRecentPages.localhost.PropertyCriteria[] { RecentlyCreatedCriterion };

    WSFindRecentPages.localhost.PageReference StartPageRef=
```

```
            new WSFindRecentPages.localhost.PageReference();
        StartPageRef.ID = 3;

        WSFindRecentPages.localhost.RawPage[] RecentPages =
            GetPagesFactory.FindPagesWithCriteria( StartPageRef, RecentlyCreatedCriteria );
        foreach ( WSFindRecentPages.localhost.RawPage CurrPageData in RecentPages ) {
            string PropsString = string.Empty;
            foreach ( WSFindRecentPages.localhost.RawProperty RawProp in CurrPageData.Property ) {
                PropsString += RawProp.DisplayName + "\t"
                    + ( RawProp.Value != null ? RawProp.Value : "<null>") + "\t";
            }
            listRecentPages.Items.Add( PropsString );
        }
    }
```

6. Compile, build, run and then click on the Button. Provided some pages have actually been made during the last seven days, the ListBox should look a bit like this.

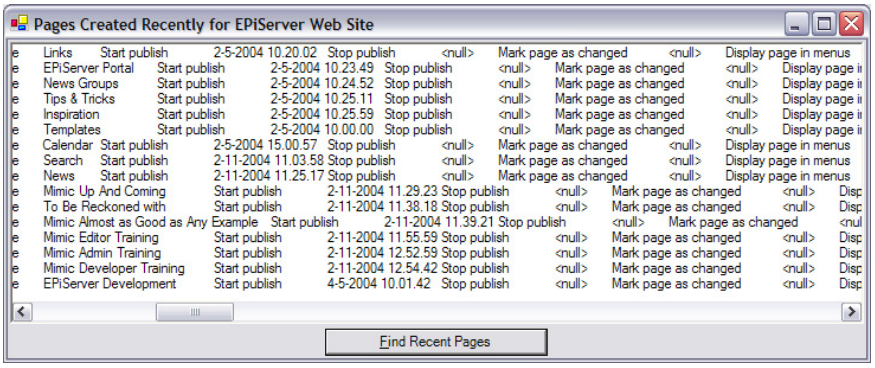


Figure 9-6: Example of results using Web Services to retrieve page information from EPiServer.

Now, there's a lot of interesting things going on in the Click event service function. Firstly, example 13-20 on page 293, has been doing the same basic assignment of retrieving recently created pages from EPiServer. The big difference between that code and the present is that this time we're far away from an EPiServer so we cannot use all the nifty constructs and data types that are available for EPiServer components. Let's start from the beginning with WSFindRecentPages.localhost.DataFactoryService. 'WSFindRecentPages' is the current name space, 'localhost' is the name of server carrying the Web Services functions. So, instead of adding EPiServer.dll and System.Web.dll and using EPiServer.DataFactory as the data type for GetPagesFactory, you have to use WSFindRecentPages.localhost.DataFactoryService. As you'll soon be aware, this holds true for every class that has an EPiServer origin.

Moving along, we skip the credentials bit – nothing special here. Next stop is the PropertyCriteria class. This is also of the WSFindRecentPages.localhost 'persuasion', and you have to adapt when assigning contents to its properties. FindPagesWithCritiera in this guise does not use a collection but rather an array of criteria. Oh well, just do what you're told. The penultimate interesting part is the return type of FindPagesWithCriteria – this returns an array of WSFindRecentPages.localhost.RawPage objects. These RawPage objects are very similar to EPiServer.Core.RawPage, so we can use information about this to help with WSFindRecentPages.localhost.RawPage. EPiServer.Core.RawPage has two fields: ACL and Property. The said RawPage.Property is of the type EPiServer.Core .RawProperty.

As you can see in the code in example 9-10, we enumerate the RawProperty objects found in the RawPage.Property collection. Doing this is quite interesting, as the name of the RawProperty (EPiServer.Core.RawProperty.Name) is the same as you would use in a EPiServer.WebControls.Property statement. The RawProperty names in table 9-4 were returned running the code above.

Table 9-4: RawProperty.Name returned to Web Services client.

RawProperty.Name	*RawProperty.Name*	*RawProperty.Name*
PageCategory	PageLink	PageTypeID
PageParentLink	PagePendingPublish	PageWorkStatus
PageFolderID	PageDeleted	PageName
PageStartPublish	PageStopPublish	PageChangedOnPublish
PageVisibleInMenu	PageCreated	PageChildOrderRule
PagePeerOrder	PageLanguageID	PageArchiveLink
PageExternalURL	PageShortcutType	PageShortcutLink
PageTargetFrame	PageLinkURL	PageTypeName
PagePeerOrderRule	PageChanged	PageSaved
PageUseOutputCache	PageCreatedBy	PageChangedBy
PageCreatedSID	MainBody	WriterName
ConferenceContainer	MainSearchPage	

Import and Export

Export and Import Functions in the EPiServer Admin Mode

EPiServer 4.3 sees the introduction of the name space EPiServer.Enterprise. It contains the classes needed to do in code what can be done with EPiServer Admin mode menu choices Export data and Import data.

Figure 9-7: EPiServer Admin mode menu, Tools section (incomplete).

The classes in EPiServer.Enterprise, as well as the Export/Import data administrative functions, allow EPiServer objects to be copied to and from one EPiServer site to another. Both parties involved must be EPiServer Web sites, as the format used is proprietary.

The reason to include the Export/Import data function in the Admin mode of EPiServer was to facilitate moving a complete EPiServer Web site off a development system and into a production environment. Moving from one production environment to another is also easy using the Export/Import functions.

Export and Import Classes in EPiServer.Enterprise

Great care and a lot of effort has gone in to the creation of the new classes in the EPiServer.Enterprise name space.

We have tried our utmost to make these classes as robust as possible and as graceful as possible when failing. An important task has been to design decision routes to handle the very many translations that are bound to appear when importing EPiServer objects into an existing Web site. As EPiServer is all about developer freedom, the import classes have been designed to make intelligent choices when handling conflicts during import. (We do not envisage many problems when exporting objects from a Web site.)

The classes are able to handle such non-trivial tasks as what to do when importing page objects which have properties categorised in specific ways into an EPiServer system which doesn't have the same category definitions.

An example of this particular problem is the import of a page utilising properties in the categories 'News' and 'Technical' into a system that defines only one of them, Technical. During import, all properties in the 'Technical' category will be imported without any problem and a warning message will be generated about the missing 'News' category. The import will then move on to the next task, meaning that there should be no interruption during import when 'likely' problems appear. If a Page Type with the same name as an existing Page Type is imported, the existing Page Type is preserved, but is updated with the properties from the importee.

The same kind of gracefulness applies to other data types, i.e. always warn, never break execution as long as the problem can be managed.

EPiServer.Enterprise

ExportImportBase

Base class for export and import of pages.

Public Properties

Table 9-5: Public properties for EPiServer.Enterprise.ExportImportBase.

Property Name	*Description*
AllowPageSync	Allow pages to be synchronized using the PageLookup during import operation
AutoCloseStream	If the input stream is to be closed automatically after the import/export method has been called
Categories	Categories that should be exported or have been imported
CurrentContext	Access the current export/import context without access to the original class
DestinationPages	Pages that should be exported or have been imported
DestinationRoot	The destination root where new pages should be created
DynamicPropertyDefinitions	Definitions of dynamic properties that should be exported or have been imported
Errors	A collection of errors which have been added by calls to the LogError method
ExternalFolderLookup	A look-up table between external folders ID on the source and destination
Files	Files which should be exported or have been imported
Frames	Frames which should be exported or have been imported
HasErrors	If any errors have been reported

Table 9-5: Public properties for EPiServer.Enterprise.ExportImportBase.

Property Name	*Description*
IncludePageDependencies	Set to true to include page dependencies, false otherwise
IsAborting	If the current operation has been aborted by a call to the Abort method
IsDone	If the current import/export is done
IsTest	If the current import should be started in test mode
LogContext	Set a context or prefix used for logging
PageLookup	A look-up table which contains source page ID and destination page ID.
PageTypes	Page types which should be exported or has been imported
ProgressLog	A collection of progress messages which have been added by calls to the LogProgress method
Stream	The stream which should be used to write or read the actual package
TabDefinitions	Tabs to be exported or have been imported
Warnings	A collection of warnings which have been added by calls to the LogWarning method

Public Methods

Table 9-6: Public methods for EPiServer.Enterprise.ExportImportBase.

Method Name	*Description*
Abort	Abort the running export/import
AddFileToExport	Add a custom file to an export package
GetDestinationPage	Get the page on the destination based on an ID number from the source
LogError	Log an error message
LogProgress	Log a debug message concerning the progress

Table 9-6: Public methods for EPiServer.Enterprise.ExportImportBase.

Method Name	*Description*
LogWarning	Log a warning message

DataExporter

Export pages to binary format.

Public Methods

Table 9-7: Public Methods for EPiServer.Enterprise.DataExporter.

Method Name	*Description*
AddFileToExport	Add a custom file to an export package
Export	Execute the export

DataImporter

Public Methods

Table 9-8: Public Methods for EPiServer.Enterprise.DataImporter.

Method Name	*Description*
Import	Execute the import

Example Code for Using the EPiServer.Enterprise Classes

Exporting a Page Type

Example 9-11: Exporting a single Page Type using EPiServer.Enterprise.DataExporter.

```
EPiServer.Enterprise.DataExporter exporter = new EPiServer.Enterprise.DataExporter();
exporter.PageTypes.Add( EPiServer.DataAbstraction.PageType.Load( "Ordinary web page") );
exporter.Stream = new System.IO.FileStream( "c:\\pagetype.epi4", System.IO.FileMode.Create );
exporter.Export();
```

Importing a Page Type

Example 9-12: Importing a single Page Type using EPiServer.Enterprise.DataImporter.

```
EPiServer.Enterprise.DataImporter importer = new EPiServer.Enterprise.DataImporter();
importer.Stream = new System.IO.FileStream( "c:\\pagetype.epi4", System.IO.FileMode.Open );
importer.Import();
```

Handling Warnings and Errors

Warning and error messages are collected in the properties Warnings and Errors in class ExportImportBase. Progress messages are collected in the property ProgressLog. (Errors, ProgressLog and Warnings all have the data type System.Collections.Specialized.StringCollection.)

Synchronizing Pages Between EPiServer Web Sites

One of many things to get excited about in EPiServer 4.3 is the possibility of synchronizing pages between EPiServer Web sites. Synchronizing is not nearly as effortless as using DataImporter and DataExporter, but then again, we've tried to make it as powerful as possible.

The synchronization is wholly under developer control, meaning you decide what to synchronize and you're also responsible for maintaining the look-up table used for synchronization.

Property AllowPageSync is the Focal Point

The focal point for page synchronization is the property AllowPageSync (EPiServer.Enterprise.ExportImportBase.AllowPageSync). Setting this property to true enables page synchronization.

PageLookup Provides Focusing

Although it is necessary to set AllowPageSync to true, it's not enough. You also have to populate the collection PageLookup. PageLookup has a data type of EPiServer.Enterprise.Util.IntLookup, which allows enumeration using System .Collections.DictionaryEntry.

PageLookup holds pairs of source and destination identification strings, in effect providing a translation table between the page exported from the source and the page already existing in the destination whose contents we wish to synchronize with the source page.

Simple and Manageable Page Synchronisation Example

Synchronized with the release of EPiServer 4.3, a comprehensive example of Web page synchronization was made available. The simplified example provided below was derived from that larger example.

We have left a lot of otherwise necessary coding out of this example. For one thing, there is currently nothing to trigger the import action. You might want to implement the importing functions as a scheduled job (see chapter 11, *Job Scheduling*). Nor will this example handle warnings, errors or bother with the progress log.

On the Exporting Web Site

The SimpleSyncSample uses a System.Data.DataSet to create a table to be used when importing. The table proper is passed between the Web sites in an XML file along with its schema.

On the exporting Web site, we hook the EPiServer.DataFactory.PublishedPage event, so each time a page is published our code will be automatically run.

The facility to copy, or move, the files from the exporting server to the importing is simply to copy the output files to a server share on the importing server.

Example 9-13: Code on the exporting side to synchronize pages between Web sites.

```
namespace SimpleSyncSample {
                                /// <summary>Synchronises page information between sites.
                                /// </summary>
    public class ReplicationPublisher {

        publicstring DropPath= "\\\\server\PageSync\DropFolder\\";

        public void Init() {
            EPiServer.Global.EPDataFactory.PublishedPage+=
                new EPiServer.PageEventHandler( OnPublish );
        }

        private void OnPublish( object sender, EPiServer.PageEventArgs e ) {
            System.Guid guid = System.Guid.NewGuid();
            ExportPage( e.PageLink, guid.ToString() + ".epi4" );
            CreateEvent( guid, "Test", e.PageLink.ID, e.TargetLink.ID, guid.ToString() + ".epi4" );
        }

        private void CreateEvent( System.Guid guid, string PageName, int sourceID, int destinationID,
            string fileName ) {
            System.Data.DataSet ds= new System.Data.DataSet( "EventPackage" );
            System.Data.DataTable tbl= new System.Data.DataTable( "Events" );
            tbl.Columns.Add( "SourceID",typeof( int ) );
            tbl.Columns.Add( "DestinationID",typeof( int ) );
            tbl.Columns.Add( "ImportFile",typeof( string ) );
            ds.Tables.Add( tbl );
            System.Data.DataRow row= tbl.NewRow();
            row[ "SourceID" ]= sourceID;
            row[ "DestinationID" ]= destinationID;
            row[ "ImportFile" ]= fileName;
            tbl.Rows.Add( row );
            ds.WriteXml( DropPath + guid.ToString() + ".event",
                System.Data.XmlWriteMode.WriteSchema );
        }
```

```
        private void ExportPage( EPiServer.Core.PageReference pageLink, string fileName ) {
            EPiServer.Enterprise.DataExporter export = new EPiServer.Enterprise.DataExporter();
            export.DestinationPages.Add( EPiServer.Global.EPDataFactory.GetPage( pageLink,
                EPiServer.Security.AccessControlList.NoAccess ) );
            export.Stream = new System.IO.FileStream( DropPath + fileName,
                System.IO.FileMode.CreateNew );
            export.Export();
        }

    }

}
```

The code in example 9-13 shouldn't be too hard to follow. Whenever a page is published it is also exported, by the function ExportPage, along with the events file (file extension 'event').

On the Importing Web Site

On the importing side of the synchronization puzzle, there's a bit more to be done. Every file in the import folder is processed and moved to another folder.

The page file proper is processed in the function ImportEPi4.

Example 9-14: Code on the importing side to synchronize pages between Web sites.

```
namespace SimpleSyncSample {
                                /// <summary>Synchronizes page information between sites.
                                /// </summary>
    public class ReplicationSubscriber {
        publicstring                        ListenPath   = "c:\\sync\\drop\\";
        publicstring                        LogPath      = "c:\\sync\\log\\";
        publicEPiServer.Core.PageReferenceSyncToPage= new EPiServer.Core.PageReference ( 44117 );

        private void ImportEvents() {
            System.IO.FileInfo[]files= new System.IO.DirectoryInfo( ListenPath ).GetFiles( "*.event" );
            System.Data.DataSetds= GetPageLookup();

            foreach ( System.IO.FileInfo file in files ) {
                System.Data.DataSet evt = new System.Data.DataSet( "EventPackage" );
                evt.ReadXml( file.FullName );

                foreach ( System.Data.DataRow row in evt.Tables[ "Events" ].Rows ) {
                    int sourceID = (int) row[ "SourceID" ];
                    int destinationID = (int) row[ "DestinationID" ];
                    string importFileName = row[ "ImportFile" ] as string;

                    if ( System.IO.File.Exists ( ListenPath + importFileName ) ) {
                        ImportEPi4( ds, ListenPath + importFileName );
                    }
```

```
            if ( importFileName != null ) {
                System.IO.File.Move( ListenPath + importFileName, LogPath + importFileName);
            }
        }

        file.MoveTo( LogPath + file.Name );
    }

    ds.WriteXml( ListenPath + "map.xml", System.Data.XmlWriteMode.WriteSchema );
}

private int LookupPage( int pageID, System.Data.DataSet ds ) {
    System.Data.DataRow[] rows = ds.Tables[ "PageLookup" ].Select(
        "SourceID=" + pageID .ToString() );
    if ( rows.Length == 0 ) {
        return -1;
    } else {
        return (int) rows[ 0 ][ "TargetID" ];
    }
}

private System.Data.DataSet GetPageLookup() {
    System.Data.DataSet ds = new System.Data.DataSet( "Sync" );

    if ( System.IO.File.Exists ( ListenPath + "map.xml" ) ) {
        ds.ReadXml ( ListenPath + "map.xml" );
    } else {
        System.Data.DataTable tbl = new System.Data.DataTable( "PageLookup" );
        tbl.Columns.Add( "SourceID", typeof ( int ) );
        tbl.Columns.Add( "TargetID", typeof ( int ) );
        ds.Tables.Add( tbl );
    }

    return ds;
}

private void ImportEPi4( System.Data.DataSet ds, string fileName ) {
    EPiServer.Enterprise.DataImporter importer = new EPiServer.Enterprise.DataImporter();
    importer.DestinationRoot = SyncToPage;
    importer.AllowPageSync = true;

    foreach ( System.Data.DataRow row in ds.Tables[ "PageLookup" ].Rows ) {
        if ( ! importer.PageLookup.Contains( (int) row[ "SourceID" ] ) ) {
            importer.PageLookup.Add( (int) row[ "SourceID" ], (int) row[ "TargetID" ] );
        }
    }
```

```
                importer.Stream = System.IO.File.OpenRead( fileName );
                importer.Import();

                foreach ( System.Collections.DictionaryEntry entry in importer.PageLookup ) {
                    int source = (int) entry.Key;
                    int target = (int) entry.Value;

                    System.Data.DataRow[] hits = ds.Tables[ "PageLookup" ].Select(
                        "SourceID=" + source );
                    if ( hits.Length > 0 ) {
                        continue;
                    }

                    System.Data.DataRow newRow = ds.Tables[ "PageLookup" ].NewRow();
                    newRow[ "SourceID" ] = source;
                    newRow[ "TargetID" ] = target;
                    ds.Tables[ "PageLookup" ].Rows.Add( newRow );
                }
            }

        }

    }
```

10
Personalization

An EPiServer information solution is a highly personalizable environment. With the classes, interfaces and enumerations in the EPiServer.Personalization name space, you can easily personalize the user environment. EPiServer.Personalization is also used to assign Tasks to Editors (or to Admins for that matter).

As part of the personalization capabilities, custom properties for every user and for every page can be stored opening up a wealth of opportunities. Portal settings, background colours and images, even skinning of the Web site, are but a few of the possibilities.

Contents of the EPiServer.Personalization Name Space

The EPiServer.Personalization name space supplies classes for personalization and subscription. The PersonalizedData class can be used to retrieve and save information about a user, in addition to storing global personalized values or page-specific values.

Table 10-1: Classes in the EPiServer.Personalization name space.

Class Name	*Description*
PersonalizedData	Data class handling personalized information for a user.
Subscription	Handles subscription for the current user through personalization
SubscriptionInfo	Handles subscription for a user other than the one currently logged in. Use Subscription for the currently logged on user.
SubscriptionJob	Job which handles subscriptions; scheduled automatically by EPiServer Scheduler. Should not be called from your code.
SubscriptionMail	The class which handles sending of e-mail notifications for subscriptions, used by SubscriptionJob.

Table 10-1: Classes in the EPiServer.Personalization name space.

Class Name	*Description*
SubscriptionPlugInAttribute	For subscription plug-ins (enter as 'Subscription-PlugIn' or 'SubscriptionPlugInAttribute').
Task	Task representation

Table 10-2: Interfaces in the EPiServer.Personalization name space.

Interface Name	*Description*
ISubscriptionHandler	Supports sending of customized subscriptions.

Table 10-3: Enumerations in the EPiServer.Personalization name space.

Enumeration Name	*Description*
TaskStatus	TaskStatus has the floowing members: NotStarted, InProgress, Completed, Rejected.

Class PersonalizedData (EPiServer.Personalization.PersonalizedData)

Being the holder of personalized information, a lot centres around class PersonalizedData.

Table 10-4: Public properties for EPiServer.Personalization.PersonalizedData.

Property Name	*Description*
Address	User address
Company	User organisation name
Current	Access personalized information for the current user [**static**]
Description	User description
DisplayName	Returns a combination of first name, last name and user name depending on which are set.
Email	User e-mail address
FirstName	Users first name
IsModified	Check to see if any value has been modified

Table 10-4: Public properties for EPiServer.Personalization.PersonalizedData.

Property Name	*Description*
IsPersonalModified	Check to see if any personal information has been changed, for example FirstName, LastName etc.
Item	Indexer.
Language	User language
LastName	User last name
Mobile	User mobile phone number
PersonalizedPages	Array of all pages which have personalized information for this user.
PostalAddress	User postal address
PostalNumber	User postal number
Telephone	User telephone number
Title	User title

There are four public methods which PersonalizedData defines itself, or three as you don't really need the Save method.

Table 10-5: Public methods for EPiServer.Personalization.PersonalizedData.

Method Name	*Description*
GetProperties	Get the collection containing all page restricted properties; only used to iterate through values.
Load	Load information on users other than the currently logged on user.
Save	Force changes to personalized data to be saved. You should not need to call this method directly, since any changes to the current data are flushed automatically on the Page.Unload event. This method primarily remains for backwards compatibility.
SetPersonalized-Properties	Add personalized properties to a page.

Item, or Storing Other Personalized Settings

Table 10-4 lists the pre-defined properties for PersonalizedData. You are free to store whatever settings you like in the Item array. These settings are either page-

global for the user or per page. Page-global settings can be thought of as dynamic properties set for the root page of the Web Page Tree.

Example 10-1: Accessing a custom personal property (from EPiServer4SDK.chm).

```
if ( PersonalizedData.Current != null ) {
    if ( PersonalizedData.Current[ "MyTestKey" ] != null ) {
        Response.Write( PersonalizedData.Current[ "MyTestKey" ] );
    }
    PersonalizedData.Current["MyTestKey"] = "NewContents";
}
```

Example 10-1 shows how to access a custom property for the currently logged-on user.

Example 10-2: Accessing a custom personal property linked to the current page (from EPiServer4SDK.chm).

```
if ( PersonalizedData.Current != null ) {
    if ( PersonalizedData.Current[ "MyTestKey", CurrentPage.PageLink ] != null ) {
        Response.Write( PersonalizedData.Current[ "MyTestKey", CurrentPage.PageLink ] );
    }
    PersonalizedData.Current[ "MyTestKey", CurrentPage.PageLink ] = "NewPageContents";
}
```

Example 10-2 shows how to access a custom property on the current page for the currently logged-on user.

Database Storage

Personalized information is stored in the EPiServer database table tblUser and tblUserProperty, see figure B-2 on page 317. The same figure also provides clues as to how the EPiServer infrastructure is fo exanple able to maintain the PersonalizedPages property for PersonalizedData objects.

Using the EPiServer.Personalization Name Space

Accessing Information for the Currently Logged-On User

Depending on the context, you can access information for the currently logged-on user in several ways. All three properties in table 10-6 return an object of the type UnifiedPrincipal (EPiServer.Security.UnifiedPrincipal).

Table 10-6: Ways to access information for the currently logged-on user.

Property Name	*Context*
EPiServer.Security.UnifiedPrincipal .**Current**	Globally accessible, can be used everywhere, even in pages and code-behind files which are not created from EPiServer Page Types nor inherit any base class (or class descendant from a base class).
EPiServer.Security.UnifiedPrincipal .**CurrentSid**	Globally accessible, just like Current. Suitable as an argument for the PersonalizedData .Load method.
EPiServer.PageBase .**CurrentUser**	Available in pages and code-behind files of pages created from Page Types which are part of the EPiServer Content Framework.
UserControlBase.PageBase .**CurrentUser**	Can be used in User Controls which inherit EPiServer.WebControls.UserControlBase or any of its descendant classes.

Armed with the UnifiedPrincipal object, its personal data is accessed via the UserData property of the type PersonalizedData (EPiServer.Personalization.PersonalizedData).

The obvious shortcut to using any of the properties in table 10-6 is the static property PersonalizedData.Current, which returns a PersonalizedData object for the currently logged-on user.

PersonalizedData.GetProperties

Method GetProperties is perhaps mostly used to enumerate the contents of the property array for the current PersonalizedData object. It has two overloads, one for retrieving all custom user property settings and one for retrieving custom user property settings connected to a certain Web page (in accordance with the Item property itself).

Example 10-3: Enumerating all properties for the currently logged-on user.

```
if ( EPiServer.Security.UnifiedPrincipal.Current != EPiServer.Security.UnifiedPrincipal.AnonymousUser ) {
  foreach( string NameString in EPiServer.Personalization.PersonalizedData.Current.GetProperties() ) {
```

```
        if ( EPiServer.Personalization.PersonalizedData.Current[ NameString ] != null ) {
...
        }
    }
}
```

GetProperties returns all personalized data, including Address, Company, etc. In doing so, the name of the property (variable NameString) is prepended with 'Personal' in the same manner as page property names are prepended with 'Page'. So, to access the 'Address' property you would write 'PersonalAddress', and so on.

Example 10-4: Enumerating all properties for the current page for the currently logged-on user.

```
if ( EPiServer.Security.UnifiedPrincipal.Current != EPiServer.Security.UnifiedPrincipal.AnonymousUser ) {
 if ( EPiServer.Personalization.PersonalizedData.Current.GetProperties( CurrentPage.PageLink ) != null ) {
        foreach( string NameString in EPiServer.Personalization.PersonalizedData.Current.GetProperties(
            CurrentPage.PageLink ) ) {
            if ( EPiServer.Personalization.PersonalizedData.Current[ NameString ] != null ) {
...
            }
        }
    }
}
```

The overloaded version of GetProperties which takes an EPiServer.Core.PageReference parameter returns null when there are no personalized properties for the referenced page. Hence we first test GetProperties for null-ness in the code in example 10-4.

PersonalizedData.Load

Example 10-5: Using EPiServer.Personalization.PersonalizedData.Load (from Profile.ascx.cs).

```
protected void SaveProfile( object sender, System.EventArgs e ) {
    SavePage();
    EPiServer.Personalization.PersonalizedData userData =
        EPiServer.Personalization.PersonalizedData.Load( (int) CurrentPage[ "SID" ] );
    if ( userData != null ) {
        SyncPersonalizedData( userData, false );
    }
    SwitchMode();
}
```

The code in example 10-5 is taken from the code-behind file of one of the Web User Controls which ship with EPiServer, Profile.ascx. Here, PersonalizedData.Load is used to load the stored personal information of the user account whose sid is found in the page property SID. This information is then used to allow the

user to update her personal information. The function SyncPersonalizedData (not shown) compares present and changed information and saves the synchronised personal information back to the database.

Example 10-6: Loading PersonalizedData for the currently logged-on user.

```
if ( EPiServer.Security.UnifiedPrincipal.Current != EPiServer.Security.UnifiedPrincipal.AnonymousUser ) {
    EPiServer.Personalization.PersonalizedData CurrUserData =
        EPiServer.Personalization.PersonalizedData.Load( EPiServer.Security.UnifiedPrincipal.CurrentSid );
    if ( CurrUserData != null ) {
...
    }
}
```

Example 10-6 shows how to load personal information for the currently logged-on user using EPiServer.Security.UnifiedPrincipal.CurrentSid.

Using Subscription (EPiServer.Personalization.Subscription)

Example 10-7: Code from the code-behind file of Web Form NewsGroupList.aspx.

```
public class NewsGroupList : NewsGroup {

    protected System.Web.UI.WebControls.LinkButton      Subscribe;
    protected System.Web.UI.WebControls.Panel           PersonalSettings;
    protected System.Web.UI.HtmlControls.HtmlTableCell  CreateEntry;

    private void Page_Load( object sender, System.EventArgs e ) {
        PersonalSettings.Visible = true;
        if ( CurrentUser.UserData == null ) {
            PersonalSettings.Visible = false;
        } else if ( ! EPiServer.Personalization.Subscription.IsSubscribingTo( CurrentNewsGroup ) ) {
            Subscribe.Text = Translate( "#subscribe" );
        } else {
            Subscribe.Text = Translate( "#unsubscribe" );
        }
        CreateEntry.Visible =
            CurrentPage.ACL.QueryDistinctAccess( EPiServer.Security.AccessLevel.Create );
...
    }
...
}
```

The code in example 10-7 is taken from the code-behind file of the NewsGroupList.aspx Web Form (part of the example Web site). The code in the Page_Load function is a good example of using personal information.

To begin with, the code checks whether there are any UserData available for the CurrentUser. If this property is null, the Panel PersonalSettings will not be

shown. Assuming that UserData exist, we want to know whether the current user already subscribes to the current news group (the code is executing inside a news group, as it were). If she's not already a subscriber, the LinkButton Subscribe will be labelled 'Subscribe' (after proper translation). If she actually is a subscriber, the same LinkButton will be labelled 'Unsubscribe'.

Last of the personal matters is to find out whether the current user has Create permission to the current page. If the use does have Create permission, she will be allowed to create new news group items (which are pages).

Figure 10-1: News group Windows (from the example Web site) as it appears to an anonymous user.

Figure 10-1 is what an anonymous user would see of the news group (i.e. a CurrentUser whose UserData property returns null. Now let's see what it looks like with the other kind of user.

Figure 10-2: News group Windows (from the example Web site) as it appears to a logged-on user.

Figure 10-2 shows the same news group as in 10-1, only this time a user has logged on. Note that the user is apparently not yet subscribing to this news group, as the LinkButton is labelled 'Subscribe'. Also notice that she obviously must have Create permission to the current page, as the CreateEntry object is visible.

11
Job Scheduling

Scheduled Jobs – Have the Computer Work for You

The EPiServer infrastructure has a member which keeps a very low profile, at least in a visual sense. We are of course talking about the EPiServer Scheduler Service, which is a Windows Service, and these rarely interact with users. In the open spirit of EPiServer solution development, you can add your own jobs to be run by EPiServer Scheduler.

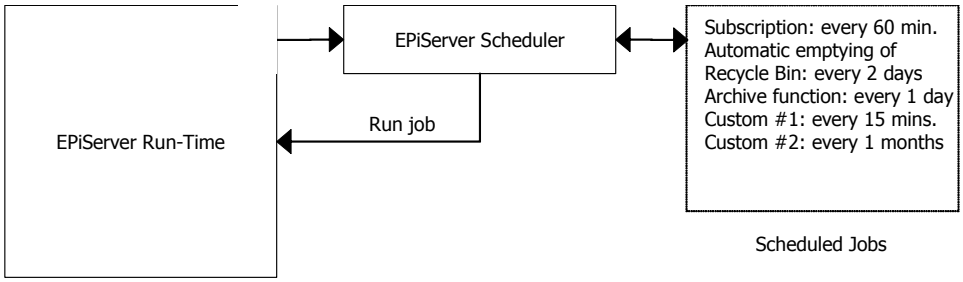


Figure 11-1: Scheduled jobs interaction between EPiServer and EPiServer Scheduler.

Figure 11-1 depicts the interactions between EPiServer and its scheduler service. It is important to notice that EPiServer is actually running the jobs; the Scheduler's assignment is to keep track of which active jobs are set to be executed. At execution time, EPiServer Scheduler sends a 'job start signal' to EPiServer, which ultimately does its deed.

Figure 11-2: Scheduled jobs section, EPiServer Admin mode.

Figure 11-2 shows the administrative interface for all scheduled jobs in EPiServer, both built-in and custom-made (by yourself or other developers).

One other perhaps obvious thing sets EPiServer Scheduler apart from the rest of EPiServer. This is that its actions never influence EPiServer Web sites – it doesn't interact with the Web server service, although it may use ASP.NET classes and objects.

Jobs Have Access to the Full Infrastructure

As is evident from figure 11-1, when creating scheduled jobs you have access to the complete EPiServer programming environment, anything you can do in a Web Form or a Web User Control you can also do in a scheduled job.

EPiServer Demands on the Scheduled Job Class

Scheduled jobs are implemented as classes and a new class file is simply added to your Web site's project. This ensures that the class is created in the correct name space.

EPiServer has but two demands on scheduled jobs classes you create yourself.

- The class takes an EPiServer.PlugIn.ScheduledPlugIn attribute.
- A public static parameter-less string method called 'Execute' is present, **or**
- A public static parameter-less void method called 'Execute' is present.

Note that the method Execute either has no return type or a 'string' return type.

Attribute EPiServer.PlugIn.ScheduledPlugIn

Assigning the ScheduledPlugIn ensures automatic recognition of the new class as a scheduled job in EPiServer Admin mode.

Table 11-1: Public Properties for EPiServer.PlugIn.ScheduledPlugIn class.

Property Name	*Description*
DisplayName	Of great importance to the administrator handling the scheduled job in EPiServer Admin mode. This property is appropriately called DisplayName, as whatever text you assign to it is displayed in EPiServer Admin mode (see figure 11-6).
HelpFile	The custom help page to display for the job.

Execute Method

Any scheduled job class must have a method called Execute. Execute should be a public string method. Any string returned is displayed on the History tab for the particular job (see figure 11-8), one row for each time the job has been run.

A Trivial Scheduled Job

Let's create a trivial class to be used as an EPiServer scheduled job. The only service the job will perform is to tell the time of its execution.

1. First create a new class in the EPiServer project, call it TellTime.

2. Check that the name space is the expected, without changes made it should be 'development'.

   Example 11-1: ScheduledPlugIn attribute added to class TellTime.

   ```
   namespace development {
   ...
   }
   ```

3. Add the EPiServer.PlugIn.ScheduledPlugIn to the class. Set its DisplayName attribute to something helpful. This text will be visible in EPiServer Admin mode.

   Example 11-2: ScheduledPlugIn attribute added to class TellTime.

   ```
   [ EPiServer.PlugIn.ScheduledPlugIn( DisplayName = "Tells time of invocation" ) ]
   public class TellTime {
   ```

4. Create a public parameter-less string method by the name of 'Execute'. Code it to return the current time.

   Example 11-3: Public parameter-less string method Execute returning the current time.

   ```
   public string Execute() {
       return System.DateTime.Now.ToString();
   }
   ```

5. Done. Your class should now be very similar to this:

   Example 11-4: Complete TellTime class.

   ```
   namespace development {
                                   /// <summary>TellTime is an EPiServer scheduled job.</summary>
       [ EPiServer.PlugIn.ScheduledPlugIn( DisplayName = "Tells time of invocation" ) ]
       public class TellTime {

           public TellTime() {
           }

           public static string Execute() {
               return System.DateTime.Now.ToString();
           }
       }
   }
   ```

6. Compile the solution and either start debugging or simply open the EPiServer Admin mode in your Web browser.
7. Verify that there's a new scheduled job called 'Tells time of invocation'.
8. Click on this job name in the left pane.

9. Click on the button 'Start manually' in the right pane.
10. Activate the History tab – you should see a new record looking similar to this (probably not the same date and time, though):

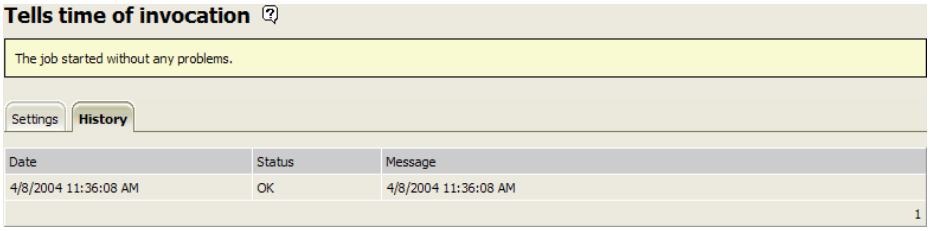


Figure 11-3: Result of running scheduled job 'Tells time of invocation'.

Elementary Troubleshooting of Scheduled Jobs

There are several things to check for when a scheduled job doesn't behave as expected. First check the demands on page 244. The Execute method in particular, must meet these demands:

- Its name must be 'Execute', with this exact spelling
- It must take no parameters; 'Execute**()**'
- It must be either **public static** or **public static string**

If your Execute method fails any of these demands EPiServer Admin mode will display an error message.

Figure 11-4: Error message from EPiServer Admin mode trying to run faulty scheduled job.

Debug Scheduled Jobs just like any other EPiServer Component

An excellent, and perhaps obvious, side-effect of the way scheduled jobs are run is that they are as easy to debug as any other EPiServer component you might build. Simply set a break point in the code and start the Visual Studio .NET debugger, open EPiServer Admin mode, click on the job name in the left pane and the click on the button 'Start manually' in the right pane. Of course, there's nothing to stop you from simply starting the Visual Studio .NET debugger and waiting for the next scheduled invocation of the job.

If you need more advanced debugging, please refer to chapter 5, *Avoiding Errors, Testing and Debugging*, starting on page 95. There's a section concerning the EPiServer Scheduler service starting on page 121 in the same chapter.

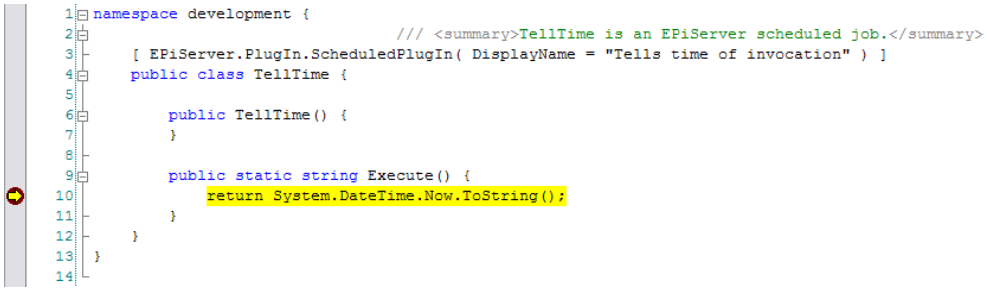


Figure 11-5: Scheduled job 'Tells time of invocation' being debugged.

Develop a Cautious Mentality

Since scheduled jobs are a little bit more involved than other EPiServer components, it's a good idea to always start with a simple basic skeleton, such as above, compile, install and test run it. With that minor hurdle out of the way, you can concentrate on the more important aspects of the scheduled job.

Scheduled Job to List All Pages Created Last Week

A more interesting job to perform might be to list all new pages created during the last week. This could be useful in an environment where there are many editors creating a lot of pages to help with planning ahead with regards to server capacity. Once created, we'll instruct EPiServer to execute this job once a week.

There are a multitude of options as far the report proper is concerned. We've settled on sending the list in an e-mail to a named account. The name of the account, as well as name of sending account and also the subject line, are stored in the web.config file for maximum flexibility. Currently there are no more scheduled jobs planned, so we boldly call the three settings EPsJobMailFrom, EPsJobMailTo and EPsJobMailSubject.

The Code

Example 11-5: Code to implement a custom scheduled job.

```
namespace development {
                                /// <summary>NewlyCreatedPages mails a list of newly created pages in
                                /// the EPiServer database (using PageCreated property).</summary>
    [ EPiServer.PlugIn.ScheduledPlugIn( DisplayName = "Mail list of newly created pages" ) ]
    public class NewlyCreatedPages {

        public NewlyCreatedPages() {
            // Nothing!
```

```
}

public static string Execute() {
    EPiServer.PropertyCriteria   PageCreatedSince = new EPiServer.PropertyCriteria();
    PageCreatedSince.Name      = "PageCreated";
    PageCreatedSince.Value     = System.DateTime.Now.Date.AddDays( -7 ).ToString();
    PageCreatedSince.Type      = EPiServer.Core.PropertyDataType.Date;
    PageCreatedSince.Required = true;
    PageCreatedSince.Condition= EPiServer.Filters.CompareCondition.GreaterThan;

    EPiServer.PropertyCriteriaCollection NewPagesCriteria =
        new EPiServer.PropertyCriteriaCollection();
    NewPagesCriteria.Add( PageCreatedSince );

    EPiServer.Core.PageDataCollection NewPages =
        EPiServer.Global.EPDataFactory.FindPagesWithCriteria(
        EPiServer.Global.EPConfig.RootPage, NewPagesCriteria,
        EPiServer.Security.AccessControlList.NoAccess );

    System.Text.StringBuilder   PageInfoStrings= new System.Text.StringBuilder();
    int                         NumNewlyCreated= 0;
    foreach( EPiServer.Core.PageData NewPage in NewPages ) {
        try {
            NumNewlyCreated++;
            PageInfoStrings.AppendFormat(
                "Page \"{0}\" created {1} by {2}. <a href=\"{3}\">{3}</a><br>\n",
                NewPage.PageName, NewPage.Created, NewPage.CreatedBy,
                Global.EPConfig.HostUrl + NewPage.LinkURL );
        }
        catch {
            // Nothing!
        }
    }

    if ( NumNewlyCreated > 0 ) {
        SendMail( PageInfoStrings );
    }

    return string.Format( "{0} page(s) created since {1}. Mail sent to {2}.",
        NumNewlyCreated, PageCreatedSince.Value,
        (string) EPiServer.Global.EPConfig[ "EPsJobMailTo" ] );
}

public static void SendMail( System.Text.StringBuilder MessageBody ) {
    System.Web.Mail.MailMessage MailMessage = new System.Web.Mail.MailMessage();
    MailMessage.From            = (string) EPiServer.Global.EPConfig[ "EPsJobMailFrom" ];
    MailMessage.To              = (string) EPiServer.Global.EPConfig[ "EPsJobMailTo" ];
```

```
            MailMessage.Subject             = (string) EPiServer.Global.EPConfig[ "EPsJobMailSubject" ];
            MailMessage.UrlContentBase      = EPiServer.Global.EPConfig.HostUrl;
            MailMessage.Headers[ "X-Mailer" ]= "EPiServer 4";
            MailMessage.Body                = MessageBody.ToString();
            MailMessage.BodyFormat          = System.Web.Mail.MailFormat.Html;
            EPiServer.Global.EPConfig.InitSmtpServer();
            System.Web.Mail.SmtpMail.Send( MailMessage );
        }

    }
}
```

The Comments

The Pseudo Code

Class NewlyCreatedPages written in pseudo code could look like this:

Example 11-6: Class NewlyCreatedPages written in pseudo code.

```
class NewlyCreatedPages
    function Execute
        create search criterion for pages, include only pages created since midnight seven days ago
        perform the search
        iterate through the collection of pages returned from the search
            tally the pages
            collect and store some information on every page
        if there at least was one new page created during the last seven days send an e-mail to —
        pre-configured recipient e-mail address passing along the collected page information
        return string summarising the operation
    function SendMail accepting MessageBody argument
        assemble e-mail message including MessageBody argument
        initialise the e-mail server
        send the assembled message
```

Search Criterion

The search criterion we decided upon was for page property Created (when used in searches and in some other instances too, Created is accessed via its Property alias PageCreated) to be no more than seven days ago. In fact, System.DateTime.Now.Date means we're actually using the midnight of the day seven days ago, meaning that the pages created at any time on that day are included. You can find more information on using EPiServer.Property, PropertyCriteriaCollection on page 195.

Performing the Search

After adding the single search criterion to the criteria collection, the search is performed using Global.EPDataFactory.FindPagesWithCriteria using the variant which accepts three arguments; the start page for the search, a criteria collection and the access permission level needed.

As the start page for the search, we use EPiServer.Global.EPConfig.RootPage, as we want every page to be tested and not just those making up the Web site (in which case we would have specified EPiServer.Global.EPConfig.StartPage). Our criteria collection, NewPagesCriteria, comprises a single criterion. Since we only wish to query the pages' properties, an access permission of EPiServer.Security.AccessControlList.NoAccess is specified.

There's more information on using FindPagesWithCriteria, together with EPiServer.Property and PropertyCriteriaCollection, on page 216.

Enumerating the Pages Found

Next in the function Execute, the page collection returned from FindPagesWithCriteria is enumerated and the pages are tallied. In addition to counting the pages, a multi-line string (using a System.Text.StringBuilder object) is built containing one line of text for each page found in the search.

Calling SendMail from Execute

The penultimate task performed in the function Execute is to call the SendMail function passing on the multi-line string built. This call is only performed if the page search returned any recently created pages.

Finally, in Execute

Lastly in Execute, a summary string is composed and returned to the caller (meaning EPiServer in this case). To create this summary string, the contents of a configuration parameter, EPsJobMailTo, are read and included.

Function SendMail

Function SendMail uses functionality in ASP.NET (i.e. System.Web.Mail) to create an e-mail message. Again, web.config is read and the contents of all three settings are read and used to create the e-mail message. The message is sent off after initialisation of the mail server.

Web.config Is Used to Store Configuration Data

Three items of configuration information are added to the web.config file.

- EPsJobMailFrom holds the sender e-mail address
- EPsJobMailTo stores the recipient e-mail address
- EPsJobMailSubject is the subject line for the mail

They have been added as good EPiServer subjects using the EPs prefix.

Discover the New Job in EPiServer Admin Mode

Having successfully compiled the new class into the EPiServer solution, it's time to start EPiServer and enter Admin mode. There is no need to manually restart the EPiServer Web site, it happens automatically.

As you open EPiServer Admin mode, you'll notice the new scheduled job in the left pane. Well, it not actually scheduled yet, that's what we're about to do.

Figure 11-6: EPiServer Admin mode, new custom scheduled job visible.

Set Up the New Job

To set up the new job, click on its name in the left pane. For this particular job we had decided on running it once weekly to adjust settings to look like in figure 11-7.

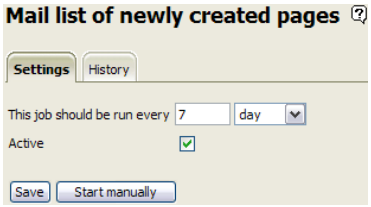


Figure 11-7: Settings for scheduled job 'Mail list of newly created pages'.

To test the job, click on 'Start manually', when sufficient time has elapsed click on the History tab, which should look something like figure 11-8.

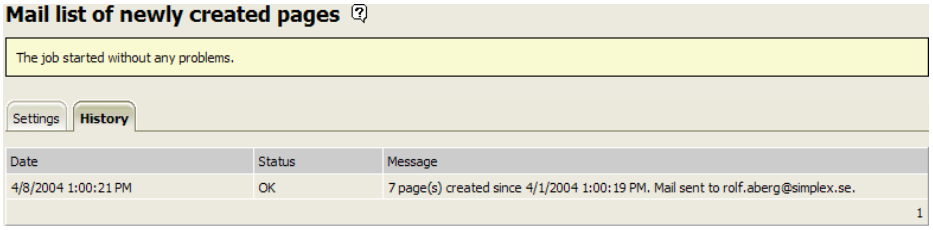


Figure 11-8: First report from new job 'Mail list of newly created pages'.

In figure 11-8, you can appreciate the beauty of letting Execute be a string returning function – whatever it returns is presented on the History tab, one row for each time the job is run.

Removing an Obsolete Scheduled Job

Removing obsolete scheduled jobs is as easy as either deleting the implementing class file or simply commenting out the ScheduledPlugIn attribute from the class name and then recompiling.

If you change the class or name space name, the old scheduled job will still be active but without a settings page. You should manually delete this job from the database, in table tblScheduledItem.

12
File and Folder Objects

File and Folder Handling in EPiServer 4.3 and Later

In EPiServer 4.30, a major change has been made in the way EPiServer handles static files. This change was made possible by a new file system, which among other things made it possible to add a virtual share giving access to files not exposed by Internet Information Server. The virtual shares are handled by EPiServer and should not be confused with virtual folders in IIS. Actually, files do not even have to be located in a traditional file system, they can be stored in any source that can represent their data in a structured way. You can write your own file system implementation to work with Microsoft SharePoint for example or your own company file system with complete integration with EPiServer. In this release, we have provided a simple sample (on the Web site) which virtualises an XML file as a complete file system.

A major new feature is that all files (actually folders) can be protected using EPiServer access permission security. To be able to see or download a file (image, document, etc.) you now need to have the correct access permissions. The files corresponding to specific pages are protected by the same access permissions as the page. This feature requires activation, as the original files need to be moved to a protected location so that EPiServer can handle all access. If you need this feature and are using file access in your templates to a protected folder using the File and Directory classes in .NET, you will have to rewrite your solution to use the classes in EPiServer.FileSystem name space.

From the Editor perspective, we added a powerful new File Manager which is integrated with the new file system. Open the FileManager in the ActionWindow by clicking on the new tool to the right on the tool-bar. You can now add files, search files, delete files, set access permissions on folders, add metadata to files and more. Look in particular at the file view and you will find a list of all pages that a file has been used on.

EPiServer.FileSystem

Important Note on Path Strings

Path strings used in EPiServer.FileSystem never include device letters or semi-colons; slashes are used instead of back-slashes.

Classes

Table 12-1: EPiServer.FileSystem classes.

Name	*Description*
FileSystemEventArgs	Argument passed as part of File System events.
UnifiedDirectory	A unified folder. Contains information on a folder and methods to delete, move and create new folders.
UnifiedFile	Unified file. Contains information on a file and methods to delete, move and copy files.
UnifiedFileSummary	Contains metadata and summary information on a file as entered by an Editor.
UnifiedFileSystem	The global class for the file systems used in EPiServer. This class contains methods to load and create folders and files.
UnifiedFileSystemConfiguration	Contains configuration information for the unified file system
UnifiedSearchHit	A class that describes a hit returned from UnifiedSearchQuery
UnifiedSearchHitCollection	Represents a collection of UnifiedSearchHit classes.
UnifiedSearchQuery	A class to build and execute search queries against the file system
WebDownloadManager	Handles all file downloads.

Delegates

FileSystemEventArgs

Table 12-2: EPiServer.FileSystem delegates.

Name	*Description*
FileSystemEventHandler	Represents the method which will handle generic file-system events

EPiServer.FileSystem.UnifiedDirectory

Public Properties

Table 12-3: Public Properties for EPiServer.FileSystem.UnifiedDirectory.

Property Name	*Description*
ACL	The access permissions defined for this folder
Configuration	The configuration which defines the various settings for this handler or virtual folder
IsPageDirectory	Whether this folder belongs to a specific page
Name	The short name of the folder
Parent	The parent folder of this folder
Path	The path to this folder
Provider	The underlying provider which handles all access

Public Methods

Table 12-4: Public methods for EPiServer.FileSystem.UnifiedDirectory.

Method Name	*Description*
CopyTo	Copy the folder to another location
CreateFile	Create a new file in the current folder
CreateSubdirectory	Create a new subfolder to the current folder
Delete	Delete the current folder and all its sub-folders and files
Get	Get the UnifiedDirectory based on the path
GetDirectories	Get all sub-folders below the current folder
GetFiles	Get all files in the current folder
GetOwnerPage	Get the owner page if this folder is considered to be a special folder for a page
MoveTo	Move this folder to a new destination.
PathIsPageDirectory	Determines whether it is a page folder based on the format of the path

Table 12-4: Public methods for EPiServer.FileSystem.UnifiedDirectory.

Method Name	*Description*
PathIsPageDirectoryOrChild	Determine whether it is a page folder or a sub-folder to a page folder based on the format of the path
QueryAccess	Get the access level for the current user on this folder
QueryDistinctAccess	Query whether a user has a certain access permission for the current folder
Rename	Change name of the folder

EPiServer.FileSystem.UnifiedFile

Public Properties

Table 12-5: Public properties for EPiServer.FileSystem.UnifiedFile.

Property Name	*Description*
Changed	When the file was last changed
Created	When the file was created
Length	The size of the file in bytes
Name	The name of the file
Parent	The parent folder of the file
Path	The full path for the file
Provider	The underlying provider which handles all access
Summary	Metadata and file summary information as entered by the editor, e.g. Title

Public Methods

Table 12-6: Public methods for EPiServer.FileSystem.Unified File.

Method Name	*Description*
CopyTo	Copy the current file to another folder
Delete	Delete the current file

Table 12-6: Public methods for EPiServer.FileSystem.Unified File.

Method Name	*Description*
Get	Get the UnifiedFile based on the path
MoveTo	Move a file to another folder
OpenRead	Open a file for reading
OpenWrite	Open a file for writing
QueryAccess	Get the access level for the current user on this file
QueryDistinctAccess	Query whether a user has a certain access permission for the current folder
Rename	Rename the file

EPiServer.FileSystem.UnifiedFileSummary

Public Properties

Table 12-7: Public properties for EPiServer.FileSystem.UnifiedFileSummary.

Property Name	*Description*
Author	The author of the file
Category	The categories for the file
Comments	The comments made for this file
Keywords	Keywords specified for this file
Subject	The subject of this file
Title	The title of this file

Public Methods

Table 12-8: Public methods for EPiServer.FileSystem.UnifiedFileSummary.

Property Name	*Description*
SaveChanges	Save changes made to summary

EPiServer.FileSystem.UnifiedFileSystem

Public Properties

Table 12-9: Public properties for EPiServer.FileSystem.UnifiedFileSystem.

Property Name	*Description*
Configuration	The global configuration object which represents the configuration section in web.config (**static**)
Root	The root folder below all virtual shares (**static**)

Public Methods

Table 12-10: Public methods for EPiServer.FileSystem.UnifiedFileSystem.

Method Name	*Description*
CreateDirectory	Overloaded. Create folder.
GetDirectory	Overloaded. Get a folder
GetFile	Overloaded. Get a file
IsValidNameSyntax	Check to see whether the syntax of a file or folder name is valid
IsValidPathSyntax	Check to see whether the syntax of a path is valid
ResolveDefaultDirectory	Resolve the default folder

Public Events

Table 12-11: Public events for EPiServer.FileSystem.UnifiedFileSystem.

Event Name	*Description*
ResolvingDefaultDirectory	Event which allows developers to change the default folder based on run-time variables

EPiServer.FileSystem.UnifiedFileSystemConfiguration

Public Properties

Table 12-12: Public properties for EPiServer.FileSystem.UnifiedFileSystemConfiguration.

Property Name	*Description*
Handlers	Array of all handler configuration objects

Public Methods

Table 12-13: Public Methods for EPiServer.FileSystem.UnifiedFileSystemConfiguration.

Method Name	*Description*
LookupConfiguration	Look-up a configuration object based on the name of the virtual share

EPiServer.FileSystem.UnifiedSearchHit

Public Properties

Table 12-14: Public properties for EPiServer.FileSystem.UnifiedSearchHit.

Property Name	*Description*
Name	Name of the file
Path	Path of the file which was found
Preview	A preview of the file contents
Rank	Rank of the file hit

EPiServer.FileSystem.UnifiedSearchHitCollection

Public Properties

Table 12-15: Public properties for EPiServer.FileSystem.UnifiedSearchHitCollection.

Property Name	*Description*
Item	Indexer.

Public Methods

Table 12-16: Public methods for EPiServer.FileSystem.UnifiedSearchHitCollection.

Method Name	*Description*
Add	Adds a UnifiedSearchHit to the end of the collection.
AddRange	Adds a collection of objects to the end of the collection.
Contains	Determines whether the collection contains a specific element.
CopyTo	Copies the entire collection to a one-dimensional array, starting at the specified index of the target array.

Table 12-16: Public methods for EPiServer.FileSystem.UnifiedSearchHitCollection.

Method Name	*Description*
IndexOf	Searches for the specified UnifiedSearchHit and returns the zero-based index of the first occurrence within the entire collection.
Insert	Inserts an element into the collection at the specified index.
Remove	Removes the first occurrence of a specific UnifiedSearchHit from the collection.

EPiServer.FileSystem.UnifiedSearchQuery

Public Properties

Table 12-17: Public properties for EPiServer.FileSystem.UnifiedSearchQuery.

Property Name	*Description*
FileNamePattern	The pattern used for matching file
FreeTextQuery	Text to search for
MatchSummary	A dictionary which contains key/value pairs to search for in file/folder summaries
ModifiedFrom	Limit search to files changed after this date
ModifiedTo	Limit search to files changed before this date
Path	Search in all files in this path

Public Methods

Table 12-18: Public methods for EPiServer.FileSystem.UnifiedSearchQuery.

Method Name	*Description*
Search	Execute the search

EPiServer.FileSystem.WebDownloadManager

Public Methods

Table 12-19: Public methods for EPiServer.FileSystem.WebDownloadManager.

Method Name	*Description*
GetMimeType	Returns the MIME type for a specified file path

Table 12-19: Public methods for EPiServer.FileSystem.WebDownloadManager.

Method Name	*Description*
IsDownloadableUnifiedFile	Check to see whether a file path belongs to the unified file system.
TransmitFile	Transmit a file to the output stream.

EPiServer.FileSystem.FileSystemEventHandler

Table 12-20: Arguments passed to the event function (EPiServer.FileSystem .FileSystemEventHandler).

Argument Name	*Description*
sender	The object which triggered the event
e	Event data which contains the Path related to this event

EPiServer Web Custom Controls that Utilise the EPiServer.FileSystem Classes

With EPiServer 4.3, two Web Custom Controls were introduced which benefit from the functionality in the classes, delegates and enumerations in the EPiServer.FileSystem name space.

Table 12-21: Classes in EPiServer.WebControls which use EPiServer.FileSystem functionality.

Class Name	*Description*
FileManager	Renders a control for managing folders and files
FileManagerNavigator	Renders a control for (typically) providing a FileManager object with context

Use of EPiServer.FileSystem in EPiServer 4.3 (and Later)

Starting with EPiServer 4.3, the new unified file system is used in both EPiServer Admin mode and Edit mode.

File Management Tool in EPiServer Admin Mode

File Management Tool in EPiServer Admin Mode

In the Tools section of the Admin mode menu there is a 'File management' tool which is dependent on the functionality of the classes in EPiServer.FileSystem.

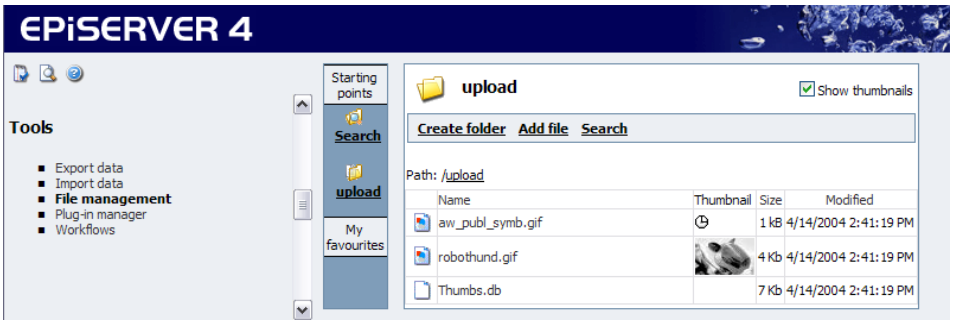


Figure 12-1: File management tool in Tools section of Admin mode menu (EPiServer 4.3 and later).

This tool is implemented by the Web User Form admin\FileManagement.aspx. However, this Web Form is no more than a pointer to edit\FileManagement-Control.ascx (more on this below).

File Management Tool on Action Window in EPiServer Edit Mode

The same File management tool which is available for administrators in EPiServer Admin mode is also available as a choice in the Action window for editors in EPiServer Edit mode. As you already know, this is the same tool which is implemented by the Web User Control edit\FileManagementControl.ascx for both modes.

Functionality in the File Management Tool

When familiarising yourself with the File management tool, you'll notice that it makes good use of the classes in EPiServer.FileSystem. You can create sub-folders, add files to folders, search for files, handle file meta information, and so on.

Using EPiServer.FileSystem

UnifiedFileSystem

Configuration Settings

EPiServer.FileSystem uses one or more file systems. You can rely on one of them being of the type EPiServer.FileSystem.Handler.NativeFileSystem. To enumerate the handling file systems and their respective settings, use the code in example 12-1.

Example 12-1: Enumeration of handling file systems and their settings.

```
foreach ( EPiServer.FileSystem.Handler.HandlerConfiguration HandlerConfig in
    EPiServer.FileSystem.UnifiedFileSystem.Configuration.Handlers ) {
    System.Collections.IDictionaryEnumerator FsConfigSetting =
        HandlerConfig.CustomSettings.GetEnumerator();
    while ( FsConfigSetting.MoveNext() ) {
        System.Diagnostics.Debug.WriteLine( FsConfigSetting.Key + "\t" + FsConfigSetting.Value );
    }
    System.Diagnostics.Debug.WriteLine( HandlerConfig.FS.GetRootDirectory().Name );
    System.Diagnostics.Debug.WriteLine( HandlerConfig.IsVirtualShare );
    System.Diagnostics.Debug.WriteLine( HandlerConfig.VirtualName );
}
```

Among the settings found in CustomSettings (EPiServer.FileSystem.Handler .HandlingConfiguration.CustomSettings) expect to find one key equal to 'PhysicalPath', the contents of which consists of the ordinary path to the root folder of the file system.

The method GetRootDirectory and the properties IsVirtualShare and VirtualName reveal information about the root folder for the file system.

For the default file system NativeFileSystem, the settings in table 12-22 are to be expected.

Table 12-22: Configuration settings for default file system handler, NativeFileSystem.

Setting Name	*Contents*
HandlerConfig.CustomSettings.Key = 'PhysicalPath'	'c:\INetPub*Web Site Name*\upload\', or similar
GetRootDirectory().Name	upload
IsVirtualShare	false
VirtualName	upload

Root Folder

Switching back to the UnifiedFileSystem, the path of the 'unified' root is found in EPiServer.FileSystem.UnifiedFileSystem.Root.Path. This will often simply be '/'.

UnifiedDirectory

GetDirectories

Example 12-2: Using EPiServer.FileSystem.UnifiedDirectory.GetDirectories method.

```
EPiServer.FileSystem.UnifiedDirectory RootDir = EPiServer.FileSystem.UnifiedFileSystem.Root;
```

```
foreach ( EPiServer.FileSystem.UnifiedDirectory CurrDir in RootDir.GetDirectories() ) {
    System.Diagnostics.Debug.WriteLine( CurrDir.Name + "\t[" + CurrDir.Path + "]" );
}
```

ACL

For an example of how to enumerate the Access Control List, see example 6-25 on page 147.

GetFiles

The GetFiles method retrieves the files (as a UnifiedFile array) for the specified folder (UnifiedDirectory object).

Example 12-3: Using EPiServer.FileSystem.UnifiedDirectory.GetFiles method.

```
EPiServer.FileSystem.UnifiedDirectory UploadDir =
    EPiServer.FileSystem.UnifiedFileSystem.GetDirectory( "/upload" );
if ( UploadDir.GetFiles() != null ) {
    foreach ( EPiServer.FileSystem.UnifiedFile CurrFile in UploadDir.GetFiles() ) {
        System.Diagnostics.Debug.WriteLine( CurrFile.Name + "\t[" + CurrFile.Path + "]" );
    }
}
```

UnifiedFile and UnifiedFileSummary

File handling and file information is what the classes UnifiedFile and UnifiedFileSummary have been charged with.

Example 12-4: Using EPiServer.FileSystem.UnifiedDirectory.UnifiedFile.QueryAccess method.

```
EPiServer.FileSystem.UnifiedDirectory UploadDir =
    EPiServer.FileSystem.UnifiedFileSystem.GetDirectory( "/upload" );
foreach ( EPiServer.FileSystem.UnifiedFile CurrFile in UploadDir.GetFiles() ) {
    System.Diagnostics.Debug.WriteLine( CurrFile.QueryAccess() );
}
```

Example 12-5: Using EPiServer.FileSystem.UnifiedDirectory.UnifiedFile.Summary property.

```
foreach ( EPiServer.FileSystem.UnifiedFile CurrFile in UploadDir.GetFiles() ) {
    System.Diagnostics.Debug.WriteLine( CurrFile.Summary.Author );
    System.Diagnostics.Debug.WriteLine( CurrFile.Summary.Category );
    System.Diagnostics.Debug.WriteLine( CurrFile.Summary.Comments );
    System.Diagnostics.Debug.WriteLine( CurrFile.Summary.Keywords );
    System.Diagnostics.Debug.WriteLine( CurrFile.Summary.Subject );
    System.Diagnostics.Debug.WriteLine( CurrFile.Summary.Title );
}
```

UnifiedSearchQuery, UnifiedSearchHitCollection and UnifiedSearchHit

The three classes UnifiedSearchQuery, UnifiedSearchHitCollection and UnifiedSearchHit co-operate in searching for files and for text inside files. Note that it is up to the implementing file system to implement the search infrastructure; thus you cannot rely on support for all types of searches in all file system handlers.

Example 12-6: Using EPiServer.FileSystem.UnifiedSearchQuery.Search

```
EPiServer.FileSystem.UnifiedSearchQuery FileSearch = new EPiServer.FileSystem.UnifiedSearchQuery();
FileSearch.FileNamePattern = "*.gif";
FileSearch.Path = "/upload";
EPiServer.FileSystem.UnifiedSearchHitCollection SearchHits = FileSearch.Search();
foreach ( EPiServer.FileSystem.UnifiedSearchHit SearchHit in SearchHits ) {
    System.Diagnostics.Debug.WriteLine( SearchHit.Name + "\t" + SearchHit.Rank + "\t" +
        SearchHit.Preview + "\t[" + SearchHit.Path + "]" );
}
```

13
Extending EPiServer

Quite a few aspects of EPiServer are open to extensions. We have already seen examples of some of the extension types, such as new property data types in chapter 8, *Custom Property Data Types and Filters* (section starting on page 201) and scheduled jobs in chapter 11, *Job Scheduling*.

Perhaps the most interesting aspect of extensibility is the editor that is available for editing LongString properties (the LongString editor is fondly known as the 'DHTML Editor'), EPiServer Admin mode and EPiServer Edit mode, respectively.

Extensible Areas of the EPiServer Admin and Edit Mode

Admin Mode Areas

The extensible areas in EPiServer Admin mode partially overlap. When extending the System settings tab strip you have the area indicated by a white line in figure 13-1 to work with.

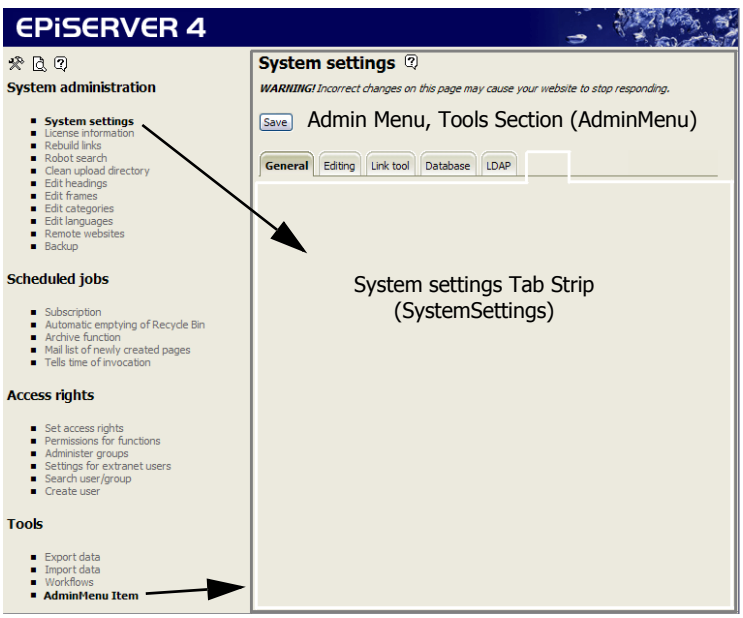


Figure 13-1: The extensible areas of EPiServer Admin mode.

When you extend by adding a new menu item to Tools section of the Admin menu, the full right pane area is available to you (indicated by a grey rectangle in figure 13-1).

Edit Mode Areas

Three areas of EPiServer Edit mode can be extended by plug-ins.

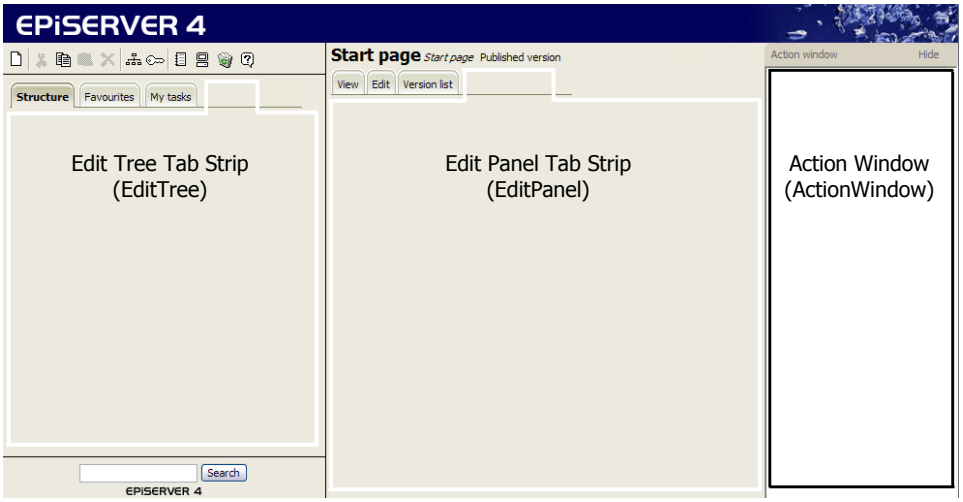


Figure 13-2: The extensible areas of the EPiServer Edit mode.

Creating Plug-Ins for EPiServer

We hope that you expect it to be almost effortless to create plug-ins for EPiServer now that you've been exposed to many of its facets. In fact, we like to think that extending EPiServer is as easy as creating a Web User Control for EPiServer use. It is our sincere hope that this chapter will prove us right.

As we don't expect every single reader of this book to take in this book from cover to cover in one sitting, we will create all the various types of plug-ins in the same manner. First, we create what's best described as a dummy plug-in which only includes those elements which identify the plug-in as such to the EPiServer infrastructure. With that out of the way, we move on and create what we hope is a useful plug-in, one which you might like to include in your own EPiServer code library. (We know that at least the web.config file editor we create as one example is going to stay with us for the foreseeable future.)

Order is always a positive and although the plug-ins we create in this chapter go in the root folder of the Web site, you might want to organise your plug-ins using a more elaborate folder structure. One suggestion is to create more folders under the templates folder.

EPiServer.PlugIn Name Space

Irrespective of which part of EPiServer you are looking to extend, all the classes, interfaces and enumerations to use are found in the EPiServer.PlugIn name space. The inheritance hierarchy for the classes is depicted in figure 13-3. Notice that all the classes on the left are direct descendants, daughter classes, of System.Object.

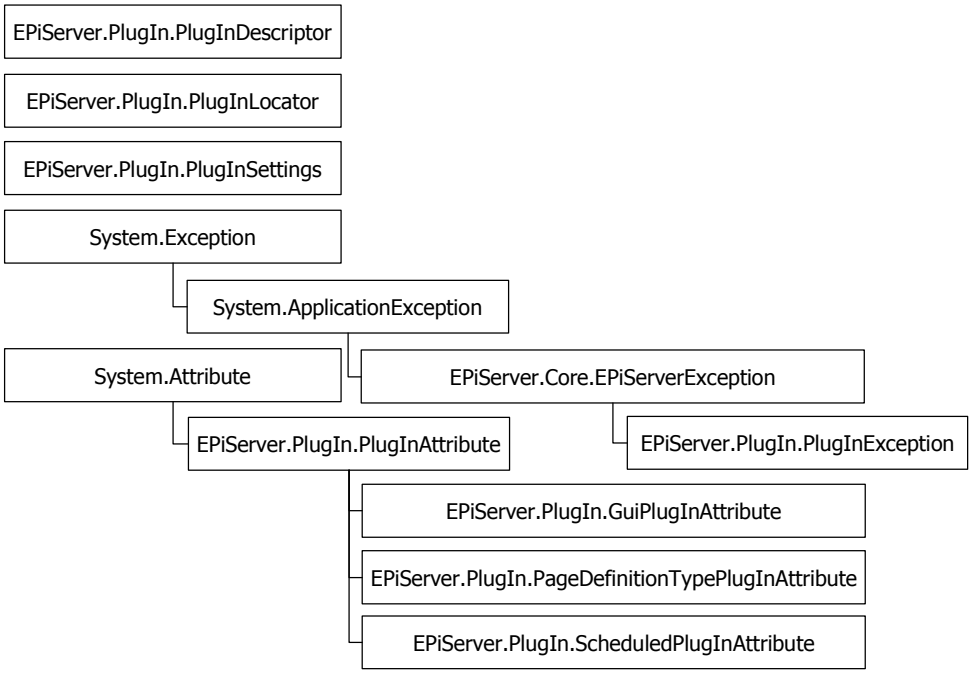


Figure 13-3: Inheritance hierarchy for classes in EPiServer.PlugIn name space.

Looking at figure 13-3, you can see that there is only one class attribute you haven't seen yet, the GuiPlugInAttribute. This attribute has a wider range of assignments compared with both the PageDefinitionTypePlugInAttribute, for custom property data types, and ScheduledPlugInAttribute, for scheduled jobs.

Classes in the EPiServer.PlugIn Name Space

Table 13-1: Classes in EPiServer.PlugIn name space.

Class Name	*Description*
GuiPlugInAttribute	Settings for plug-ins in graphical modes
PageDefinitionTypePlugInAttribute	Used for new Property Data Types.

Table 13-1: Classes in EPiServer.PlugIn name space.

Class Name	*Description*
PlugInAttribute	The base class for all specialised plug-in types.
PlugInDescriptor	Describes a plug-in at run-time without having to create it.
PlugInException	Exception thrown by the plug-in framework when a error occurs while loading or interacting with a plug-in.
PlugInLocator	Handles all internal locating of plug-ins.
PlugInSettings	Class for handling simple plug-in settings as a DataSet.
ScheduledPlugInAttribute	A plug-in to have a scheduled job with a simple user interface in Admin mode.

Interfaces in the EPiServer.PlugIn Name Space

Table 13-2: Interfaces in EPiServer.PlugIn name space.

Interface Name	*Description*
ICustomPlugInLoader	Override the default behaviour when plug-ins are loaded.

Enumerations in the EPiServer.PlugIn Name Space

Table 13-3: Enumerations in EPiServer.PlugIn name space.

Enumeration Name	*Description*
PlugInArea	The various areas which are currently supported. See below.

EPiServer.PlugIn.PlugInArea Enumeration

Using the members in the PlugInArea enumeration, you decide which part of EPiServer Admin mode or Edit mode your plug-in will be integrated with.

Table 13-4: Members on the enumeration EPiServer.PlugIn.PlugInArea.

Member Name	*Description*
None	No plug-in area

Table 13-4: Members on the enumeration EPiServer.PlugIn.PlugInArea.

Member Name	*Description*
ActionWindow	Item in listing in the Action window in EPiServer Edit mode, see figure 13-2.
EditPanel	A custom tab item on the Edit mode edit panel tab strip, see figure 13-2.
EditTree	A custom tab item on the Edit mode Edit Tree tab strip, see figure 13-2.
SystemSettings	A custom tab item on the EPiServer Admin mode System settings tab strip, see figure 13-1.
AdminMenu	A custom menu item on the Admin mode menu in the Tools section, see figure 13-1.

EPiServer.PlugIn.PlugInAttribute

As seen in figure 13-3, the class PlugInAttribute is the mother of all the other attribute classes (including EPiServer.Editor.EditorPlugInAttribute, which will be discussed later). You should make sure to always add pertinent contents to the DisplayName attribute, as those contents will always be displayed in EPiServer Admin mode or Edit mode

Following what has become Microsoft .NET development tradition, you can enter the class name when used in the source file either as 'PlugInAttribute' or in its short form as 'PlugIn'. This is also true for all classes which inherit EPiServer .PlugIn.PlugInAttribute.

Table 13-5: Public properties for EPiServer.PlugIn.PlugInAttribute.

Property Name	*Description*
Description	Balloon help text or, in the case of an Action window plug-in, a description line following the menu item line.
DisplayName	Tab name, menu entry name, etc.
LanguagePath	Path to node in language files where translation can be found

EPiServer.PlugIn.GuiPlugInAttribute

Public Properties for GuiPlugInAttribute

Table 13-6: Public properties for EPiServer.PlugIn.GuiPlugInAttribute.

Property Name	*Description*
Area	The area this plug-in supports; one of the members in the EPiServer.PlugIn.PlugInArea enumeration.
Url	The URL for the resource file (Web User Form or Web User Control) according to specification Area.

Public Methods for GuiPlugInAttribute

Table 13-7: Public methods for EPiServer.PlugIn.GuiPlugInAttribute.

Method Name	*Description*
Match	Overridden. When overridden in a derived class, returns a value indicating whether this instance equals a specified object.

More Information on the GuiPlugInAttribute

GuiPlugInAttribute extends its mother class, EPiServer.PlugIn.PlugInAttribute, with two properties and one method. Correctly using both properties is paramount to the successful deployment of this group of plug-ins.

EPiServer.PlugIn.GuiPlugInAttribute.Area

The Area property describes to EPiServer where this plug-in is to be utilised. There are five different places in which a plug-in can be employed, described above for enumeration PlugIn.PlugInArea.

EPiServer.PlugIn.GuiPlugInAttribute.Url

Acting as a file locator property the Url property, points to the main file of the Web Form or Web User Control implementing the plug-in. There are only two file extensions expected in the contents of Url – 'ascx' or 'aspx'. Moreover, in only one instance is the file extension 'aspx' and that's when you're creating a plug-in for the Tools section of the EPiServer Admin mode menu. In this case, Area is set to AdminMenu.

EPiServer.PlugIn.PageDefinitionTypePlugInAttribute

Public Methods

Table 13-8: Public methods for EPiServer.PlugIn.PageDefinitionTypePlugInAttribute.

Method Name	*Description*
Start	Auto-start method which is called upon application startup

EPiServer.PlugIn.PlugInDescriptor

Public Properties

Table 13-9: Public properties for EPiServer.PlugIn.PlugInDescriptor.

Property Name	*Description*
AssemblyName	The name of the assembly
ID	The identifier assigned by EPiServer to this plug-in
PlugInType	The System.Type for the plug-in
TypeName	The name of the class.

Public Methods

Table 13-10: Public methods for EPiServer.PlugIn.PlugInDescriptor.

Method Name	*Description*
Create	Create an instance of the plug-in
GetAttribute	Get a plug-in attribute of a special type.
GetAttributeArray	Get all PlugInAttribute of a given type from an array of PlugInDescriptor.
GetAttributes	Get all plug-in attributes.
Load	Overloaded. Load plug-in information based on a plug-in ID.

EPiServer.PlugIn.PlugInLocator

Public Methods

Table 13-11: Public methods for EPiServer.PlugIn.PlugInLocator.

Method Name	*Description*
FindPlugInAttributes	Search for unique plug-in attribute classes.

Table 13-11: Public methods for EPiServer.PlugIn.PlugInLocator.

Method Name	*Description*
Search	Overloaded. Search for plug-ins.

EPiServer.PlugIn.PlugInSettings

Public Methods

Table 13-12: Public methods for EPiServer.PlugIn.PlugInSettings.

Method Name	*Description*
Populate	Populate DataSet with data for plug-in
Save	Save DataSet with settings to database.

EPiServer.PlugIn.ScheduledPlugInAttribute

Public Properties

Table 13-13: Public properties for EPiServer.PlugIn.ScheduledPlugInAttribute.

Property Name	*Description*
HelpFile	The help page to display for the job.

Plug-Ins for the ActionWindow (EPiServer Edit Mode)

Unlike the other extensibility areas, as you will see later, the Action window has duties which do not directly deal with the Web Page Tree. To view the Action window, open Edit mode and click on the Open Action window tool-bar button,  .

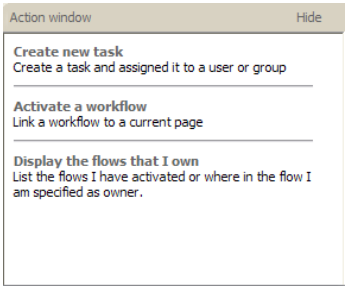


Figure 13-4: Edit mode Action window.

Simple Plug-In for the Action Window

Following these steps will guarante that you will end up with a plug-in for the Action window.

1. In Visual Studio .NET, right-click on the current project name, select Add and then 'Add Web User Control'.
2. Name this Web User Control 'ActionWindowSimpleton'.
3. Switch to HTML mode and enter the text: 'Hello Action window!'.
4. Open the code-behind file and add the all-important GuiPlugInAttribute

Example 13-1: The GuiPlugInAttribute for ActionWindowSimpleton.

```
[ EPiServer.PlugIn.GuiPlugIn( DisplayName="ActionWindowSimpleton",
    Description="Action window simpleton plug-in",
    Area=EPiServer.PlugIn.PlugInArea.ActionWindow,
    Url="~/ActionWindowSimpleton.ascx" ) ]
public class ActionWindowSimpleton : System.Web.UI.UserControl {
```

5. Compile and run.
6. Open EPiServer Edit mode and then the Action window, which should look something like this:

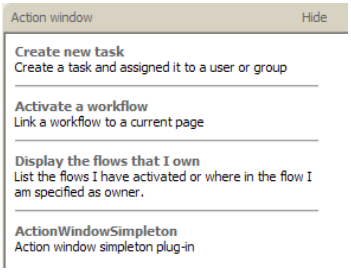


Figure 13-5: ActionWindowSimpleton plug-in in the Action window.

7. Clicking on the ActionWindowSimpleton link should produce this salutation:

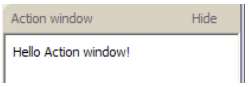


Figure 13-6: ActionWindowSimpleton plug-in open.

From figure 13-5, you can deduce how to use the GuiPlugIn attribute properties DisplayName and Description. The contents of both DisplayName and Description are shown on the first page of the Action window – make them helpful to the person using the Action window.

Create a Live Clock Plug-In for the Action Window

You might have seen this example plug-in before. It is the same plug-in that you can find in the EPiServer Open Development Architecture white paper. The plug-in creates a clock in the action window. Using a small JavaScript, the plug-in continuously displays the time.

Create the Web User Control ActionWindowClock

To set things rolling, we create a new Web User Control and call it ActionWindowClock.

The HTML Part for ActionWindowClock

All the action will be in the HTML part, which is not uncommon for plug-ins. We add the JavaScript in 13-2.

Example 13-2: The JavaScript to add to the HTML part of ActionWindowClock.

```
<script Language="JavaScript">
    function ShowTime() {
        var time = new Date()
        var hour = time.getHours()
        var minute = time.getMinutes()
        var second = time.getSeconds()
        var temp = "" + ( (hour < 10) ? "0" : "") + hour
        temp += ( (minute < 10) ? ":0" : ":") + minute
        temp += ( (second < 10) ? ":0" : ":") + second
        document.getElementById( "ShowTime" ).innerHTML = temp
        id = setTimeout( "ShowTime()", 1000 )
    }
</script>
<div id="ShowTime"></div>
<script>ShowTime()</script>
```

The Code-Behind File for ActionWindowClock

Example 13-3: The code-behind file for ActionWindowClock Web User Control.

```
namespace development {

    [ EPiServer.PlugIn.GuiPlugIn( DisplayName="Clock", Description="Shows the time in 24 hr format",
        Area=EPiServer.PlugIn.PlugInArea.ActionWindow, Url="~/ActionWindowClock.ascx" ) ]
    public abstract class ActionWindowClock : System.Web.UI.UserControl {

...
    }
}
```

Nothing much happens in the code-behind file, the only important thing is the presence of the GuiPlugIn attribute and its attributes which correctly identify the Web User Control as an Action window plug-in.

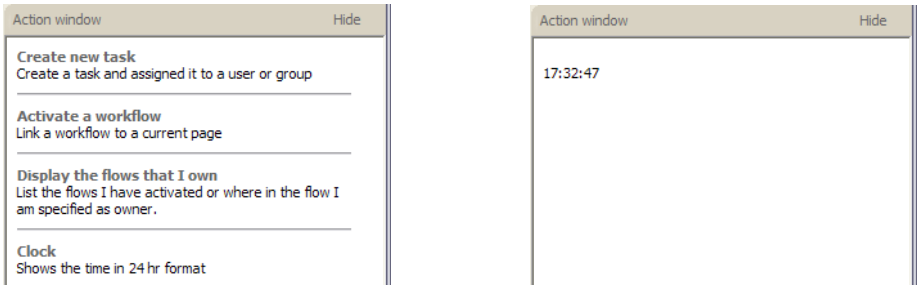


Figure 13-7: ActionWindowClock plug-in: on the left in the Action window menu, on the right in action.

Plug-Ins for the Edit Panel Tab Strip (EPiServer Edit Mode)

Extending the Edit panel tab strip is ideal for extensions which deal with single Web pages, the Web currently being handled in Edit mode.

Very Simple Edit Panel Tab Strip Extension

1. In Visual Studio .NET, right-click on the current project name, select Add and then 'Add Web User Control'.
2. Name this Web User Control 'EditPanelSimpleton'.
3. Switch to HTML mode and enter the text: 'Hello Edit panel tab strip!'.
4. Open the code-behind file and add the all-important GuiPlugInAttribute

   Example 13-4: The GuiPlugInAttribute for EditPanelSimpleton.

   ```
   [ EPiServer.PlugIn.GuiPlugIn( DisplayName="EditPanelSimpleton",
       Description="EditPanelSimpleton is a simple plug-in for the Edit panel tab strip",
       Area=EPiServer.PlugIn.PlugInArea.EditPanel, Url="~/EditPanelSimpleton.ascx" ) ]
   public class EditPanelSimpleton : System.Web.UI.UserControl {
   ```

5. Compile and run.
6. Open EPiServer Edit mode, click on one of the Web Pages and then look at the right pane. It should look something like this:

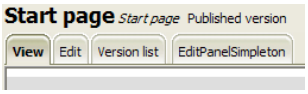
   

   Figure 13-8: EditPanelSimpleton plug-in in the Edit Panel tab strip.

7. Clicking on the EditPanelSimpleton tab should produce this salutation:

Figure 13-9: EditPanelSimpleton plug-in open.

A Page Information Plug-In for the Edit Panel Tab Strip

As the Edit Panel Tab Strip is used for functions which deal with one page, let's create a plug-in which displays some pertinent information for the currently selected page. Given the fact that plug-ins for the Edit panel tab strip will be Web User Controls and our need to access page data effortlessly, we let the Web User Control class inherit EPiServer.UserControlBase.

Create the Web User Control EditPanelPageInfo

A new Web User Control EditPanelPageInfo is created and named.

The HMTL Part of EditPanelPageInfo

The HTML part of EditPanelPageInfo looks like this:

Example 13-5: HTML part of EditPanelPageInfo.

```
<%@ Control Language="c#" AutoEventWireup="false" Codebehind="EditPanelPageInfo.ascx.cs"
    Inherits="development.EditPanelPageInfo"
    TargetSchema="http://schemas.microsoft.com/intellisense/ie5"%>
Page Information<br/>
<table>
    <tr><td>Page Name:</td>        <td><%= CurrentPage.PageName%></td></tr>
    <tr><td>Created: </td>         <td><%= CurrentPage.Created%></td></tr>
    <tr><td>Created by:</td>       <td><%= CurrentPage.CreatedBy%></td></tr>
    <tr><td>Last changed:</td>     <td><%= CurrentPage.Changed%></td></tr>
    <tr><td>Last changed by:</td>  <td><%= CurrentPage.ChangedBy%></td></tr>
</table>
```

As you'll soon see, the class inherits EPiServer.UserControlBase and this enables the simple use of CurrentPage which is seen in example 13-5. This time the code render blocks do not use the special Data Expression Syntax, '<%#', but rather the more ordinary '<%=' simply beacuse the HTML part is not enclosed in a templated control or any other kind of data bound control.

The Code-Behind File for EditPanelPageInfo

Getting the GuiPlugIn attribute and its properties right is the only challenge presented by the code-behind file along with the class inheritance.

Example 13-6: The code-behind file for EditPanelPageInfo.

```
namespace development {
                                    /// <summary>EditPanelPageInfo displays page information, it's
                                    /// a plug-in for the Edit panel tab strip.</summary>
    [ EPiServer.PlugIn.GuiPlugIn( DisplayName="EditPanelPageInfo",
        Description="EditPanelPageInfo displays page information (plug-in for the Edit panel tab strip)",
        Area=EPiServer.PlugIn.PlugInArea.EditPanel, Url="~/EditPanelPageInfo.ascx" ) ]
    public class EditPanelPageInfo : EPiServer.UserControlBase {
...
    }
}
```

Using the Plug-In

When the EditPanelPageInfo plug-in is executed, the results look like this.

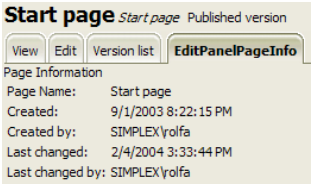


Figure 13-10: EditPanelPageInfo plug-in at work.

Plug-Ins for the EditTree Tab Strip (EPiServer Edit Mode)

Choosing to extend either the Edit Panel area or the Edit Tree area is a rather easy decision to make.

- Plug-ins in the Edit Panel area should deal with a single page, the selected page (see the earlier section).
- Plug-ins in the Edit Tree area can handle the whole Web Page Tree or other system- wide settings (system-wide from the Editor point of view); its area of interest is confined to the left pane in the Edit mode.

Creating a Simple Plug-In for the EditTree Tab Strip

These are the steps to follow when creating plug-ins for the EditTree tab strip.

1. In Visual Studio .NET, right-click on the current project and select first Add and then 'Add Web User Control'.
2. Name this Web User Control 'EditTreeSimpleton'.
3. Switch to HTML mode and enter the text: 'Hello Edit Tree tab strip!'.
4. Open the code-behind file and add the all-important GuiPlugInAttribute

Example 13-7: The GuiPlugInAttribute for EditTreeSimpleton.

```
[ EPiServer.PlugIn.GuiPlugIn( DisplayName="EditTreeSimpleton",
    Description="Edit Tree tab strip simpleton plug-in",
    Area=EPiServer.PlugIn.PlugInArea.EditTree,
    Url="~/EditTreeSimpleton.ascx" ) ]
public class EditTreeSimpleton : System.Web.UI.UserControl {
```

5. Compile and run.
6. Open EPiServer Edit mode. The Edit Tree tab strip should look something like this:

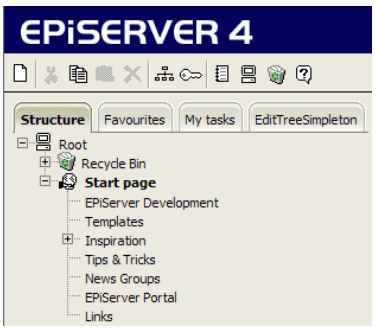


Figure 13-11: EditTreeSimpleton plug-in in the Edit Tree tab strip.

7. Clicking on the EditTreeSimpleton tab should produce this salutation:

Figure 13-12: EditTreeSimpleton plug-in open.

The complete Web User Control is presented next.

Example 13-8: HTML part of EditTreeSimpleton.

```
<%@ Control Language="c#" AutoEventWireup="false" Codebehind="EditTreeSimpleton.ascx.cs"
    Inherits="development.EditTreeSimpleton"
    TargetSchema="http://schemas.microsoft.com/intellisense/ie5"%>
Hello Edit Tree tab strip!
```

Create a 'My Pages' Edit Tree Extension

In an environment where many Editors create Web pages, it can be useful to have a way of dealing only with the pages created by a single editor, e.g. the logged-on user.

Again we'll turn to FindPagesWithCriteria for assistance in selecting the pages whichs were created by the currently logged-on user account. If you look at the specification for EPiServer.Core.PageData, you'll find just the property we need, CreatedBy. As we are going to use FindPagesWithCritiera, the property CreatedBy will be specified as 'PageCreatedBy'. Prepending 'Page' to PageData properties is typical when you handle properties via EPiServer.WebControls.Property or, as in this case, FindPageWithCriteria. (Should you fail to find the 'Page' alias for a certain PageData property, just prepend 'Page' and it should work.)

ExplorerTree (EPiServer.WebControls.ExlporerTree) is the ideal control for the job, and yet it's not perfect. ExplorerTree is the easiest control to use which lets us preserve the Edit mode functionality of being able to click the page name and have it presented in the right pane for further processing. At the same time, EplorerTree fails to present the pages hierarchically if some are missing (small wonder). For this example, we'll accept this small drawback.

Create the Web User Control EditTreeMyPages

First we create a new Web User Control and name it EditTreeMyPages.

HTML Part

Example 13-9: HTML Part of EditTreeMyPages.ascx.

```
<%@ Control Language="c#" AutoEventWireup="false" Codebehind="EditTreeMyPages.ascx.cs"
    Inherits="development.EditTreeMyPages"
    TargetSchema="http://schemas.microsoft.com/intellisense/ie5"%>
<%@ Register TagPrefix="EPiServer" Namespace="EPiServer.WebControls" Assembly="EPiServer" %>
<EPiServer:ExplorerTree EnableVisibleInMenu="False" ShowRootPage="False"
    ShowIcons="False" ClickScript="window.parent.navigateEvent( window, '{ PageLink }' );"
    id="MyPages" runat="server"
/>
```

Now, isn't that the perfect size for the HTML part of a Web User Control – just three HTML code lines! A lot of things must happen out of sight for this to work.

A very important part of the ExplorerTree is the attribute ClickScript. The contents of this attribute, 'window.parent.navigateEvent(window, '{ PageLink }');', were borrowed from the Edit mode Structure tab. It's this script which is responsible for sending the selected page to the right pane of Edit mode.

Code-Behind File for Web User Control EditTreeMyPages

There should be no surprises in the code-behind file for EditTreeMyPages.

Example 13-10: Code-Behind File for Web User Control EditTreeMyPages

```
namespace development {

                    /// <summary>EditTreeMyPages list only the pages the current user has
                    /// created, in a PageTree control.</summary>
```

```
    [ EPiServer.PlugIn.GuiPlugIn( DisplayName="My Pages",
        Description="List only the Web pages you've created",
        Area=EPiServer.PlugIn.PlugInArea.EditTree,
        Url="~/EditTreeMyPages.ascx" ) ]
    public class EditTreeMyPages : System.Web.UI.UserControl {

        protected EPiServer.WebControls.ExplorerTreeMyPages;

        private void Page_Load( object sender, System.EventArgs e ) {
            EPiServer.PropertyCriteriaCollection CurrentUserCriteria =
                new EPiServer.PropertyCriteriaCollection();

            EPiServer.PropertyCriteria CurrentUserCriterion = new EPiServer.PropertyCriteria ();
            CurrentUserCriterion.StringCondition = EPiServer.Filters.StringCompareMethod.Identical;
            CurrentUserCriterion.Type = EPiServer.Core.PropertyDataType.String;
            CurrentUserCriterion.Name = "PageCreatedBy";
            CurrentUserCriterion.Value =
                EPiServer.Security.UnifiedPrincipal.Current.UserData.DisplayName;
            CurrentUserCriterion.Required = true;
            CurrentUserCriteria.Add( CurrentUserCriterion );

            MyPages.DataSource = EPiServer.Global.EPDataFactory.FindPagesWithCriteria(
                EPiServer.Global.EPConfig.RootPage, CurrentUserCriteria );
            MyPages.DataBind();
        }
...
    }
}
```

Figure 13-13: Web User Control EditTreeMyPages at work.

Extending the Extension

Since we are using FindPagesWithCriteria and building our own PageDataCollection (EPiServer.Core.PageDataCollection), we can add any processing we choose before handing off the collection to the ExplorerTree control.

In cases where single editors are responsible for a whole sub-tree, ExplorTree should be fine.

Plug-Ins for the System Settings Area of EPiServer Admin Mode

Being number four in a litter of five cannot be easy, but the System settings area of EPiServer Admin mode is still very handy for some plug-ins. To demonstrate, we'll create a plug-in to list and edit the settings in the appSettings section of the web.config file, but first we'll create the simplest possible System settings area plug-in.

Simplest Possible System Settings Area Plug-In

To create any plug-in for the System settings area, follow these first steps.

1. In Visual Studio .NET, right-click on the current project and select first Add and then 'Add Web User Control'.
2. Name this Web User Control 'SystemSettingsSimpleton'.
3. Switch to HTML mode and enter the text: 'Hello System settings!'.
4. Open the code-behind file and add the all-important GuiPlugInAttribute

   Example 13-11: The GuiPlugInAttribute for SystemSettingsSimpleton.

   ```
   [ EPiServer.PlugIn.GuiPlugIn( DisplayName="Simpleton",
       Description="System settings simpleton plug-in",
       Area=EPiServer.PlugIn.PlugInArea.SystemSettings,
       Url="~/SystemSettingsSimpleton.ascx" ) ]
   public class SystemSettingsSimpleton : System.Web.UI.UserControl {
   ```

5. Compile and run.
6. Open EPiServer Admin mode, click on System settings in the right pane.

7. Your System setting right pane should look something like this:

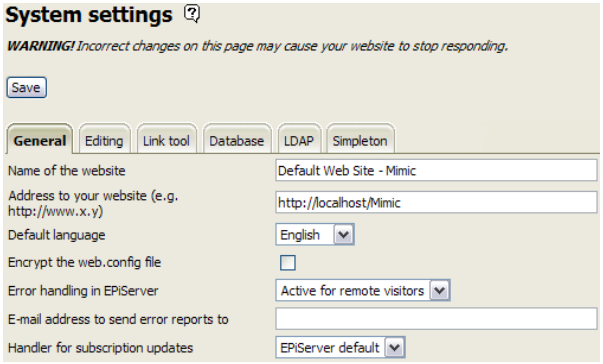


Figure 13-14: SystemSettingsSimpleton plug-in in the System settings tab strip.

8. Clicking on the Simpleton tab should produce this salutation:

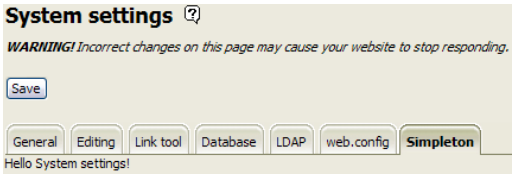


Figure 13-15: SystemSettingsSimpleton plug-in open.

The complete Web User Control is presented next.

Example 13-12: HTML part of SystemSettingsSimpleton.

```
<%@ Control Language="c#" AutoEventWireup="false"
    Codebehind="SystemSettingsSimpleton.ascx.cs" Inherits="development.SystemSettingsSimpleton"
    TargetSchema="http://schemas.microsoft.com/intellisense/ie5" %>
Hello System settings!
```

Example 13-13: Code-behind file for SystemSettingsSimpleton.

```
namespace development {

                        /// <summary>SystemSettingsSimpleton is a very simpleton plug-in for
                        /// System settings area of Admin mode.</summary>
    [ EPiServer.PlugIn.GuiPlugIn( DisplayName="Simpleton",
        Description="System settings simpleton plug-in",
        Area=EPiServer.PlugIn.PlugInArea.SystemSettings,
        Url="~/SystemSettingsSimpleton.ascx" ) ]
    public class SystemSettingsSimpleton : System.Web.UI.UserControl {

        private void Page_Load( object sender, System.EventArgs e ) {
                        // Put user code to initialize the page here
        }
```

```
...
}
```

Armed with the knowledge of how to create plug-ins for the System settings area, we quickly move on to create a really useful one.

Web.config Editor for the System Settings Area in EPiServer Admin Mode

Remember that the appSettings section in the web.config file holds many interesting settings for your EPiServer solution. Access to these settings is via EPiServer.Global.EPConfig.ConfigFile (see page 161). In particular there's a fast way to get at all the settings in the appSettings section using the public property AllAppSettings (EPiServer.Global.EPConfig.ConfigFile.AllAppSettings). It returns a collection of settings name–settings value pairs (see page 161).

For presentation and visual support, we'll use a templated control, the DataList (System.Web.UI.WebControls.DataList).

From the point of view of the DataList control (i.e. its DataSource property), AllAppSettings return a string array (string[]) of all names of the settings found in the appSettings section.

We boldly name this plug-in WebConfigEditor, although you might find it lacking in some respects.

DataList Information

We will be using the DataList control both for presentation and as a means to select items for editing as well as support during editing.

HTML Part of the Plug-In

The HTML part of the plug-in is straightforward, as we can rely on EPiServer Admin mode to take care of many visual aspects for us.

Example 13-14: HTML part of System setting area plug-in WebConfigEditor.

```
<%@ Control Language="c#" AutoEventWireup="false" Codebehind="WebConfigEditor.ascx.cs"
    Inherits="development.WebConfigEditor"
    TargetSchema="http://schemas.microsoft.com/intellisense/ie5" %>
<table border="0" cellpadding="1" cellspacing="1" width="100%">

<asp:DataList id="ConfigFileSettings" runat="server"
    OnEditCommand="EditItem"
    OnCancelCommand="CancelItem"
    OnUpdateCommand="UpdateItem"
>
    <ItemTemplate>
        <tr>
            <td><%# Container.DataItem.ToString() %></td>
            <td><%# EPiServer.Global.EPConfig.ConfigFile.GetAppSetting(
```

```
                Container.DataItem.ToString() ) %></td>
            <td><asp:Button Text="Edit" CommandName="Edit" runat="server" ID="edit" /></td>
        </tr>
    </ItemTemplate>
    <EditItemTemplate>
        <tr>
            <td><asp:Label Text="<%# Container.DataItem.ToString() %>" runat="server"
                ID="SettingNameInEditMode" /></td>
            <td><asp:TextBox Text="<%# EPiServer.Global.EPConfig.ConfigFile.GetAppSetting(
                Container.DataItem.ToString() ) %>" runat=server ID="SettingValueInEditMode"
                Width="100%" Font-Bold="True" /></td>
            <td><asp:Button Text="Cancel" CommandName="Cancel" runat="server" ID="cancel" />
            </td>
            <td><asp:Button Text="Update" CommandName="Update" runat="server" ID="update" />
            </td>
        </tr>
    </EditItemTemplate>
</asp:DataList>
</table>
```

Obviously we're going for a tabular look where settings names go in the first column, the contents of the settings in the next column followed by one or two column containing Button objects.

The more interesting parts of the HTML code in example 13-14 have been marked in bold face font.

The opening tag for the DataList control binds three commands to functions in the code-behind file (DataList can handle five named commands: Cancel, Delete, Edit, Select and Update). (As there is no way from code tood deleting settings in the appSettings section, we have no need for a Delete command.)

Two of DataList's templates are used, ItemTemplate and EditItemTemplate. Only the single item being edited will be rendered using the EditItemTemplate, all the others adhere to the ItemTemplate.

In the EditItemTemplate, we use a Label and a TextBox control to render the settings name and it value, respectively. The TextBox is used for the obvious purpose of being able to edit the settings value. But why a Label control? It's not needed in ItemTemplate. Well, it is needed so we can read the name of the setting in the code-behind file. Speaking of which, let's turn our attention to the code-behind file.

Code-Behind File for the Plug-In

Example 13-15: Code-behind file for System setting area plug-in WebConfigEditor.

```
namespace development {
```

```
                    /// <summary>WebConfigEditor adds web.config editing tab to
                    /// EPiServer Admin mode, System settings.</summary>
[ EPiServer.PlugIn.GuiPlugIn( DisplayName="web.config",
    Description="web.config Settings Editor",
    Area=EPiServer.PlugIn.PlugInArea.SystemSettings, Url="~/WebConfigEditor.ascx" ) ]
public class WebConfigEditor : System.Web.UI.UserControl {

    protectedSystem.Web.UI.WebControls.DataListConfigFileSettings;

    private void Page_Load( object sender, System.EventArgs e ) {
        if ( ! IsPostBack ) {
            DataBind();
            ConfigFileSettings.DataSource = EPiServer.Global.EPConfig.ConfigFile.AllAppSettings;
            ConfigFileSettings.DataBind();
        }
    }

    private void UpdateDataListDataBinding() {
        ConfigFileSettings.DataSource = EPiServer.Global.EPConfig.ConfigFile.AllAppSettings;
        ConfigFileSettings.DataBind();
    }

    public void EditItem( object sender, System.Web.UI.WebControls.DataListCommandEventArgs e)
    {
        ConfigFileSettings.EditItemIndex = e.Item.ItemIndex;
        UpdateDataListDataBinding();
    }

    public void CancelItem( object sender,
        System.Web.UI.WebControls.DataListCommandEventArgs e ) {
        ConfigFileSettings.EditItemIndex = -1;
        UpdateDataListDataBinding();
    }

    public void UpdateItem( object sender,
        System.Web.UI.WebControls.DataListCommandEventArgs e ) {
        EPiServer.Global.EPConfig.ConfigFile.SetAppSetting(
        string SettingName = ( (System.Web.UI.WebControls.Label) e.Item.FindControl(
            "SettingNameInEditMode" ) ).Text;
        string SettingValue = ( (System.Web.UI.WebControls.TextBox) e.Item.FindControl(
            "SettingValueInEditMode" ) ).Text;
        EPiServer.Global.EPConfig.ConfigFile.SetAppSetting( SettingName, SettingValue );
        EPiServer.Global.EPConfig.ConfigFile.Persist();
        ConfigFileSettings.EditItemIndex = -1;
        UpdateDataListDataBinding();
    }
```

```
...
    }
}
```

The code in example 13-15 should be easy to follow. Three of the functions are the service functions for the DataList commands already mentioned (Edit, Cancel and Update). There's one helper function UpdateDataListDataBinding, which updates the DataList.DataSource property and then calls DataList.DataBind.

Plug-In Class Attribute GuiPlugIn

Compared to other GuiPlugIn attributes, this one is just plain ordinary. The DisplayName property string shows up as a name on the new tab in the System settings tab strip.

Function EditItem

Function EditItem has got it made; all it has to do is to set DataList.EditItemIndex to the index of the currently selected item (chosen by clicking on the Edit Button to the right of the setting in Admin mode).

For the DataList control, whenever EditItemIndex is set to a value in the allowed range (0 to DataList.Items.Count – 1) the Item whose index is DataList.EditItemIndex will be rendered using the EditItemTemplate.

Function CancelItem

In our case the Cancel Button is used to escape out of settings edit mode so in CancelItem DataList.EditItemIndex is set to –1, meaning no item will be rendered using EditItemTemplate.

Function UpdateItem

In function UpdateItem, we need to retrieve the name of the setting being edited as well as the new value for the setting. For this purpose, the ID of the ASP.NET control objects used to render the setting name and its contents is used. That's why, in DataList Edit mode, the Label was given an ID of 'SettingNameInEditMode' and the TextBox an ID of 'SettingValueInEditMode'.

The new setting value is written to the web.config file using EPiServer.Global.EPConfig.ConfigFile.SetAppSetting, and, lastly, we call EPiServer.Global.EPConfig.ConfigFile.Persist to convince EPiServer we really meant to save the new setting.

Cancelling DataList Edit mode is the penultimate thing to happen in UpdateItem.

Look (and Feel) of WebConfigEditor

Having successfully compiled WebConfigEditor, you open EPiServer Admin mode and click on System settings in the left pane. In the right pane, there is now a new tab in the tab strip.

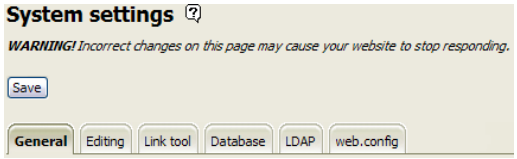


Figure 13-16: WebConfigEditor plug-in in the System settings tab strip.

Clicking on the tab called 'web.config' lets us see the appSettings section of the web.config file in all its glory.

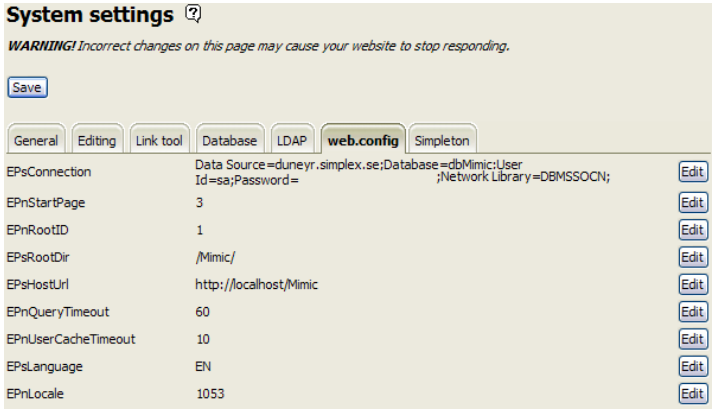


Figure 13-17: WebConfigEditor plug-in open (screen dump has been trimmed).

Clicking on the Edit button 'opens' the edit mode for that particular setting.

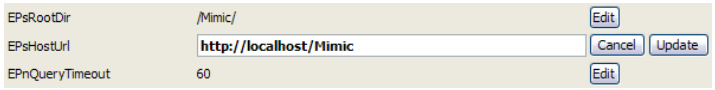


Figure 13-18: Result of clicking on the Edit Button.

In Edit mode, you are free to enter whatever new value you wish for the setting. Pressing Cancel negates any changes whilst clicking on Update writes the new content to the web.config file.

Conclusion

Using very few lines of HTML and corresponding C# code, we've built ourselves a complete WYSIWYG (What You See Is What You Get) editor for the appSettings section in the web.config file. Quite impressive and a testament to the richness of the ASP.NET and EPiServer development environment. However, there

are some obvious improvements we'd like you to consider implementing yourself.

Improvements Left to the Reader

1. Add new setting. A button to add a new setting could either go in the HeaderTemplate of the DataList control or outside the HTML table altogether. Use EPiServer.Global.EPConfig.ConfigFile.SetAppSetting.
2. User entry control. For settings which adhere to the 'EP*xNomen*' naming scheme, it would be a good idea to check user input. Use System.Boolean .Parse and System.Int32.Parse, respectively.
3. Save current value of setting. You could save a few CPU cycles by saving the current setting value, in a class-level string variable, as you enter Edit mode (in EditItem) and then, in UpdateItem, compare the two values.
4. Read and display any settings comment from web.config using EPiServer .Global.EPConfig.ConfigFile.GetComment. Could likely be done in a statement very much like the existing way of reading the value, '<%# EPiServer .Global.EPConfig.ConfigFile.GetComment(Container.DataItem.ToString()) %>'
5. Don't display settings which are controlled via other means.
6. Don't display settings which should be considered read-only.

Plug-Ins for the AdminMenu (EPiServer Admin Mode)

The last of the GUI plug-ins go in the EPiServer Admin mode menu, in the left pane under the Tools heading. A big difference between these extensions and the other four under the GuiPlugIn umbrella is that plug-ins for the Admin mode menu are Web Forms (aspx), not Web User Controls (ascx). Otherwise, they have to abide by the same rules as the others, e.g. the Url property of the GuiPlugIn attribute is equally important for AdminMenu plug-ins.

Very Simple Plug-In for the Admin Mode Menu

Now we have to be careful to make sure we create a Web Form to serve as an Admin mode menu extension, otherwise it's a lot like other GUI extensions.

1. In Visual Studio .NET, right-click on the current project and select first Add and then 'Add **Web Form**'.
2. Name this Web Form 'AdminMenuSimpleton'.
3. Switch to HTML mode and enter the text: 'Hello Admin mode menu!', inside the HTML form.

4. Change the id property of the form to 'AdminMenuSimpleton' (the HTML Title tag has automatically been set to 'AdminMenuSimpleton').

5. Open the code-behind file and add the all-important GuiPlugInAttribute (remember that since the extension is a Web Form, the file extension is aspx.

   Example 13-16: The GuiPlugInAttribute for AdminMenuSimpleton.

```
[ EPiServer.PlugIn.GuiPlugIn( DisplayName="AdminMenuSimpleton",
    Description="Admin Menu Simpleton", Area=EPiServer.PlugIn.PlugInArea.AdminMenu,
    Url="~/AdminMenuSimpleton.aspx" ) ]
public class AdminMenuSimpleton : System.Web.UI.Page {
```

6. Compile and then either start debugging and open EPiServer Admin mode or switch to the browser and open Admin mode.

7. The Tools section should have a new addition.

   Figure 13-19: AdminMenuSimpleton plug-in in the Admin menu, Tools section.

8. Clicking on the AdminMenuSimpleton link should produce this salutation in the right pane:

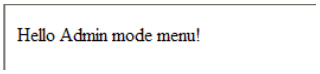


   Figure 13-20: AdminMenuSimpleton plug-in open.

Web Forms control their visual appearance themselves; this is the reason for the all-default appearance of the open AdminMenuSimpleton Web form.

A Perhaps Useful Addition to the Admin Mode Menu

Knowing all there is to know about GUI plug-ins for EPiServer, we set out to create a useful Admin mode menu extension. Our proposed extension is one that displays a list of the Web pages created for the Web site during the last week. This is the same list that we used in chapter 11, *Job Scheduling*, but this time we'll use an EPiServer templated control, PageList, to display information about the new pages. FindPagesWithCriteria (Global.EPDataFactory) will be used to select the pages.

Also, for this last GUI extension we'll be presented with the small challenge of making our contribution look like it was always present in EPiServer Admin mode, from the point of view of visual appearance. Of course, in order to do this in the most effective way, we'll make sure we benefit from the visual infrastructure present in EPiServer. But, all in good time; first we'll make the extension work, then we can make it pleasing to the eye.

Create the Web Form

First we create a new Web Form (remember, Admin mode menu extensions are the only Web Form GUI plug-ins) and name it 'AdminMenuRecentlyCreatedList'.

The HTML Part

In the HTML part, we add one line to make it possible to use control objects from EPiServer.WebControls name space:

Example 13-17: Declaration of EPiServer.WebControls name space in the HTML part of the Web Form AdminMenuRecentlyCreatedList.aspx.

```
<%@ Register TagPrefix="EPiServer" Namespace="EPiServer.WebControls" Assembly="EPiServer" %>
```

The contents of the id property of the HTML Form tag are changed to 'AdminMenuRecentlyCreatedList'. Inside the form we add the DataList parts.

Example 13-18: EPiServer.WebControls.DataList added to the Web Form AdminMenuRecentlyCreatedList.aspx.

```
<table>
<EPiServer:PageList runat="server" ID="RecentlyCreatedPages">
    <ItemTemplate>
        <tr>
            <td><%# Container.CurrentPage.Created.ToString( "r" ) %></td>
            <td><%# Container.CurrentPage.PageName %></td>
            <td><%# Container.CurrentPage.CreatedBy %></td>
        </tr>
    </ItemTemplate>
</EPiServer:PageList>
</table>
```

The DataList control is embedded in an HTML table. Its name is set to RecentlyCreatedPages, as we use this in the code-behind file. Only the ItemTemplate (for reasons which will become clear later) of the DataList control is used. An item is made up of three pieces of information:

- Date–time the page was created
- Name of the page
- Name of user account of creator

This is what the completed HTML part looks like.

Example 13-19: Complete HTML part of AdminMenuRecentlyCreatedList.aspx.

```
<%@ Page language="c#" Codebehind="AdminMenuRecentlyCreatedList.aspx.cs"
    AutoEventWireup="false" Inherits="development.AdminMenuRecentlyCreatedList" %>
```

```
<%@ Register TagPrefix="EPiServer" Namespace="EPiServer.WebControls" Assembly="EPiServer" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<html>
    <head>
        <title>AdminMenuRecentlyCreatedList</title>
        <meta content="Microsoft Visual Studio .NET 7.1" name="GENERATOR">
        <meta content="C#" name="CODE_LANGUAGE">
        <meta content="JavaScript" name="vs_defaultClientScript">
        <meta content="http://schemas.microsoft.com/intellisense/ie5" name="vs_targetSchema">
    </head>
    <body MS_POSITIONING="FlowLayout">
        <form id="AdminMenuRecentlyCreatedList" method="post" runat="server">
            <table>
                <EPiServer:PageList runat="server" ID="RecentlyCreatedPages">
                    <ItemTemplate>
                            <tr>
                                <td><%# Container.CurrentPage.Created.ToString( "r" ) %></td>
                                <td><%# Container.CurrentPage.PageName %></td>
                                <td><%# Container.CurrentPage.CreatedBy %></td>
                            </tr>
                    </ItemTemplate>
                </EPiServer:PageList>
            </table>
        </form>
    </body>
</html>
```

The Code-Behind File

The code-behind file shouldn't present any major obstacles. We have to get the GuiPlugIn attribute right. It's OK to let the class inherit from System.Web.UI .Page, but if necessary, you can let the class inherit from EPiServer.SystemPage, EPiServer.SimplePage or any other class in the EPiServer.PageBase inheritance chain.

Example 13-20: Code-behind file for Web Form AdminMenuRecentlyCreatedList.

```
namespace development {
                                /// <summary>AdminMenuRecentlyCreatedList extends the Admin mode
                                /// menu Tools section with 'Recent pages' choice (see DisplayName).
                                /// </summary>
    [ EPiServer.PlugIn.GuiPlugIn( DisplayName="Recent pages", Description="Recently created pages",
      Area=EPiServer.PlugIn.PlugInArea.AdminMenu, Url="~/AdminMenuRecentlyCreatedList.aspx" ) ]
    public class AdminMenuRecentlyCreatedList : System.Web.UI.Page {

        protected EPiServer.WebControls.PageListRecentlyCreatedPages;
```

```
        private void Page_Load( object sender, System.EventArgs e ) {
            if ( ! IsPostBack ) {
                EPiServer.PropertyCriteriaCollection RecentlyCreatedCriteria =
                    new EPiServer.PropertyCriteriaCollection();

                EPiServer.PropertyCriteria RecentlyCreatedCriterion = new EPiServer.PropertyCriteria ();
                RecentlyCreatedCriterion.Condition = EPiServer.Filters.CompareCondition.GreaterThan;
                RecentlyCreatedCriterion.Type = EPiServer.Core.PropertyDataType.Date;
                RecentlyCreatedCriterion.Name = "PageCreated";
                RecentlyCreatedCriterion.Value = System.DateTime.Now.Date.AddDays( -7 ).ToString();
                RecentlyCreatedCriterion.Required = true;
                RecentlyCreatedCriteria.Add( RecentlyCreatedCriterion );

                RecentlyCreatedPages.DataSource =
                    EPiServer.Global.EPDataFactory.FindPagesWithCriteria(
                    Global.EPConfig.RootPage, RecentlyCreatedCriteria );
                RecentlyCreatedPages.DataBind();
            }
        }
...
    }
}
```

Compiling the project and then opening EPiServer Admin mode reveals a new addition to the Admin mode menu Tools section.

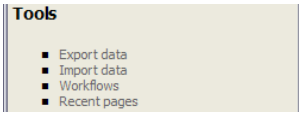


Figure 13-21: Admin mode menu Tools section with new menu choice 'Recent pages'.

Clicking the link 'Recent pages' serves up a nice list of recently created pages and some pertinent page information.

Mon, 05 Apr 2004 10:01:42 GMT EPiServer Development	SIMPLEX\rolfa
Mon, 05 Apr 2004 14:44:47 GMT Global.asax	SIMPLEX\rolfa
Mon, 05 Apr 2004 14:54:09 GMT Global.asax CreatingPage	SIMPLEX\rolfa
Mon, 05 Apr 2004 14:59:54 GMT CreatingPage	SIMPLEX\rolfa
Mon, 05 Apr 2004 15:07:53 GMT CreatingPage II	SIMPLEX\rolfa
Mon, 05 Apr 2004 15:26:08 GMT C3	SIMPLEX\rolfa
Mon, 05 Apr 2004 15:29:10 GMT C4	SIMPLEX\rolfa

Figure 13-22: Result of running AdminMenuRecentlyCreatedList Web Form.

But, and this is big visual but, it certainly looks very odd with the all-default HTML page in the midst of this slick environment. We have to do something

about this. Luckily, help is close at hand. Peeking into another Web Form used in Admin mode, we spot an interesting looking line in the header.

Example 13-21: Style sheet specification line on the header of an Admin mode Web Form.

```
<link href="../util/styles/system.css" type="text/css" rel="stylesheet">
```

We'll borrow this line (see example 13-21) for our Admin mode Web Form. But, beware, here be dragons! Unless we change the relative path statement in the contents of the href attribute, the style sheet won't be picked up and we'll be none the wiser as to why. In this particular case, the Web Form was saved in the root folder of our Web project so this is the line we add to AdminMenuRecentlyCreatedList .aspx.

Example 13-22: Style sheet specification added to the header section of AdminMenuRecentlyCreatedList.aspx.

```
<link href="Util/styles/system.css" type="text/css" rel="stylesheet">
```

Recompiling and running AdminMenuRecentlyCreatedList, there are no complaints about the artistic impression, it's 6.0 all around. While we're at it, we'll add a few more bits of information and visual pleasantries. For example we spotted in the standard Admin mode Web Form that it used two classes from the style sheet: EP-systemHeading and EP-systemInfo. Using these and introducing a footer template section for the PageList produces this completed look for the AdminMenuRecentlyCreatedList Web Form.

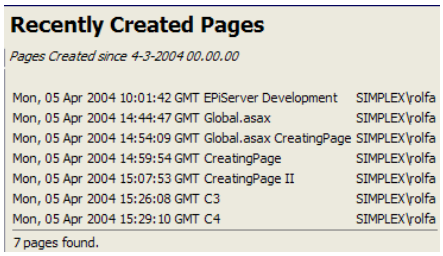


Figure 13-23: Final visual appearance of AdminMenuRecentlyCreatedList.

The HTML part wasn't changed very much.

Example 13-23: Finalised HTML part for AdminMenuRecentlyCreatedList.aspx.

```
<%@ Page language="c#" Codebehind="AdminMenuRecentlyCreatedList.aspx.cs"
AutoEventWireup="false" Inherits="development.AdminMenuRecentlyCreatedList" %>
<%@ Register TagPrefix="EPiServer" Namespace="EPiServer.WebControls" Assembly="EPiServer" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<html>
    <head>
        <title>AdminMenuRecentlyCreatedList</title>
        <meta content="Microsoft Visual Studio .NET 7.1" name="GENERATOR">
        <meta content="C#" name="CODE_LANGUAGE">
```

```
    <meta content="JavaScript" name="vs_defaultClientScript">
    <meta content="http://schemas.microsoft.com/intellisense/ie5" name="vs_targetSchema">
    <link href="Util/styles/system.css" type="text/css" rel="stylesheet">
</head>
<body MS_POSITIONING="FlowLayout">
    <h1 class="EP-systemHeading">Recently Created Pages</h1>
    <span class="EP-systemInfo">Pages Created since
        <%= System.DateTime.Now.Date.AddDays( -7 ) %>
    </span>
    <form id="AdminMenuRecentlyCreatedList" method="post" runat="server">
        <table>
            <EPiServer:PageList runat="server" ID="RecentlyCreatedPages">
                <ItemTemplate>
                    <tr>
                        <td><%# Container.CurrentPage.Created.ToString( "r" ) %></td>
                        <td><%# Container.CurrentPage.PageName %></td>
                        <td><%# Container.CurrentPage.CreatedBy %></td>
                    </tr>
                </ItemTemplate>
                <FooterTemplate>
                    <tr bgcolor="Gray" height="1px"><td colspan="3"></td></tr>
                    <tr>
                        <td colspan="3">
                            <%# RecentlyCreatedPages.DataCount %> page(s) found.</td>
                    </tr>
                </FooterTemplate>
            </EPiServer:PageList>
        </table>
    </form>
</body>
```

As can be seen in example 13-23, the reason we introduce the FooterTemplate for the PageList control is to be able to display the number of pages found in the search. The first table row in the FooterTemplate section is used to produce the horizontal grey line seen in figure 13-23.

Elementary Troubleshooting of GUI Plug-Ins

Some mistakes are unique to EPiServer plug-ins, two of which are presented here.

1. Letting plug-in class inherit from the incorrect mother class, e.g. letting a plug-in for the System setting area inherit from anything but 'System.Web.UI .Control' (or a descendant class, such as EPiServer.UserControlBase) pro-

duces the message 'Only UserControl objects are supported on a tabstrip' when opened in EPiServer Admin mode.

2. Forgetting to name the plug-in in the GuiPlugIn attribute's Url property. Failing to do so produces the message 'User control source files must have a .ascx file extension.' for a System settings area plug-in when attempting to open it.

Extending the DHTML Editor

The DHTML Editor is automatically invoked in Edit mode, or quick-edit mode, to enable editing of properties/values of the type 'Long string (>255)'. Considering the fact that the bulk of text on a Web site is likely to be held in that kind of property/value and the number of people using the DHTML Editor, it's easy to envisage the need for extensibility in the DHTML Editor.

As editors in general are not good candidates for any client–server processing, you will find that the extensibility model of the DHTML Editor relies on a combination of server code and client scripts, written in Microsoft's JavaScript (ECMAScript). In short, the server code is used to set up all that's needed so the client script is able to process whatever is entered in the editor.

When extending the DHTML Editor, we switch names spaces to EPiServer.Editor and the plug-in attribute to use is now EditorPlugInAttribute. There's another big change – DHTML Editor plug-ins are neither Web Forms nor Web User Controls, they are regular C# classes (or Visual Basic .NET classes, for that matter). The reason for this is that the DHTML Editor in itself is a highly visual component; the plug-ins you create are tools to supplement its own functions.

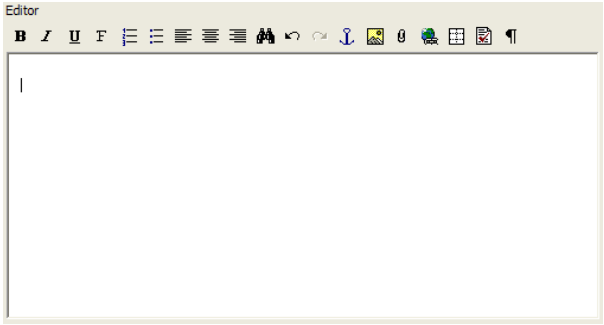


Figure 13-24: The DHTML Editor for LongString properties/values.

An interesting aspect of DHTML Editor plug-ins is that they do not have to be created in the same Visual Studio .NET Solution as the main Web site. If you create them in their own solutions, you may have as many solutions as you have plug-ins. Be sure to either have the resulting DLLs created in the bin folder of the main solution or copied to this folder. (It's probably best to go for the file copy

options – it's less messy.) If you have access to Visual Studio .NET 2003, you can create Web Control Libraries as separate projects.

EPiServer.Editor.EditorPlugInAttribute Class

The class EditorPlugInAttribute inherits directly from EPiServer.PlugIn.GuiPlugInAttribute. Properties and methods declared for GuiPlugInAttribute, or higher in the inheritance chain, are not listed in the tables below.

Of the mother class properties, DisplayName is still very important and you should assign DisplayName the same importance as for the plug-ins discussed earlier in this chapter.

Public Properties for EditorPlugInAttribute

Table 13-14: Public properties for EditorPlugInAttribute.

Property Name	*Description*
CommandIdentifier	DHTML command identifier which the tool can react to. If for example the identifier 'Bold' is selected, the tool will toggle between on and off in tool-bars and menus when the cursor moves around in the editor body.
LanguageKey	Used to handle translation of tool tips
LanguageKeyMenu	Used to handle translation of texts in the context menu
MenuGroup	The name of the menu group in which the plug-in will appear. All plug-ins which have the same group and menu names share the same group in the same menu. The plug-ins in the group are sorted by their MenuIndex. If the group name is empty, the plug-in will show up outside the groups in a position defined by the MenuIndex. When sorting groups, the plug-ins with the lowest MenuIndex in each group are used as the sort keys.
MenuIndex	Position in the menu. If the plug-in belongs to a menu group, the position is within the group.
MenuName	The name of the menu in which the plug-in will appear. If the menu name is empty, the plug-in will appear in the top menu.
MenuSortKey	Complex sort key consisting of period-separated keys, format: 'SubMenuName.MenuGroup.MenuIndex'. Non-empty keys are always prefixed with an underscore, '_'. Empty keys are always set to 'x', which guarantees that the non-empty keys are always sorted before empty keys.

Table 13-14: Public properties for EditorPlugInAttribute.

Property Name	*Description*
SubMenuName	The name of an optional sub menu which will be opened when the mouse pointer is positioned over this plug-in in the menu.
ToolbarIndex	Determines where on the tool-bar the plug-in should be displayed. If not specified, the plug-in will be displayed on the far left on the tool-bar. A high value ToolbarIndex causes the plug-in to be displayed on the far right of the tool-bar area.
Usage	One or more of the members in the EPiServer.Editor.ToolUsage enumeration (see below). The enumeration members may be Or'ed together.

Public Methods for EditorPlugInAttribute

Table 13-15: Public methods for EditorPlugInAttribute.

Method Name	*Description*
IsUsage	Check whether one or more ToolUsage flags are enabled on this object. The flags can be Or'ed before the check.
Match	Overridden.

EPiServer.Editor.ToolUsage Enumeration

The ToolUsage enumeration is used by the Usage property of the EditorPlugIn attribute to specify which part of the DHTML Editor the plug-in is extending.

Table 13-16: Members in the ToolUsage Enumeration.

Member Name	*Description*
None	Plug-in has no visual presence.
Toolbar	Plug-in extends the DHTML Editor tool-bar.
ContextMenu	Plug-in extends the shortcut (right-click) menu.
Keyboard	Plug-in is attached to a keyboard shortcut (not yet supported).
All	Shorthand for 'ToolBar | ContextMenu | Keyboard'.

EPiServer.Editor.Tools.ToolBase Class

Extension classes for DHTML Editor plug-ins must inherit the EPiServer.Editor .Tools.ToolBase class. This is the mother class to a lot of classes, some of which we will acquaint ourselves with in this section.

Public Properties for ToolBase

Table 13-17: Public properties for ToolBase.

Property Name	*Description*
Availability	Gets or sets the modes which the tool supports in the editor.
ClientScriptBlock	Gets or sets the client-side script block to register with the tool.
ClientScriptKey	Gets or sets the key which will be used when registering the client-side script on the aspx page.
ClientSideEnabled	Gets or sets the client-side javascript method(s) which will be used to determine if the tool is currently enabled or not.
ClientSideOnClick	Gets or sets the client-side javascript method(s) which will be used in the OnClick attribute of the icon.
Disabled	Gets or sets value indicating whether you want the tool to be rendered in the toolbar.
Height	Gets or sets the height of the icon.
IconOff	Gets or sets the image name when the icon is off.
IconOver	Gets or sets the image name when the icon is over.
Label	Gets or sets the label to use for the tool.
MenuLabel	Gets or sets the label to use in context menus for the tool.
Name	Gets or sets the unique name of the tool.
ParentEditor	Gets or sets the ParentEditor reference.
ResourceJsName	Gets or sets the resource name for the Client-Side API Java-Script file of the tool.
StartupScriptBlock	Gets or sets the startup client-side script to register with the tool.

Table 13-17: Public properties for ToolBase.

Property Name	*Description*
StartupScriptKey	Gets or sets the key which will be used when registering the startup client-side script on the ASPX page.
Width	Gets or sets the width of the icon.

Protected Methods for ToolBase

Table 13-18: Protected methods for ToolBase.

Method Name	*Description*
RenderTool	Renders the tool to the specified HtmlTextWriter object. Usually a Page.

EPiServer.Editor.Tools.IInitializableTool Interface

This interface is implemented by plug-ins which need to perform setup processing, such as initialisations, client-side scripts handling, attaching to events and more. There is a single method in this interface, Initialize. Any processing which your plug-in needs is put in this method.

Example 13-24: EPiServer.Editor.Tools.IInitializableTool.Initialize function.

```
void Initialize( EPiServer.Editor.HtmlEditor editor );
```

The 'editor' argument passed to the function is the DHTML Editor.

A Skeleton DHTML Editor Extension Plug-In

The class listed in example 13-25 is a complete DHTML Editor extension plug-in. It compiles OK and when you open EPiServer Edit mode you will see the text 'DhtmlEditorSkeleton' (the contents of the attribute DisplayName) in the toolbar area.

Example 13-25: Skeleton DHTML Editor extension plug-in.

```
namespace development {
    [ EPiServer.Editor.EditorPlugIn( DisplayName="DhtmlEditorSkeleton",
        Usage=EPiServer.Editor.ToolUsage.Toolbar ) ]
    public class DhtmlEditorSkeleton : EPiServer.Editor.Tools.ToolBase {
        public DhtmlEditorSkeleton() {
        }
    }
}
```

So, a DHTML Editor plug-in needs to abide by only five rules:

1. It must be a class.
2. The class must inherit EPiServer.Editor.Tools.ToolBase.
3. It must have an EditorPlugIn attribute (EPiServer.Editor.EditorPlugIn).
4. The EditorPlugIn attribute must have a DisplayName property.
5. The EditorPlugIn attribute must have a Usage property.

Basic DHTML Editor Plug-In

To create a basic, really a dummy, plug-in for the DHTML Editor, follow these steps.

1. In Visual Studio .NET, right-click on the current project name, select Add and then 'Add **Class**'.
2. Name this class 'DhtmlEditorSimpleton'.
3. Add the all-important EditorPlugIn attribute

   Example 13-26: The GuiPlugInAttribute for SystemSettingsSimpleton.

   ```
   [ EPiServer.Editor.EditorPlugIn( DisplayName="DhtmlEditorSimpleton",
       Description="DhtmlEditorSimpleton extends the DHTML Editor tool-bar",
       Usage=EPiServer.Editor.ToolUsage.Toolbar ) ]
   ```

4. Let the class inherit EPiServer.Editor.Tools.ToolBase.

   Example 13-27: Class DhtmlEditorSimpleton inherits EPiServer.Editor.Tools.ToolBase.

   ```
   public class DhtmlEditorSimpleton : EPiServer.Editor.Tools.ToolBase {
   ```

5. Add a client side script to the constructor for DhtmlEditorSimpleton.

   Example 13-28: Client-side script in the constructor for DhtmlEditorSimpleton.

   ```
   public DhtmlEditorSimpleton() {
       ClientSideOnClick = "alert( 'Hello DHTML Editor!') ";
   }
   ```

6. Compile, build, and run. Open EPiServer Edit mode, select a page containing a LongString property/value and activate the Edit tab in the right pane.

(Or log on and select Quick-edit from the right-click menu, the shortcut menu.) The DHTML Editor now has an addition to its tool-bar.

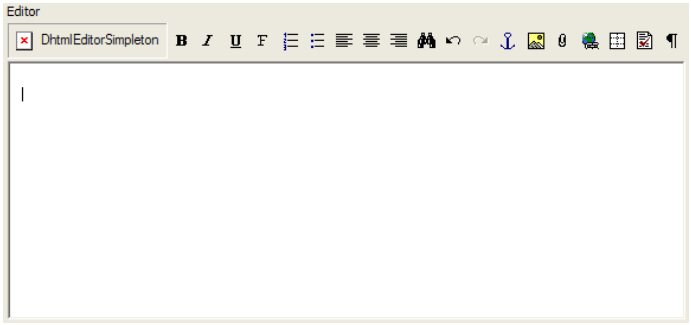


Figure 13-25: The DHTML Editor with the DhtmlEditorSimpleton plug-in visible in the tool-bar.

7. Clicking on the DhtmlEditorSimpleton tool-bar button produces this salutation:

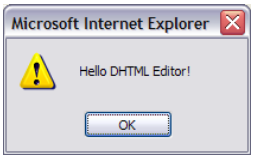


Figure 13-26: DhtmlEditorSimpleton plug-in in action.

Congratulations! You now know how to create plug-ins for all the extensible areas of EPiServer. We do have a little more work to do before we conclude this chapter though.

Before we move on, please take a moment to reflect on this skeleton plug-in for the DHTML Editor. Minimal it may be, but it still demonstrates the dichotomy of the extensibility model. Server processing was used for setup and configuration, including configuring the client-side script 'alert('Hello DHTML Editor!')'. The script is passed to the client upon instantiation of the object, in the constructor. Later, when the button is clicked, there is no server-side processing; everything happens on the client, as the script was bound to the 'click event' by setting the property ClientSideOnClick to the JavaScript statement.

We will create two more plug-ins for the DHTML Editor. Actually they are both variations on the same theme – you decide which you prefer.

The first plug-in will be made part both of the DHTML tool-bar and the context menu (shortcut, right-click, menu) in the Editor. The second will not have any visible part at all; it will act covertly, unbeknown to the user.

As for actions, both will convert some of the letters in the text entered in the Editor into HTML entities, e.g. 'æ' into 'æ'. Plug-in number one, the visible one, will be made to toggle between actual letters and their HTML entity counterparts.

Both plug-ins will also be created outside your main Web site Visual Studio .NET solution, and we'll start by going over a few things you need to think about when doing so.

Creating DHTML Editor Plug-Ins as Separate Visual Studio .NET Solutions

When you create DHTML Editor plug-ins in their own Visual Studio .NET Solutions, there are a few things to consider.

1. Create the plug-ins as 'Class Library' solutions.
2. Add EPiServer.dll from the correct EPiServer binaries daughter folder. (You will find the correct folder as a daughter folder to '%ProgramFiles%\EPiServer4\binaries\EPiServer', its name is its version number.).
3. Add System.Web.dll (found in the list on the .NET tab when adding References).
4. Subsequent to the compile and build steps, copy the resulting library file to the EPiServer Web site's bin folder.

A Visible DHTML Editor Plug-In to Toggle between Letter and HTML Entity

For this plug-in, we'll be introducing several new pieces of information, but fret ye not, we'll explain them all.

Folders Used

EPiServer has been set up so that the client-side DHTML Editor scripts go in the folder Util\javascript\Editor folder and images to use for tool-bar button have their home in the Util\images\Editor folder. With client-side Editor scripts and images in their respective folders, you only have to specify files names in the code.

Create a New Class Library in Visual Studio .NET Solution

We start by creating a new Class Library solution in Visual Studio .NET. Its name is set to 'DhtmlEditorLetterToEntityVisible' and both the class source file name and the class itself are given this name.

Two references, for EPiServer and for System.Web, in the form of the library files EPiServer.dll and System.Web.dll, are added.

Example 13-29: Source code for plug-in class DhtmlEditorLetterToEntityVisible.

```
namespace DhtmlEditorLetterToEntityVisible {
                                /// <summary>DhtmlEditorLetterToEntityVisible is a plug-in for the
                                /// DHTML Editor, converting letters to HTML entities, and back.
                                /// </summary>
    [ EPiServer.Editor.EditorPlugIn( DisplayName="Letter <-> HTML Entity",
        Description="DhtmlEditorLetterToEntityVisible extends the DHTML Editor tool-bar",
        Usage=EPiServer.Editor.ToolUsage.ContextMenu | EPiServer.Editor.ToolUsage.Toolbar ) ]
```

```
    public class DhtmlEditorLetterToEntityVisible : EPiServer.Editor.Tools.ToolBase,
        EPiServer.Editor.Tools.IInitializableTool {

        void EPiServer.Editor.Tools.IInitializableTool.Initialize( EPiServer.Editor.HtmlEditor editor){
            ClientScriptBlock    = editor.GetScriptTag( "LetterToHtmlEntity.js" );
            ClientScriptKey      = "LetterToHtmlEntity.js";
            ClientSideOnClick    = string.Format( "ConvLetterToHtmlEntity( '{0}' );", editor.ClientID );
            IconOff              = "LetterToHtmlEntity_off.gif";
            IconOver             = "LetterToHtmlEntity_over.gif";
        }

    }
}
```

Perusing the code in example 13-29, several interesting pieces of information may be noted:

- The plug-in will be available both on the Editor tool-bar and in the shortcut menu ('Toolbar | ContextMenu' is specified for the Usage property).
- The plug-in also implements the interface IInitializableTool (EPiServer.Editor.Tools.IInitializableTool).
- In the function EPiServer.Editor.Tools.IInitializableTool.Initialize, a client-side script file is specified for ClientScriptBlock and also ClientScriptKey.
- Which of the possible client-side script functions to execute when the user clicks on the tool-bar button for DhtmlEditorLetterToEntityVisible, or selects it in the menu, is specified in the contents of the ClientSideOnClick property. Note in particular the use of the argument 'editor', editor.ClientID (see the client-side script below).
- Tool-bar button images are specified by the contents of properties IconOff and IconOver. Place the images in the folder mentioned above. Also provide images for tool-bar states Disabled and Selected following the same naming scheme.
- After initialisation, all the processing takes place on the client.

Create Client-Side JavaScript File LetterToHtmlEntity.js

The client-side script is stored in a file on the server and physically sent to the client computer as part of the processing in the Initialize function on the class.

We decided to make the script toggle between letter and HTML Entity representation. One lesson we learned is that you cannot simply include the letters in a JavaScript file; it will be messed up as it reaches the client. So, instead of being able to write nice obvious statements as in example 13-30, we had to write them using String.fromCharCode as seen in the code (example 13-31).

Example 13-30: Desired JavaScript replace statements using 'Æ' directly.

```
InnerText = InnerText.replace( "Æ", "&AElig;" );
...
InnerText = InnerText.replace( "&AElig;", "Æ" );
```

Another involuntary lesson was that this String.replace function in JavaScript seems to replace only the first occurrence of the specified string, hence the do–while loops used in the script.

Example 13-31: Client-side JavaScript for DhtmlEditorLetterToEntityVisible class.

```
function ConvLetterToHtmlEntity( id ) {
    var InnerText = getEditor( id ).document.body.innerText;
    var ToEntityNames= false;

    if ( InnerText != null ) {

        ToEntityNames = ToEntityNames || ( InnerText.indexOf( String.fromCharCode( 198 ) ) >= 0 );
        ToEntityNames = ToEntityNames || ( InnerText.indexOf( String.fromCharCode( 230 ) ) >= 0 );

        if ( ToEntityNames ) {
            do {
                InnerText = InnerText.replace( String.fromCharCode( 198 ), "&AElig;" );
            } while ( InnerText.indexOf( String.fromCharCode( 198 ) ) >= 0 );
            do {
                InnerText = InnerText.replace( String.fromCharCode( 230 ), "&aelig;" );
            } while ( InnerText.indexOf( String.fromCharCode( 230 ) ) >= 0 );
        } else {
            do {
                InnerText = InnerText.replace( "&AElig;",String.fromCharCode( 198 ) );
            } while ( InnerText.indexOf( "&AElig;" ) >= 0 );
            do {
                InnerText = InnerText.replace( "&aelig;",String.fromCharCode( 230 ) );
            } while ( InnerText.indexOf( "&aelig;" ) >= 0 );
        }
    }
    getEditor( id ).document.body.innerText = InnerText;
}
```

The script as written is only capable of converting between the letter 'Æ' and HTML Entity 'Æ' and the letter 'æ' and HTML Entity 'æ', respectively. To include more letters, please make use of table D-1 on page 325. To ensure that the function successfully toggles between the letter representation and the HTML Entity representation, you must always test for the presence of all letters which you replace, for variable ToEntityNames.

You might want to make use of the fact that in the ANSI character table there is a fixed distance of 32 between the upper and lower case versions of letters.

Using the Plug-In

Once all JavaScript quirks have been ironed out, the plug-in works very smoothly. To test it, open EPiServer Edit mode and enter some text containing at least one of the letters 'Æ' or 'æ' (type Alt+0230 and Alt+0198 if they're not on your keyboard). We used the simple sentence 'Ægteskab ær hustru og ægtemand' (we're aware of the narrow-minded view, please bear with us).

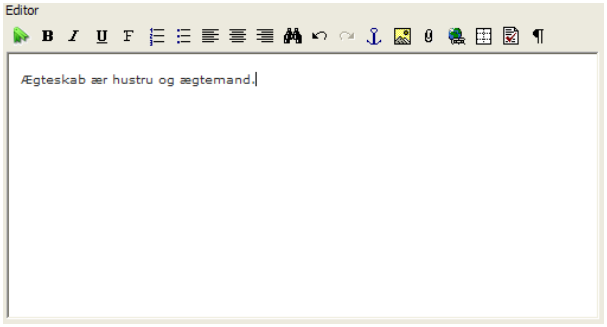


Figure 13-27: The DHTML Editor with the DhtmlEditorLetterToEntityVisible plug-in visible as a new button in the tool-bar.

Pressing the tool-bar button once nicely converts all eligible letters to their HTML Entities as can be seen in figure 13-28.

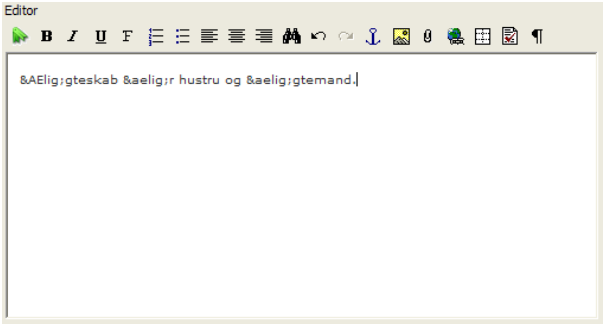


Figure 13-28: The DhtmlEditorLetterToEntityVisible plug-in has been called once.

Next we'll convert the HTML Entities back to proper letters by invoking the shortcut menu (right-click) and then selecting the menu item 'Letter <-> HTML Entity' (it's at the bottom of the menu).

Visual Tuning of the Plug-In

If you're not content with the placement of the tool-bar button or the placement in the shortcut menu, both are controllable via properties of the EditorPlugIn attribute.

Tool-bar button placement is controlled via the integer property ToolbarIndex. The rule is: the higher the value, the farther the placement to the right on the tool-bar.

Menu item placement is similarly controlled via the integer property MenuIndex. This time the rule is: the higher the value, the closer the placement to the bottom of the menu.

An Invisible DHTML Editor Plug-In to Toggle between Letter and HTML Entity

To demonstrate the use of invisible plug-ins, we'll create a plug-in which does exactly what the DhtmlEditorLetterToEntityVisible plug-in does, but using another approach. Instead of exposing the ANSI–HTML Entity toggle in the user interface, this time we'll convert all text to ANSI before passing it to the client. When the user's done editing, all eligible letters in the text will be converted to HTML Entities before passing the text back to the browser. In other words, we divide the work between the server and the client.

The rationale for this plug-in is simplicity for Editors; they can work in their own language and a higher degree of standards abidance as the HTML code in Web pages presented to Web site visitors will feature HTML Entity names instead of non-English letters.

N.B. This plug-in requires EPiServer 4.3 or later.

Create the DhtmlEditorLetterToEntityVisible Class Library

This plug-in will also be created outside of your main Visual Studio .NET solution. Open the Start Page, click New Project and select Class Library. Name the library, the class file and the class itself DhtmlEditorLetterToEntityCovert. Add EPiServer.dll and System.Web.dll as references (see page 304 for procedure).

This is what the class source code looks like, comments to follow.

Example 13-32: Source code for DhtmlEditorLetterToEntityCovert.cs.

```
namespace development {
                                /// <summary>DhtmlEditorLetterToEntityCovert is a plug-in for the
                                /// DHTML Editor, converting letters to HTML entities, and back, without
                                /// user intervention.</summary>
    [ EPiServer.Editor.EditorPlugIn( DisplayName="DhtmlEditorLetterToEntityCovert",
        Description="DhtmlEditorLetterToEntityCovert does without user intervention",
        Usage=EPiServer.Editor.ToolUsage.None ) ]
    public class DhtmlEditorLetterToEntityCovert : EPiServer.Editor.Tools.ToolBase,
        EPiServer.Editor.Tools.IInitializableTool {

        void EPiServer.Editor.Tools.IInitializableTool.Initialize( EPiServer.Editor.HtmlEditor editor) {
            editor.Text             = EntityToLetter( editor.Text );
            ClientScriptKey         = "LetterToHtmlEntityCovert.js";
            ClientScriptBlock       = editor.GetScriptTag( "LetterToHtmlEntityCovert.js" );
            ClientSideOnSubmit      = string.Format( "ConvertLetterToHtmlEntity( '{0}' );",
```

```
                editor.ClientID );
        }

        private string EntityToLetter( string TextToConvert ) {
            System.Text.StringBuilder TempString= new System.Text.StringBuilder( TextToConvert );
            TempString.Replace( "&AElig;","Æ" );
            TempString.Replace( "&aelig;","æ" );
            return TextToConvert;
        }

    }
}
```

Just a few things to note here.

- Usage is set to None, meaning this plug-in does not have any visual representation or a keyboard presence.
- ClientSideSubmit is new for EPiServer 4.3, it is used to set a script to run on the client upon submission.
- 'Pre-processing' of the editor content is done on the server. In order to keep the load low, we use StringBuilder instead String.

Since we don't have to supply a toggling function, the code to convert letters and entities is much simpler. However, we process the contents in two different places and it's easy for them to go out of sync.

Create Client-Side JavaScript File LetterToHtmlEntityCovert.js

The JavaScript file contains a single function.

Example 13-33: JavaScript file LetterToHtmlEntityCovert.js

```
function ConvertLetterToHtmlEntity( id ) {
    var InnerText = getEditor( id ).document.body.innerText;

    if ( InnerText != null ) {
        do {
            InnerText = InnerText.replace( String.fromCharCode( 198 ), "&AElig;" );
        } while ( InnerText.indexOf( String.fromCharCode( 198 ) ) >= 0 );
        do {
            InnerText = InnerText.replace( String.fromCharCode( 230 ), "&aelig;" );
        } while ( InnerText.indexOf( String.fromCharCode( 230 ) ) >= 0 );
    }
    getEditor( id ).document.body.innerText = InnerText;
}
```

The JavaScript function ConvertLetterToHtmlEntity is quite simple; it converts the letters 'Æ' and 'æ' to their respective HTML Entities.

Shadow Folders

Whenever ElektroPost updates EPiServer, you simply download the new version to your Web site, overwriting the previous version. Any changes made to the user interface parts of the Admin and Edit modes are thus lost. With the aid of Shadow Folders, you can control which files are overwritten and which are preserved. However, this places the burden on the developer to maintain compatibility between her own code and new versions of EPiServer.

Switching on Shadow Folders is as easy as adding one line to the settings file web.config:

Example 13-34: Enabling Shadow Folders in web.config.

```
<add key="EPfEnableAlternateFiles" value="True" />
```

Next stop is to add new folders to the EPiServer installation. These folders have fixed names: 'admin_', 'edit_' and 'util_', i.e. the ordinary EPiServer folder names, but with an underscore, '_', appended. When the EPiServer run-time searches for files, it looks in 'admin_' before "admin" and so on. As you add to or replace EPiServer functionality, you put the new files in the new folders which are never overwritten by a new EPiServer version.

NB! Access permissions for the Shadow Folders must be specified in web.config, as they are not automatically inherited.

A
Finding Information

Information on the Internet

As ASP.NET developers, we are blessed with a lot of information and Web sites offering ASP.NET information and source code samples of almost every possible kind and flavour. Searching the Internet for 'ASP.NET' yields a response of more than a million. When it comes to EPiServer development information, the situation is a little different. There are something like five thousand Web pages containing EPiServer information, most of them from our own site www.episerver.com (of course we hope this will change over time as more and more developers start sharing EPiServer information and code samples over the Internet).

A Lot Is Available on the EPiServer Web Site

As one of the major uses for EPiServer 4 is Web-based information solutions, it is no wonder that there's a whole Web site dedicated to EPiServer at http://www.episerver.com.

Having been operating for a few years, the EPiServer Web site is quite comprehensive. On the site you'll find information not only on EPiServer products and customers but also:

- Support, e.g. help with particular problems
- Support tools
- EPiServer Developer Community
- Code Samples
- EPiServer KnowledgeBase
- FAQ (Frequently Asked Questions, i.e. Common Questions) lists
- Technical notes and White Papers
- Product releases
- EPiServer Software Development Kit, SDK
- EPiServer Manuals

EPiServer Developer Community

The mission statement for the EPiServer Developer Community is: 'EPiServer Developer Community is a site dedicated to developers, you can browse code samples or discuss different topics in the forum. We will continuously add more content and features to help developers in their daily work.'

Developer Forums

There are currently three developer forums at the Developer Community Web site:

- Developer to developer – 'Discuss different solutions on developer issues'
- Feature requests – 'New feature requests for EPiServer, from UI to API'
- Problems and bugs – 'Report errors and strange behaviour'

Code Samples

The Code Samples at the Developer Community Web site are written by both ElektroPost developers and other EPiServer developers. There are currently six different categories for code samples:

- Custom property types – 'Custom property types render their own user interfaces for both editing and viewing'
- Editor Tools – 'Tools adding features and extending the functionality of the new DHTML Editor available as of EPiServer version 4.20'
- Navigation and listings – 'Controls and classes which render site navigation'
- Problem solving – 'Useful tools for problem-solving'
- Productivity – 'Specialised tools and classes which make life easier for EPiServer solution developers'
- Subscription – 'Custom subscription mailer and other subscription-related samples'

Frequently Asked Questions (Common Questions) Lists

Every question sent to the EPiServer support department at ElektroPost is a candidate for addition to the Frequently Asked Questions (FAQ) lists. The FAQ lists are categorised for easier access:

- Configuration: Common issues
- Configuration: LDAP and Active Directory
- Editor

- Error messages
- Hotfixes and known issues
- Installation
- Programming: Common questions
- Programming: Templates
- Programming: Web Services
- Scheduler
- Upgrading from EPiServer 3

Technical notes and White Papers

On the EPiServer Web site, you will find both Technical Notes which address a specific technique and White Papers which are broader in scope than Technical Notes. These are some of the White Papers that are available:

- 'Technical Overview of EPiServer'
- 'Getting started with EPiServer 4'
- 'Security in EPiServer 4'
- 'Developing User Controls for EPiServer 4'
- 'Filters in EPiServer 4'
- 'Creating Templates for EPiServer'
- 'Integrating SharePoint and EPiServer'

EPiServer Software Development Kit

No doubt the major information source for EPiServer developers is the EPiServer Software Development Kit, SDK. These are the major components of the SDK:

- EPiServer4SDK.chm – Windows help file documenting EPiServer classes, etc.
- EPiServer.xsd – syntax definition file, see below
- ReadMe.htm (you know what this file is)
- Sample source code from EPiServer itself: PropertyNumber, Edit, Filters and Web Controls (ASP.NET Web Custom Control)

Syntax Definition File

The syntax definition file will enable IntelliSense for EPiServer Web Custom Controls in Visual Studio .NET, giving you auto-completion, option lists, etc. when entering EPiServer tags in Web Forms. To install the file, copy EPiServer.xsd to folder Program Files\Microsoft Visual Studio .NET\Common7\Packages\Schemas\ xml.

To benefit from the IntelliSense, you must include a name space declaration in a tag which wraps the part on which you are working with EPiServer Web Custom Controls. Typically you will include this declaration in the <body> tag. It should look like this:

Example A-1: XML Name Space declaration for EPiServer Web Custom Controls.

```
<body xmlns:EPiServer="http://schemas.episerver.com">
```

You might find a strange first character in EPiServer.xsd – this is a Unicode byte order mark, Visual Studio .NET will not complain about it. (EPiServer.xsd is Unicode encoded and the byte order mark is duplicated and consists of two identical Unicode characters with the character code '0xff 0xfe'.)

B
Database Queries

Important Database Tables

A lot of EPiServer database processing deals with about two handfuls of tables. Most aspects of Page Types, Properties and Web Pages are stored in the database. The most important tables in this respect are listed in table B-1.

Table B-1: The most important tables in the database for Page Types, Properties and Web Pages.

Table Name	*Description*
tblPage	Web Pages, a great deal of the contents are used to instantiate the PageData object.
tblPageDefinition	Properties and the Web Pages they are attached to.
tblPageDefinitionType	Property Data Types, both built-in and user-defined.
tblPageType	Defined Page Types.
tblProperty	All Properties for all Web Pages and their values.
tblACL	Access Control List, ACL, for a Web page.
tblSID	EPiServer Security Identifiers for users and groups.
tblSIDGroup	Connector between groups and users (connects entries in tblSID)
tblUser	Registered users on the Web site.

Please note that all queries below assume that the query is executed inside the pertinent database.

The relationships between the tables can be seen in figures B-1 and B-2.

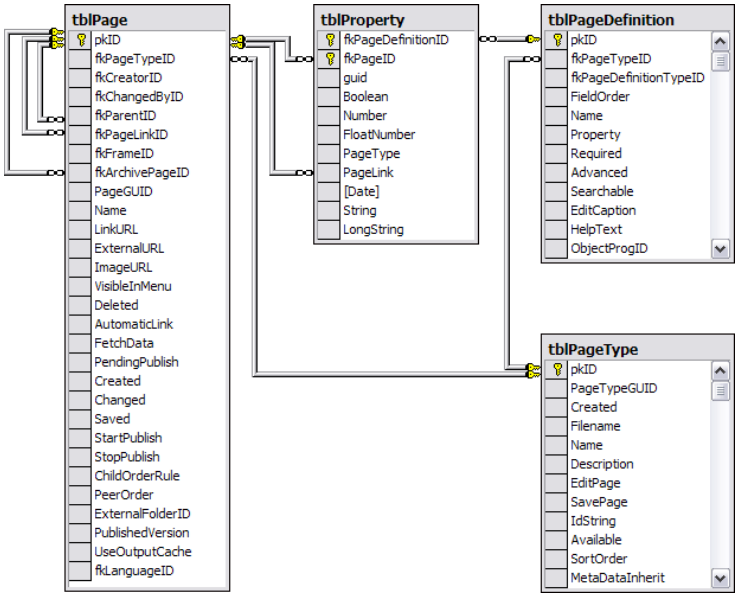


Figure B-1: Relationships between the database tables, tblPage, tblProperty, tblPageDefinition and tblPageType.

In figure B-1, note for example that in tblPage there are two internal relationships. One of them, between columns fkParentID and pkID, is used to realise the page tree hierarchy. Every page in the page tree has a parent except for the page pointed to by EPiServer.Global.EPConfig.RootPage, which has a null entry for fkParentID.

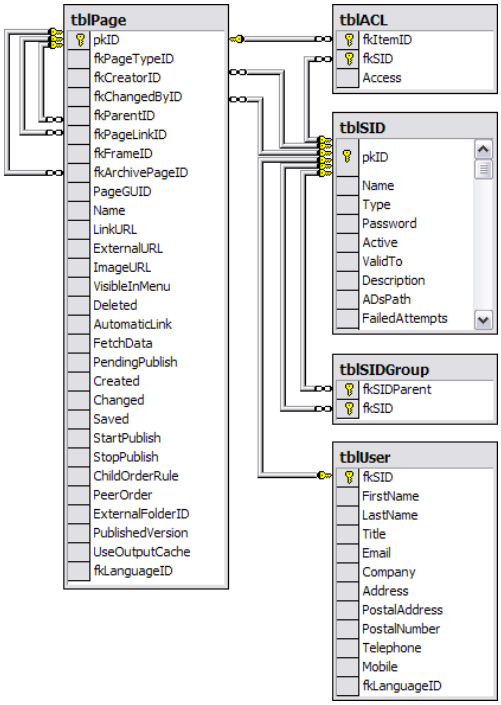


Figure B-2: Relationships between tblPage, tblACL, tblSID, tblSIDGroup and tblUser.

Figure B-2 depicts four different sets of relationships:

- Linking user to EPiServer Security ID, SID, and EPiServer Group
- Linking access permission (Access Control List, ACL) to an EPiServer Security ID
- Linking page creators and changers to EPiServer Security ID

SQL Queries to Retrieve Page Types, Properties and Web Pages

List All Defined Page Types

Example B-1: SQL query to list all defined Page Types.

```
select Name, Filename from tblPageType
order by Name
```

There is more interesting information in the Page Type table, e.g. the column pkID, which is the primary key. You'll find this primary key used as a foreign key in several other tables, mostly under the name fkPageTypeID.

List All Page Template Files, Page Types and Web Pages

Example B-2: SQL Query to list all Page Template Files, Page Types and Web Pages.

```
select distinct tblPageType.FileName 'Page Template File', tblPageType.Name 'Page Type Name',
tblPageType.Description 'Page Type Description', tblPage.Name 'Web Page Name',
tblPage.LinkURL 'Web Page URL'
from tblPageType
inner join tblPage
    on tblPageType.pkID = tblPage.fkPageTypeID
order by tblPage.Name
```

List All Defined Data Types

Example B-3: SQL query to list all Property Data Types.

```
select Name, TypeName, AssemblyName from tblPageDefinitionType
order by TypeName, Name
```

Executing this query against a newly installed EPiServer Web site database would yield a result set much like this:

Table B-2: EPiServer Data Types defined at installation.

Data Type Name	*Full Data Type Name*	*Assembly Name*
Boolean	NULL	NULL
Category	NULL	NULL
Date	NULL	NULL
FloatNumber	NULL	NULL
Form	NULL	NULL
LongString	NULL	NULL
Number	NULL	NULL
PageReference	NULL	NULL
PageType	NULL	NULL
String	NULL	NULL
DocumentUrl	EPiServer.SpecializedProperties.Property-DocumentUrl	EPiServer
Frame	EPiServer.SpecializedProperties.PropertyFrame	EPiServer

Table B-2: EPiServer Data Types defined at installation.

Data Type Name	*Full Data Type Name*	*Assembly Name*
ImageUrl	EPiServer.SpecializedProperties.PropertyImageUrl	EPiServer
Language	EPiServer.SpecializedProperties.PropertyLanguage	EPiServer
Password	EPiServer.SpecializedProperties.PropertyPassword	EPiServer
Selector	EPiServer.SpecializedProperties.PropertySelector	EPiServer
Sid	EPiServer.SpecializedProperties.PropertySid	EPiServer
SortOrder	EPiServer.SpecializedProperties.PropertySortOrder	EPiServer
Url	EPiServer.SpecializedProperties.PropertyUrl	EPiServer
WeekDay	EPiServer.SpecializedProperties.PropertyWeekDay	EPiServer

List All Defined Property Types and Their Data Type

Example B-4: SQL Query to list all defined Property Types and their Data Type.

```
select distinct tblPageDefinition.Name 'Property Name', tblPageDefinitionType.Name 'Data Type',
tblPageDefinition.EditCaption, tblPageDefinition.HelpText
from tblPageDefinition inner join tblPageDefinitionType
    on tblPageDefinition.fkPageDefinitionTypeID=tblPageDefinitionType.pkID
order by tblPageDefinition.Name
```

List All Page Types and Their Properties

Example B-5: SQL query list all defined Page Types and their Properties.

```
select tblPageType.Name 'Page Type Name', tblPageType.Description 'Page Type Description',
tblPageType.FileName 'Page Template File', tblPageDefinition.Name 'Property Name',
tblPageDefinition.EditCaption 'Edit Heading', tblPageDefinition.HelpText 'Help Text'
from tblPageType inner join tblPageDefinition
    on tblPageType.pkID = tblPageDefinition.fkPageTypeID
order by tblPageType.Name, tblPageDefinition.FieldOrder
```

List All Dynamic Properties

Example B-6: SQL query to list all Dynamic Properties.

```
select distinct tblPageDefinition.Name 'Property Name', tblPageDefinitionType.Name 'Data Type',
tblPageDefinition.EditCaption, tblPageDefinition.HelpText
from tblPageDefinition inner join tblPageDefinitionType
    on tblPageDefinition.fkPageDefinitionTypeID=tblPageDefinitionType.pkID
```

```
    where tblPageDefinition.fkPageTypeID is null
order by tblPageDefinition.Name
```

List All Web Pages with Their Properties and Current Values

Example B-7: SQL query to list all Web Pages with their Properties and current values.

```
select tblPage.LinkURL 'Web Page URL', tblPage.Name 'Web Page Name', tblPageDefinition.Name
'Property Name', 'Property Value' =
    case
        when tblProperty.Number is null and tblProperty.FloatNumber is null and
            tblProperty.PageType is null and tblProperty.PageLink is null and tblProperty.Date is null and
            tblProperty.String is null and tblProperty.LongString is null
            then 'Boolean: ' + cast( tblProperty.Boolean as varchar( 40 ) )
        when tblProperty.Number is not null then 'Number: '+ cast( tblProperty.Number as varchar( 40 ) )
        when tblProperty.FloatNumber is not null then 'FloatNumber: '+ cast( tblProperty.FloatNumber as
            varchar( 40 ) )
        when tblProperty.PageType is not null then 'PageType: '+ cast( tblProperty.PageTypeas
            varchar( 40 ) )
        when tblProperty.PageLink is not null then 'PageLink: '+ cast( tblProperty.PageLinkas
            varchar( 40 ) )
        when tblProperty.Date is not null then 'Date: '+ cast( tblProperty.Dateas varchar( 40 ) )
        when tblProperty.String is not null then 'String: '+ cast( tblProperty.Stringas varchar( 40 ) )
        when tblProperty.LongString is not null then 'LongString: '+ cast( tblProperty.LongStringas
            varchar( 40 ) )
        else cast( 'Error Determining Value!' as varchar( 40 ) )
    end
from tblProperty inner join tblPage
    on tblProperty.fkPageID = tblPage.pkID inner join tblPageDefinition
    on tblProperty.fkPageDefinitionID = tblPageDefinition.pkID
order by tblPage.LinkURL, tblPage.Name, tblPageDefinition.Name
```

SQL Query to List All User Tables and Their Columns In a SQL Server Database

Example B-8: SQL query to list all user tables and their columns in a Microsoft SQL Server database.

```
select sysobjects.Name, syscolumns.Name
    from sysobjects inner join syscolumns on sysobjects.id = syscolumns.id
where sysobjects.xtype = 'U'
order by sysobjects.Name, syscolumns.colorder
```

SQL Server Procedure to Display the Web Page Hierarchy

SQL Server Books Online presents a stored procedure to display hierarchies in stored data (see 'Expanding Hierarchies'). We found a more elegant, recursive, stored procedure to solve the same problem on the Internet (see http://vyaskn .tripod.com/hierarchies_in_sql_server_databases.htm).

This is what it looks like adapted for tblPage:

Example B-9: SQL commands to produce and execute a recursive stored procedure to display the Web Page hierarchy in table tblPage.

```
IF EXISTS (SELECT name FROM sysobjects
        WHERE name = 'ShowHierarchy' AND type = 'P')
    DROP PROCEDURE ShowHierarchy
go

CREATE PROC dbo.ShowHierarchy ( @Root int ) AS BEGIN
    SET NOCOUNT ON
    DECLARE @PageID int, @PageName varchar(30)

    SET @PageName = (SELECT Name FROM dbo.tblPage WHERE pkID = @Root)
    PRINT REPLICATE( '-', @@NESTLEVEL * 4) + @PageName

    SET @PageID = (SELECT MIN( pkID ) FROM dbo.tblPage WHERE fkParentID = @Root)

    WHILE @PageID IS NOT NULL
    BEGIN
        EXEC dbo.ShowHierarchy @PageID
        SET @PageID = (SELECT MIN( pkID ) FROM dbo.tblPage
            WHERE fkParentID = @Root AND pkID > @PageID)
    END
END
go

ShowHierarchy 1
go
```

Assuming that the smallest pkID value is 1, the commands in example B-9 would produce the complete hierarchy for all pages in tblPage.

To see only the Web pages which are marked as VisibleInMenu (value set to 1), alter the above procedure to look like this:

Example B-10: Recursive stored procedure to list the hierarchy of all visible Web pages.

```
CREATE PROC dbo.ShowHierarchy ( @Root int ) AS BEGIN
    SET NOCOUNT ON
    DECLARE @PageID int, @PageName varchar(30)

    SET @PageName = ( SELECT Name FROM dbo.tblPage WHERE pkID = @Root )
    PRINT REPLICATE('-', @@NESTLEVEL * 4) + @PageName

    SET @PageID = (SELECT MIN( pkID ) FROM dbo.tblPage WHERE fkParentID = @Root
        and VisibleInMenu = 1 )

    WHILE @PageID IS NOT NULL
    BEGIN
```

```
            EXEC dbo.ShowHierarchy @PageID
            SET @PageID = ( SELECT MIN( pkID ) FROM dbo.tblPage WHERE fkParentID = @Root
            and VisibleInMenu = 1 AND pkID > @PageID )
        END
    END
```

C
Developers' Book List

This is a short list, but we can whole-heartedly recommend each and everyone of these books!

Code Complete, Steve McConnell (Microsoft Press 1993, ISBN 1-55615-484-4)

Design Patterns, Erich Gamma, Richard Helm, Ralph Johnsson & John Vlissides (Addison-Wesley 1995, ISBN 0-201-63361-2)

Mr. Bunny's Guide to ActiveX, Carlton Egremont III (Carlton Egremont III 1998, ISBN 0-201-48536-2)

Refactoring: Improving the Design of Existing Software, Martin Fowler (Addison-Wesley 2000, ISBN 0-201-48567-2)

About Face 2.0: The Essentials of Interaction Design, Alan Cooper & Robert Reimann (John Wiley & Sons Inc 2003, ISBN 0-7645-2641-3)

Designing with Web Standards, Jeffrey Zeldman (New Riders 2003, ISBN 0-7357-1201-8)

Writing Secure Code, 2:nd edition, Michael Howard and David LeBlanc (MS Press 2003, ISBN 0-7356-1722-8)

D
ANSI To HTML Entity Table

Table D-1: ANSI character code to HTML Entity Names.

Code	*Char*	*HTML Entity*	*Code*	*Char*	*HTML Entity*
128	€	€	192	À	À
129			193	Á	Á
130	‚	‚	194	Â	Â
131	*ƒ*	ƒ	195	Ã	Ã
132	„	„	196	Ä	Ä
133	…	…	197	Å	Å
134	†	†	198	Æ	Æ
135	‡	‡	199	Ç	Ç
136	ˆ	ˆ	200	È	È
137	‰	‰	201	É	É
138	Š	Š	202	Ê	Ê
139	‹	‹	203	Ë	Ë
140	Œ	Œ	204	Ì	Ì
141			205	Í	Í
142	Ž	Ž	206	Î	Î
143			207	Ï	Ï
144			208	Ð	Ð
145	‘	‘	209	Ñ	Ñ

Table D-1: ANSI character code to HTML Entity Names.

Code	*Char*	*HTML Entity*	*Code*	*Char*	*HTML Entity*
146	’	’	210	Ò	Ò
147	“	“	211	Ó	Ó
148	”	”	212	Ô	Ô
149	o	•	213	Õ	Õ
150	–	–	214	Ö	Ö
151	—	—	215	×	×
152	˜	˜	216	Ø	Ø
153	™	™	217	Ù	Ù
154	š	š	218	Ú	Ú
155	›	›	219	Û	Û
156	œ	œ	220	Ü	Ü
157			221	Ý	Ý
158	ž	ž	222	Þ	Þ
159	Ÿ	Ÿ	223	ß	ß
160			224	à	à
161	¡	¡	225	á	á
162	¢	¢	226	â	â
163	£	£	227	ã	ã
164	¤	¤	228	ä	ä
165	¥	¥	229	å	å
166	¦	¦	230	æ	æ
167	§	§	231	ç	ç
168	¨	¨	232	è	è
169	©	©	233	é	é
170	ª	ª	234	ê	ê

Table D-1: ANSI character code to HTML Entity Names.

Code	*Char*	*HTML Entity*	*Code*	*Char*	*HTML Entity*
171	«	«	235	ë	ë
172		¬	236	ì	ì
173	-		237	í	í
174	®	®	238	î	î
175	¯	¯	239	ï	ï
176	°	°	240	ð	ð
177	±	±	241	ñ	ñ
178	²	²	242	ò	ò
179	³	³	243	ó	ó
180	´	´	244	ô	ô
181	µ	µ	245	õ	õ
182	¶	¶	246	ö	ö
183	·	·	247	÷	÷
184	¸	¸	248	ø	ø
185	¹	¹	249	ù	ù
186	º	º	250	ú	ú
187	»	»	251	û	û
188	¼	¼	252	ü	ü
189	½	½	253	ý	ý
190	¾	¾	254	þ	þ
191	¿	¿	255	ÿ	ÿ

List of Figures

List of Tables

List of Examples

Index