

# Beyond Express Edition: Special Features of Visual Studio

## LEARNING OBJECTIVES

- To understand how Visual Web Developer Express Edition differs from the full version provide with Visual Studio
- To be aware of the various mark-up languages for mobile devices
- To be able to use Visual Studio to build mobile web applications
- To be able to localize a web application
- To be aware of current developments in .NET

## INTRODUCTION

So far in *Dynamic Web Application Development using ASP.NET* we have looked at features of Visual Web Developer that are available in the Express Edition of the product. However the Express Edition does not have all of the features available with the complete edition of Visual Web Developer, which is shipped with Visual Studio. In this chapter we introduce a number of important features of the full edition of the product, including the ability to create mobile web forms and localization. We also outline some of the other differences between the Express Editions of Visual Web Developer, and also SQL Server, and the professional versions typically used in commercial web development.

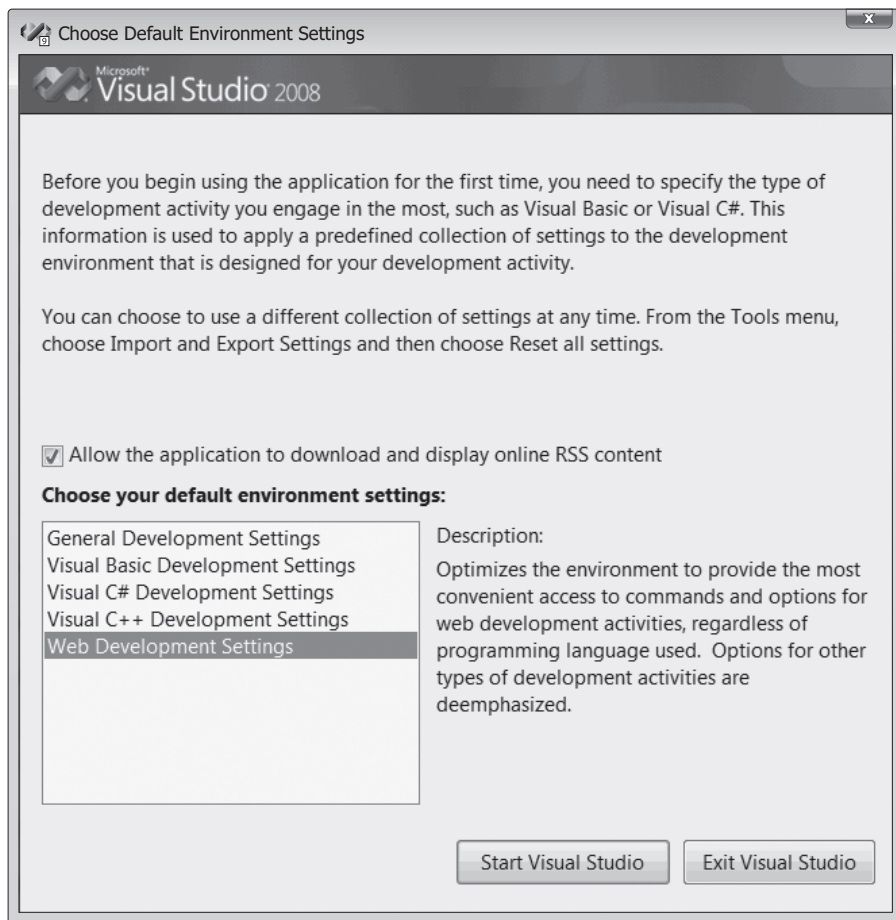
### **15.1** Beyond Visual Web Developer Express Edition

So far, this book has covered ASP.NET web development using the Express Editions of Visual Web Developer and SQL Server. Professional or commercial web developers are more likely to use a commercial edition of this software, so it is worthwhile pointing out here the main differences. There is no standalone commercial edition of Visual Web

Developer. Instead, you should buy one of the versions of Visual Studio such as Standard, Professional or Team System, where the complete version of Visual Web Developer is one of the installation options. Figure 15.1 shows the installation dialog of Visual Studio 2008, where Visual Web Developer can be seen as one of the installation and maintenance options, under the 'Language Tools' category.

In general, each of these installation categories provides a superset of the features in Visual Web Developer Express Edition. For example, they all support the development of desktop Windows applications, and a wider range of programming languages including for example C++ as well as C# and Visual Basic. These extra features of course can be a disadvantage while you are still learning about ASP.NET web development. Visual Web Developer Express Edition is already a rich and powerful environment which takes time to learn how to use fluently. Eventually, however, it is likely you will feel the need for some of the additional features provided by one of the full Visual Studio products. These include enhanced debugging and refactoring facilities, richer support for projects and code sharing, more extensive integration with the Microsoft Developer Network (MSDN) and support for external tools or plug-ins, and finally a wider range of target platforms, notably mobile devices.

**FIGURE 15.1** Visual Web Developer is one of the 'Language Tools' options in the Visual Studio installation/maintenance process



As well as these broad aspects, there are some specific differences which you will notice when you look at the user interface of these different products. Some of these relate to the broader scope of Visual Studio, but even restricting attention to web development, there are some significant differences, for example the following features:

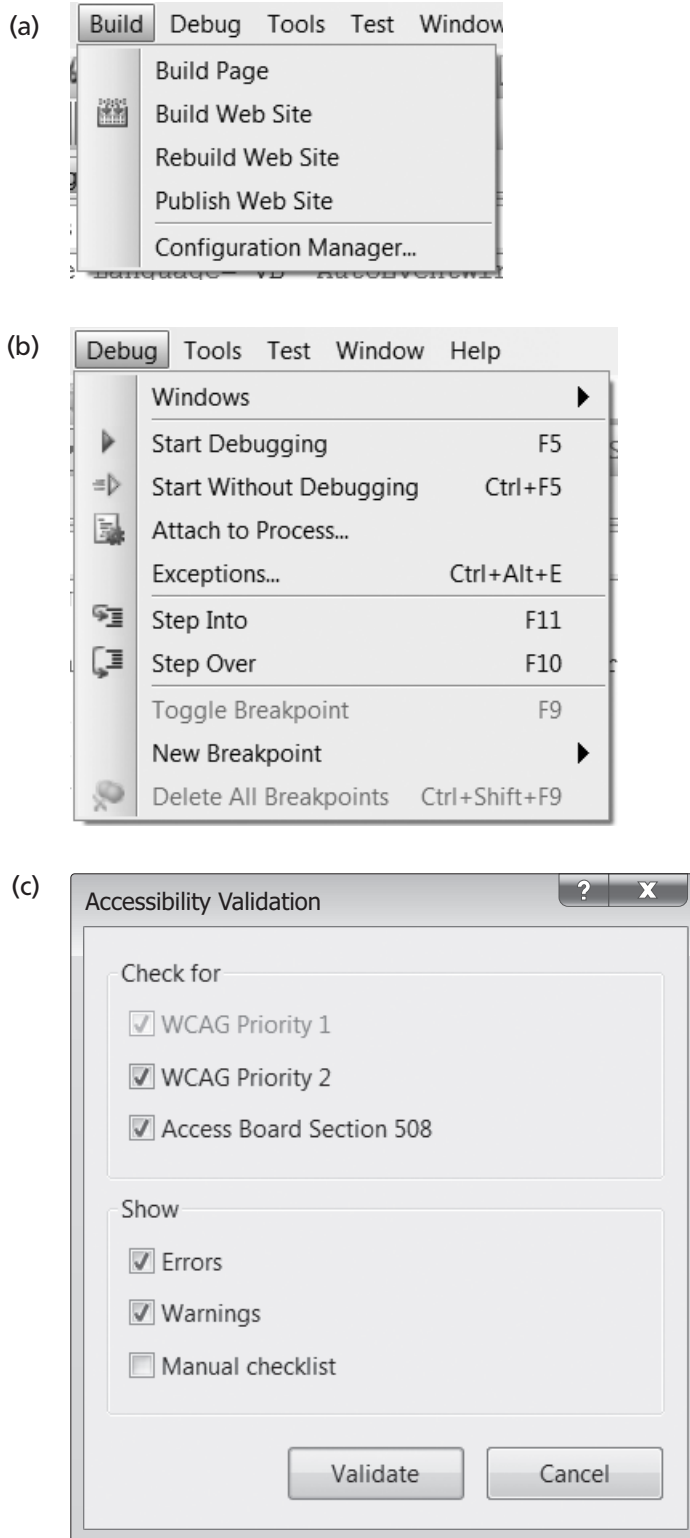
- Visual Studio includes an accessibility checker for web pages and sites. This allows you to validate your HTML against accessibility standards such as WCAG and section 508.
- Visual Studio also includes a facility to ‘publish’ a website, not just copy it. This facility in fact allows you to compile and build your website, as one or more binary files. Doing so helps to protect your source code (particularly useful if you are using a third party Internet Service Provider to host your site) and has a slight performance advantage relative to the standard ‘just-in-time’ approach to compilation.
- Visual Studio does not include the ASP.NET Website Administration Tool. This tool is replaced by the Build Configuration Manager, which is less useful for web development. This means it is more likely you will need to edit your site’s web.config configuration file directly, or else make use of additional tools such as the configuration tools provided with IIS or SQL Server. For example, to set up the membership provider database used by the ASP.NET Login and Web Parts controls, you will need to run the aspnet\_regsql utility.
- Visual Studio does not include a Database Explorer window. Instead, it has a Server Explorer window which allows you to monitor and control a range of servers, including, but not limited to, databases. In addition, the Server Explorer allows you access to Crystal Reports, event logs and message queues. The functions provided are a superset of Visual Web Developer’s Database Explorer. To access a database, open the Database Connections node, and thereafter the same facilities are provided, for example to view and edit database tables, which behave the same as in Visual Web Developer Express Edition.

Note that when running Visual Studio for the first time, it asks you to specify the type of development activity you most often engage in. You should choose Web Development, and the appropriate settings will be applied, and appropriate options emphasized. You can reset these settings and options later, of course, using the ‘Tools’ → ‘Settings’ menu options. In Web Development mode, Visual Studio appears like an enhanced version of Visual Web Developer Express Edition, with one or two more menu items, and a few additional commands on each menu. The screen shots in Figure 15.2 give some indication of the scope and nature of these additional items. There are additional commands available in the Build and Debug menus in Visual Studio compared to Visual Web Developer Express Edition, which include the ability to build, rebuild and publish a website, and the ability to attach to and debug a running process. The third screen shot shows the website accessibility validation dialog.

As mentioned previously, the Database Explorer window is replaced by the Server Explorer. This can be launched using the ‘View’ → ‘Server Explorer’ menu command which, by default, can be found in the same position in the same menu as when using Express edition. As the screen shots in Figure 15.3 demonstrate, the Server Explorer facilities are, as usual, a superset of those found in the Express Edition of Visual Web Developer.

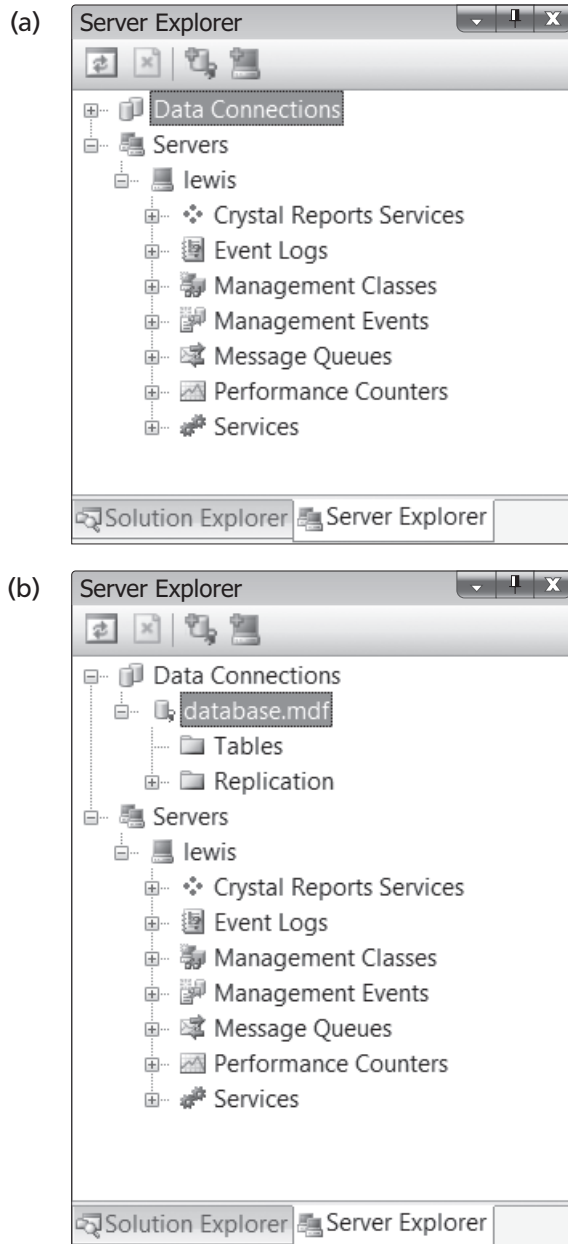
Similarly, when creating a new file, you will note that a few additional types are available such as ADO.NET Data Service, Crystal Report, or Class Diagram.

**FIGURE 15.2** Additional commands available in Visual Studio



**FIGURE 15.3**

Visual Studio's Server Explorer window allows you to view and monitor all types of server, not just databases. If you open up the Data Connections node, however, you will find it offers the same view and facilities as Express Edition's Database Explorer



The current version of Visual Studio is known as 2008, and the next major release is scheduled for 2010. No doubt you will find descriptions on-line of the expected improvements, as well as beta versions of this software for you to download and experience for yourself.

## 15.1.1 SQL Server

The free Express Edition of SQL Server has ample functionality for all the examples we have covered in this book. However the commercial edition of SQL Server has a wealth of additional features which make it a better choice for use in a commercial production environment. Note that the Express Edition license warns that Microsoft may not provide support services for this product.

The additional features of the full version of SQL Server relate mostly to monitoring, administration and scalability. For example, if you wish to spread your database across multiple database servers, you will need the full edition of SQL Server. Even for smaller systems, however, you should certainly benefit from some of the other ease of administration features.

One difference you will notice when switching from the Express Edition to the full SQL Server is that it runs using a different process name. Thus your connection strings must change from using a name such as `.\SQLEXPRESS`, signifying the instance of SQL Server Express Edition running on the local host, to ones such as `remoteServer\SQLEXPRESS`, signifying an instance of SQL Server running on a remote host. It is likely that the connection string will also vary in other ways, relating for example to the different security policy. Note that there is a whole website devoted to explaining and sharing connection strings.

A further difference is, as noted above, the need to use the `aspnet_regsql` utility to set up the membership database for use by the ASP.NET Login and Web Parts controls (if you are using these).

## 15.1.2 Testing your website

To start with, testing a website is easy, as you can simply browse around your site using the internal web browser provided with Visual Web Developer or an external one. As your site grows, however, you will find that testing becomes somewhat tedious and repetitive. At this point, you should consider making use of automated web testing tools.

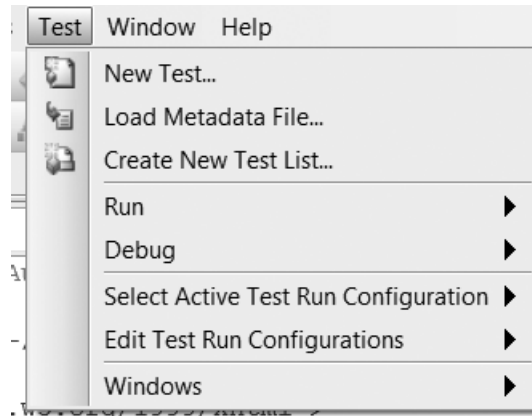
Fortunately, Visual Studio and Microsoft .NET include tools for unit testing of classes and methods. This can be accessed using the Test menu in Visual Studio, as illustrated in Figure 15.4.

The most popular third party automated testing tool for Microsoft .NET is NUnit, a variant of the popular JUnit and SUnit tools for Java and Smalltalk respectively (these various unit testing tools are known collectively as the 'xUnit family'). NUnit is a freely available tool, funded by voluntary contributions. It is relatively mature, being in its fifth major release at the time of writing. You should note, however, that its main focus is testing standard .NET code. This means you can use it to test your business logic, or data access objects, provided you have created these as independent classes, which of course is the correct way to design such code.

For testing ASP.NET pages, you need more specialized testing software. One such system, used internally by the ASP.NET QA team, is the 'Lightweight Test Automation

**FIGURE 15.4**

Microsoft Visual Studio includes support, inside the development environment, for creating and running test cases. You can specify, using the test run configuration, whether to test locally or remotely, the required code coverage instrumentation, and so on



Framework for ASP.NET', which is freely available, but unsupported. Alternatively, you will probably wish to consider Microsoft's Team System Test Edition, which provides a comprehensive range of testing tools for web applications and services. Beyond this, a range of commercial testing tools are available, either generic ones from companies such as IBM/Rational, or a more specialized ones which are typically produced by smaller companies.

### **15.1.3** Website development features in Visual Studio

So far in this chapter we have been reviewing some general differences in the feature set between Visual Web Developer Express Edition and the full version of Visual Studio. In many cases the differences are primarily about the full version providing a more professional environment, with different ways of achieving a similar result. In some cases, however, there are some important website development tools that are not available in Express Edition. Two of these are mobile web page development and localization. In the remainder of this chapter we will look at how both of these features can be implemented in the full edition of Visual Studio.

## **15.2** Mobile mark-up language evolution

In the first few years of the World Wide Web, we saw an evolution from static to dynamic content, but a more recent evolution has been from single format content to adaptive content. One of the most important aspects of an adaptive web application is the ability for the content delivery to be adapted to the capabilities of the client device, so that the same content can be delivered to a range of devices including desktop computers, PDAs, mobile phones, set top boxes, games consoles, etc. It is becoming increasingly necessary

to adapt presentation to these different device types, particularly as the penetration of mobile phones that are capable of web browsing has become widespread. The issue to address of course is that different devices have different presentation capabilities, for example you cannot run AJAX on every mobile device, because not all phones support JavaScript enabled web browsers, and mobile browsers do not all support the same mark-up languages. Another aspect of the limitations of mobile browsers is that many of them are not able to process XSL transformations, so are unable to render XML documents. Stylesheets are also problematical, because although many mobile browsers support Stylesheets they are not the same as the CSS used in desktop browsers. Despite these difficulties, it is possible to develop web applications that can adapt to different mobile device browsers by using appropriate tools. In this chapter we will look at the evolution and characteristics of the various types of mark-up that are supported by mobile browsers and see how Visual Web Developer makes it possible to write device-adaptive web applications.

We saw in Chapter 3 that HTML has evolved through several versions, eventually being superseded by XHTML. However these mark-up languages have been primarily oriented towards the desktop PC browser. In parallel with the evolution of desktop browser mark-up, there have been a number of different types of mark-up specifically designed for mobile devices. Early examples of this type of mark-up included cHTML (Compact HTML) for iMODE phones, used primarily in Japan but also in some parts of Europe, and HDML (Handheld Device Mark-up Language), which was designed as a more generic mark-up language by Unwired Planet. There was also a W3C note regarding 'HTML 4.0 Guidelines for Mobile Access' (Kamada, 1999). The approach of these mark-up languages was to provide a simplified subset of desktop browser mark-up more suited to the restrictions (display, processing power, memory, etc.) of mobile devices.

### **15.2.1 The Wireless Access Protocol (WAP) and the Wireless Mark-up Language (WML)**

In 1997, Nokia, Ericsson, Motorola and Unwired Planet cooperated to launch the Wireless Application Protocol (WAP) and provide an industry standard platform for mobile web access. The group became the WAP Forum in 1998, and expanded to include members from across the mobile communications industry. The forum had 500 members by 2001. In 2003 the WAP Forum became the Open Mobile Alliance (OMA), supporting more general standardization efforts within the mobile communications industry.

Part of the WAP platform was the Wireless Mark-up Language (WML). cHTML and WML had rather different approaches to supporting the mobile Web. cHTML was designed as a subset of HTML compatible with all its major versions (2.0, 3.2 and 4.0). In order to make sure that pages could be rendered on the simplified browsers available in iMODE phones, Stylesheets, tables, background colors and multiple fonts were excluded from the specification. One advantage of cHTML was that its pages could also be rendered on a standard desktop browser. Although a note to the W3C provided a suggested DOCTYPE for cHTML:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD Compact HTML 1.0 Draft//EN">
```



cHTML documents do not have to be well formed and the DOCTYPE has not been used in practice.

WML had a very different approach, with many of its concepts based on HDML, and was much more ambitious. It included many features that were intended to leverage the specific characteristics of the mobile phone platform, such as a 'deck of cards' architecture, which meant a single page could be downloaded that included multiple 'cards'. Each card provided a different view in the browser. Effectively this meant that a single download provided multiple web pages. WML also included its own scripting language and Stylesheets. Unlike cHTML, WML was not a subset of HTML and had its own mark-up syntax, though this syntax was XML compliant, meaning that it was well formed, and valid against the following DTD (for version 1.1):

```
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
```

The system doctype can also be directed to:

```
http://www.openmobilealliance.org/tech/DTD/wml_1_1.dtd
```

Here is an example of some WML mark-up. Note that while some of the mark-up is compatible with XHTML, other tags are not, in particular the 'wml' root element and the 'card' element, which identifies one of the cards in the current deck.

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
  <template>
    <do type="prev" label="Back"><prev/></do>
  </template>
  <card id="w" title="Insurance Claim Details">
    <p>Policy Number:
    <input type="text" name="policyNumber" value="" size="10"/>
    <br/>Amount claimed:
    <input type="text" name="amount" value="" size="10"/><br/>
    <select name="type">
      <option value="buildings">Buildings</option>
      <option value="contents">Contents</option>
    </select>
    <br/>Description of claim:<br/>
    <input type="text" name="description" value="" size="30"/><br/>
    ...etc
  </card>
</wml>
```

While cHTML was successful in the Japanese market, WAP did not find major market success in the early years. Slow mobile connections made it difficult to access the mobile web, and the restrictions of the mobile phone form factor also discouraged users. It was only with the introduction of the first WAP portal, Vodafone Live!, in 2001, which was supported by customized handsets that could automatically access the portal to make access easier, that WAP began to become more popular.

## 15.2.2 XHTML-Basic and XHTML-Mobile Profile

The experience of both cHTML and WML led to standardization efforts across the mobile communications industry, to provide a global mark-up for all mobile devices. The outcome of this was XHTML-Basic, a subset of XHTML that is

*'designed for Web clients that do not support the full set of XHTML features; for example, Web clients such as mobile phones, PDAs, pagers, and set top boxes.'*  
(McCarron *et al.*, 2007)

The first version (1.0) was defined in 2000 and version 1.1 in 2006. Like previous mobile mark-up, XHTML-Basic provides a simple set of tags that do not place undue burdens on the mobile device's display, processor or memory. Table 15.1 summarizes the elements that comprise XHTML-Basic.

There are a few elements in XHTML-Basic that we have not introduced in previous chapters, so we will briefly cover them here. Most of them are text formatting elements that in practice render the text either in italics or in a monospace font. The 'object' element is more complex, and of course the types of object that might be embedded into the page would be constrained by the capabilities of a given mobile device. Table 15.2 lists these elements and briefly describes their meanings.

XHTML-Basic is a generic approach to providing a subset of XHTML for a generic range of limited devices, but does not specify a particular type of device. In contrast, the mobile phone industry required a mark-up language that was intended specifically for mobile phones, and therefore did not need a language that was totally generic. Therefore the industry developed the specification for XHTML-MP (Mobile Profile), to produce a '*richer authoring language*' than XHTML-Basic (OMA, 2006). The OMA have adopted XHTML-MP as the migration path for WML, providing an updated version of WML (WML-2) for backward compatibility, which is otherwise superseded by XHTML Mobile Profile as the mark-up language used in WAP 2.0.

**TABLE 15.1** Elements in XHTML-Basic

Module	Element
Structure	body, head, html, title
Text	dfn, div, em, h1, h2, h3, h4, h5, h6, kbd, p, pre, q, samp, span, strong, var
Hypertext	a
List	dl, dt, dd, ol, ul, li
Basic forms	form, input, label, select, option, textarea
Basic tables	caption, table, td, th, tr
Image	img
Object	object, param
Meta information	meta
Link	link
Base	base

XHTML-MP is a superset of XHTML-Basic, which includes some additional elements and attributes from the full version of XHTML (Table 15.3). This table shows the additional elements and attributes in XHTML-MP version 1.2 (earlier versions (1.0 and 1.1) supported only some of these).

One important aspect of XHTML-MP is its support for the 'style' element and attribute, enabling Stylesheets to be applied. These are not, however, intended for use with standard CSS but enable the use of WAP CSS (WCSS), a special type of stylesheet definition that is defined in the WAP 2.0 specification. XHTML-MP also provides support for scripting using ECMA Script Mobile Profile, another initiative of the OMA. However it should be noted that an XHTML-MP browser may not provide support for all aspects of the specification. In addition, there are many other aspects to designing for the mobile phone format than simply using a particular mark-up (Passani, 2007).

**TABLE 15.2** Elements from XHTML-Basic not previously introduced

Element	Meaning
Dfn	Definition: surrounds the definition of a term
Kbd	Keyboard: describes characters to be typed in at the keyboard
Pre	Preformatted: maintains existing line feeds and spaces in the text
Q	Quotation: adds quotation marks
Samp	Example: describes text which is an example of something
Var	Variable: describes text that is being used as a variable
Object	A multi-media element that enables objects such as applets, images, plugins and other documents to be embedded in the page
Param	Parameter: a parameter value used with the object element to define parameters to the embedded object
Base	Base URL: enables a different URL to be used for relative references other than the one the page actually came from

**TABLE 15.3** Additional elements and attributes in XHTML-MP, not present in XHTML-Basic

Module	Element/Attributes
Forms	fieldset, optgroup
Lists	'start' attribute in ordered lists 'value' attribute in list items
Presentation	b, big, hr, i, small
Stylesheet	'style' element
Style attribute	'style' attribute

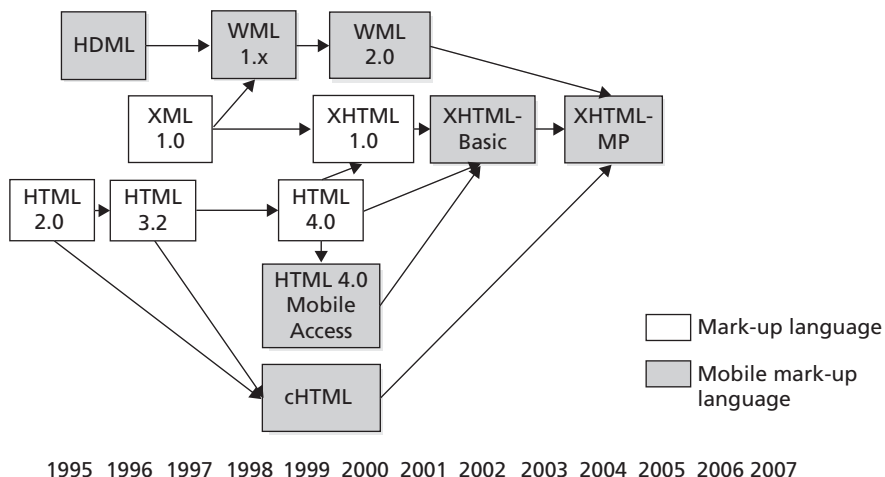
In XHTML-MP there is one element that we have not previously introduced, the 'optgroup', which is used with a 'select' tag. It is useful where a select list has a large number of entries that can be grouped in some way to make them easier to navigate.

The following example shows some XHTML-MP mark-up. Of course much of an XHTML-MP document will look exactly like a standard XHTML document. The main differences, however, are the DOCTYPE (this example is for XHTML-MP version 1.2) and, in this case, the use of a 'style' attribute that refers to a WAP CSS style:

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//OMA//DTD XHTML Mobile 1.2//EN"
"http://www.openmobilealliance.org/tech/DTD/xhtml-mobile12.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head>
<title>Welcome to WebHomeCover</title>
</head>
<body>
<p>
<div style="display: -wap-marquee">Welcome to the claims department at
WebHomeCover.com</div>
<br/>click&nbsp;
<a href="claimdetails.jsp">here</a>
&nbsp;to enter your claim
</p>
</body>
</html>
```

Figure 15.5 summarizes the relationships between the various mark-up languages that we have looked at so far in this chapter. It can be seen from this diagram that any future development of applications using mobile mark-up should be using XHTML-MP rather than any earlier standards, particularly as it has been given very strong support from the OMA.

**FIGURE 15.5** The evolution of mark-up languages for mobile devices (shaded boxes are types of mobile mark-up)



### 15.2.3 The .mobi Top Level Domain

With the move towards XHTML-MP as a standard mark-up for mobile device browsers, there has been an effort on behalf of part of the mobile communications industry to enable mobile web users to more easily identify websites that can be browsed using mobile devices. Thirteen mobile and Internet organizations formed the mTLD (.mobi Top Level Domain) group to promote the adoption of a new top level Internet domain with a '.mobi' extension. Any web application that uses this extension is expected to provide pages specifically for mobile devices, so that users of the mobile web know which sites are likely to work effectively on their mobile browsers. The mechanism for this is simply to encourage developers to create mobile mark-up using XHTML-MP. The guide document for .mobi developers states that

*'the response must be encoded in XHTML-MP unless the device accessing it is known to support an alternative choice of mark-up.'*

(Cremin and Rabin, 2006)

However there are some that object to the approach of having a specific domain name extension for mobile web applications, in particular Tim Berners-Lee, who objected strongly to the original proposals, and wrote:

*'The Web must operate independently of the hardware, software or network used to access it, of the perceived quality or appropriateness of the information on it, and of the culture, and language, and physical capabilities of those who access it.'*

(Berners-Lee, 2004)

Nevertheless, the '.mobi' top level domain was approved by ICANN in 2006 and seems to have found some popularity, at least according to the mTLD website (<http://pc.mtld.mobi/>).

## 15.3 Device adaptivity with Mobile Web Forms

In the previous section we saw that mark-up used with mobile devices has developed through various different syntaxes, some that were subsets of HTML and some XML syntaxes. In addition there are many different types of mobile device, and new models are being introduced to the market all the time. These devices vary in physical form factors such as screen size and resolution, colors supported, layout of control buttons, etc., as well as varying in the types of mobile browser that they support. This makes it difficult for the developer of a web application who wants to be able to support a wide range of mobile clients. Ideally, we would like to be able to provide customized versions of our web applications for each and every device capability, but this would be too time-consuming and difficult to maintain. Although there has been a gradual move towards XHTML-MP as the standard mark-up language for mobile browsers, there are still a wide range of mobile device browsers in use, supporting many of the mark-up languages we have introduced. This means that in order to support all types of mobile browsers, we need to write web applications that can generate dynamic content in a range of different mark-up languages. Doing this manually would be very arduous, but fortunately there is support for creating adaptive web pages within the full version of Visual Studio using Mobile Web Forms.

**NOTE**

Unfortunately the ability to create adaptive pages for the mobile web is not included in Visual Web Developer Express Edition.

### 15.3.1 Creating a mobile adaptive web form in Visual Studio

Visual Studio 2005 automatically included a ‘Mobile Web Form’ as one of its installed templates. Unfortunately, updates to the framework in Visual Studio 2008 have meant that this template now has to be installed separately. The following section explains how to install and create mobile web forms in Visual Studio 2008.

**NOTE**

When using the mobile web form template in Visual Studio 2008, you have to work in Source view, since Design view is not compatible with this template.

The first step in creating an adaptive web page in Visual Studio 2008 is to manually install the page templates available from Omar Khan’s blog site at Microsoft:

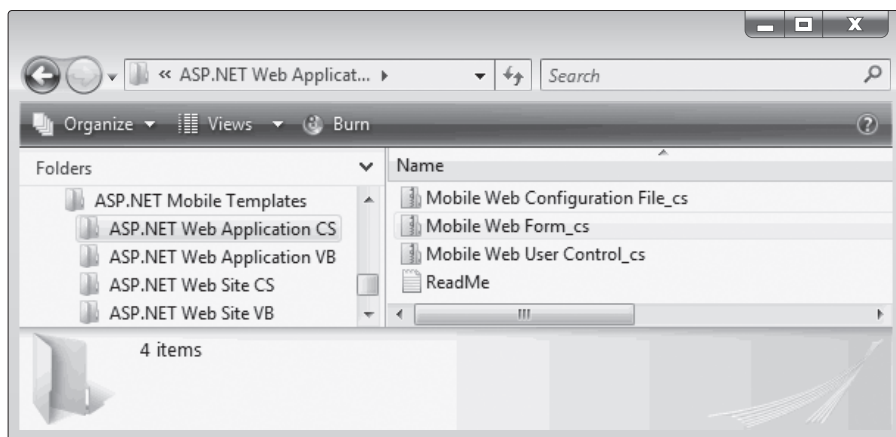
<http://blogs.msdn.com/webdevtools/archive/2007/09/17/tip-trick-asp-net-mobile-development-with-visual-studio-2008.aspx>

The first step is to download the template zip file from the blog site. Once you have unzipped this file into a folder, you will see that it consists of a main folder (‘ASP.NET Mobile Templates’) with four subfolders (Figure 15.6).

The zip files that appear in these folders have to be copied to specific folders. Information about how to do this is included in the ‘readme’ files, but basically all you have to do is:

- Copy all the zip files with filenames ending with “\_cs” in the ‘ASP.NET Web Application CS’ folder to:  
[My Documents]\Visual Studio 2008\Templates\ItemTemplates\Visual C#

**FIGURE 15.6** The subfolders in the Mobile Templates download



- Copy all the zip files with filenames ending with “\_vb” in the ‘ASP.NET Web Application VB’ folder to:

[My Documents]\Visual Studio 2008\Templates\ItemTemplates\Visual Basic

- Copy all the zip files with filenames ending with “\_cs” in the ‘ASP.NET Website CS’ folder, and all the zip files with filenames ending with “\_vb” ASP.NET Website VB’ folder, to:

[My Documents]\Visual Studio 2008\Templates\ItemTemplates\Visual Web Developer

You will need to restart your machine before Visual Studio will be able to properly integrate these templates. Once you have done so, it should now be possible to create a new ‘Mobile Web Form’ using the ‘Project’ → ‘Add Module’ dialog (Figure 15.7). In this example, the new file has been named ‘MobileForm.aspx’.

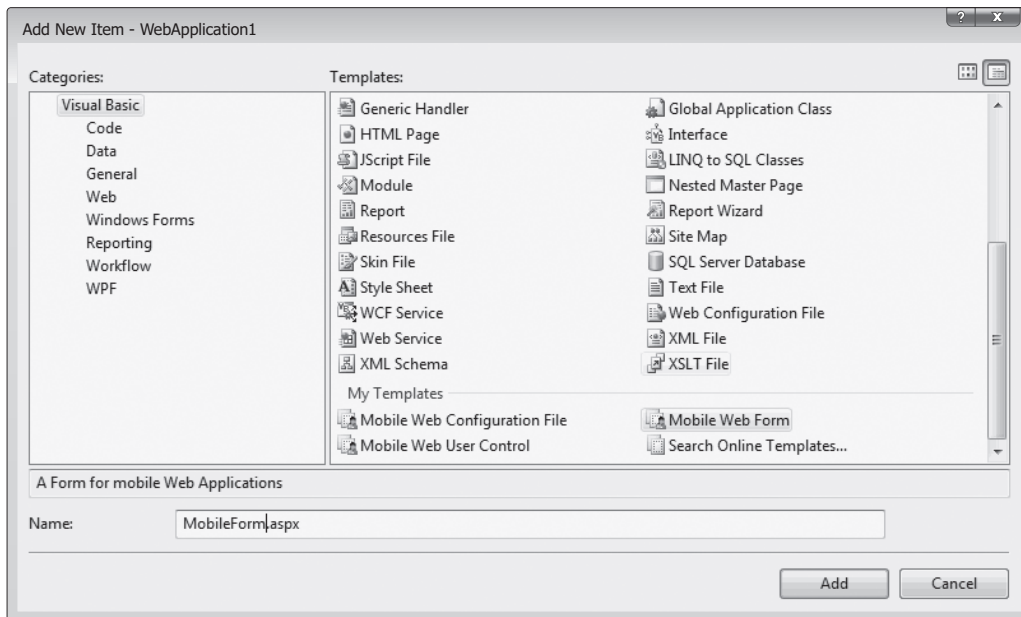


**NOTE**

For some reason the mobile templates do not appear in the Add New Item dialog if you choose the ‘File’ → ‘New File’ menu option, so you must use ‘Project’ → ‘Add Module’.

The generated code for the mobile web form that appears in Source view is somewhat different from that usually generated for a web form. Instead of the page including a standard DOCTYPE, such as XHTML 1.0, the page leaves the generation of the appropriate DOCTYPE to the framework, since different mobile devices will require different types of mark-up. The body of the page contain a ‘mobile:Form’ element, which will act as a container for special mobile controls.

**FIGURE 15.7** The mobile web form templates that appear in the Add New Item dialog if you select ‘Project’ → ‘Add Module’



```

<%@ Page Language="VB" AutoEventWireup="false"
Inherits="WebApplication1.MobileForm" Codebehind="MobileForm.aspx.vb"
%>
<%@ Register TagPrefix="mobile"
Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile"
%>

<html xmlns="http://www.w3.org/1999/xhtml" >
<body>
    <mobile:Form id="Form1" runat="server">

        </mobile:Form>
</body>
</html>

```

Once a mobile web form has been created, controls may be added to it from the 'Mobile Web Forms' group of controls in the Toolbox. This group of controls includes mobile versions of some familiar controls such as TextBoxes, Labels and validation controls. It also includes some special controls for mobile device development such as the 'PhoneCall' control (Figure 15.8).

Here is the source of a mobile web form with some simple controls added to it: a Label, a TextField and a Command. The Command acts in a similar way to how a Button would be used on a standard web page. However some mobile devices will map commands to soft keys on the device, rather than buttons on the screen. This means that the Command button (with the text 'Ring Number') may or may not appear as a visual button on the mobile device. Therefore we need to also set the 'SoftkeyLabel' property of this control, because this will be used if the device maps commands to keys on the mobile device keypad.

```

<%@ Page Language="C#" AutoEventWireup="true"
Inherits="MobileWebCSharp.MobileForm" Codebehind="MobileForm.aspx.cs"
%>
<%@ Register TagPrefix="mobile"
Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile"
%>

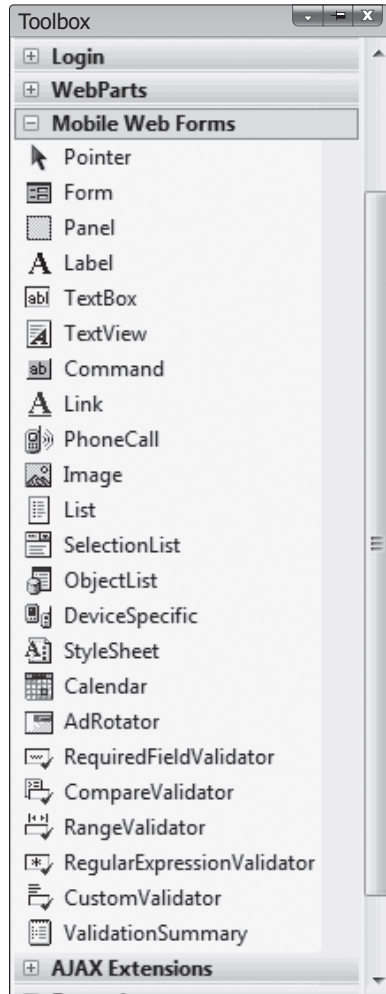
<html xmlns="http://www.w3.org/1999/xhtml" >
<body>
    <mobile:Form id="Form1" runat="server">
        <mobile:Label ID="Label1" Runat="server">Enter Phone Number
        </mobile:Label>
        <mobile:TextBox ID="TextBox1" Runat="server">
        </mobile:TextBox>
        <mobile:Command ID="Command1" Runat="server" SoftkeyLabel="Ring">
        Ring Number
        </mobile:Command>
    </mobile:Form>
</body>
</html>

```

Mobile web forms are flexible enough to generate pages for standard desktop browsers as well as mobile browsers. Figure 15.9a and b shows our mobile web form rendered on



**FIGURE 15.8** The Mobile Web Forms group of controls in the Toolbox



two different clients; a mobile device simulator and a standard desktop browser (Internet Explorer 7). Note that when the form is displayed in the desktop browser, the Command appears as a standard 'submit' button, but when it is displayed on the mobile device emulator, there is a soft key mapping, and no visible button on the screen.



**NOTE**

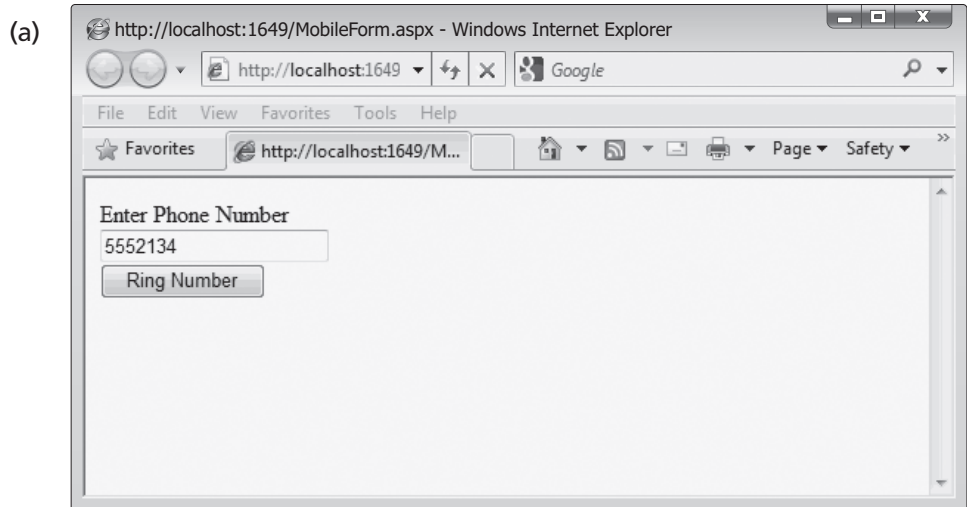
The soft key command will read 'edit' rather than 'ring' if the TextField has focus

It is interesting to compare the generated mark-up for these two types of browser. If you view the source of the page in the desktop browser, we can see that (as well as some JavaScript) part of the page is standard HTML, for example the rendering of the Command button:

```
<input name="Command1" type="submit" value="Ring Number"/>
```

**FIGURE 15.9**

A mobile web form generating different mark-up for mobile and desktop browsers. The image on the right is from the Openwave simulator. This simulator, which was used to create all the mobile browser screen images in this section, can be downloaded from the Openwave Developer Network ([http://developer.openwave.com/dvl/tools\\_and\\_sdk/phone\\_simulator/](http://developer.openwave.com/dvl/tools_and_sdk/phone_simulator/))



(b)



In contrast, the version of the page sent to the mobile device is marked up in WML:

```
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
'http://www.wapforum.org/DTD/wml_1.1.xml'>
<wml>
```

```

<head>
  <meta http-equiv="Cache-Control" content="max-age=0" />
</head>
<card>
<onevent type="onenterforward">
  <refresh>
    <setvar name="TextBox1" value="" />
  </refresh>
</onevent>
<do type="accept" label="Ring">
  <go href="/Mobile/MobileForm.aspx" method="post">
    <postfield name="__EVENTTARGET" value="Command1" />
    <postfield name="TextBox1" value="{$(TextBox1)}" />
  </go>
</do>
<p>Enter Phone Number<input name="TextBox1" />
</p>
</card>
</wml>

```

## 15.4 Integrating mobile web forms with a web application

We need some way of integrating mobile web forms mark-up into our existing web applications while continuing to support desktop browser clients. Although it is possible to write an application entirely using mobile web forms, this is rather limiting, because it means that desktop browsers will produce a very basic set of view components. If we want to maintain our support for the richness of desktop browsers in a web application, along with support for mobile devices, we will have to support two streams of web page generation, one for powerful full size browsers, supported by the dynamic page generation we already have in place, and another for other types of browser that we will support using mobile web forms. There are basically two approaches to this problem. One is to take the ‘.mobi’ domain path, and simply create two entirely separate web applications, one using standard web forms and the other using mobile web forms. Alternatively, we can try to support both types of client within the same application. In our example, we will address the second approach.

We can cater for different types of client by analyzing the ‘User-Agent’ header information sent within the browser’s HTTP request. In fact this is exactly what the .NET mobile controls do in order to identify different types of mobile device, but all we need to do is filter out the main desktop browsers. Table 15.5 shows the ‘User-Agent’ header sent to the server by seven of the major web browsers: Flock, Google Chrome, Internet Explorer, Mozilla Firefox, Safari (Windows version), Netscape Navigator and Opera (some aspects of these will vary from machine to machine).

We can see from Table 15.5 that six of the major browsers’ user agent strings contain the word ‘Mozilla’. Therefore it is quite easy to identify these browsers. The Opera user agent string includes the word ‘Opera’, but because there are different versions of Opera for different types of client (i.e. there are Opera Mini and Opera Mobile browsers for mobile devices, and a version for the Nintendo games console) just identifying the ‘Opera’ string will not be

TABLE 15.5

The 'User-Agent' header information sent to the server by seven of the major web browsers

Browser	Version	'User-Agent' header
Flock	2	Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.0.5) Gecko/2008121620 Firefox/3.0.5 Flock/2.0.3
Google Chrome	1	Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US) AppleWebKit/525.19 (KHTML, like Gecko) Chrome/1.0.154.65 Safari/525.19
Internet Explorer	8	Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; .NET CLR 1.1.4322; .NET CLR 2.0.50727; .NET CLR 3.0.04506.648; .NET CLR 3.5.21022; InfoPath.1; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729; OfficeLiveConnector.1.3; OfficeLivePatch.0.0)
Mozilla Firefox	3	Mozilla/5.0 (Windows; U; Windows NT 5.1; en-GB; rv:1.9.0.9) Gecko/2009040821 Firefox/3.0.9 (.NET CLR 3.5.30729)
Safari	3	Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US) AppleWebKit/525.27.1 (KHTML, like Gecko) Version/3.2.1 Safari/525.27.1
Netscape Navigator	9 (final version)	Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.12) Gecko/20080219 Firefox/2.0.0.12 Navigator/9.0.0.6
Opera	9	Opera/9.64 (Windows NT 5.1; U; en) Presto/2.1.1

enough to specify an Opera desktop browser. Therefore we would need to also check for the 'Windows' string to ensure that we can differentiate the different versions of Opera.

The following 'if' statement can be used in server-side code to filter out the five major desktop browsers. Using just these substrings ('Mozilla', or 'Opera' and 'Windows') means that the code should be resilient against changes in details such as version numbers:

```
if(userAgent.startsWith("Mozilla") || (userAgent.contains("Opera") &&
userAgent.contains("Windows")))
```

Browsers that match these criteria can be sent pages generated using standard XHTML. In contrast, the 'User-Agent' header for other browsers will be very different, for example the OpenWave simulator Version 7 sends the header string:

```
OPWV-SDK UP.Browser/7.0.2.3.119 (GUI) MMP/2.0 Push/PO
```

Browsers such as this can be handled separately and pages can be generated from mobile web forms library. Of course any desktop browsers that are not picked up by this 'if'

statement will be sent pages generated by mobile web forms. However they will still work. An associated problem is that some mobile devices will send user agent headers that include the word 'Mozilla'. For example the Nokia 6630 cell phone will send the following header:

```
Mozilla/4.0 (compatible; MSIE 5.0; Series60/2.8 Nokia6630/4.06.0 Profile/MIDP-2.0  
Configuration/CLDC-1.1)
```

One way of dealing with this is to look for the 'Windows' string with Mozilla browsers, but this will exclude non-Windows desktop browsers such as Safari running on the Mac. Alternatively we could exclude user agents that contain 'Nokia', though this would not deal with other mobile browsers that might include the 'Mozilla' string but are not Nokia phones. In the end, there will probably be one or two browsers that slip through the cracks, but we will cater correctly for the vast majority of clients.

In order to integrate mobile web forms with our existing pages and processes, we would need to provide two paths through the web application, one that uses the existing web forms and one that uses the mobile version. To branch to different paths we would start from a common default start page which would identify the browser category and forward to the appropriate home page.

This page would not render anything on the client browser. Instead, it will contain some code in its page load event handler for forwarding the client to the appropriate page, depending on the type of browser they are using. The following page load event handler for the start page ('Start.aspx'), contains the code to check the 'User-Agent' header (using the 'UserAgent' property of the 'Request' object) and then sets the value of a Boolean variable called 'isMobile' to either *true* or *false*, depending on the value of the user agent header. Depending on the contents of that string, control will be passed to either a mobile web form or a standard web form.

## VB Code

```
Partial Class Start  
    Inherits System.Web.UI.MobileControls.MobilePage  
  
    Protected Sub Page_Load(ByVal sender As Object,  
        ByVal e As System.EventArgs) Handles Me.Load  
        Dim UserAgent As String = Request.UserAgent  
        Dim isMobile As Boolean = True  
        If UserAgent.StartsWith("Mozilla") Or (UserAgent.Contains("Opera")  
            And UserAgent.Contains("Windows")) Then  
            isMobile = False  
        End If  
        If isMobile Then  
            Server.Transfer("mobile web form")  
        Else  
            Server.Transfer("standard web form")  
        End If  
    End Sub  
End Class
```

## 15.4.1 A mobile web form

As a simple example of a mobile web form that has some useful functionality, we will implement a simple form that will display details of an insurance policy, when the user enters a valid policy number. The mobile web form is very simple. It consists of a `TextBox` into which the user can enter a policy number, a command button (which will by default invoke the page load event) and a blank label that will contain the details of the policy when read from the database.

```
<%@ Page Language="C#" AutoEventWireup="true"
Inherits="MobileWebCSharp.Default2"
Codebehind="MobilePolicyForm.aspx.cs" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls"
Assembly="System.Web.Mobile" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<body>
<mobile:Form id="Form1" runat="server">
<mobile:Label ID="Label1" Runat="server">Enter a policy number
</mobile:Label>
<mobile:TextBox ID="PolicyTextBox" Runat="server">
</mobile:TextBox>
<mobile:Command ID="Command1" Runat="server">Command
</mobile:Command>
<mobile:Label ID="Result" Runat="server"></mobile:Label>
</mobile:Form>
</body>
</html>
```

In the code behind, we need to access the `PolicyDAO` to read a policy from the database, so we need the following delegation method in the `ModelFacade`:

```
public PolicyDTO getPolicyByNumber(int number)
{
    PolicyDAO myDAO = new PolicyDAO();
    return myDAO.getPolicyByNumber(number);
}
```

The code behind is relatively simple, with everything taking place in the page load event handler. If the `TextBox` contains anything, we try to parse the value in the box into an integer. If this does not throw an exception the value is then used to try to retrieve a policy from the database. If a matching policy is found, policy details are displayed (only the type and premium in this simple example):

```
public partial class Default2 :
System.Web.UI.MobileControls.MobilePage
{
    protected void Page_Load(object sender, EventArgs e)
    {
        if (PolicyTextBox.Text != "")
```

```

{
ModelFacade facade = new ModelFacade();
try
{
int policyNum = Int32.Parse(PolicyTextBox.Text);
PolicyDTO policy = facade.getPolicyByNumber(policyNum);
if (policy != null)
{
Result.Text = "Policy type: " + policy.policyType +
", Premium: " + policy.annualPremium;
}
else
{
Result.Text = "No matching policy";
}
}
catch (Exception ex)
{
Result.Text = "Not a valid policy number";
}
}
}
}

```

Figure 15.10 shows the mobile form after a valid policy number has been entered.

**FIGURE 15.10** The mobile web form displaying details of a policy



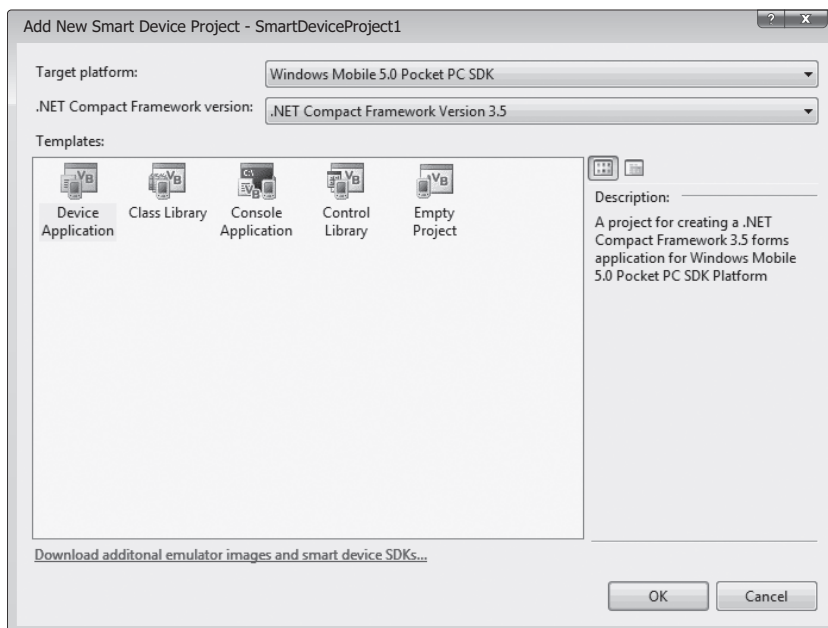
## 15.5 Alternative approaches to mobile systems

In this chapter we have taken a particular approach to adaptivity, namely utilizing the features of mobile web forms to generate different mark-up for different client devices. Another approach that is sometime used with XHTML data (as opposed to ASP.NET web forms) is to use XML transformations to generate different mark-up for different types of client. Yet another option to adaptivity would be simply to rely on the mobile browser to transform standard web content for the mobile device. This is increasingly possible with the continuing development of the Opera Mini and Opera Mobile Browsers. Opera Mini is a generic mobile browser that is able to render standard web pages on a small screen by converting the original pages. Opera Mobile is a more powerful browser, which is targeted to the specific mobile device on which it is installed, so there are different versions of the browser available for a range of mobile phones. Opera Mobile is a more fully featured mobile browser than Opera Mini, and includes support for JavaScript, making it possible to develop mobile AJAX applications.

### 15.5.1 Non-adaptive mobile systems

Instead of creating mobile web forms that adapt to mobile devices, another significant .NET development theme is to create mobile web pages for specific Windows mobile devices using target platforms such as Windows Mobile. In Visual Studio, we develop this type of software by selecting 'File' → 'New Project' then selecting a language (C#, VB, . . .) and finally selecting 'Smart Device' project. Within this project, you can choose your target platform (Figure 15.11).

**FIGURE 15.11** Selecting a target device platform for a Smart Device Project





When working with a smart device projects, the Toolbox will contain specialized controls, and design view will emulate the target device (Figure 15.12).

### EXERCISE 15.1

Take the contents details form from the 'get insurance quote' use case and write a mobile version of it using a mobile web form. Note that when using the mobile controls you will need to replace the check box and radio buttons with Selection List controls. Test the page renders correctly by using a suitable mobile phone simulator.

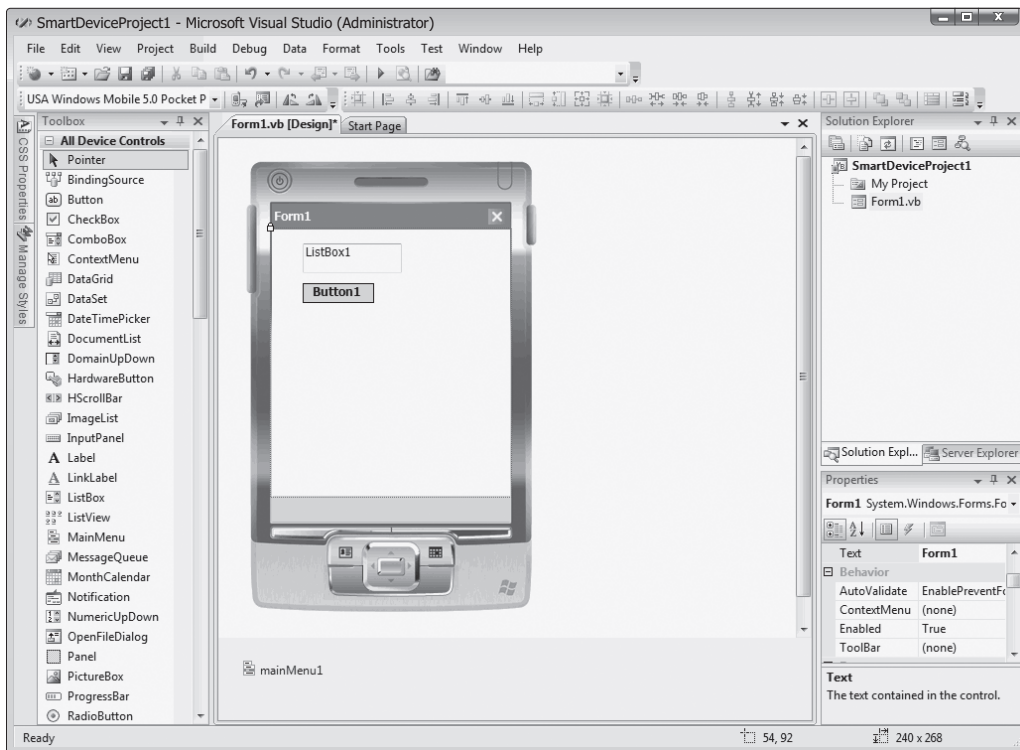
### EXERCISE 15.2

Write a mobile version of the buildings details form from the 'get insurance quote' use case.

### EXERCISE 15.3

Implement a set of mobile web forms that enable a mobile user to enter an account number and view the related policy holder details.

**FIGURE 15.12** Specialized Toolbar options and Design view in a smart device project



## 15.6 Localization

Localization means providing the same web application in different languages. It is also known as ‘internationalization’, with ‘i18n’ as a commonly used shorthand because there are 18 letters between the ‘i’ and the ‘n’. However Microsoft tends to use the term ‘localization’, so we will use this term in this chapter.

Visual Web Developer makes it relatively easy to provide dynamic web pages in different languages because you don’t need to replicate all the pages for each language you want to support. Instead, resource files are provided for each language and used by the running web application based on the preference settings of the requesting browser.

### 15.6.1 Localized resource files

Resource files can be created from within Design view (*not* Source view) by opening a web form in an editing window. Once the web form is open, select any of the controls on the page and then select ‘Generate Local Resource’ from the ‘Tools’ menu. This will create a resource file in the App\_LocalResources folder of the website. Each web form will have its own set of resource files, so the file name will be based on the file name of the web form, with an additional ‘resx’ extension. For example, if you generate a resource file from a web form called ‘MyForm.aspx’, then the resource file will be called ‘MyForm.aspx.resx’.

To successfully internationalize a page, all the text (with the exception of the title element) needs to be in controls rather than HTML elements. As an example, here is a rewritten version of ‘ClaimForm1.aspx’, which we have seen a number of times before in this book. However you will note that in this version the headings and page text appear in Label controls rather than in HTML elements. Note also that the labels have been given meaningful names. This is useful when adding internationalization as it makes the labels easier to identify. You will also note that we are not using the master page in this example. This is because the master page, too, would need to be rewritten to contain only controls rather than HTML elements, and would need its own internationalization. To keep this example simple, we have not included the master page.


```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head runat="server">
    <title>Make a Claim</title>
  </head>
  <body>
    <form id="form1" runat="server">
      <h1>
        <asp:Label ID="HeadingLabel" runat="server"
          Text="WebHomeCover insurance claim form"></asp:Label>
      </h1>
      <p>
        <asp:Label ID="MessageLabel" runat="server"
          Text="Please enter your policy number in the text box below,
          and also select the type of insurance claim you wish to make.
```

```

    Then press the 'Submit' button">
</asp:Label>
</p>
<div>
  <asp:Label ID="PolicyNumberLabel" runat="server"
    Text="Policy Number: ">
  </asp:Label>
  <asp:TextBox ID="PolicyNumber" runat="server"></asp:TextBox>
  <br />
  <asp:RadioButtonList ID="PolicyType" runat="server">
  <asp:ListItem Value="contents">Contents Insurance Claim
  </asp:ListItem>
  <asp:ListItem Value="buildings">Buildings Insurance Claim
  </asp:ListItem>
  </asp:RadioButtonList>
  <br />
  <asp:Button ID="SubmitButton" runat="server" Text="Submit" />
</div>
</form>
</body>
</html>

```

When you generate the resource file, then all the controls in the form will also be automatically modified to include an additional 'meta:resourcekey' attribute. Each of these attributes is given a value that identifies the control with which it is associated. Here is the modified 'ClaimForm1.aspx' page with the generated 'meta:resourcekey' attributes highlighted in bold.

 <b>NOTE</b>	If you re-generate the resource keys more than once you may find the names get out of synch with the names of the controls. You can manually edit the resourceKey names if this happens.
---	--

```

<%@ Page Language="VB" AutoEventWireup="false"
CodeFile="ClaimForm1.aspx.vb" Inherits="ClaimForm1"
  Culture="auto" meta:resourcekey="PageResource1" UICulture="auto"
%>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head runat="server">
    <title>Make a Claim</title>
  </head>
  <body>
    <form id="form1" runat="server">
      <h1>
        <asp:Label ID="HeadingLabel" runat="server"
          Text="WebHomeCover insurance claim form"

```

```

    meta:resourcekey="HeadingLabelResource1">
</asp:Label>
</h1>
<div>
  <p>
    <asp:Label ID="MessageLabel" runat="server"
      Text="Please enter your policy number in the text box below,
      and also select the type of insurance claim you wish to make.
      Then press the 'Submit' button"
    meta:resourcekey="MessageLabelResource1">
    </asp:Label>
  </p>
  <asp:Label ID="PolicyNumberLabel" runat="server"
    Text="Policy Number: "
    meta:resourcekey="PolicyNumberLabelResource1">
  </asp:Label>
  <asp:TextBox ID="PolicyNumber" runat="server"
    meta:resourcekey="PolicyNumberResource1">
  </asp:TextBox>
  <br />
  <asp:RadioButtonList ID="PolicyType" runat="server"
    meta:resourcekey="PolicyTypeResource1">
    <asp:ListItem Value="contents"
      meta:resourcekey="ListItemResource1">
      Contents Insurance Claim
    </asp:ListItem>
    <asp:ListItem Value="buildings"
      meta:resourcekey="ListItemResource2">
      Buildings Insurance Claim
    </asp:ListItem>
  </asp:RadioButtonList>
  <br />
  <asp:Button ID="SubmitButton" runat="server" Text="Submit"
    meta:resourcekey="SubmitButtonResource1"/>
  </div>
</form>
</body>
</html>

```

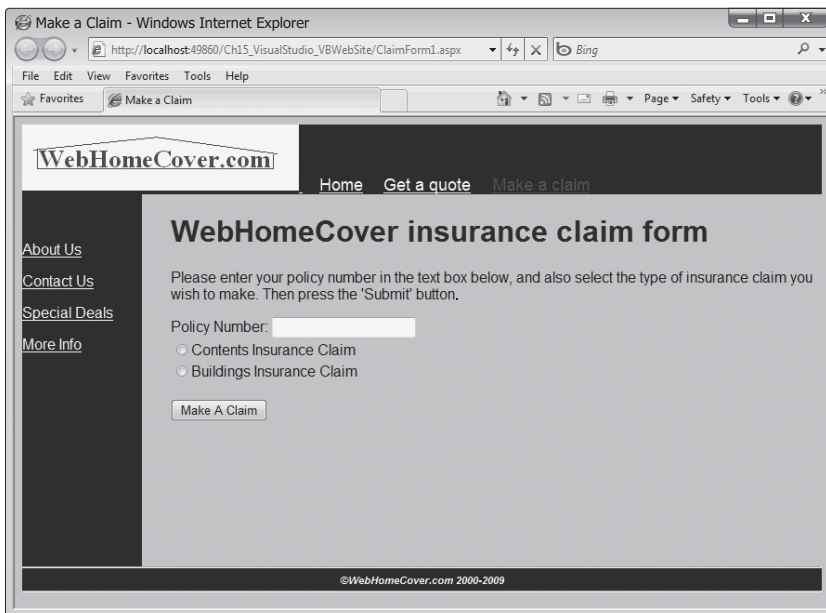
You will notice that the various ‘Text’ and ‘Value’ attributes (in English) are still included in the elements for the controls. However these values can now be overridden by the contents of the resource file. If you open up the generated ‘ClaimForm1.aspx.resx’ file in an editor window, you will see that it contains a series of name → value pairs (Figure 15.13). Where there are values already specified for some of these items, such as the text in the first label, then those values are listed in the file. Where we have not provided values, such as the ToolTip entries, the values are blank.

To show that this file can override the values that appear in the web form, try changing the ‘SubmitButtonResource1.Text’ value from ‘Submit’ to ‘Make a Claim’. If you do this and then view the page in a browser, you should see that the button text has been changed to the value contained in the resource file (Figure 15.14).

**FIGURE 15.13** The contents of the default (English language) resource file for the first claim form page

Name	Value	Comment
HeadingLabelResource1.Text	WebHomeCover insurance claim form	
HeadingLabelResource1.ToolTip		
ListItemResource1.Text	Contents Insurance Claim	
ListItemResource1.Value	contents	
ListItemResource2.Text	Buildings Insurance Claim	
ListItemResource2.Value	buildings	
MessageLabelResource1.Text	Please enter your policy number in the text box below, and also select the type of insurance claim you wish to make. Then press the 'Submit' button	
MessageLabelResource1.ToolTip		
PageResource1.Title	Make a Claim	
PolicyNumberLabelResource1.Text	Policy Number:	
PolicyNumberLabelResource1.ToolTip		
PolicyNumberResource1.Text		
PolicyNumberResource1.ToolTip		
PolicyTypeResource1.ToolTip		
SubmitButtonResource1.Text	Submit	
SubmitButtonResource1.ToolTip		
*		


**FIGURE 15.14** The web form with the button text being set from the resource file



The file that we have created so far will be the default resource file. However in order to cater for different languages we need to add other resource files in different languages. To specify the language of a resource file, we incorporate the ISO Language Code in to the resource file name, between the web form file name and the '.resx' extension. For example we might want to provide the web application in French as well as

our default language, which is English. The name of the second resource file would therefore be:

ClaimForm1.aspx.fr.resx

	<p><b>NOTE</b></p> <p>You do not need to supply the complete language codes. 'fr' for example would match all the cultural locales that begin with 'fr' (France, Canada, Belgium, etc.).</p>
---	--

To create additional resource files, simply copy the default resource file in the App-LocalResources folder, and paste the copy back into the same folder, then rename it to suit the required language.

The content of the copied file would be the same set of keys but with values written in the chosen language. Figure 15.15 shows the French language resource file, with some dubious translations, courtesy of Babel Fish (<http://babelfish.yahoo.com>).

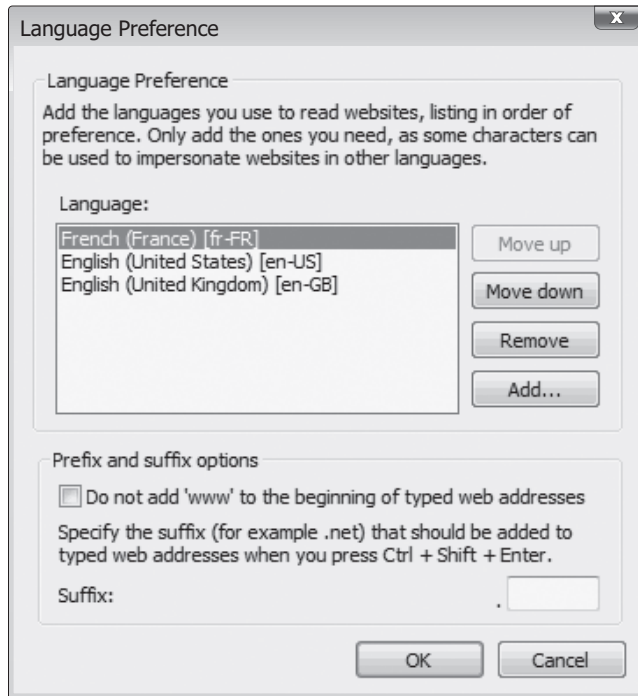
## 15.6.2 Setting up the browser to test localized pages

To test whether your localized resources are working, you have to change the language preferences in your browser. In Internet Explorer you select 'Tools' from the menu bar, then 'Internet Options'. In the resulting dialog you press the 'Languages' button, which will show you the dialog in Figure 15.16. In this dialog you can add new languages (with the 'add' button) and change their order of preference. In Figure 15.16 we have added

**FIGURE 15.15** The French language resource file, 'ClaimForm1.aspx.fr.resx'

Name	Value	Comment
HeadingLabelResource1.Text	Forme de déclaration de sinistre de WebHomeCover	
HeadingLabelResource1.ToolTip		
ListItemResource1.Text	Déclaration de sinistre de contenu	
ListItemResource1.Value	contenu	
ListItemResource2.Text	Déclaration de sinistre de bâtiments	
ListItemResource2.Value	bâtiments	
MessageLabelResource1.Text	Veillez introduire votre nombre de politique dans la boîte des textes ci-dessous, et choisissez également le type de déclaration de sinistre que vous souhaitez introduire. Pressez alors le ' ; Submit' ; bouton	
MessageLabelResource1.ToolTip		
PageResource1.Title	Introduisez une réclamation	
PolicyNumberLabelResource1.Text	Nombre de politique :	
PolicyNumberLabelResource1.ToolTip		
PolicyNumberResource1.Text		
PolicyNumberResource1.ToolTip		
PolicyTypeResource1.ToolTip		
SubmitButtonResource1.Text	Introduisez une réclamation	
SubmitButtonResource1.ToolTip		
*		

**FIGURE 15.16** The language preferences dialog in Internet Explorer 7



French and moved it to the top of our list of preferences. The browser will now include French as its preferred language when it sends HTTP requests.



**NOTE**

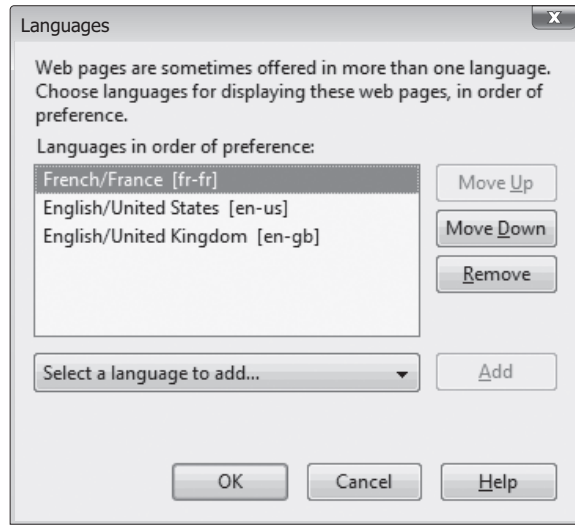
You have to restart the browser for changes in language settings to take effect.

Setting up the language preferences in Mozilla Firefox 3 is very similar. Again we choose 'Tools' from the menu bar, then select 'Options' and press the 'Choose' button next to the 'Languages' label on the dialog. Figure 15.17 shows the Firefox language dialog. In Opera, select 'Tools', then 'Preferences', then press the 'Details' button next to the 'Language' drop down list.

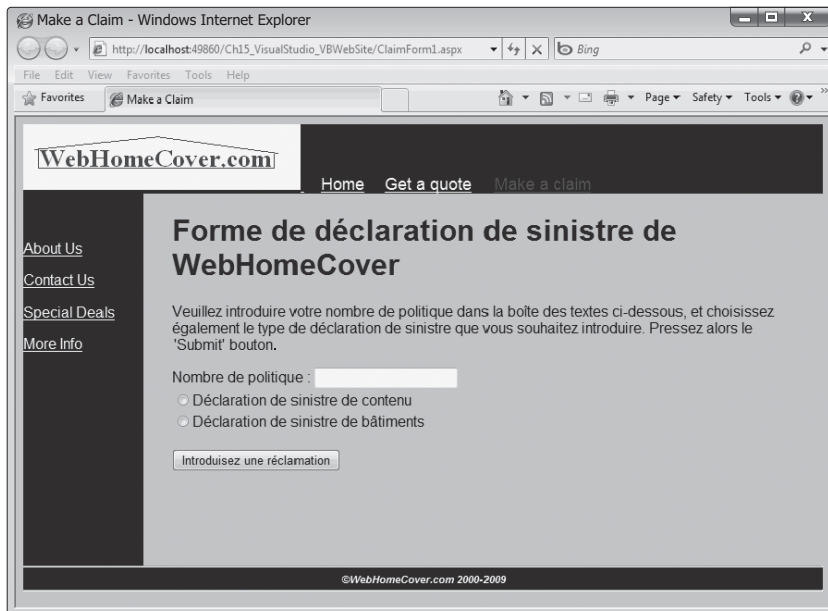
If we change the language settings and view our claim form in a browser, we will see the text properties of the controls appear in French (Figure 15.18).

The localization approach we have described in this chapter is known as 'implicit localization'. There is also an approach known as 'explicit localization' which uses global resource files rather than local ones. For more information on localization, see the article, 'Walkthrough: Using Resources for Localization with ASP.NET', at <http://msdn.microsoft.com/en-us/library/fw69ke6f.aspx>.

**FIGURE 15.17** The language preferences dialog in Mozilla Firefox 3



**FIGURE 15.18** A localized web application with the browser language preference set to French



## 15.7 Additional features of ASP.NET

One of the exciting aspects of ASP.NET is that it continues to be developed actively by Microsoft. There are always new innovations being released for you to learn about. This section reviews some of the enhancements that have just been made available at the time of writing. Two major new features of ASP.NET which are currently in beta testing are the



MVC (model-view-controller) and Dynamic Data frameworks, which are summarized in the following paragraphs.

MVC is a classic pattern for use in interactive applications. It separates the code into three main classes or packages, a model which encapsulates the persistent state, a view which provides an abstract view of a window displaying this state, and the controller which processes user input and directs it, as appropriate, to the model or view for action. The MVC approach is said to promote a clean separation of concerns, thereby enhancing maintainability and testability. The ASP.NET MVC framework replaces the post-back event handling mechanism currently in use, and instead to a single Controller class. It also includes a facility to map URLs directly to MVC actions. Existing ASP.NET web forms are treated as MVC style views, so that existing pages can be incorporated unchanged within the framework.

ASP.NET Dynamic Data is designed to simplify the development of data-driven applications by generating automatically the code and mark-up needed to display SQL relational database fields and records, rather than requiring you to produce it manually. As you have recognized, developing pages using the DetailsView, FormView, GridView and ListView controls can sometimes be somewhat repetitive, which is why the Dynamic Data framework has been created to boost web application developers' productivity. Of course, code generation does not suit every application, or every application developer, and its use is of course optional. In many cases, however, your web applications will be able to take advantage of this facility. For example, if a certain database field is defined as char(20) and non-nullable, then the Dynamic Data framework can automatically generate appropriate mark-up for web forms displaying and editing this item on your web pages, and this frees up your time to concentrate on other more challenging aspects of your website.

Finally, one of the most significant features of the recent release of ASP.NET was the inclusion of Silverlight version 2. It is claimed by Microsoft that Silverlight '*powers rich application experiences and delivers high quality, interactive video across the web and mobile devices through the most powerful runtime available on the web.*' That is to say, it is a client-side technology delivery rather than more powerful interaction and animation than, say, AJAX. This is possible because to view web pages using Silverlight, users must first download the Silverlight run-time, which is essentially a version of the .NET runtime. Thus, with Silverlight client-side interactions and animations can be programmed using standard .NET languages and libraries. This is in many ways a more satisfactory experience for the developer than having to switch to JavaScript code for client-side scripting. On the other hand, there are powerful competitors on the client, notably Flash and the AIR runtime from Adobe, JavaFX from Sun, and, of course, JavaScript itself, which has received a boost from Google's Chrome browser which includes a performance-enhanced version of this standard scripting language. The focus of *Dynamic Web Application Development using ASP.NET*, moreover, has been database-driven websites and so it was decided not to cover Silverlight development, itself the topic of a number of other books in its own right, in this text.

## Self Study Questions

1. In the Express Edition of Visual Web Developer, we have access to the ASP.NET Website Administration Tool. What is the alternative tool in the full edition of Visual Studio?
2. What is the name of a popular third party automated testing tool for Microsoft .NET?
3. Which early example of mark-up for mobile devices was targeted at iMODE phones?
4. Which mobile mark-up is associated with the WAP platform?
5. Which type of mark-up has been adopted by the Open Mobile Alliance, and is required for '.mobi' websites?
6. What are the control buttons on a mobile phone known as, when mapping these buttons to mobile web form controls?
7. What is the logic behind the term 'i18n' to mean internationalization?
8. What is the preferred term for internationalization when using .NET?
9. What must you do after changing your language preferences in the browser, in order for these changes to take effect?
10. Which Microsoft technology is similar in intent to Flash?

## Exercises

For these exercises you can either continue to use French, or choose some other example localized language.

- 15.4 Modify the WebHomeCover master page so that all its text elements are replaced by controls. Then provide localized text resource files for the master page.
- 15.5 Convert the localized 'ClaimForm1.aspx' page from this chapter to use the localized master page.
- 15.6 Localize the other two web forms in the insurance claim web flow.

### SUMMARY

In this chapter we reviewed a number of features of Visual Web Developer that are only available in the full version of the product which is shipped as part of Visual Studio. We looked at a number of differences in the way the tool provides services such as access to database administration tools, and website publishing. We looked at two particular areas in detail, where support is not provided in the express edition of the product, namely mobile web forms and localization. We concluded the chapter by acknowledging current developments in .NET including the MVC and Dynamic Data Frameworks, and Silverlight.

## References and further reading

- ASP.NET Dynamic Data*, Scott Guthrie, Microsoft 2007,  
<http://weblogs.asp.net/scottgu/archive/2007/12/14/new-asp-net-dynamic-data-support.aspx>  
*ASP.NET Dynamic Data* (official website), <http://www.asp.net/dynamicdata/>  
*ASP.NET MVC Framework*, Scott Guthrie, Microsoft 2007,  
<http://weblogs.asp.net/scottgu/archive/2007/10/14/asp-net-mvc-framework.aspx>  
*ASP.NET MVC Framework*, (official website), <http://www.asp.net/mvc/>  
*How do we write test automation for ASP.NET*, ASP.NET QA Team, 2008  
(<http://weblogs.asp.net/asptest/archive/2008/09/25/how-do-we-write-test-automation-for-asp-net.aspx>)
- Berners-Lee, T. 2004. *New Top Level Domains .mobi and .xxx Considered Harmful*  
<http://www.w3.org/DesignIssues/TLID>
- Cremin, R. and Rabin, J. 2006. dotMobi Switch On! Web Developer Guide  
[http://dev.mobi/files/dotmobi\\_Switch\\_On\\_Web\\_Developer\\_Guide\\_1.0\\_External\\_Draft.html](http://dev.mobi/files/dotmobi_Switch_On_Web_Developer_Guide_1.0_External_Draft.html)
- Kamada, T., Asada, T., Ishikawa, M. and Matsui, S. 1999. *HTML 4.0 Guidelines for Mobile Access*,  
<http://www.w3.org/TR/NOTE-html40-mobile>
- McCarron, S., Ishikawa, M., Baker, M., Ishikawa, M., Matsui, S., Stark, P., Wugofski, T. and Yamakami, T. 2007. *XHTML™ Basic 1.1 W3C Working Draft*  
<http://www.w3.org/TR/xhtml-basic>
- Microsoft Team System 2008 Test Edition*, <http://msdn.microsoft.com/en-gb/vsts2008/test/default.aspx>
- OMA. 2006. *XHTML Mobile Profile Approved Version 1.1* [http://www.openmobilealliance.org/release\\_program/docs/browsing/v2\\_2-20061020-a/oma-wap-xhtmlmp-v1\\_1-20061020-a.pdf](http://www.openmobilealliance.org/release_program/docs/browsing/v2_2-20061020-a/oma-wap-xhtmlmp-v1_1-20061020-a.pdf)
- Passani, L. 2007. *Global Authoring Practices for the Mobile Web*. <http://www.passani.it/gap/>
- Silverlight 2 in Action*, Chad Campbell and John Stockton, Manning 2008.
- Walkthrough: Using Resources for Localization with ASP.NET*  
<http://msdn.microsoft.com/en-us/library/fw69ke6f.aspx>

