XAML IN A NUTSHELL

A Desktop Quick Reference

O'REILLY®

Lori A. MacVittie

XAML IN A NUTSHELL

"Strong code examples and an efficient, conversational style take the tedium out of learning XAML and make the subject understandable—even interesting."

—Bradley F. Shimmin, Business Analyst, CMP Media LLC

The introduction of Windows Vista heralds a new presentation subsystem and a fresh new declarative language for building user interfaces. From the programmer to the interface designer, XAML offers a wealth of new controls and elements with exciting capabilities, including animation and rendering of 3-D graphics. While XAML removes the requirement for interface designers to write code, it also provides the programmer with the option to create, interact with, and manipulate XAML elements within one of several .NET languages.

This book covers everything necessary to begin designing user interfaces and .NET applications that take advantage of the Windows Presentation Foundation. Prerequisites such as Microsoft's new unified build system, MSBuild, and core XAML constructs and syntax—including shortcuts—are all presented with plenty of examples to get you started.

XAML in a Nutshell digs even deeper into syntax rules and attributes for all XAML elements in a series of quick-reference chapters. It divides XAML elements into logical categories of controls, shapes and geometry, layout, animations, and transformations for easy reference.

Lori MacVittie is currently a senior technology editor with *Network Computing Magazine*. In past lives, she has been a software developer, a network administrator, and an enterprise architect specializing in web-based technologies. Through the course of her career, she has nearly coded her way through the alphabet, starting with Apple BASIC, hitting "L" for LISP while consulting for Autodesk, and is currently on the letter "Y". Lori holds an M.S. in Computer Science from Nova Southeastern University and lives with her husband and children in the technological mecca of the Midwest, Green Bay, Wisconsin. She is also the coauthor of *Object-Oriented Programming: A New Way of Thinking* (CBM Books).

O'REILLY® www.oreilly.com

US $29.99 CAN $41.99
ISBN-10: 0-596-52673-3
ISBN-13: 978-0-596-52673-3

9 780596 526733 52999

XAML

IN A NUTSHELL

Other resources from O'Reilly

Related titles

- Programming Windows Presentation Foundation
- Programming C#
- C# Cookbook™
- Programming Visual Basic 2005
- Visual Basic 2005 in a Nutshell
- Programming ASP.NET
- ASP.NET 2.0 Cookbook™
- XML in a Nutshell
- HTML & XHTML: The Definitive Guide

oreilly.com

oreilly.com is more than a complete catalog of O'Reilly books. You'll also find links to news, events, articles, weblogs, sample chapters, and code examples.

oreillynet.com is the essential portal for developers interested in open and emerging technologies, including new platforms, programming languages, and operating systems.

Conferences

O'Reilly brings diverse innovators together to nurture the ideas that spark revolutionary industries. We specialize in documenting the latest tools and systems, translating the innovator's knowledge into useful skills for those in the trenches. Visit *conferences.oreilly.com* for our upcoming events.

Safari Bookshelf (*safari.oreilly.com*) is the premier online reference library for programmers and IT professionals. Conduct searches across more than 1,000 books. Subscribers can zero in on answers to time-critical questions in a matter of seconds. Read the books on your Bookshelf from cover to cover or simply flip to the page you need. Try it today for free.

XAML

IN A NUTSHELL

Lori A. MacVittie

O'REILLY®
Beijing • Cambridge • Farnham • Köln • Paris • Sebastopol • Taipei • Tokyo

XAML in a Nutshell
by Lori A. MacVittie

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*safari.oreilly.com*). For more information, contact our corporate/institutional sales department: (800) 998-9938 or *corporate@oreilly.com*.

Editor: Jeff Pepper
Production Editor: Matt Hutchinson
Copyeditor: Rachel Monaghan
Proofreader: Matt Hutchinson

Indexer: Ellen Troutman
Cover Designer: Karen Montgomery
Interior Designer: David Futato
Illustrators: Robert Romano, Jessamyn Read, and Lesley Borash

Printing History:

March 2006: First Edition.

This book uses RepKover™, a durable and flexible lay-flat binding.

ISBN: 0-596-52673-3
[M]

Table of Contents

Preface . . . . . **ix**

Part I. Introducing XAML

1. Introducing XAML **3**
The Benefits of XAML 4
What XAML Is Not 5
XAML Development Resources 6

2. Getting Started with XAML **8**
XAML Prerequisites 8
Defining XAML Applications 9
Building XAML Applications 11
XAML Applications and Visual Studio 15

Part II. XAML Concepts

3. The Basics of XAML **23**
Core XAML Syntax 23
Elements 27
Attributes 31
Attached Properties 36
Binding Properties 37
codebehind 41

4. Layout and Positioning 44
StackPanel and DockPanel 45
Using Width and Alignment 48
Margins and Padding 52
Grid 56
Absolute Positioning 57

5. Resources 63
Using Resources 63
Using Styles 66
Triggers 70

6. Storyboards and Animations 75
Storyboards 76
Controlling Animations 80
Animation Using Key Frames 86

Part III. Core XAML Reference

7. Elements 91

8. Controls 117
Base Control Reference 118
Common Event Reference 122
Core Control Reference 124

9. Shapes and Geometry 157

10. Layout 184

11. Animations and Transformations 200

12. Events 237
Routing Strategies 239
Event Argument Reference 240
Event Reference 245

Part IV. Appendixes

A. System.Windows.Controls 257

B. System.Windows.Documents 259

C. System.Windows.Shapes 261

D. System.Windows 262

E. System.Windows.Media 263

F. System.Windows.Input.ApplicationCommands 265

G. Predefined Colors 267

H. XAML Interface in Code 269

Index 273

Preface

Windows Vista is on its way, and with it comes a brand-new mechanism for defining user interfaces. XAML is one of many new technologies appearing in Windows Vista and promises to be a pervasive part of core Windows programming across a variety of yet-to-be-introduced Windows frameworks. XAML completely removes the need for user-interface designers to understand code. Third-party visual layout applications can now generate valid XAML for use in building sophisticated Windows Vista applications.

The Windows Presentation Foundation (WPF), and therefore XAML, offer many sophisticated user-interface features that are not available in other declarative markup languages such as HTML or XUL. Scaling and rotation of both text and graphics, animation, and extensibility are all core parts of WPF and accessible to XAML developers. While HTML was developed primarily for displaying text and graphics on the Web, XAML's primary target is native Windows applications (although it can also target web-based deployments).

The close relationship between runtime objects and the elements in a XAML file make XAML an easy choice for user-interface design on the Windows platform. It offers the means to create rich, or "smart," clients that act more like a full-featured interface than a web-based application.

XAML can be used to design user interfaces without the need for code, or it can be used in conjunction with supported .NET languages such as C# and VB.NET. XAML is the preferred method of developing interfaces for applications on the Windows Vista platform because its powerful features allow developers to create interfaces that go above and beyond traditional interface design. XAML and the WPF open up endless possibilities for exciting new user interfaces, and this book will provide an understanding of the language and the framework upon which those interfaces are developed.

Who Should Read This Book

This book is intended for both .NET developers and user-interface designers familiar with HTML and the basics of XML. Developers intending to write full applications should have a good understanding of an existing .NET language such as C# or Visual Basic, as application logic requires development of code in a .NET-supported language.

Familiarity with other declarative markup languages such as HTML or XUL will help you quickly grasp the concepts and user-interface elements used to design interfaces with XAML.

Even if you are not familiar with a .NET language or other declarative markup languages, this book will be invaluable in providing you with an understanding of XAML.

What This Book Covers

This book covers XAML as it exists in the WinFX SDK (Community Technology Preview, October 2005). It covers core XAML constructs and discusses syntax as it relates to interfacing with the WinFX runtime—the WPF. The book provides examples and documentation of all core components and presents detailed discussions on features such as animation, resources, and layout that will jump-start you on your way to becoming a XAML developer.

There are already several flavors of XAML, each created to enable the design of user interfaces for a specific Windows API, such as Windows Workflow Foundation. This book focuses on the core XAML language as intended for use in building user interfaces for Avalon and will not explore API-specific subsets.

Organization

This book is organized into four sections. Each section focuses on a particular set of topics that are grouped together logically.

Part I, Introducing XAML

This part of the book introduces the basics of XAML. It details the prerequisites necessary to begin building user interfaces in XAML and introduces MSBuild, Microsoft's new unified build system.

Because XAML supports many new features such as animation and resources, Part II has been devoted to covering these unique concepts. The basics will be covered here, but new concepts such as animation and transformations are given in-depth attention later on.

Chapter 1, *Introducing XAML*
: This chapter provides you with a quick introduction to XAML and includes a list of references to tools available for developing XAML applications.

Chapter 2, *Getting Started with XAML*
: This chapter details the system prerequisites and basics necessary to begin developing and building XAML applications. It introduces Microsoft's new unified build system, MSBuild, and describes how to use it to build XAML applications. The chapter also walks you through an example of using Microsoft's Visual Studio tools to create and build an application.

Part II, XAML Concepts

This part of the book delves into the details of XAML. You'll learn about elements, controls, styles, and animations, and how to use them to create your own user interface.

There are many specific elements not discussed directly in other sections of this book. These elements, in conjunction with all core XAML elements, are detailed here for quick and easy access.

Chapter 3, *The Basics of XAML*
: This chapter describes the core XAML syntax and delves into the types of elements used to create XAML applications. Attributes, attached properties, and event handler coding techniques are explained and accompanied by examples of how to use them.

Chapter 4, *Layout and Positioning*
: This chapter details how to position individual elements using a variety of techniques, including panels and absolute positioning.

Chapter 5, *Resources*
: This chapter provides an overview of resources, focusing on the use of global resources to create a customized look and feel for your interface. It describes how to define and reference resources and details the use of triggers to apply styles based on events.

Chapter 6, *Storyboards and Animations*
: This chapter details the mechanisms available for animating XAML elements. It includes examples of animating properties, such as position and size of elements.

Part III, Core XAML Reference

This part of the book details syntax rules and attributes for XAML in a series of quick-reference chapters. This section divides XAML elements into logical categories of elements, controls, shapes and geometry, layout, animations, and transformations.

Chapter 7, *Elements*
: This reference chapter details and provides examples for the basic elements used within XAML, including `Brush` and `Pen`, `ListItem`, and elements used for text decoration, such as `Inline`, `Bold`, and `Italic`.

Chapter 8, *Controls*
: This reference chapter details the control elements available within XAML, such as `Button`, `CheckBox`, `ImageViewer`, and `Expander`. It also contains a reference to common events.

Chapter 9, *Shapes and Geometry*
: This reference chapter explains the differences between shape and geometry classes and details the `Shape` and `Geometry` elements available within XAML.

Chapter 10, *Layout*
: This reference chapter details the XAML elements used to lay out user interfaces such as `Grid` and `Panel`, and describes supporting elements such as `Trigger`, `Style`, and `Border`.

Chapter 11, *Animations and Transformations*
: This reference chapter details the types of animations and transformations available to XAML elements.

Chapter 12, *Events*
: This reference chapter explains the WPF event system and details the events available to XAML elements.

Part IV, Appendixes

The appendixes detail the CLR classes in the WinFX runtime that are available through XAML, list all of the predefined `Color` values supported by XAML, and present a complete code-only example of building a XAML application.

Appendix A, *System.Windows.Controls*
: Lists the elements found in the `System.Windows.Control` namespace

Appendix B, *System.Windows.Documents*
: Lists the elements found in the `System.Windows.Documents` namespace

Appendix C, *System.Windows.Shapes*
: Lists the elements found in the `System.Windows.Shapes` namespace

Appendix D, *System.Windows*
: Lists the elements found in the `System.Windows` namespace

Appendix E, *System.Windows.Media*
: Lists the elements found in the `System.Windows.Media` namespace

Appendix F, *System.Windows.Input.ApplicationCommands*
: Lists the elements found in the `System.Windows.Input.ApplicationCommands` namespace

Appendix G, *Predefined Colors*
: Lists the available predefined colors supported by XAML

Appendix H, *XAML Interface in Code*
: Contains a XAML declaration used to build a simple application

Conventions Used in This Book

The following list details the font conventions used in the book:

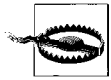
`Constant width`
: Indicates anything that might appear in a XAML document, including element names, tags, attribute values, and entity references, or anything that might appear in a program, including keywords, operators, method names, class names, and literals.

`Constant width bold`
: Indicates user input or emphasis in code examples and fragments.

`Constant width italic`
: Denotes replaceable elements in code statements.

Italic
: Indicates emphasis in body text, new terms when they are defined, pathnames, filenames, program names, and host and domain names.

This icon signifies a tip, suggestion, or general note.



This icon indicates a warning or caution.

Significant code fragments, complete applications, and documents generally appear in a separate paragraph, like this:

```
<Page xmlns="http://schemas.microsoft.com/winfx/avalon/2005"
      xmlns:x="http://schemas.microsoft.com/winfx/xaml/2005">
  <StackPanel>
     <TextBlock>Hello World</TextBlock>
  </StackPanel>
</Page>
```

When a property has a fixed set of values from which to choose, those choices will be displayed as a pipe-separated list:

```
SelectionMode="Single|Multiple|Extended" >
```

XAML, like XML, is case-sensitive. The `Page` element is not the same as the `PAGE` or `page` element. Both are also character-encoding-sensitive, and the smart quotes found in a Microsoft Word document or in the help files accompanying the WinFX SDK are not considered the same as the double quotes produced by applications such as Microsoft's Visual Studio or Notepad. Smart quotes are not valid within a XAML document, so it is important that you use the "Copy code" option in the WinFX SDK help system or turn off smart quotes in Microsoft Word if you wish to use either program to create XAML applications.

Using Code Examples

Most of the examples in this book have very little real-world value and are unlikely to be reused, although they work well as templates to get you started in designing your own user interfaces with XAML. In general, you may use the code in this book in your programs and documentation. Permission is not required unless you're reproducing a significant portion of the code. For example, writing a program that uses several blocks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books *does* require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation *does* require permission.

Attribution is appreciated, but not required. An attribution usually includes the title, author, publisher, and ISBN. For example: "*XAML in a Nutshell,* by Lori A. MacVittie. Copyright 2006 O'Reilly Media, Inc., 0-596-52673-3."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at *permissions@oreilly.com*.

Comments and Questions

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
(800) 998-9938 (in the U.S. or Canada)
(707) 829-0515 (international/local)
(707) 829-0104 (fax)

There is a web page for this book that lists errata, examples, or any additional information. You can access this page at:

http://www.oreilly.com/catalog/xamlian

To comment or ask technical questions about this book, send email to:

bookquestions@oreilly.com

For more information about books, conferences, Resource Centers, and the O'Reilly Network, see the O'Reilly web site at:

http://www.oreilly.com

Safari® Enabled

When you see a Safari® Enabled icon on the cover of your favorite technology book, that means the book is available online through the O'Reilly Network Safari Bookshelf.

Safari offers a solution that's better than e-books: it's a virtual library that lets you easily search thousands of top tech books, cut and paste code samples, download chapters, and find quick answers when you need the most accurate, current information. Try it for free at *http://safari.oreilly.com*.

Acknowledgments

Jeff Pepper, the editor who proposed the book and got things rolling.

Brad Shimmin, for bringing the opportunity to my attention in the first place.

My husband, Don, for encouraging me to agree to this undertaking and putting up with long hours spent staring at the screen trying to figure out why something wasn't working the right way, and for a second set of technically minded eyes.

The reviewers of early versions of the manuscript were invaluable in this effort. Thanks especially to Tim Patrick and Filipe Fortes for their thorough reviews and helpful comments.

Introducing XAML

1

Introducing XAML

XAML (pronounced "Zamel") stands for e*X*tensible *A*pplication *M*arkup *L*anguage. It is Microsoft's new declarative language for defining application user interfaces. XAML provides an easily extensible and localizable syntax for defining user interfaces separated from application logic, similar to the object-oriented technique for developing *n*-tier applications with a MVC (Model-View-Controller) architecture.

XAML was created by Microsoft expressly for the purpose of interfacing with its .NET Framework on its Windows Vista (formerly codenamed "Longhorn") operating system through the WinFX (codename "Avalon") presentation subsystem runtime environment. XAML gives developers the ability to control the layout of all .NET user-interface elements such as text, buttons, graphics, and listboxes, using XML. Because XAML is XML-based, your code must be well-formed XML. Every XAML tag corresponds directly to a .NET Framework class whose properties are controlled through the use of XML attributes. For example, the `<Button>` tag corresponds directly to the `System.Windows.Controls.Button` class. XAML elements represent a Common Language Runtime (CLR) class, the runtime engine for Microsoft's .NET framework. The CLR is similar to the Java Virtual Machine (JVM), except that the JVM can only run Java language programs, while the CLR can run applications written in a number of .NET languages, such as C#, J#, and VB.NET.

Because XAML elements represent CLR objects (this book focuses on those in the Windows Presentation Foundation [WPF]), anything that can be done with XAML can also be accomplished with procedural code. There are some things, however, that can be done by manipulating the object model programmatically that are not accessible through XAML. Properties that are read-only are not exposed through XAML; only those properties that are public and have both a `get` and a `set` method are accessible to XAML developers.

Events and handlers can also be specified by XAML attributes, and the necessary code behind the handlers, `codebehind`, can be written in .NET-supported

languages—currently C# and VB.NET. This code can be inlined in the XAML file or placed in the `codebehind` file, similar to what is done with ASP.NET code. If procedural code is embedded in a XAML page, you must compile the application before you can run it; if there is no procedural code in the XAML page, you can display it on a Windows Vista system by double-clicking the page file (just as you would with HTML pages). On Windows XP, however, the XAML pages must be "compiled" into an executable application before they can be displayed or loaded into a browser.

XAML is similar to other markup languages designed for rendering in web browsers, such as XHTML and HTML, and uses mechanisms similar to Cascading Style Sheets (CSS) for designating properties of XAML elements. Just as HTML objects are parsed to build out a Document Object Model (DOM) tree, XAML elements are parsed to build out an `ElementTree`.

XAML is inherently object-oriented since its elements represent CLR classes. This means that an element derived from another XAML element inherits the attributes of its parent. For example, a `System.Windows.Controls.Button` derives from `System.Windows.Controls.ButtonBase`, which derives from `System.Windows.Controls.ContentControl`, which derives from `System.Windows.FrameworkElement`, which derives from `System.Windows.UIElement`. Therefore, the `Button` element has very few attributes of its own but still boasts a lengthy list of attributes that it has inherited from classes above it in the hierarchy, such as `Width` and `Height`. It is necessary to understand the nature of inheritance in order to take advantage of XAML and its ability to be extended. Custom controls can be created for XAML by creating subclasses in one of the supported .NET languages (C# or VB.NET), for example, and then exposing the class to XAML developers for use in user-interface design.

Some XAML elements require children and attributes to be of a specific type, usually one of the base classes. Because of the nature of object-oriented programming, any element requiring that its children be of type `UIElement` can be declared as an element derived from `UIElement`. The `Brush` object is a very common attribute type for XAML elements, yet an instance of `Brush` is rarely used as an attribute. Instead, one of `Brush`'s subclasses, such as `SolidColorBrush` or `LinearGradientBrush`, is often used. The nature of object-oriented programming allows an attribute to be broadly defined as a base class and lets the designer choose which specialized subclass will be used.

Because of XAML's object-oriented nature, not all attributes will be listed with the element. It is necessary to understand an element's hierarchy to fully understand all of the attributes available to describe the element. In Part III, I have included each element's hierarchy—as well as a description of abstract elements—to facilitate this understanding. While abstract elements are rarely, if ever, declared in XAML, their description and attributes are used by derived classes and will therefore be fully described.

The Benefits of XAML

XAML offers similar benefits to other markup-based application interface mechanisms such as XUL (e*X*tensible *U*ser-interface *L*anguage), HTML (*H*yper*T*ext

*M*arkup *L*anguage), and Flex. Markup-based interfaces are quick to build and easily modifiable. They require less code than traditional structured programming. For example, creating and defining the properties of a `Button` with XAML requires just one line of syntax, as opposed to multiple lines in C# or VB.NET:

```
<Button Click="OnClickHandler" Background="Green" Content="Submit" />
```

The same `Button` object created using C# requires four lines:

```
Button myBtn = new Button( );
myBtn.Background = Brushes.Green;
myBtn.Text="Submit";
myBtn.Click += new System.EventHandler(OnClickHandler);
```

While HTML has limited programmatic functionality and control, XAML and other new-generation declarative markup languages offer back-end scripting language support to circumvent this limitation. While XAML separates the user interface from application logic, it still provides a mechanism by which the two can easily interact. This separation offers several benefits, including easily localized user interfaces and the ability for developers to modify application logic without affecting the user interface, and vice versa.

XAML also opens up user-interface design to a wider group of developers, namely graphic designers and markup developers. Anyone with experience using HTML or other web-oriented markup languages will find XAML to be intuitive; they will be able to jump in and begin developing user interfaces in a short period of time. This alleviates the burden placed on .NET developers and allows them to focus on developing application logic, while others determine the look and feel of the user interface.

XAML is toolable, which offers third-party developers opportunities to create applications that support it. Several third-party applications already exist that offer visual environments for developing XAML. Additional products are expected as Windows Vista begins to be generally deployed.

XAML is extensible, as its name implies. XAML can easily be extended by developers creating custom controls, elements, and functionality. Because XAML is essentially the XML representation of objects defined by the WPF, XAML elements can easily be extended by developers using object-oriented programming techniques. Custom controls and composite elements can be developed and exposed to user-interface designers or shared with other developers.

Finally, by using XAML, Windows applications can be delivered unchanged via the Web to Windows clients. *Smart clients*, Microsoft's term for rich user interfaces with full Windows functionality, can be delivered to any connected Windows machine over the Internet through a web browser without requiring the overhead of a managed desktop to deploy full-featured thick-client applications.

What XAML Is Not

XAML is purely a markup language designed for describing user-interface components and arranging them on the screen. Though there are components of XAML that appear to be programmatic in nature, such as the `Trigger` and `Transform`

elements, XAML is not a procedural programming language and is not designed to execute application logic.

XAML is interpreted, not compiled—though it can be compiled. Microsoft recommends that XAML be compiled by compacting it into Binary Application Markup Language (BAML). Both XAML and BAML are interpreted by the WPF and then rendered on the screen in a manner similar to HTML. Unlike HTML, however, XAML is strongly typed. HTML defaults to ignoring tags and attributes it doesn't understand, while XAML requires that every tag and attribute be understood, including the typing of attributes. Although all attributes initially appear to be strings, don't let that fool you. The string represents an object, and because those objects must be understood by WPF, XAML is strongly typed.

Finally, XAML is not HTML. Although there are similarities in the declaration of elements, application of styles, and assignment of event handlers, XAML is an XML-based interface to the Windows Presentation Framework, while HTML is a markup language that is rendered within the context of the browser and operating system in which it is loaded. XAML is far more than a mechanism for displaying information and soliciting basic user input. It is a complete user-interface design and development markup language that reaches beyond the scope of simple HTML elements by including advanced features such as 3-D element rendering and rich vector-based drawing capabilities.

XAML Development Resources

XAML can be developed in myriad ways. XAML can be written in any text editor. For example, all the code included in this book was written in Notepad and then compiled using MSBuild.

There are much easier ways to develop a XAML user interface, however, and most of them involve a visual layout tool. There are several third-party tools, as well as tools from Microsoft that support XAML. Some are focused on only one aspect of XAML, such as development of 3-D interfaces, while others are more generally applicable. Some popular tools available as of this writing include:

Electric Rain ZAM D XAML Tool (*http://www.erain.com/products/zam3d/*)
: A tool that supports visual development of 3-D interface elements for XAML.

Xamlon Pro and XAML Converter (*http://www.xamlon.com/*)
: Xamlon Pro supports development of XAML user interfaces in a visual environment. XAML Converter converts other formats to XAML.

MyXAML (*http://www.myxaml.com/*)
: An open source project dedicated to XAML development. Includes a mailing list and forums focused on discussion of XAML and the sharing of tips, tricks, and techniques.

Mobiform Aurora XAML Editor (*http://www.mobiform.com/2005/XAML/xamlhome.htm*)
: A visual editor for XAML from Mobiform.

XamlViewer (*http://weblogs.asp.net/gmilano/archive/2004/11/24/269082.aspx*)
: A visual editor for XAML that integrates into Visual Studio 2005.

XamlPad
: A simple, real-time visual editor for XAML. XamlPad does not support visual layout of elements, but it does offer a visual representation, in real time, of XAML elements. XamlPad is included in the WinFX SDK.

Microsoft's Visual Studio 2005 Extensions for WinFX
: Tools that include XAML Intellisense support through schema extensions for the editor and project templates for the WPF, the Windows Communication Foundation (formerly known as "Indigo"), and WinFX SDK documentation integration. These tools do not include a graphical design surface for either the WPF or the Windows Communication Foundation.

Microsoft Expression Interactive Designer (formerly "Sparkle") (*http://www.microsoft.com/products/expression/en/interactive_designer/default.aspx*)
: A forthcoming Microsoft visual-design tool for developing WinFX applications.

2

Getting Started with XAML

As with most development-oriented tools, it's important to have the proper environment before you can start developing user interfaces with XAML. This chapter discusses the prerequisites necessary to define and run XAML applications and later details the basic structure of a XAML project, as well as how to compile and run that application.

This chapter assumes that you have a working knowledge of XML and are at least somewhat familiar with other user-interface markup languages, such as ASP.NET and HTML.

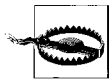
XAML Prerequisites

Although XAML is designed specifically for Windows Vista, it's also available on Windows XP and Windows Server 2003, given that certain system requirements are met. This makes it possible for developers to become familiar with XAML and the WinFX SDK before Windows Vista is officially available.

XAML can be used to develop applications on the following operating systems:

- Windows XP SP2
- Windows Server 2003 SP1
- Windows Vista

On Windows XP SP2 and Windows Server 2003 SP1, you will first need to install the WinFX runtime, which contains, among other things, the Windows Presentation Foundation (Avalon). Regardless of the operating system you choose, you'll need to install the WinFX SDK. The SDK contains the libraries, build tools, and documentation necessary to begin developing user interfaces with XAML. Depending on the operating system you choose, the WinFX SDK may also have prerequisites that must be met.



If you plan on using the WinFX Extensions to Visual Studio 2005, you *must* install Visual Studio 2005 before installing the WinFX SDK.

Defining XAML Applications

A XAML application comprises two types of elements: an application element and the set elements that make up the user interface. The XAML files contain the user-interface definition for your application. The `codebehind` files will contain the application logic and the code that handles event processing. XAML does not provide a mechanism for handling events, but it can direct the runtime engine to call event handlers written in C# or VB.NET. If you're a developer, you'll code the event handlers and application logic just as you always have, but because the user-interface code is separate, you'll have to pay a bit more attention to the names of the handlers and elements you reference because you don't define them—they're declared and named in the XAML file.

You can define XAML applications completely using C# or VB.NET. The CLR classes represented by XAML are all accessible through code, and you can write applications just as you always have, if you so desire. XAML offers you the ability to completely separate the presentation layer (user interface) from the application logic, thus making it easier to split up development responsibilities and isolate UI changes from the code. Appendix H provides an example of an application declared in XAML, as well as entirely in C#.

The most common application element is of type `NavigationApplication`. `NavigationApplication` defines an application that behaves like a web application or wizard in that it consists of pages between which a user navigates using hyperlinks and forward and back buttons.

The application definition is generally declared in its own file. It requires two properties to be set, the namespace and the startup URI, which is the URI of the first page that should be loaded when the application starts. For our purposes in this chapter, the application definition file will be called *MyApp.xaml*. It is detailed in Example 2-1.

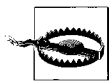
Example 2-1. MyApp.xaml

```
<NavigationWindow
    xmlns="http://schemas.microsoft.com/winfx/avalon/2005"
    StartupUri="Page1.xaml" />
```

In XAML, element names correspond to CLR object names, and attributes represent properties. The exception to this rule is with standard XML elements, such as `xmlns`, which is used to declare the namespace used within the XML file. The namespace used here is the default namespace for the application and identifies the Avalon types. If we did not specify the Avalon namespace as the default, all core XAML elements would need to include a reference to it. That's a lot of extra typing. It is much easier to use the Avalon namespace as the default, unless you

will be primarily using custom elements defined in your own namespace, in which case, it is probably easier to specify your own namespace as the default and explicitly identify XAML elements instead. All the examples in this book will declare the Avalon namespace as the default. Every XAML element requires either explicit references to the namespace on a per-element basis or the declaration of the Avalon namespace as the default of the root element. Of course the latter is recommended, as it will alleviate the requirement to explicitly reference the namespace for every XAML element in the file.

The first element declared in any XAML file is called the *root element*. The root element must contain a reference to the namespace in which it is defined. For XAML elements, the namespace is `http://schemas.microsoft.com/winfx/avalon/2005`.



The default namespace will change when WPF officially ships.

Root elements are containers that hold other XAML elements. The most common root element for the application definition is `NavigationWindow`. The most common root elements for a page definition are `Panel` and its subclasses, `DockPanel` and `StackPanel`, and `Page`. `Window` is also used, though less often than the aforementioned elements.

In Example 2-1, the `StartupUri` attribute of the `NavigationWindow` specifies the XAML page that will be loaded when the application starts, in this case *Page1.xaml*. Additional attributes of `NavigationWindow` can be specified. For a complete description of `NavigationWindow`, see Chapter 8.

Page1.xaml will contain the actual definition for the user interface. Any subsequent pages will be referenced through allowable mechanisms, such as the `HyperLink` element. Like all XAML files, *Page1.xaml* requires a root element. The file is shown in Example 2-2.

Example 2-2. Page1.xaml

```
<StackPanel xmlns="http://schemas.microsoft.com/winfx/avalon/2005">
   <TextBlock>Hello World</TextBlock>
   <Button Width="100">Click Me</Button>
</StackPanel>
```

`StackPanel` is fully described in Chapter 7. Like `DockPanel`, it is used to hold elements, and that is all you need to know for now. The `TextBlock` element holds text, and the `Button` element represents a standard user-interface button. Interpreting the code in XamlPad produces the output shown in Figure 2-1.

This is an extremely simple example of a XAML application with absolutely no attention paid to style, layout, or usefulness. Refining these aspects of user-interface design is a subject for subsequent chapters. For now, it is only important that the file declares the minimum requirements for a XAML application. With a successfully defined application definition (*MyApp.xaml*) and a page definition (*Page1.xaml*), it's time to build the application into a Windows executable.

Figure 2-1. A simple XAML page previewed in XamlPad

Building XAML Applications

While XAML can be used to create libraries and modules that can be shared and used to build other applications (in the same way that C# or VB.NET can be used to build DLLs or shared assemblies), it is more likely that you will use XAML to generate an application. There are two types of XAML applications: express and installed. *Express applications* are hosted in a web browser. *Installed applications* are traditional desktop applications and can be either Windows applications or console applications. The type of application generated is determined by a property value in the project file MSBuild uses to assemble the application.

MSBuild is one of the new features in Windows Vista and Visual Studio 2005. With the release of Visual Studio 2005, Microsoft has moved to a unified build environment. All projects now use MSBuild facilities to generate CLR assemblies. The most exciting, and beneficial, aspect of this change is that Visual Studio is no longer required to compile and build applications; builds can be completely automated without it. MSBuild is distributed with the WinFX SDK.

If you're using Visual Studio to edit XAML and associated `codebehind` files, don't worry about the details of MSBuild. The relevant files are generated automatically by Visual Studio.

MSBuild is similar to ANT and Unix/Linux *make* facilities. MSBuild reads in XML-based project files, conventionally named with a *.proj* extension, and executes the tasks contained in the project file to produce the desired target.

There are a number of XML elements that can be used in a project file. This discussion covers only the basic elements and the typical ways that they are used

to create an Avalon project file. The following list describes the key elements in an Avalon project file:

`Project`
: Functions as the root element for all project files

`PropertyGroup`
: Contains project property settings, such as the build configuration setting (`Debug` or `Release`)

`ItemGroup`
: Contains the list of items, such as source or resource files, that make up the project

`Import`
: Allows you to import other project files, such as target files, into your project

There are a multitude of options that can be configured with MSBuild. It is a very rich schema designed to handle building targets in a dynamic environment. The following code illustrates the minimum requirements for a project file:

```
<Project
    xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
    <PropertyGroup>
        <AssemblyName>MyFirstApplication</AssemblyName>
        <TargetType>winexe|exe|library|module</TargetType>
        <OutputPath>.\</OutputPath>
    </PropertyGroup>
    <Import Project="$(MSBuildBinPath)\Microsoft.CSharp.targets" />
    <Import Project="$(MSBuildBinPath)\Microsoft.WinFX.targets" />
    <ItemGroup>
        <ApplicationDefinition Include="MyApp.xaml" />
        <Page Include="Page1.xaml" />
    </ItemGroup>
    <ItemGroup>
    <Reference Include="System">
          <Private>false</Private>
    </Reference>
    <Reference Include="System.Xml">
          <Private>false</Private>
    </Reference>
    <Reference Include="System.Data">
          <Private>false</Private>
    </Reference>
    <Reference Include="WindowsBase">
          <Private>false</Private>
    </Reference>
    <Reference Include="PresentationCore">
          <Private>false</Private>
    </Reference>
    <Reference Include="PresentationFramework">
          <Private>false</Private>
    </Reference>
    <Reference Include="WindowsUIAutomation">
          <Private>false</Private>
    </Reference>
```

```
    <Reference Include="UIAutomationProvider">
        <Private>false</Private>
    </Reference>
  </ItemGroup>
</Project>
```

The most important piece of the project file is the `ItemGroup`, which specifies the inclusion of the XAML files that make up your project. You'll need one `ApplicationDefinition` file, identified by the `<ApplicationDefinition .../>` element, and one or more page definition files, included through the use of the `<Page .../>` element.

You can set a few optional attributes in the `PropertyGroup` element:

`HostInBrowser`
: This `Boolean` value is set to `true` to generate express applications or `false` to generate an installed application. The default value is `false`.

`Install`
: This `Boolean` value determines the type of deployment file to generate. When set to `true`, a deployment file for an installed application is generated. When set to `false`, a deployment file for an express application is created. If `HostInBrowser` is set to `true`, the default value for this property is `false`. If `HostInBrowser` is `false`, the default value for this property is `true`.

`Configuration`
: This `String`-based value determines the type of configuration to build: `Debug` or `Release`. The default is `Release`.

MSBuild relies on a number of environment variables related to the location of libraries and the identification of the .NET Framework version used to build the application. The WinFX SDK includes a batch file to appropriately set these environment variables. The necessary variables are:

```
SET FrameworkVersion=v2.0.50215
SET FrameworkDir=%windir%\Microsoft.NET\Framework
SET WinFX=%ProgramFiles%\Reference Assemblies\Microsoft\WinFX\
%FrameworkVersion%
SET URT=%FrameworkDir%\%FrameworkVersion%
SET WinFXSDK=C:\Program Files\Microsoft SDKs\WinFX
SET FrameworkSDKDir=%WinFXSDK%\
SET WinFXSDKTOOLPATH=%WinFXSDK%\bin
SET PATH=%URT%;%WinFXSDKTOOLPATH%;%WinFXSDK%\vc\bin;%path%;
SET INCLUDE=%WinFXSDK%\Include;%WinFXSDK%\vc\Include;
SET LIB=%WinFXSDK%\Lib;%WinFXSDK%\vc\Lib;
```

After the environment variables have been set and the project file is appropriately configured for your application target, execute MSBuild on the command line to generate your application. When completed, you will see several files:

MyFirstApplication.exe
: The executable application. This file is always generated.

MyFirstApplication.xbap
: The express application. This file is recognized by the Windows IE browser and can be run by opening the file within IE. This file is generated only when