Оглавление

Предисловие

Эта книга не про WPF (Windows Presentation Foundation), не про Silverlight, не про программирование для Windows 10 или для WinRT. Эта книга не о возможностях UWP (Universal Windows Platform) и не о технологии Xamarin. Эта книга о том общем звене (почти для всех перечисленных инструментов и платформ), в качестве которого выступает язык декларативной разметки XAML. Именно этот язык применяется во всех названных инструментах и позволяет разрабатывать универсальный интерфейс пользователя, пригодный и для разных операционных систем (Windows, iOS, Android, Linux), и для разных видов приложений (работающих на телефонах, планшетах, настольных компьютерах, устройствах Surface Hub, консолях Xbox и HoloLens).

Сегодня XAML используется очень широко. Особенно это справедливо для технологий, развиваемых Microsoft. Хотя язык XAML построен на основе XML, но он принципиально отличается от других языков разметки, подобных HTML. Основное отличие в том, что XAML с помощью применяемых в XAML объектов накрепко «привязан» к сборкам исполняющей среды CLR, точнее к типам платформы .NET Framework.

Первоначально фирма Microsoft планировала, что XAML будет новым дескриптивным языком разработки гибкого и адаптивного пользовательского интерфейса для платформы .NET Framework при использовании средств WPF. Однако XAML вышел далеко за пределы первоначально поставленных задач.

В последнее время XAML вытесняет средства Windows Forms, применяемые для проектирования пользовательского интерфейса Windows-приложений. Кроме того, XAML является основой для разработки богатых веб-интерфейсов, мультимедиапотоков и управляемых данными приложений Line-of-Bisiness (LOB). Современные операционные системы, точнее универсальная платформа Windows (UWP), также широко используют возможности XAML.

Разнообразие областей применения XAML привело к тому, что в настоящее время (2017–2018 гг.) мир XAML стал фрагментарным: наиболее самобытные диалекты XAML существуют для WPF, Silverlight, UWP и Xamarin.Forms. Если учесть, что родные инстру-

менты фирмы Microsoft (WPF, Silverlight) все больше соответствуют философии UWP, а проект Xamarin SDK разрабатывался до 2016 г. отдельно компанией Xamarin, то понятно, что наиболее резкая граница сейчас существует между XAML-диалектами для UWP и вариантом XAML, применяемым в Xamarin.Forms. Тот факт, что компания Xamarin в 2016 г. была куплена фирмой Microsoft и сейчас Xamarin SDK включена в состав IDE Microsoft Visual Studio в качестве бесплатного инструмента, пока не устранил различий между диалектом XAML для UWP и диалектом XAML для Xamarin.Forms. В настоящее время ведется разработка стандарта спецификации XAML, цель которой – выработать набор принципов, позволяющих согласовать и унифицировать диалекты XAML. При стандартизации XAML предполагается не отбрасывание тех или иных средств конкретного диалекта XAML, а расширение диалектов за счет добавления отсутствующих в них уникальных средств из других диалектов. Такой подход к разработке стандарта XAML позволяет не изменять конкретные инструментальные среды, которыми пользуются разработчики, применяя UWP или Xamarin.Forms.

Возможно, в связи с тем, что разработка стандарта спецификации XAML пока не завершена, в настоящее время практически отсутствуют книги (по крайней мере, на русском языке), посвященные именно декларативному языку XAML, а не его «проекциям» на ту или иную технологию (WPF, Silverlight, Xamarin.Forms). Язык XAML и его применения описываются либо в книгах по указанным технологиям, либо в пособиях, посвященных инструментальным средствам автоматизации проектирования пользовательского интерфейса (например, Blend или Xamarin.Forms). В такие пособия включают XAML в качестве одного из средств. Именно такой поход приводит к затруднениям в изучении WPF, SilverLight или Xamarin.Forms. В фундаментальном справочнике по C# кратко, но с похвалой, упоминая о технологии WPF, авторы (Албахари Д., Албахари Б.), перечислив ее возможности и достоинства, завершают перечисление грустной фразой о том, что технология WPF очень хороша, но слишком сложна для изучения.

В чем же сложность изучения WPF и, соответственно, Silverlight, UWP и Xamarin.Forms?

Если сравнивать WPF, Silverlight и Xamarin.Forms с предыдущими средами программирования, то основное затруднение связано с необходимостью при разработке приложений пользоваться, по

крайней мере, двумя языками программирования: императивным языком (C#, VB или C++) и декларативным языком XAML. Еще оно затруднение – необходимость ориентироваться в большом объеме справочного материала по библиотеке .NET Framework – было присуще и предыдущим технологиям, поэтому это затруднение можно в нашей книге не обсуждать.

Необходимость применять в инструментальных средах разработки для UWP и Xamarin.Forms два языка (императивный и декларативный) объясняется очень важной целью – отделить представление пользовательского интерфейса от кода. Предлагается представление интерфейса описывать на декларативном языке XAML, а логику работы создаваемого программного продукта кодировать на императивном языке. Такое строгое отделение представления пользовательского интерфейса от императивного кода позволяет дизайнерам и программистам-разработчикам императивного кода работать достаточно независимо и тем самым более продуктивно. Дизайнерам достаточно владеть инструментальными средствами автоматизированного проектирования пользовательского интерфейса и хорошо понимать декларативную разметку на языке XAML, которую формирует применяемый ими инструмент (например, Blend или Xamarin.Forms). Роль программиста-разработчика теоретически не меняется, он должен кодировать на императивном языке. Но это только теоретически. Во-первых, не всегда существуют отдельный профессионал-дизайнер и отдельный профессионал-программист. Часто это одно лицо. Во-вторых, даже получив от дизайнера результат его труда в виде разметки на языке XAML, профессиональный программист должен хорошо знать возможности не только императивного языка, но и языка XAML и уметь вручную (без использования визуального интерактивного конструктора) программировать на XAML. И вот тут-то начинаются трудности именно из-за особенностей XAML, хотя при первом знакомстве кажется, что язык, построенный на основе XML, должен быть таким же понятным и простым, как, например, HTML. Все так и не совсем так.

Проиллюстрируем трудности изучения синтаксиса XAML на следующем примере XAML-документа, который выводит на экран браузера Internet Explorer приветствие (рис. 1). Если читатель не знаком с XML и/или никогда не работал на HTML, он может пропустить этот пример и перейти к основным главам книги, там все объясняется.

```
<!-- Предисловие_01.xaml - автономный XAML-документ -->
<Page
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation">
  <Label>
    <Label.FontSize>
      30
    </Label.FontSize>
    <Label.FontWeight>
      Bold
    </Label.FontWeight>
    <Label.Foreground>
      <SolidColorBrush>
        <SolidColorBrush.Color>
          Red
        </SolidColorBrush.Color>
      </SolidColorBrush>
    </Label.Foreground>
    <Label.Content>
      Hello, XAML!
    </Label.Content>
  </Label>
</Page>
```

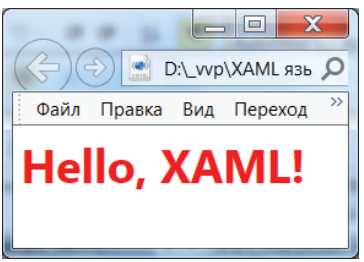

Рис. 1. Отображение элемента Label на экране браузера

Основу документа составляет XAML-элемент Page, в который вложен только один элемент Label. Для этого элемента определены значения четырех свойств: FontSize (Размер шрифта), FontWeight (Начертание шрифта), Foreground (Окраска текста), Content (Содержимое в виде текста). Свойства определены с помощью элементов свойств, в обозначения которых входит имя элемента Label. Размер шрифта «30» (Label.FontSize), начертание шрифта «Bold» (Label.FontWeight), содержимое (текст) «Hello, XAML!» (Label.Content).

Необычным является декларация свойства Foreground:

```
<Label.Foreground>
  <SolidColorBrush>
    <SolidColorBrush.Color>
```

```
          Red
        </SolidColorBrush.Color>
      </SolidColorBrush>
    </Label.Foreground>
```

Значением свойства Foreground служит XAML-элемент SolidColorBrush (Кисть однородного цвета), для которого определено свойство Color. А вот для свойства Color установлено значение «Red». Именно такой цвет у выводимого на экран текста из элемента Label.

Собственно синтаксисом XAML мы подробно займемся в следующих главах, а сейчас покажем другую форму той же XAML-разметки, точнее приведем другой XAML-документ, результат отображения которого полностью совпадает с изображением на рис. 1.

```
<!-- Предисловие_02.xaml - автономный XAML-документ -->
<Label
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  FontSize = "30"
  FontWeight="Bold"
  Foreground="Red"
  Content = "Hello, XAML!»/>
```

Как видите, текст разметки сократился в три раза, но результат отображения тот же. В новом документе нет элемента Page, все четыре свойства элемента Label определены с помощью атрибутов, об элементе SolidColorBrush нет даже упоминания. О других различиях этих двух документов не будем упоминать, с ними познакомимся в основных главах книги.

Конечно, программисты предпочитают более короткий код. Но всегда ли это возможно и всегда ли целесообразно? Мы привели пример только некоторых из возможностей сокращения разметки, но в языке XAML их достаточно много, и каждая из них требует и от дизайнера, и от программиста понимания того, почему и что он сокращает, чем можно заменить конкретный фрагмент разметки и что нельзя удалить из разметки, не нарушив требуемую функциональность создаваемого XAML-документа. Затруднения могут возникать и в тех случаях, когда текст на языке XAML формируется автоматически, а от дизайнера или программиста требуется внести в него небольшие изменения.

Цель предлагаемой книги – изложить XAML в чистом виде, вне какой-либо из существующих инструментальных сред для автоматизации проектирования интерфейса пользователя или создания

приложений, ориентированных на UWP и Xamarin.Forms. Именно в таком виде материал по синтаксису и применению XAML будет полезен как для дизайнеров, так и для программистов. Книга служит введением в XAML, а также объясняет его синтаксические особенности и конструкции. В книге изложены основные концепции XAML: его элементы, свойства элементов, атрибуты, система зависимых свойств, механизм расширения разметки, присоединенные свойства, привязка данных, растровая и векторная графика, преобразования изображений, триггеры, анимация и т. д.

Кроме основного содержания (объяснительного контента), каждая глава включает примеры автономных XAML-документов, которые читатель может расширять и использовать в своих разработках.

Книга предназначена для профессиональных разработчиков (дизайнеров и программистов), использующих платформу .NET. Однако для изучения XAML нет необходимости устанавливать на компьютере те или иные специализированные программные продукты. Обработка автономных XAML-документов требует наличия на компьютере только платформы .NET Framework и браузера (например, Internet Explorer), имеющего доступ к этой платформе. На сегодняшний день, после разработки стандарта спецификации .NET Framework, эта платформа доступна во всех основных операционных системах, а браузер Internet Explorer распространяется свободно. Исходя из этих возможностей, в книгу включены примеры использования языка XAML только в виде автономных XAML-документов.

В книге не рассмотрены различия между существующими на сегодняшний момент диалектами XAML. Так как все иллюстративные эксперименты с XAML-документами выполнены с применением браузера Internet Explorer, то подробнее всего показаны возможности диалекта XAML, применяемого в UWP. Для знакомства с особенностями диалекта XAML, используемого в Xamarin.Forms, придется обратиться к документации по Xamarin.Forms. Для изучения особенностей взаимосвязей XAML-разметки с императивным кодом в WPF или в Silverlight следует познакомиться с публикациями по WPF или по Silverlight. Однако без изучения основных механизмов языка XAML, которым посвящена эта книга, изучить программирование для UWP и Xamarin.Forms можно только поверхностно, на уровне применения чужих заготовок и рецептов.

Конечно, все примеры XAML-документов в тексте снабжены иллюстрациями тех результатов, которые получены при их визуализации (как на рис. 1). Но будет полезно, если читатель наберет в любом текстовом редакторе (например, в Блокноте) код декларативной разметки XAML-документа и, сохранив его в файле с расширением «.xaml», передаст его браузеру для обработки. Особенно это важно для изучения особенностей привязки данных, трансформации изображений, триггеров, анимации, то есть в тех случаях, когда документ создает не статическую картинку, а формирует интерактивный интерфейс, реагирующий на действия пользователя или выполняющий анимацию.

Иногда в предисловиях к книгам авторы перечисляют условия, при которых усвоение материала книги невозможно. Например, помещают фразу: «Кто НЕ должен читать эту книгу!» Далее авторы перечисляют требования к подготовке читателя. В нашем случае требования к читателю предельно просты: он должен обладать минимальной компьютерной грамотностью, чтобы уметь в любом текстовом редакторе набрать текст XAML-документа, сохранить его в файле, а затем отправить браузеру для воспроизведения.

В Интернете опубликованы многочисленные статьи и достаточно подробные руководства по разработке XAML-документов. В них приводятся образцы XAML-разметки, предназначенные для иллюстрации особенностей применения того или иного механизма XAML. (В разделе «Литература и ссылки на электронные ресурсы» приведены ссылки на некоторые источники из Интернета.) Почему же трудно изучать язык XAML только по примерам XAML-документов?

Обычно приводимые примеры XAML-разметки снабжены подробными комментариями и могут служить хорошими образцами для самостоятельных разработок. Однако большинство образцов не иллюстрирует наиболее общих принципов синтаксиса языка, так как в текстах разметки используются многочисленные «недомолвки», позволяющие сократить текст за счет применения правил умолчания.

Например, в XAML-элементе Storyboard (сейчас не стоит объяснять назначение этого элемента) практически всегда опускают теги свойства Children. Никак не комментируют тот факт, что в список, введенный как значение свойства Children, можно непосредственно включать элементы анимации с типами производных от абстрактного класса AnimationTimeline, не «обрамляя» их

тегами элементов Storyboard. Такое сокращение текста разметки очень разумно в реальных проектах, но затрудняет понимание общих принципов синтаксиса XAML как языка декларативной разметки. Особенные трудности эти недомолвки и сокращения вызывают в тех случаях, когда текст на языке XAML формируется автоматически с помощью инструментов для визуального проектирования пользовательского интерфейса. Получив автоматически созданный XAML-документ, автор (дизайнер интерфейса) или программист-разработчик императивного кода должен понимать, какие фрагменты разметки являются существенными и где можно или нужно вручную внести изменения, чтобы сделать интерфейс пользователя наиболее соответствующим требованиям решаемой задачи.

Не приводя обзора глав книги (смотрите оглавление), отметим, что подробный предметный указатель позволяет использовать книгу в качестве справочника по элементам, свойствам, классам и перечислениям, применяемым в XAML-разметке.

Глава 1

XAML как XML-приложение

Язык декларативного программирования XAML (eXtensible Application Markup Language – произносится как «зэмл», а переводится как «расширяемый язык разметки приложений») является одним из языков, построенных на основе языка XML (eXtensible Markup Language – «расширяемый язык разметки»).

Глубокого изучения XML для понимания синтаксиса XAML не требуется, но нужно знать, что такое XML-документ, как определяются его элементы и атрибуты. Начнем с того, что XML (eXtensible Markup Language) – это не язык разметки, как, например, HTML, а основа и средство для разработки специализированных языков разметки.

Разметка (markup) – это добавление в текст документа специальных дескрипторов (именованных меток) для обозначения и выделения частей документа. Упомянутые дескрипторы разметки называются тегами. В общем случае выделяемый или размечаемый с помощью XML-средств фрагмент (участок) текста ограничен двумя тегами:

<*имя_тега*> и </*имя_тега*>.

Первый из них называют начальным тегом, в нем, кроме имени тега, могут помещаться атрибуты, разделяемые пробелами. Второй из тегов называют конечным тегом.

С помощью тегов:

- указываются границы участка (фрагмента) текста;
- определяется роль (назначение) выделяемого фрагмента в тексте документа;
- определяется расположение (позиция) фрагмента относительно других участков текста;
- определяется вложенность фрагментов текста;

- определяются связи фрагмента текста с ресурсами, размещёнными за пределами документа.

Фрагмент между начальным и конечным тегами вместе с этими тегами называют XML-элементом. При XML-разметке весь размечаемый текст ограничивается начальным и конечным тегами, которые определяют размечаемый текст как отдельный XML-документ. XML-документ называют корневым элементом (root element). В корневой элемент XML-документа обычно включают некоторый текст документа и набор элементов более низкого уровня. Каждый элемент вводится и ограничивается своей парой тегов. Имя тега определяет имя или название соответствующего XML-элемента. Таким образом, XML-документ в общем случае – это дерево элементов, в котором XML-документ – это основная единица или корень дерева.

В начальный тег каждого элемента (как уже упомянуто), кроме его названия, обычно включают атрибуты, определяющие какие-либо свойства элемента. Каждый атрибут имеет уникальное для данного тега имя и значение, отделяемое от имени атрибута знаком =. Значение атрибута задается в виде строки, ограниченной кавычками или апострофами. Количество атрибутов может быть произвольным. Их имена зависят от вида (типа) определяемого элемента.

Приведем синтаксически верный XML-документ:

```
<!-- №_01_00.xml -->
<Label FontSize = "30" Background="LightGray">
   Label - это "ярлык"!
</Label>
```

Первая строка документа – комментарий XML. Он начинается лексемой из четырех символов (<!--) и завершается лексемой из трех символов (-->). Между ними – достаточно произвольный текст. В приведенном XML-документе – только один элемент, вводимый тегом с именем Label. Зная, как используется элемент управления Label в технологиях WinForms и WPF, название этого элемента можно, наверное, переводить на русский язык как «ярлык» или «наклейка с надписью». Часто этот элемент называют «меткой», но для декларативных языков (к числу которых относится XML и созданные на его основе языки разметки) этот перевод мало подходит по смыслу. Вводя ярлык, нужно указать, что будет на нем написано (содержимое), каким шрифтом, на каком фоне и т. п. Именно

эта информация в данном случае определена в приведенном документе. Между его начальным тегом <Label…> и конечным тегом </Label> помещено содержимое элемента – в данном случае текст «Label – это"ярлык"!». В начальном теге для элемента Label определены два разделенных пробелом атрибута: FontSize и Background. Первый определяет размер шрифта текста, используемый при визуализации документа, второй задает цвет фона изображения элемента Label. Обратите внимание, что значения атрибутов заключены в кавычки. (Возможно применение и апострофов.)

Хотя приведенный текст соответствует правилам оформления XML-документа, но программа его обработки должна (по крайней мере) «знать», что означают использованные в декларации документа идентификаторы (Label, FontSize, Background и т. п.). Как же это становится известно программе, и какие программы могут правильно интерпретировать XML-документ?

В отличие от предшествующих языков, XML не является сам по себе языком разметки. Он не определяет теги разметки, а даёт возможность определять эти теги пользователю XML. Пользователь может вводить собственные имена тегов, либо брать за основу кем-то созданный язык разметки и применять, либо расширять набор тегов этого чужого языка.

Однако, создавая собственный язык разметки, его автор должен следовать строгим правилам языка XML. Одно из них заключается в требовании к структуре документа. Структура документа должна однозначно определять правила его интерпретации разными XML-процессорами. Именно документ с такой структурой считается корректным с точки зрения XML-языка (well_formed).

Язык разметки, созданный с помощью XML, называют XML-приложением (XML application). В настоящее время существует очень много XML-приложений, каждое из которых создано для тех или иных целей.

Документ, размеченный с помощью любого из XML-приложений, должен соответствовать минимальному набору правил XML. Для соблюдения этих правил и правил конкретного языка разметки предусматривается создание спецификации создаваемого языка разметки. Спецификация представляет собой набор требований, которым должен соответствовать правильно размеченный документ.

При разметке документа требуется определить его *структуру* и его *представление*. Представление (presentation) определяет внешний вид документа при его визуализации. В таком языке, как

HTML (который появился раньше, чем был разработан XML), правила представления документа и правила идентификации его элементов объединены. Точнее, набор тегов HTML содержит как теги объявления элементов, так и теги, определяющие их представление при отображении. К первым, т. е. к тегам разметки, в HTML относятся, например, теги, выделяющие заголовок документа или его тело. Представление, то есть формат отображения документа или его фрагмента, в HTML определяют теги, указывающие, например, нужный шрифт вывода текста.

В XML объединение тегов разметки с тегами представления элементов, по крайней мере, *не одобряется*. Не приводя примеров затруднений, которые возникают при одновременном включении в язык разметки как тегов, определяющих структуру документа, так и тегов для задания формата представления, отметим, что корректные XML-документы не должны содержать описаний форматов отображения элементов.

XML-документ определяет с помощью тегов разметки структуру документа, а форматы отображения элементов размещаются в отдельном документе (или в отдельном фрагменте документа), названном таблицей стилей (хотя термин «Style sheet» было бы точнее перевести как «лист стилей», но в русскоязычной литературе установилось название «таблица стилей»). Такое выделение таблицы стилей позволяет, меняя таблицу стилей, по-разному представлять документ, например, при выводе на печать и при отображении на экране.

Пространства имен

Так как в крупных проектах могут быть совместно использованы фрагменты, закодированные с помощью разных языков разметки, то возможно совпадение имен тегов из разных языков разметки. Для исключения такой возможной неоднозначности в XML введен механизм пространств имен. Конкретное пространство имен объявляется в XML-документе при помощи атрибута xmlns. Атрибут xmlns помещают в начальном теге того элемента, в котором использована разметка на соответствующем языке. Объявление пространства имен относится к тому элементу, в котором использован этот атрибут, и ко всем вложенным в него элементам. Название пространства имен должно быть уникальным для каждого XML-приложения.

Не углубляясь более в общие особенности построения XML-приложений, укажем только, что язык XAML – только одно из XML-приложений, причем существуют подмножества этого языка, ориентированные на применения в разных технологиях. (Назовем: Silverlight; WCF – Windows Communication Foundation; WPF – Windows Presentation Foundation; WWF – Windows Workflow Foundation; WCS – Windows CardSpace. И наконец, наиболее свежие: Universal Windows Platform – UWP и Xamarin.Forms.)

Для обозначения пространства имен при применении XAML в среде выполнения ОС Windows используется атрибут:

```
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
```

Строка – значение атрибута xmlns – это просто название, которое фирма Microsoft ввела для обозначения того пространства имен, которому «приписаны» элементы языка XAML в среде выполнения ОС Windows. Пространство имен xmlns определяет термины словаря языка XAML (Label, Button, ...), относящиеся к среде исполнения ОС Windows. При наличии в документе такого объявления пространства имен xmlns-термины (имена элементов и атрибутов) могут использоваться в декларации XAML-документа по умолчанию (то есть без явного указания того, что термины принадлежат именно к этому пространству имен). Обратим внимание, что строка, определяющая пространство имен xmlns, не имеет никакого отношения к интернет-адресации, несмотря на наличие префикса «http:». Добавление этого атрибута xmlns в начальный тег приведенного выше синтаксически верного XML-документа сделает его XAML-документом, пригодным для обработки и выполнения в среде ОС Windows:

```
<!-- №_01_01.xaml - автономный XAML-документ -->
<Label FontSize = "30" Background="LightGray"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation">
   Label - это "ярлык"!
</Label>
```

Автономные XAML-документы

Если сохранить приведенный документ в текстовом файле с расширением «.xaml» (используя кодировку UTF-8), то получим автономный XAML-файл или «автономный XAML-документ». Автономный XAML-файл можно обработать и визуализировать,

например, веб-браузером Internet Explorer (см. рис. 1.1). Для этого достаточно указать в браузере путь к файлу или щелкнуть мышью на имени файла в каталоге.

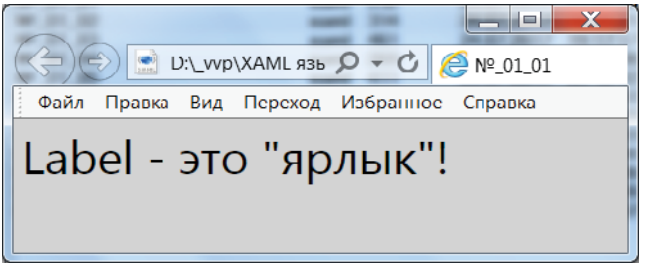

Рис. 1.1. Отображение автономного XAML-файла браузером

Следует отметить, что в ряде случаев для корректного отображения XAML-документа потребуется «настройка» браузера Internet Explorer.

Последовательность шагов настройки браузера:

1. В строке меню браузера выберите «Сервис».
2. В ниспадающем меню нажмите «Свойства браузера».
3. В диалоговом окне свойств браузера выберите «Безопасность».
4. В поле безопасности щелкните кнопку «Другой...».
5. В окне параметров безопасности установите флажки «Включить» для пункта «XAML-приложения веб-обозревателя» и для пункта «Свободный XAML».
6. Щелкните на кнопке «ОК».
7. Во всплывающем окне разрешите изменить настройку.
8. Для завершения настроек щелкните на кнопке «ОК».

Обратите внимание (см. рис. 1.1), что текст содержимого элемента Lable размещен в левом верхнем углу окна браузера, и все окно окрашено в цвет, определенный атрибутом Background. Немного усовершенствуем XAML-код. Добавим к элементу Label еще два атрибута: VerticalAlignment="Center" и HorizontalAlignment="Center". Эти атрибуты «заставят» размещать изображение элемента Label (вместе с содержимым) в центре окна браузера. Автономный XAML-документ примет вид:

```
<!-- №_01_02.xaml – автономный XAML-документ -->
<Label
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
```

```
   FontSize = "30"
   VerticalAlignment = "Center"
   HorizontalAlignment = "Center"
   Background="LightGray">
      Label - это "ярлык"!
</Label>
```

Сохраним полученный XAML-документ в файле №_01_02.xaml (используя кодировку UTF-8). Его представление в окне браузера показано на рис. 1.2.

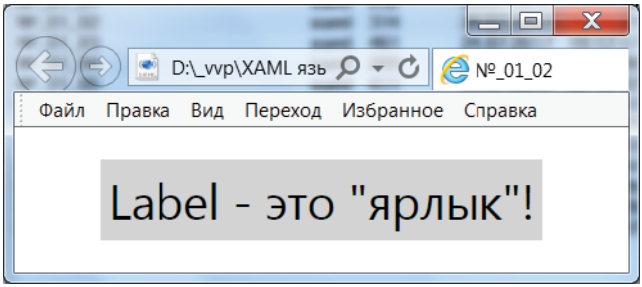

Рис. 1.2. Представление автономного XAML-файла в окне браузера

Обратите внимание, что в этом случае для размещения текста из ярлыка (из элемента Lable) выделен прямоугольник, и только он закрашен цветом, определенным атрибутом Background. Каким образом можно явно определить размеры этого прямоугольника, рассмотрим позже. В данном документе они выбираются по умолчанию с учетом размеров надписи.

Синтаксис XML и синтаксис XAML

Прежде чем переходить к XAML, еще раз остановимся на тех особенностях *синтаксиса XML*, которые нужно учитывать, используя XAML-разметку.

Итак, нам понятен формат декларации (объявления) XML-элемента:

<***имя_элемента*** *атрибуты*>
Содержимое
</***имя_элемента***>

Началом объявления элемента служит начальный тег с именем элемента. В конце декларации элемента – закрывающий тег с именем того же элемента. Особое назначение в тексте разметки

имеют символы: пробел, табуляция, перевод строки, знак равенства (присваивания), кавычка, апостроф и символы < («меньше») и > («больше»). Эти символы зарезервированы и играют служебные роли. Они используются в качестве разделителей имен элементов, имен атрибутов и значений атрибутов, поэтому эти символы недопустимы в именах элементов. Закрывающий тег никогда не имеет атрибутов, и его имя совпадает с именем элемента. Если в элементе отсутствует содержимое (или оно вводится с помощью специального атрибута, как будет показано позже в примерах XAML-документов), то такой «пустой» элемент можно описать одним тегом:

<*имя_элемента атрибуты*/>

Это так называемая самозакрывающаяся форма записи тега.

Между открывающей угловой скобкой тега (<) и именем элемента не должно быть пробельных символов, но лишние пробелы в любом другом месте тега и в декларации элемента допустимы. Это позволяет разбивать объявление элемента на несколько строк, что продемонстрировано выше в тексте примера №_01_02.xaml.

При разметке документа нужно соблюдать правила вложения его элементов. Закрывающий тег элемента должен помещаться после открывающего тега. Открывающий и закрывающий теги элемента должны размещаться в пределах охватывающего их элемента более «высокого» уровня.

Внутри объявления элемента (между тегами) все, что не соответствует по синтаксису объявлениям вложенных элементов, воспринимается как текст. В тексте не должны присутствовать символы, которые «зарезервированы» для специальных целей. Например, символ < воспринимается как начало тега и не может применяться непосредственно в тексте. Таким образом, поместив в качестве содержимого в документ текст: «3 < 5», получим ошибку при воспроизведении документа.

В случае необходимости для правильного представления в содержимом зарезервированных символов используются подстановки вида «&*код*;». Такие конструкции иногда называют эскейп-последовательностями. В руководствах по XML такую подстановку называют символьная сущность (character entity). Код в эскейп-последовательности – это условное обозначение (мнемоника) зарезервированного символа либо символ #, за которым следует числовое значение (десятичное или шестнадцатеричное целое

число). Чтобы не отсылать читателя к таблицам специальных символов (см., например, https://dev.w3.org/html5/html-author/charref), приведем наиболее полезные для разметки XML- и XAML-документов:

- & или & – амперсанд (&);
- > или > – правая угловая скобка (>);
- < или < – левая угловая скобка (<);
- ' или ' – апостроф (');
- " или " – кавычка (");
-   или   – неудаляемый пробел;
-
 или
 – переход на новую строку.

В нашем примере правильной будет запись: "3 < 5". Такое содержимое отобразится браузером в виде текста: 3 < 5.

Как уже показано, элементы XML могут включать атрибуты, которые служат для декларации значений свойств элементов или для определения его поведения при отображении документа. Значения атрибутов задаются в виде строк, ограниченных либо кавычками (""), либо апострофами (''). Если в строку, определяющую значение атрибута, необходимо поместить кавычку ("), то строка обрамляется апострофами, если в строке нужен апостроф (') – строка обрамляется кавычками. Если в строке нужны и апостроф, и кавычка – для их представления используют подстановки: (') – для апострофа и (") – для кавычки.

При использовании автономных XAML-файлов применяется правило XML: все стоящие рядом пробельные символы заменяются одним пробелом. Напомним, что к пробельным символам относятся не только пробелы, но и символы табуляции, перехода к началу строки и перехода на следующую строку. Чтобы полностью избежать применения этого правила, применяется атрибут

```
xml:space="preserve"
```

Если нужно только в одном месте XAML-документа в тексте поставить рядом несколько пробелов, то можно вместо названного атрибута использовать для каждого из пробелов эскейп-последовательность « ».

Обратите внимание, что, в отличие от HTML, в записи имен элементов и атрибутов XML учитывается регистр. Существуют еще некоторые особенности XML, которые не влияют на синтаксис XAML, поэтому объяснять их нет необходимости.