Adam Nathan

Full Color
Figures and code appear as they do in Visual Studio.

XAML

UNLEASHED

SAMS

FREE SAMPLE CHAPTER

SHARE WITH OTHERS

Adam Nathan

XAML

UNLEASHED

SAMS | 800 East 96th Street, Indianapolis, Indiana 46240 USA

XAML Unleashed

ISBN-13: 978-0-672-33722-2
ISBN-10: 0-672-33722-3

Library of Congress Control Number: 2014953616

Printed in the United States of America

First Printing December 2014

EDITOR-IN-CHIEF
Greg Wiegand

EXECUTIVE EDITOR
Joan Murray

DEVELOPMENT EDITOR
Mark Renfrow

MANAGING EDITOR
Kristy Hart

SENIOR PROJECT EDITOR
Betsy Gratner

INDEXER
Tim Wright

PROOFREADER
Chuck Hutchinson

TECHNICAL EDITOR
Jeremy Likness

EDITORIAL ASSISTANT
Cindy Teeters

COVER DESIGNER
Mark Shirar

COMPOSITOR
Nonie Ratcliff

Contents at a Glance

Part IV Exploiting XAML Features

Part V Advanced Features

Table of Contents

About the Author

Adam Nathan is a principal software architect for Microsoft, a best-selling technical author, and a prolific developer of apps for Windows. He introduced XAML to countless developers through his books on a variety of Microsoft technologies. Currently a part of the Windows team, Adam has previously worked on Visual Studio and the Common Language Runtime. He was the founding developer and architect of Popfly, Microsoft's first Silverlight-based product, named by *PCWorld* as one of its year's most innovative products. He is also the founder of PINVOKE.NET, the online resource for .NET developers who need to access Win32. His apps have been featured on Lifehacker, Gizmodo, ZDNet, ParentMap, and other enthusiast sites.

Adam's books are considered required reading by many inside Microsoft and throughout the industry. Adam is the author of *Windows 8.1 Apps with XAML and C# Unleashed* (Sams, 2013), *101 Windows Phone 7 Apps* (Sams, 2011), *Silverlight 1.0 Unleashed* (Sams, 2008), *WPF 4.5 Unleashed* (Sams, 2013), *.NET and COM: The Complete Interoperability Guide* (Sams, 2002), and several others. You can find Adam online at www.adamnathan.net or @adamnathan on Twitter.

Dedication

To Tyler and Ryan.

Acknowledgments

I'd like to give special thanks to Ashish Shetty, Tim Heuer, Mark Rideout, Jonathan Russ, Joe Duffy, Chris Brumme, Eric Rudder, Joan Murray, Loretta Yates, Betsy Gratner, Bill Chiles, Valery Sarkisov, Dwayne Need, Daniel Lehenbauer, Andy Sterland, Tim Rice, and Michelle McCarthy. As always, I thank my parents for having the foresight to introduce me to Basic programming on our IBM PCjr when I was in elementary school. That small decision changed the entire course of my life.

We Want to Hear from You!

As the reader of this book, *you* are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

We welcome your comments. You can email or write to let us know what you did or didn't like about this book—as well as what we can do to make our books better.

Please note that I cannot help you with technical problems related to the topic of this book, and that due to the high volume of mail I receive, I might not be able to reply to every message.

When you write, please be sure to include this book's title and author as well as your name and email address. We will carefully review your comments and share them with the author and editors who worked on the book.

Email: feedback@samspublishing.com

Mail: Joan Murray
Executive Editor
800 East 96th Street
Indianapolis, IN 46240 USA

Reader Services

Visit our website and register this book at informit.com/register for convenient access to any updates, downloads, or errata that might be available for this book.

INTRODUCTION

In This Chapter

Ever since XAML was publicly introduced in 2003 (as part of the framework that would eventually be named Windows Presentation Foundation), it has gotten considerable attention for the ways in which it revolutionizes the process of creating software—especially for traditional Windows programmers. It's relatively easy to create fun, useful, and shareable XAML samples that demonstrate all kinds of techniques that are difficult to accomplish in other technologies.

Over the years, Microsoft has shipped many (*too many*) UI frameworks based on XAML that are extremely similar but not identical. The introduction of universal Windows apps hopes to consolidate people's development efforts on a single technology, but there's still plenty of WPF and Silverlight development out in the real world.

That's where this book comes in. It examines the XAML language common to a number of technologies, although many of its examples are based on the latest XAML-based UI framework. I wrote this book with the following goals in mind:

- To provide a solid grounding in the underlying concepts, in a practical and approachable fashion
- To answer the questions most people have when learning XAML and to show how commonly desired tasks are accomplished
- To be an authoritative source, thanks to input from members of the teams involved with XAML over the years
- To be an easily navigated reference that you can constantly come back to

Who Should Read This Book?

This book is for software developers who are interested in creating user interfaces for Windows. Regardless of whether you're creating line-of-business applications, consumer-facing applications, or reusable controls, and no matter what types of devices you're targeting, this book contains a lot of content that helps you get the most out of XAML. It's designed to be understandable even for folks who are new to Microsoft technologies. Examples in this book appear in XAML and C#.

Code Examples

The source code for examples in this book can be downloaded from www.informit.com/title/9780672337222. (Click the Downloads tab.)

How This Book Is Organized

This book is arranged into five main parts, representing the progression of feature areas that you typically need to understand. But if you're dying to jump ahead and learn about a topic such as 3D or animation, the book is set up to allow for nonlinear journeys as well. The following sections provide a summary of each part.

Part I: The XAML Language

This part includes the following chapters:

- Chapter 1, "What Is XAML?"
- Chapter 2, "Child Elements and Keywords"
- Chapter 3, "Mixing XAML with Code"
- Chapter 4, "Extending XAML"

This part of the book delves into the syntax of XAML and examines it broadly across its applications.

Part II: Graphics

This part includes the following chapters:

- Chapter 5, "Arranging Elements"
- Chapter 6, "Automatic Layout"
- Chapter 7, "2D Graphics"
- Chapter 8, "3D Graphics"

Part II equips you with the knowledge to assemble and arrange controls (and other elements) in a user interface.

Part III: Controls

This part includes the following chapters:

- ➔ Chapter 9, “Content Controls”
- ➔ Chapter 10, “Items Controls”
- ➔ Chapter 11, “Images”
- ➔ Chapter 12, “Text”
- ➔ Chapter 13, “Audio and Video”
- ➔ Chapter 14, “Other Controls”
- ➔ Chapter 15, “User Controls and Custom Controls”

Part III provides a tour of controls built into various XAML-based UI frameworks. There are many that you’d expect to have available, plus several that you might not expect. It ends with details on how to create your own controls.

Part IV: Exploiting XAML Features

This part includes the following chapters:

- ➔ Chapter 16, “Resources”
- ➔ Chapter 17, “Data Binding”
- ➔ Chapter 18, “Styles, Templates, and Visual States”
- ➔ Chapter 19, “Animation”

This part of the book covers the XAML features that typically get the most attention. They enable you to create a stunning experience, and/or greatly enhance the development process.

Part V: Advanced Features

This part includes the following chapters:

- ➔ Chapter 20, “Layout with Custom Panels”
- ➔ Chapter 21, “Fun with XAML Readers and Writers”

The topics covered in Part V are relevant for advanced developers, or just curious people who want to delve into some lesser-known functionality.

Conventions Used in This Book

Various typefaces in this book identify new terms and other special items. These typefaces include the following:

Typeface	Meaning
Italic	Italic is used for new terms or phrases when they are initially defined and occasionally for emphasis.
`Monospace`	`Monospace` is used for screen messages, code listings, and command samples, as well as filenames. In code listings, *`italic monospace type`* is used for placeholder text. Code listings are colorized similarly to the way they are colorized in Visual Studio. `Blue monospace type` is used for XML elements and C#/C++ keywords, `brown monospace type` is used for XML element names and C#/C++ strings, `green monospace type` is used for comments, `red monospace type` is used for XML attributes, and `teal monospace type` is used for type names in C# and C++.

When a line of code is too long to fit on a line in the printed book, a code-continuation arrow (➥) is used.

Throughout this book, you'll find a number of sidebar elements:

What is a FAQ sidebar?

A FAQ sidebar presents a question readers might have regarding the subject matter in a particular spot in the book—and then provides a concise answer.

Digging Deeper Sidebars

A Digging Deeper sidebar presents advanced or more detailed information on a subject than is provided in the surrounding text. Think of Digging Deeper material as stuff you can look into if you're curious but can ignore if you're not.

A tip is a bit of information that can help you in a real-world situation. Tips often offer shortcuts or alternative approaches to produce better results or to make a task easier or quicker.

A warning alerts you to an action or a condition that can lead to an unexpected or unpredictable result—and then tells you how to avoid it.

This page intentionally left blank

Chapter 2

In This Chapter

CHILD ELEMENTS AND KEYWORDS

A XAML file, like all XML files, must have a single root object element. Therefore, it should come as no surprise that object elements can support children that are more than just the property elements introduced in the preceding chapter. Property elements aren't true children as far as XAML is concerned.

This chapter examines the types of children that object elements can have. It also summarizes all the keywords in the XAML language namespace, although many of them are discussed in depth in the two chapters that follow.

Children of Object Elements

An object element can have three types of children:

- A value for a content property
- Collection items
- A value that can be type-converted to the object element

We'll look at the first two types of children now, but save the third type for Chapter 4, "Extending XAML."

The Content Property

Most classes designate a property (via a custom attribute) that should be set to whatever content is inside the XML element. This property is called the *content property*, and it is really just a convenient shortcut to make the XAML representation more compact.

`Button`'s `Content` property is (appropriately) given this special designation, so the following `Button`:

```
<Button xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  Content="OK"/>
```

could be rewritten as follows:

```
<Button xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation">
  OK
</Button>
```

Or, more usefully, this `Button` with more complex content:

```
<Button xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation">
<Button.Content>
  <Rectangle Height="40" Width="40" Fill="Black"/>
</Button.Content>
</Button>
```

could be rewritten as follows:

```
<Button xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation">
  <Rectangle Height="40" Width="40" Fill="Black"/>
</Button>
```

There is no requirement that the content property must actually be called `Content`; classes such as `ComboBox` and `ListBox` use their `Items` property as the content property. For example, this `ListBox`:

```
<ListBox Width="100">
  <ListBox.Items>
    <ListBoxItem>1</ListBoxItem>
    <ListBoxItem>2</ListBoxItem>
    <ListBoxItem>3</ListBoxItem>
  </ListBox.Items>
</ListBox>
```

is equivalent to the following `ListBox`:

```
<ListBox Width="100">
  <ListBoxItem>1</ListBoxItem>
  <ListBoxItem>2</ListBoxItem>
  <ListBoxItem>3</ListBoxItem>
</ListBox>
```

Both produce the result in Figure 2.1.

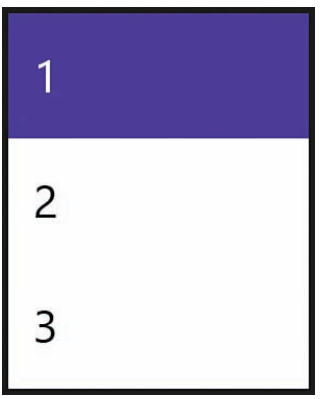

FIGURE 2.1 A simple `ListBox` has three children in its `Items` collection.

Collection Items

XAML enables you to add items to the two main types of collections that support indexing: lists and dictionaries.

Lists

A *list* is any collection that implements `System.Collections.IList`, such as `System.Collections.ArrayList` or numerous collection classes defined by various frameworks. For example, the following XAML adds two items to a `ListBox` control whose `Items` property is an `ItemCollection` that implements `IList`:

```
<ListBox xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation">
<ListBox.Items>
  <ListBoxItem Content="Item 1"/>
  <ListBoxItem Content="Item 2"/>
</ListBox.Items>
</ListBox>
```

This is equivalent to the following code:

C# (WPF and Silverlight):

```
System.Windows.Controls.ListBox listbox = new System.Windows.Controls.ListBox();
System.Windows.Controls.ListBoxItem item1 =
  new System.Windows.Controls.ListBoxItem();
System.Windows.Controls.ListBoxItem item2 =
  new System.Windows.Controls.ListBoxItem();
item1.Content = "Item 1";
item2.Content = "Item 2";
listbox.Items.Add(item1);
listbox.Items.Add(item2);
```

C# (Windows Store and Universal Apps):

```
Windows.UI.Xaml.Controls.ListBox listbox =
  new Windows.UI.Xaml.Controls.ListBox();
Windows.UI.Xaml.Controls.ListBoxItem item1 =
  new Windows.UI.Xaml.Controls.ListBoxItem();
Windows.UI.Xaml.Controls.ListBoxItem item2 =
  new Windows.UI.Xaml.Controls.ListBoxItem();
item1.Content = "Item 1";
```

```
item2.Content = "Item 2";
listbox.Items.Add(item1);
listbox.Items.Add(item2);
```

Furthermore, because `Items` is the content property for `ListBox`, you can shorten the XAML even further, as follows:

```
<ListBox xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation">
  <ListBoxItem Content="Item 1"/>
  <ListBoxItem Content="Item 2"/>
</ListBox>
```

In all these cases, the code works because `ListBox`'s `Items` property is automatically initialized to any empty collection object. If a collection property is initially `null` instead (and is read/write, unlike `ListBox`'s read-only `Items` property), you need to wrap the items in an explicit element that instantiates the collection. `ListBox` does not act this way, so an imaginary `OtherListBox` element demonstrates what this could look like:

```
<OtherListBox>
<OtherListBox.Items>
  <ItemCollection>
    <ListBoxItem Content="Item 1"/>
    <ListBoxItem Content="Item 2"/>
  </ItemCollection>
</OtherListBox.Items>
</OtherListBox>
```

Dictionaries

`ResourceDictionary` is a commonly used collection type in all XAML-based frameworks that you'll see more of in Chapter 16, "Resources." It implements `System.Collections.IDictionary`, so it supports adding, removing, and enumerating key/value pairs in procedural code, as you would do with a typical hash table. In XAML, you can add key/value pairs to any collection that implements `IDictionary`. For example, the following XAML adds two `Color`s to a `ResourceDictionary`:

```
<ResourceDictionary
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <Color x:Key="1">White</Color>
  <Color x:Key="2">Black</Color>
</ResourceDictionary>
```

This leverages the XAML `Key` keyword (defined in the secondary XML namespace), which is processed specially and enables us to attach a key to each `Color` value. (The `Color` type does not define a `Key` property.) Therefore, the XAML is equivalent to the following code:

C# (WPF and Silverlight):

```
System.Windows.ResourceDictionary d = new System.Windows.ResourceDictionary();
System.Windows.Media.Color color1 = System.Windows.Media.Colors.White;
System.Windows.Media.Color color2 = System.Windows.Media.Colors.Black;
d.Add("1", color1);
d.Add("2", color2);
```

C# (Windows Store and Universal Apps):

```
Windows.UI.Xaml.ResourceDictionary d = new Windows.UI.Xaml.ResourceDictionary();
Windows.UI.Color color1 = Windows.UI.Colors.White;
Windows.UI.Color color2 = Windows.UI.Colors.Black;
d.Add("1", color1);
d.Add("2", color2);
```

Special Attributes Defined by the W3C

In addition to keywords in the XAML language namespace, XAML also supports two special attributes defined for XML by the World Wide Web Consortium (W3C): `xml:space` for controlling whitespace parsing and `xml:lang` for declaring the document's language and culture. The `xml` prefix is implicitly mapped to the standard XML namespace; see http://www.w3.org/XML/1998/ namespace.

XAML Keywords

The XAML language namespace (`http://schemas.microsoft.com/winfx/2006/xaml`) defines a handful of keywords that must be treated specially by any XAML compiler or parser. Table 2.1 lists them all, along with the chapter that discusses each one further. Several of them require a specific context in order to be usable. For example, several are only valid for WPF.

TABLE 2.1 Keywords in the XAML Language Namespace, Assuming the Conventional `x` Namespace Prefix

Keyword	Valid As	Chapter	Meaning
`x:Array`	An element	4	Represents an array. An `x:Array` element's children are the elements of the array. It must be used with `x:Type` to define the type of the array.
`x:AsyncRecords`	Attribute on root element	3	Controls the size of asynchronous XAML-loading chunks.
`x:Arguments`	Attribute on or element inside any element	4	Specifies an argument (or multiple arguments in the element syntax) to be passed to the element's constructor. When used with `x:FactoryMethod`, specifies argument(s) for the factory method.
`x:Boolean`	An element	4	Represents a `System.Boolean`.
`x:Byte`	A XAML2009 element	4	Represents a `System.Byte`.
`x:Char`	A XAML2009 element	4	Represents a `System.Char`.
`x:Class`	Attribute on root element	3	Defines a class for the root element that derives from the element type, optionally prefixed with a namespace.
`x:ClassAttributes`	Attribute on root element and must be used with `x:Class`	N/A	Contains attributes relevant for Windows Workflow Foundation activities.
`x:ClassModifier`	Attribute on root element and must be used with `x:Class`	3	Defines the visibility of the class specified by `x:Class` (which is public by default). The attribute value must be specified in terms of the procedural language being used (for example, `public` or `internal` for C#).
`x:Code`	Element anywhere in XAML but must be used with `x:Class`	3	Embeds procedural code to be inserted into the class specified by `x:Class`.
`x:ConnectionId`	Attribute	N/A	Not for public use.
`x:Decimal`	A XAML2009 element	4	Represents a `System.Decimal`.
`x:Double`	An element	4	Represents a `System.Double`.
`x:FactoryMethod`	Attribute on any element	4	Specifies a static method to be called to retrieve the element instance instead of its constructor.

Keyword	Valid As	Chapter	Meaning
`x:FieldModifier`	Attribute on any nonroot element but must be used with `x:Name` (or equivalent)	3	Defines the visibility of the field to be generated for the element (which is internal by default). As with `x:ClassModifier`, the value must be specified in terms of the procedural language (for example, `public`, `private`, … for C#).
`x:Int16`	A XAML2009 element	4	Represents a `System.Int16`.
`x:Int32`	An element	4	Represents a `System.Int32`.
`x:Int64`	A XAML2009 element	4	Represents a `System.Int64`.
`x:Key`	Attribute on an element whose parent implements `IDictionary`	2	Specifies the key for the item when added to the parent dictionary.
`x:Members`	Inside an `Activity` class	4	Defines additional members for the root class specified by `x:Class`.
`x:Name`	Attribute on any nonroot element but must be used with `x:Class`	3	Chooses a name for the field to be generated for the element, so it can be referenced from procedural code.
`x:Null`	A property element or attribute value	4	Represents a `null` reference.
`x:Object`	A XAML2009 element	4	Represents a `System.Object`.
`x:Property`	Inside an `x:Members` element	4	Defines a new property.
`x:Reference`	A property element or attribute value	17	A reference to a named element.
`x:Shared`	Attribute on any element in a `ResourceDictionary`, but only works if XAML is compiled	16	A WPF-specific concept that doesn't really belong in the XAML language namespace.
`x:Single`	A XAML2009 element	4	Represents a `System.Single`.
`x:Static`	A property element or attribute value	4	References any static property, field, constant, or enumeration value.
`x:String`	An element	4	Represents a `System.String`
`x:Subclass`	Attribute on root element and must be used with `x:Class`	3	Specifies a subclass of the `x:Class` class that holds the content defined in XAML, optionally prefixed with a namespace (used with languages without support for partial classes).
`x:SynchronousMode`	Attribute on root element	3	Specifies whether the XAML content is allowed to be loaded asynchronously.

Keyword	Valid As	Chapter	Meaning
`x:TimeSpan`	A XAML2009 element	4	Represents a `System.TimeSpan`.
`x:Type`	A property element or attribute	4	Represents a `System.Type`, just like the `typeof` operator in C#.
`x:TypeArguments`	Attribute on any element in XAML2009, or attribute on root element that must be used with `x:Class` in XAML2006	4	Makes the class generic (for example, `List<T>`) with the specified generic argument instantiations (for example, `List<Int32>` or `List<String>`). Can be set to a comma-delimited list of generic arguments, with XML namespace prefixes for any types not in the default namespace.
`x:Uid`	Attribute on any element	16	Marks an element with an identifier used for localization.
`x:Uri`	A XAML2009 element	4	Represents a `System.Uri`.
`x:XData`	Element used as the value for any property of type `IXmlSerializable`	17	An arbitrary XML data island that remains opaque to the XAML parser.

Summary

XAML excels at expressing deep hierarchies of objects concisely, thanks to the features described in this chapter.

The concept of an element's *content* is a crucial one in XAML, for more reasons than simply the syntactic shortcut enabled by content properties. The behavior of styling, the topic of Chapter 18, "Styles, Templates, and Visual States," and a large set of controls, covered in Chapter 9, "Content Controls," are centered around the notion of an element's content.

The ability to fill collections with items is leveraged throughout XAML-based frameworks. You'll see many examples in Chapter 10, "Items Controls," and Chapter 16, "Resources."

INDEX

Numerics

A

B

C

D

E

F

G

H

I

J-K

L

M

N

O

P

Q-R

S

T

U

V

W

X

Y-Z