Содержание

Об авторе

Элвин Эшкрафт – инженер-программист, лидер технического сообщества с 25-летним стажем разработки ПО. Работает преимущественно с Microsoft Windows, вебом и облачными технологиями, в сферу интересов входит в основном здравоохранение. 11 раз был удостоен звания Microsoft MVP, в последний раз как разработчик Windows.

Элвин работает ведущим инженером-программистом в глобальной компании Allscripts по разработке ПО в области здравоохранения со штабквартирой в Филадельфии. Также является членом правления фонда TechBash, где отвечает за организацию ежегодных конференций разработчиков TechBash. Раньше работал в компаниях Oracle, Genzeon, CSC и ITG Pathfinders.

Родился в Аллентайне, штат Пенсильвания, теперь проживает в районе Уэст-Гроув, тоже в Пенсильвании, с женой и тремя детьми.

Хочу поблагодарить всех близких, кто поддерживал меня, особенно свою жену Стелену и трех дочерей.

О рецензенте

Ник Рэндольф в настоящее время возглавляет компанию Built to Roam, специализирующуюся на разработке мобильных приложений с развитым функционалом. Ник удостоен звания Microsoft MVP в знак признания заслуг и за опыт работы на платформах Microsoft.

Ника не раз приглашали на различные мероприятия, в т. ч. Tech Ed и Ignite Australia & NZ, DDD, NDC и в местные группы пользователей. Он автор нескольких книг по разработке для Windows в среде Visual Studio, был членом жюри в финалах международного конкурса Imagine Cup. Ник работал над многочисленными мобильными приложениями и помогал сотням разработчиков создавать свои мобильные приложения. Участвовал в разработке приложений для таких хорошо известных торговых марок, как Domain.com.au, ninemsn, AFL, NRL, Qantas, JB Hi-Fi, NAB, Stan и Boost Juice.

Предисловие

WinUI 3.0 – первый шаг Microsoft в направлении унифицированной платформы разработки Windows. Эта инициатива, получившая название Project Reunion, является попыткой объединить UWP, WPF и другие каркасы для разработки пользовательского интерфейса (UI) на одной платформе. WinUI позволяет быстро создавать приложения Windows, стиль которых адаптируется к платформе. По мере отработки WinUI разработчики получат возможность выбирать в качестве целевой платформы настольные компьютеры под управлением Windows, Xbox, HoloLens, Surface Hub и др.

Разработчики приложений Win32 также могут пользоваться элементами управления WinUI, чтобы модернизировать внешний вид своих приложений с помощью островков XAML из комплекта инструментов Windows Community Toolkit. Этот комплект инструментов с открытым исходным кодом содержит десятки элементов управления и других вспомогательных библиотек для WinUI, UWP и Win32. Вы научитесь выбирать подходящие элементы управления для своих приложений и использовать их в нескольких проектах.

Из этой книги вы узнаете, как разрабатывать, отлаживать, собирать и развертывать приложения с помощью Visual Studio и облачных инструментов из Microsoft Azure и GitHub. Вы откроете для себя, как сделать так, чтобы ваше приложение WinUI могло попасть в руки пользователей как из потребительского, так и из корпоративного сектора. Дочитав книгу до конца, вы будете хорошо понимать, как создавать, модернизировать и распространять приложения Windows на платформе WinUI 3.0.

Предполагаемая аудитория

Эта книга адресована всем, кто хочет разрабатывать приложения для Windows 10 с помощью WinUI 3.0. Читатели, знакомые с UWP, WPF и WinForms, тоже смогут почерпнуть из нее много полезного для углубления своих представлений о разработке в Windows и модернизации существующих приложений. Для чтения книги необходимо знакомство с языком C# и платформой .NET, но предварительные знания о разработке интерфейсов с применением WinUI, UWP или XAML не требуются.

Структура книги

В главе 1 «Введение в WinUI» рассматривается история каркасов UI в Windows и истоки WinUI. Мы создадим свой первый проект WinUI 3.0 в Visual Studio 2019.

В главе 2 «Конфигурирование среды разработки и создание проекта» мы познакомимся с проектом, над которым будем работать на протяжении большей части этой книги. Мы объясним внутреннее устройство проекта WinUI и поговорим об основных особенностях платформы, в т. ч. о жизненном цикле приложения и привязке к данным.

В главе 3 «MVVM как средство, обеспечивающее удобство сопровождения и тестирования» объясняется паттерн проектирования «Модель–представление–представление модели» (MVVM), играющий важнейшую роль в разработке приложений WinUI. Вы также освоите основы автономного тестирования проектов WinUI.

В главе 4 «Дополнительные концепции MVVM» мы продолжим изучение MVVM в WinUI, добавив NuGet-пакет **внедрения зависимостей** (dependency injection – DI) и сервисные классы для страничной навигации и выборки данных.

В главе 5 «Элементы управления WinUI» изучаются элементы управления, имеющиеся в галерее элементов управления XAML для WinUI 3.0. Это приложение с открытым исходным кодом иллюстрирует элементы управления (с примерами), доступные в проектах WinUI. Затем мы добавим два новых элемента из галереи в свой проект.

Глава 6 «Использование данных и служб» продолжает разговор о службах данных, начатый в главе 4, и добавляет хранилище на основе SQLite для сохранения данных нашего приложения. Здесь же мы познакомимся с механизмами контроля, которые препятствуют вводу недопустимых данных пользователями.

В главе 7 «Система проектирования текучих интерфейсов для приложений Windows» рассматривается кросс-платформенная система проектирования текучих интерфейсов от Microsoft. Дается исторический обзор эволюции идей проектирования интерфейсов в Windows. Здесь же мы познакомимся с редактором Fluent XAML Theme Editor, предназначенным для создания собственной темы интерфейса для приложения WinUI.

В главе 8 «Построение приложений WinUI в .NET 5» объясняется, как создать WinUI-интерфейс для проекта классического приложения, ориентированного на платформу .NET 5, а не UWP. Здесь же иллюстрируется создание элемента управления в .NET 5, допускающего использование в нескольких проектах классических WinUI-приложений.

В главе 9 «Улучшение приложений с помощью Windows Community Toolkit» мы познакомимся с комплектом инструментов с открытым исходным кодом Windows Community Toolkit и входящим в него приложением, иллюстрирующим применение Windows.Controls, вспомогательных классов и других библиотек, которые мы добавим в свои проекты.

Глава 10 «Модернизация существующих приложений Win32 с помощью островков XAML» посвящена элементам управления из Windows Community Toolkit на основе технологии островков XAML. Объясняется, как интегрировать их в приложения WPF и WinForms в качестве первого шага на пути к модернизации приложения и полному переходу на WinUI.

В главе 11 «Отладка WinUI-приложений в Visual Studio» мы ближе познакомимся с инструментами отладки XAML, имеющимися в Visual Studio 2019.

Они позволяют разработчикам WinUI оптимизировать и отладить основанный XAML пользовательский интерфейс и код привязки к данным.

В главе 12 «Размещение приложения ASP.NET Core Blazor в WinUI» описывается, как создать и развернуть одностраничное приложение (single-page application – SPA) Blazor в Azure с помощью Visual Studio Code и действий GitHub. Затем развернутое приложение Blazor будет размещено в приложении Windows с помощью нового элемента управления WebView2.

В главе 13 «Сборка, выпуск и мониторинг приложений с помощью Visual Studio App Center» рассматриваются имеющиеся в Visual Studio App Center средства сборки, развертывания и мониторинга приложений Windows. Вы научитесь оснащать свой код инструментальными средствами для получения в реальном времени аналитических данных и данных об аварийных отказах развернутых приложений.

Глава 14 «Упаковка и развертывание WinUI-приложений» посвящена различным методам развертывания WinUI-приложений. Описывается, как использовать для этой цели Visual Studio, Microsoft Partner Center, Microsoft Store и WinGet.

Что необходимо для чтения этой книги

Чтобы получить максимум пользы от чтения книги, необходимо установить Visual Studio 2019 версии 16.9 или более поздней (https://visualstudio.microsoft.com/downloads/) со следующими рабочими нагрузками и последней версией NuGet-пакета WinUI 3.0 (https://marketplace.visualstudio.com/items?itemName=Microsoft-WinUI.WinUIProjectTemplates):

- Universal Windows Platform Development;
- .NET Desktop Development (включает .NET 5).

Код и инструкции должны работать и для следующих версий рекомендуемого ПО. Самая свежая информация о WinUI 3.0 находится на сайте Microsoft Docs по адресу https://docs.microsoft.com/en-us/windows/apps/winui/winui3/#installwinui-3-preview-4.

Программное и аппаратное обеспечение, используемое в книге	Требования к ОС
Visual Studio 2019 версии 16.9	Windows 10 версии 1803 или более поздней
Шаблоны WinUI 3.0	
Visual Studio Code	Windows 10, macOS или Linux

Поскольку в 2021 году WinUI 3 и Project Reunion все еще пребывают в фазе активной разработки, не исключено, что в будущих версиях имена некоторых упоминаемых в книге проектов, пакетов и библиотек изменятся. Островки XAML поначалу будут недоступны, а UWP-клиенты поддерживаются не полностью. С полным перечнем всего, что планируется включить в первую стабильную версию WinUI 3.0, можно ознакомиться в дорожной карте команды разработчиков на GitHub по адресу https://github.com/microsoft/microsoft-ui-xaml/blob/master/docs/roadmap.md#winui-30-feature-roadmap.

Для главы, посвященной веб-разработке с применением Blazor, рекомендуется скачать Visual Studio Code (https://code.visualstudio.com/). Если вы еще не установили рабочую нагрузку .NET Desktop Development для Visual Studio, то для разработки с применением Blazor понадобится .NET 5 SDK (https://dotnet.microsoft.com/download/dotnet/5.0).

Тем, кто пользуется цифровой версией книги, мы рекомендуем набирать код самостоятельно или скачать его из репозитория на GitHub (ссылка приведена в следующем разделе). Это поможет избежать потенциальных ошибок из-за копирования и вставки кода.

Дополнительные сведения о концепциях разработки для Windows можно почерпнуть из пособия «Develop Windows 10 applications Learning Path» на сайте **Microsoft Learn** (https://docs.microsoft.com/en-us/learn/paths/develop-windows10-apps/).

Скачивание кода примеров

Код примеров для этой книги можно скачать, зайдя в свою учетную запись на сайте www.packt.com. Если вы купили книгу где-то еще, то можете зарегистрироваться на странице www.packt.com/support, тогда файлы будут отправлены вам по электронной почте.

Код примеров размещен также на сайте GitHub по адресу https://github.com/PacktPublishing/Learn-WinUI-3.0. Все обновления выкладываются в репозиторий на GitHub.

В разделе https://github.com/PacktPublishing/ есть и другие пакеты кода для нашего обширного каталога книг и видео. Не пропустите!

Скачивание цветных изображений

Мы также предлагаем PDF-файл, содержащий цветные изображения снимков экрана и рисунков. Его можно скачать по адресу https://static.packt-cdn.com/downloads/9781800208667_ColorImages.pdf.

Обозначения и графические выделения

В этой книге применяется ряд соглашений о наборе текста.

`CodeInText`: код в тексте, имена таблиц базы данных, папок и файлов, расширения имен файлов, пути к файлам и данные, вводимые пользователем. Например: «Для перехвата события запуска следует переопределить метод `OnLaunched` класса приложения».

Отдельно стоящие фрагменты кода набраны так:

```
while (query.Read())
  {
```

```
    var medium = new Medium
    {
      Id = query.GetInt32(0),
      Name = query.GetString(1),
      MediaType = (ItemType)query.GetInt32(2)
    };
```

Желая привлечь внимание к какой-то части кода, мы выделяем ее полужирным шрифтом, например:

```
private void OnSuspending(object sender, SuspendingEventArgs e)
{
  var deferral = e.SuspendingOperation.GetDeferral();
  deferral.Complete();
}
```

Полужирный: новые термины и важные слова, а также части пользовательского интерфейса. Так выделяются команды меню и текст в диалоговых окнах, например: «После того как приложение начнет выполняться, вы увидите, что новый стиль применен к кнопкам **Submit** и **Cancel**, хотя непосредственно к ним вы никаких стилей не применяли».

Советы выглядят так.

Примечания выглядят так.

Часть I

ВВЕДЕНИЕ В ПРИЛОЖЕНИЯ WINUI И WINDOWS

WinUI 3.0 – новый каркас UI от Microsoft для разработчиков на платформе Windows. В этой части мы начнем с изучения недавней истории XAML и каркасов UI для Windows, а затем познакомим читателей с WinUI. Для изучения концепций WinUI мы создадим простой проект, а потом добавим в него элементы управления и различные возможности, следуя паттернам проектирования и рекомендованным практикам, в т. ч. MVVM (Model-View-ViewModel). Мы позаботимся о тестопригодности проекта WinUI и воспользуемся механизмом внедрения зависимостей (DI) для включения зависимостей от служб в компоненты приложения.

В эту часть входят следующие главы:

- глава 1 «Введение в WinUI»;
- глава 2 «Конфигурирование среды разработки и создание проекта»;
- глава 3 «MVVM как средство, обеспечивающее удобство сопровождения и тестирования»;
- глава 4 «Дополнительные концепции MVVM»;
- глава 5 «Элементы управления WinUI»;
- глава 6 «Использование данных и служб».

Глава 1

Введение в WinUI

WinUI – это набор элементов управления и библиотек с открытым исходным кодом, предназначенных для использования в приложениях для **универсальной платформы Windows** (Universal Windows Platform – UWP) и Win32. Разработчики для UWP используют **пакет средств разработки** (**SDK**) и должны указывать версию SDK в свойствах проекта. Выделив элементы управления UWP и компоненты пользовательского интерфейса (**UI**) из Windows SDK и выпустив их в виде библиотек с открытым исходным кодом под названием WinUI, Microsoft смогла сократить цикл разработки по сравнению с самой Windows (поскольку версии Windows SDK привязаны к версиям Windows). Такое разделение позволяет также использовать элементы управления в более старых версиях Windows 10. Хотя в настоящее время рекомендуется строить приложения UWP и Win32 с помощью WinUI, важно понимать место WinUI и UWP в более широкой картине разработки для Windows.

Из этой книги вы узнаете, как создавать приложения для Windows с применением библиотек WinUI 3.0. На протяжении всей книги мы будем разрабатывать реальное приложение, пользуясь рекомендуемыми паттернами и приемами.

Прежде чем приступать к созданию приложения WinUI, важно хорошо ориентироваться в разработке клиентов Windows, в различных типах разметки UI на **расширяемом языке разметки приложений** (Extensible Application Markup Language – **XAML**) и в отличиях WinUI от других каркасов разработки классических приложений для Windows. Поэтому мы начнем с рассмотрения основ UWP и WinUI.

В этой главе рассматриваются следующие темы:

- что такое UWP и почему Microsoft создала еще один каркас разработки приложений;
- как можно использовать XAML для создания замечательных пользовательских интерфейсов, работающих на устройствах разных размеров и из разных семейств;
- зачем был создан WinUI и как он соотносится с UWP;
- каково место WinUI в общей картине средств разработки для Windows;
- что несет с собой WinUI 3.0.

Не волнуйтесь! Рассмотрение этих основополагающих вещей не займет много времени, зато поместит разработку приложения WinUI в нужный

контекст. Уже в следующей главе мы начнем писать код своего первого проекта WinUI.

Технические требования

Для проработки представленных в этой главе примеров потребуется следующее программное обеспечение:

- Windows 10 версии 1803 или более поздней. Узнать версию своей Windows вы можете в разделе **Параметры | Система | О программе**;
- Visual Studio 2019 версии 16.9 или более поздней со следующими рабочими нагрузками: .NET Desktop Development и UWP Development;
- шаблоны проектов WinUI 3.0 – на момент написания книги эти шаблоны можно скачать с сайта Visual Studio Marketplace по адресу https://marketplace.visualstudio.com/items?itemName=Microsoft-WinUI.WinUIProjectTemplates. После выпуска WinUI 3.0 шаблоны, вероятно, будут включены в состав Visual Studio.

Исходный код для этой главы доступен на GitHub по адресу https://github.com/PacktPublishing/-Learn-WinUI-3.0/tree/master/Chapter01.

В разделе сайта Microsoft Docs, посвященном WinUI 3.0, имеется обновленное пособие по подготовке рабочей станции к разработке для WinUI: https://docs.microsoft.com/en-us/uwp/toolkits/winui3/.

До UWP – приложения Windows 8 XAML

Перед тем как в 2015 году в Windows 10 появились приложения UWP, существовали приложения XAML для Windows 8 и 8.1. Синтаксис XAML и многие **интерфейсы прикладного программирования** (**API**) не изменились, а сама идея стала очередной попыткой Microsoft добиться универсализации приложений на настольной, мобильной и других платформах (Xbox, дополненная реальность и т. д.). XAML-приложение можно было написать для Windows 8 и Windows Phone. При этом генерировались различные наборы двоичных файлов, которые можно было устанавливать на ПК или Windows Phone.

У этих приложений было много других ограничений, снятых в современных приложениях UWP. Например, они могли работать только в полноэкранном режиме, как показано на рис. 1.1.

Многие другие ограничения приложений для Windows 8 были ослаблены или полностью устранены в приложениях UWP. Эти изменения документированы на рис. 1.2.

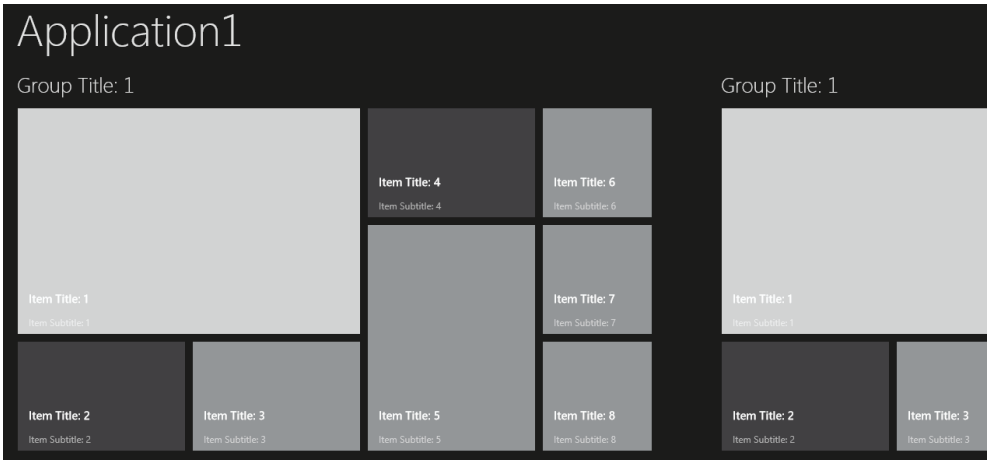


Рис. 1.1 ❖ Полноэкранное приложение для Windows 8 (взято с сайта Stack Overflow; воспроизводится по лицензии CC BY-SA 4.0. https://creativecommons.org/licenses/by-sa/4.0/)

	Приложения Windows и XAML	Приложения Windows 10 UWP
Тип окна	Только полноэкранное	С изменяемым размером
Тип устройства	Работает только на ПК	Различные устройства под управлением Windows 10
Количество экземпляров	1	1 (по умолчанию) или несколько
Поддержка консольных приложений	Нет	Да
Доступ к файловой системе	В песочнице – только локальное хранилище	По умолчанию в песочнице Приложение может запросить дополнительный доступ к пользовательским папкам и съемным устройствам

Рис. 1.2 ❖ Сравнение приложений для Windows 8 и Windows 10

Дизайн UI-приложения для Windows

Термином *стиль Metro* обозначали дизайн и структурную схему расположения приложений для Windows 8. Стиль Metro предназначался для сенсорного ввода, а также ввода с помощью мыши, клавиатуры или стилоса. Движущей силой, стоявшей за стилем Metro, стал вывод на рынок Windows Phone. Впоследствии, с появлением линейки устройств Microsoft Surface, стиль Metro Style превратился в «современный дизайн UI» (Modern UI). Некоторые черты Metro живы и поныне, в приложениях UWP и в Windows 10.

Живые плитки родились вместе со стилем Metro. Эти плитки на начальном экране Windows 8 и в меню Пуск Windows 10 могут показывать обновления без открытия соответствующего приложения. Большинство приложений, написанных самой Microsoft для Windows, поддерживают живые плитки. При-

ложение «Погода» может показывать на плитке актуальную информацию о погоде в зависимости от текущего местоположения пользователя. Живые плитки мы будем рассматривать подробнее в главе 5.

Windows Runtime (WinRT)

Еще один термин, уходящий корнями в разработку приложений для Windows 8, – WinRT. Буквы RT стали источником множества недоразумений. WinRT – сокращение от Windows Runtime, набора API, лежащего в основе XAML-приложений для Windows. Существовала также версия Windows 8 под названием Windows RT, которая поддерживала процессоры с архитектурой **Advanced RISC Machines** (**ARM**). Первым ПК в линейке Surface стал Surface RT, который работал под управлением операционной системы Windows 8 RT.

Хотя слово WinRT и по сей день можно использовать для обозначения WinRT API, потребляемых приложениями UWP, встречается оно нечасто. Мы тоже будем избегать его в этой книге, употребляя термины UWP или Windows API.

Отпор пользователей и путь к Windows 10

Как бы упорно Microsoft ни старалась увлечь пользователей современным дизайном UI, новой моделью приложений, устройствами Surface PC и ОС Windows 8 и 8.1, идея полноэкранного приложения, рассчитанного в первую очередь на сенсорный интерфейс и отводившего ПК второстепенную роль, так и не овладела умами потребителей. Выяснилось, что пользователи очень любят меню Пуск, к которому привыкли за годы работы с Windows XP и Windows 7.

Следующий шаг в разработке приложений для Windows оказался большим – настолько большим, что Microsoft приняла решение пропустить номер и сразу перейти от версии Windows 8.1 к Windows 10.

Windows 10 и разработка приложений UWP

Сделав гигантский скачок – выпуск Windows 10, – Microsoft включила в новую систему все лучшее, что зарекомендовало себя в предыдущих версиях. Меню Пуск вернулось, но стало сильно напоминать начальный экран Windows 8. Помимо алфавитного списка всех установленных приложений, имеется допускающая изменение размера область закрепленных плиток приложений. На самом деле при работе Windows в режиме планшета меню Пуск можно преобразовать в начальный экран в стиле Windows 8, который больше подходит для сенсорных устройств.

Вместе с Windows 10 Microsoft предложила разработчикам и UWP-приложения. Хотя корнями они уходят в XAML-приложения для Windows 8, есть и существенные различия, которые дают значительные преимущества при создании приложений для новой платформы.

Ключевое преимущество – универсальность приложений. Microsoft собирает версии Windows 10, работающие на устройствах из разных семейств, а именно:

- настольные (ПК);
- Xbox;
- мобильные (Windows Phone);
- HoloLens;
- IoT (интернет вещей);
- IoT без интерфейса;
- коллективные (Surface Hub).

Разработчики могут ориентировать UWP-приложения на любое из этих устройств. Существует единый базовый набор Windows API, общий для всех целевых устройств, и специализированные SDK, содержащие API для конкретных устройств из некоторых семейств, например Mixed Reality Toolkit (для устройств дополненной реальности) и SDK для HoloLens. UWP позволяет создать один проект приложения для многих семейств, скажем для настольных компьютеров, Xbox и коллективной платформы.

Поскольку UWP XAML для построения интерфейса приложения один и тот же, кривая обучения кросс-платформенной разработке не так крута, а возможности повторного использования кода очень широки. По самой своей природе XAML обеспечивает гибкость, необходимую для адаптации к различным размерам и отношениям сторон.

Выбор языка при разработке на платформе UWP

Хотя сами UWP API написаны на C++, разработчику UWP-приложений на выбор предоставляется несколько языков. UWP-проект может быть создан на любом из следующих популярных языков:

- C#;
- C++;
- F#;
- Visual Basic .NET (VB.NET);
- JavaScript.

Возможно, вас удивляет присутствие в этом списке JavaScript. Во времена Windows 8.x разработчики могли создавать приложения на JavaScript с помощью API, известного под названием WinJS. Теперь Microsoft создала ветвь React Native for Windows для разработчиков в Windows. Такие клиентские JavaScript-приложения имеют полный доступ к тем же Windows API, что и другие UWP-приложения, их можно упаковывать и развертывать через магазин Windows Store.

React Native for Windows – это проект с открытым исходным кодом, размещенный на GitHub по адресу https://github.com/Microsoft/react-native-windows.

Хотя многие UWP-приложения для Windows 10, созданные самой Microsoft, написаны на C++, большинство сторонних разработчиков предпочитают C#. Мы в этой книге тоже будем использовать C#.

Снятие ограничений

Как мы уже обсуждали, приложения для Windows 8 имели ряд ограничений, которые были ослаблены или вовсе сняты в UWP.

Прежде всего современные UWP-приложения могут работать в окнах изменяемого размера, как и любые другие классические приложения Windows. Но за это приходится платить – разработчик должен сам проверять и обрабатывать изменение размера приложения, правда, сам размер может быть почти любым. В силу своей динамической природы XAML отлично справляется с изменением размера, но до определенного нижнего предела, а потом нужно использовать полосы прокрутки.

Для конечных пользователей одним из преимуществ UWP-приложений является внутренне присущая им безопасность, поскольку они имеют лишь ограниченный доступ к файловой системе ПК. По умолчанию каждое приложение может обращаться только к собственному локальному хранилищу. В 2018 году команда разработчиков Windows объявила о новой возможности для авторов UWP-приложений. Благодаря добавлению конфигурационных параметров приложение может запросить доступ к дополнительным частям файловой системы, в том числе:

- пользовательским библиотекам, включая Documents, Pictures, Music и Videos;
- загрузкам;
- съемным устройствам.

Можно запросить и дополнительные права на доступ к файловой системе. Полный перечень см. в документации по адресу https://docs.microsoft.com/en-us/windows/uwp/files/file-access-permissions.

Все запрашиваемые дополнительные разрешения объявляются в параметрах приложения в Windows Store.

Для UWP-приложений в Windows 10 возможны некоторые менее распространенные сценарии. Добавив конфигурационные параметры и специальный код в начале программы, разработчик может разрешить запуск нескольких экземпляров своего приложения. Хотя весь смысл UWP-приложения вроде бы состоит в его пользовательском XAML-интерфейсе, теперь возможно написать консольное UWP-приложение. Это приложение будет запускаться из командной строки и иметь доступ к универсальной среде выполнения C. Разработчики, желающие попробовать свои силы в написании консольных приложений, могут найти шаблоны на сайте Visual Studio Marketplace по адресу https://marketplace.visualstudio.com/items?itemName=AndrewWhitechapelMSFT.ConsoleAppUniversal.

Обратная совместимость UWP

UWP-приложения несовместимы ни с какой версией Windows, предшествующей Windows 10. Кроме того, любое UWP-приложение должно объявить **целевую версию** (Target Version) и **минимальную версию** (Minimum Version) Windows, с которыми оно совместимо. Целевая версия – это рекомендуемая вами версия, в которой доступна вся функциональность приложения. Ну а минимальная, как легко догадаться, – версия Windows, которая должна быть установлена, чтобы приложение вообще можно было скачать из Windows Store.

Visual Studio попросит выбрать эти версии при создании нового UWP-проекта. Если они совпадают, то задача упрощается. Приложению будут доступны все API из данной версии SDK. Если целевая версия больше минимальной, то нужно будет добавить условный код, чтобы сделать доступными средства из версий, больших минимальной. Приложение должно быть все же полезно пользователям, работающим с минимальной версией, в противном случае рекомендует увеличить минимальный номер. То же самое рекомендуется сделать, если какие-то API или элементы управления, появившиеся в более поздних версиях, принципиально важны для работы приложения.

Дополнительные сведения о написании условного или зависящего от версии кода см. в документации по адресу https://docs.microsoft.com/en-us/windows/uwp/debug-test-perf/version-adaptive-code.

Если вы пишете библиотеки .NET, на которые будет ссылаться ваш UWP-проект, и хотите сделать их доступными на других платформах, например из мобильного приложения на базе Xamarin, то целевой версией для разделяемой библиотеки должна быть .NET Standard. В настоящее время самой распространенной версией .NET Standard является .NET Standard 2.0. Чтобы можно было сослаться на проект .NET Standard 2.0 из UWP-проекта, целевая версия UWP-проекта должна иметь номер 16299 или выше.

Основное преимущество WinUI по сравнению с UWP заключается в меньшей зависимости приложений Windows от конкретной версии Windows. Элементы управления, стили и API вынесены за пределы Windows SDK и сопровождаются отдельно. На момент написания книги минимальной версией, необходимой для приложения WinUI 3.0, была 17134, а целевая версия должна была быть не ниже 18362. Посмотрите в документации по WinUI текущие требования к минимальной версии.

Есть надежда, что по мере развития проекта WinUI больше элементов управления и возможностей станет доступно в большем числе поддерживаемых версий Windows 10.

Что такое XAML?

XAML основан на расширяемом языке разметки (**XML**). И на первый взгляд, это хорошо, потому что XML – гибкий язык разметки, знакомый большинству разработчиков. Он действительно гибкий и мощный, но имеет кое-какие недостатки.

Основная проблема с реализациями XAML от Microsoft заключается в том, что с годами для различных платформ разработки было создано множество вариаций языка XAML. В настоящее время приложения UWP, **Windows Presentation Foundation** (**WPF**) и Xamarin.Forms пользуются XAML в качестве языка разметки UI. Но все эти реализации имеют разные схемы, а разметку нельзя перенести с одной платформы на другую. В прошлом приложения для Windows 8, Silverlight и Windows Phone тоже пользовались XAML с разными схемами.

Если вы никогда не работали с XAML раньше, то, наверное, горите желанием увидеть пример разметки UI. В следующем фрагменте определена сетка `Grid`, содержащая несколько простых элементов управления WinUI (код для этой главы можно скачать с GitHub по адресу https://github.com/PacktPublishing/-Learn-WinUI-3.0/tree/master/Chapter01):

```
<Grid Width="400" Height="250" Padding="2"
    HorizontalAlignment="Center"
    VerticalAlignment="Center">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="Auto"/>
        <ColumnDefinition Width="*"/>
    </Grid.ColumnDefinitions>

    <TextBlock Grid.Row="0" Grid.Column="0"
               Text="Name:"
               Margin="0,0,2,0"
               VerticalAlignment="Center"/>
    <TextBox Grid.Row="0" Grid.Column="1"
             Text=""/>
    <Button Grid.Row="1" Grid.Column="1" Margin="0,4,0,0"
            HorizontalAlignment="Right"
            VerticalAlignment="Top"
            Content="Submit"/>
</Grid>
```

Рассмотрим этот фрагмент XAML по частям. На верхнем уровне окна UWP-приложения расположена страница `Page`. В UWP навигация организована постранично, и эта навигация верхнего уровня имеет место в корневом контейнере `Frame` в файле `App.xaml` проекта. Подробнее о страничной навигации мы поговорим в главе 4. Элемент `Page` должен содержать единственный до-

черний элемент, обычно какую-то макетную панель типа Grid или StackPanel. По умолчанию вставляется панель Grid. Другие типы панелей, подходящих на роль родительского контейнера, мы обсудим в следующей главе. Я изменил некоторые параметры Grid.

Свойства Height и Width задают статический размер, а свойства HorizontalAlignment и VerticalAlignment центрируют сетку на странице Page. На этом уровне XAML фиксированные размеры – редкость, потому что они ограничивают гибкость макета, но в данном случае позволяют проиллюстрировать некоторые из имеющихся атрибутов.

Grid – это макетная панель, позволяющая организовать элементы в ячейках сетки. Размеры строк и столбцов могут быть фиксированы, задаваться друг относительно друга или вычисляться автоматически в зависимости от содержимого. Дополнительные сведения смотрите в статье «Responsive layouts with XAML» по адресу https://docs.microsoft.com/en-us/windows/uwp/design/layout/layouts-with-xaml.

Блок Grid.RowDefinitions определяет количество и поведение строк сетки. В нашей сетке будет всего две строки. Для первой задан атрибут Height="Auto", что означает, что ее высота будет адаптироваться к содержимому при условии, что места достаточно. Для второй строки задан атрибут Height="*"; это означает, что под нее отводится все остальное место в сетке по вертикали. Если высота определена таким образом для нескольких строк, то имеющееся место равномерно распределяется между ними. Другие способы задания размеров мы обсудим в следующей главе.

Блок Grid.ColumnDefinitions делает то же самое для столбцов сетки. В нашей сетке определено два столбца. Для первого атрибут Height равен Auto, а для второго Height="*".

Элемент TextBlock определяет метку в ячейке на пересечении первой строки и первого столбца. В XAML индексы нумеруются с 0. В данном случае первая строка и первый столбец имеют индекс 0. Свойство Text определяет отображаемый текст, а свойство VerticalAlignment в данном случае означает, что текст центрируется по вертикали. По умолчанию VerticalAlignment для TextBlock равно Top. Свойство Margin добавляет поле – пустое место снаружи от элемента управления. Если ширина полей с каждой стороны одинакова, то можно задать ее одним числовым значением. Но в нашем случае мы хотим только добавить два пикселя справа от элемента, чтобы отделить его от TextBox. Формат ввода полей имеет вид «<LEFT>,<TOP>,<RIGHT>,<BOTTOM>» – в нашем примере "0,0,2,0".

Элемент TextBox – текстовое поле, занимающее ячейку на пересечении первой строки и второго столбца.

Наконец, во второй столбец второй строки мы добавили элемент управления Button. Чтобы отделить его от элементов сверху, мы добавили верхнее поле величиной 4 пикселя. Свойство VerticalAlignment равно Top (по умолчанию Center), а HorizontalAlignment – Right (по умолчанию Center). Для задания текста кнопки не используется свойство Text, как в TextBlock. На самом деле никакого свойства Text здесь вообще нет, а используется свойство Content. Это специальное свойство, которое мы обсудим подробнее в следующей главе. Пока что достаточно знать, что свойство Content может включать любой

другой элемент управления: текст, изображение `Image` и даже сетку `Grid`, содержащую несколько дочерних элементов. Возможности безграничны.

На рис. 1.3 показан пользовательский интерфейс, соответствующий этой разметке.

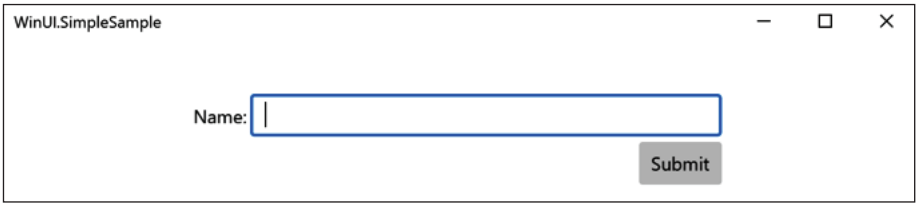


Рис. 1.3 ❖ Результат отрисовки XAML-разметки в WinUI

Это очень простой пример, но он дает почувствовать, что можно сделать с помощью XAML. Дальше мы увидим, насколько велики возможности этого языка.

Создание адаптивного UI для любого устройства

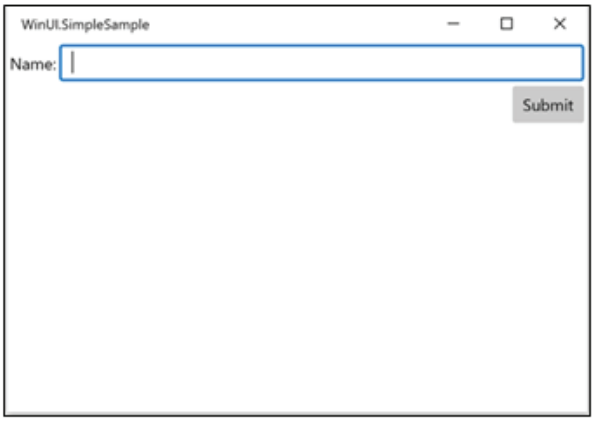
В предыдущем примере сетка `Grid` имела фиксированную высоту и ширину. Я упоминал, что задание фиксированных размеров ограничивает гибкость UI. Давайте теперь уберем фиксированные размеры и воспользуемся свойствами выравнивания, чтобы указать, как должны отрисовываться элементы UI при разных размерах и отношениях сторон:

```
<Grid VerticalAlignment="Top" HorizontalAlignment="Stretch"
Padding="2">
```

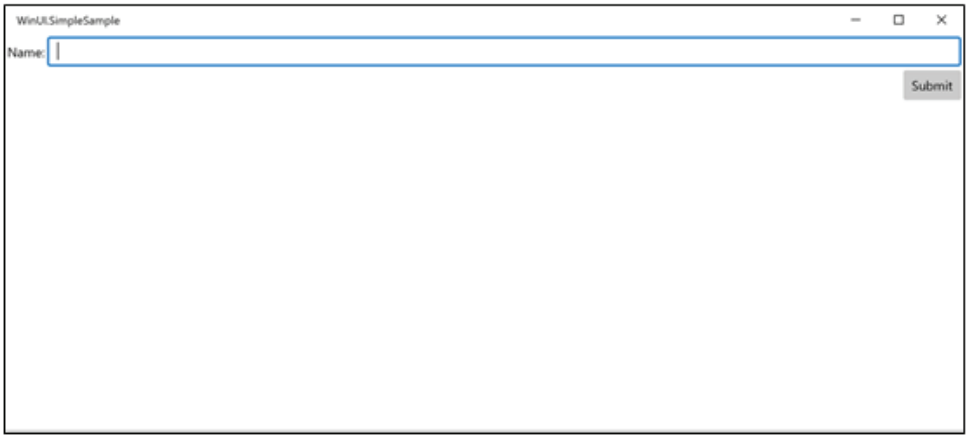
Больше в разметке ничего не меняется. В результате поле `TextBox` изменяет размер вместе с шириной окна, а кнопка `Button` остается привязанной к правому краю окна. На рис. 1.4 показано, как выглядит интерфейс при двух разных размерах окна.

Если запустить это приложение на планшете, то содержимое окна было бы перестроено, так чтобы занять все доступное место. Так проявляется адаптивная природа XAML. При построении UI мы обычно предпочитаем относительные и адаптивные свойства, например выравнивание, фиксированным размерам и положениям.

Именно благодаря адаптивному макету XAML так хорошо работает на мобильных устройствах совместно с Xamarin, и именно по этой причине WPF-разработчики так полюбили его с момента появления в Windows Vista.



Более узкое окно



Более широкое окно

Рис. 1.4 ❖ Окна разного размера

Привязка к данным

Еще одна причина популярности UWP и других каркасов на основе XAML – простота и мощь привязки к данным. Почти все свойства элементов управления UWP могут быть привязаны к данным. Источником данных может быть объект или список объектов. Чаще всего источником является класс `ViewModel`. Рассмотрим без подробностей использование синтаксиса привязки к свойству класса `ViewModel` в UWP.

1. Сначала создадим простой класс `MainViewModel` со свойством `Name`:

```
public class MainViewModel : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;
    private string _name;
```

```
    public MainViewModel()
    {
          _name = "Bob Jones";
    }
    public string Name
    {
        get
        {
            return _name;
        }
        set
        {
            if (_name == value) return;
            _name = value;
            PropertyChanged?.Invoke(this, new
              PropertyChangedEventArgs(nameof(Name)));
        }
    }
}
```

Класс `MainViewModel` реализует интерфейс `INotifyPropertyChanged`. Это ключ к получению пользовательским интерфейсом событий обновления при изменении привязанных к данным свойств. Обычно этот интерфейс обернут каркасом **модель–представление–представление модели** (**MVVM**), например Prism либо MvvmCross, или вашим собственным классом, производным от `ViewModelBase`. Но сейчас мы напрямую вызываем событие `PropertyChanged` из метода установки свойства `Name`. О классах `ViewModel` и интерфейсе `INotifyPropertyChanged` мы будем говорить в главе 3.

2. Следующий шаг – создание экземпляра класса `MainViewModel` и задание его в качестве свойства `ViewModel` на странице `MainPage`. Это делается в коде страницы в файле `MainPage.xaml.cs`, как показано ниже:

```
public sealed partial class MainPage : Page
{
    public MainPage()
    {
        this.InitializeComponent();
        this.ViewModel = new MainViewModel();
    }
    public MainViewModel ViewModel { get; set; }
}
```

Мы добавили свойство `ViewModel` в класс `MainPage` и в конструкторе присвоили ему ссылку на экземпляр нашего класса `MainViewModel`.

Любой код, добавленный в конструктор страницы, должен находиться после вызова метода `InitializeComponent()`.

3. Теперь пора добавить код привязки к данным в XAML-разметку элемента `TextBox`:

```
<TextBox Grid.Row="0" Grid.Column="1" Text="{x:Bind
  Path=ViewModel.Name, Mode=TwoWay}"/>
```

Поле `Text` устанавливается с помощью расширения разметки `x.Bind`. В качестве пути привязки к данным `Path` используется свойство `Name` объекта `ViewModel`, который был создан в коде странице на предыдущем шаге 2. Поскольку задан режим двусторонней привязки к данным `TwoWay`, обновления `ViewModel` отображаются в UI, а обновления, произведенные пользователем с помощью UI, сохраняются в свойстве `Name` класса `MainViewModel`. Теперь при запуске приложения в имя автоматически запишется значение, заданное в конструкторе `ViewModel`, как показано на рис. 1.5.

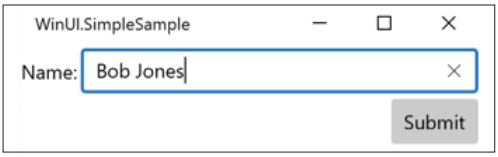


Рис. 1.5 ❖ Привязка TextBox к данным

4. Для иллюстрации привязки к другому свойству другого элемента UI сначала модифицируем сетку, добавив имя элемента:

```
<Grid x:Name="ParentGrid"
      VerticalAlignment="Top"
      HorizontalAlignment="Stretch"
      Padding="2">
```

5. Затем добавим в сетку еще одну строку `RowDefinition`, в которой разместим новый элемент UI:

```
<Grid.RowDefinitions>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="*"/>
</Grid.RowDefinitions>
```

6. Далее добавим элемент `TextBlock` и воспользуемся расширением разметки `Binding`, чтобы привязать его свойство `Text` к свойству `ActualWidth` элемента с именем (`ElementName`) **ParentGrid**. Также добавим еще один элемент `TextBlock`, который будет содержать метку этого значения – `Actual Width`:

```
<TextBlock Grid.Row="1" Grid.Column="0"
           Text="Actual Width:"
           Margin="0,0,2,0"
           VerticalAlignment="Center"/>
<TextBlock Grid.Row="1" Grid.Column="1"
           Text="{Binding ElementName=ParentGrid,
                  Path=ActualWidth}"/>
```

7. Переместим кнопку **Submit** в строку 2, изменив свойство `Grid.Row`.
8. Теперь в новом элементе `TextBlock` отображается ширина сетки `ParentGrid` в момент загрузки страницы. Отметим, что при изменении размера окна это значение не изменяется. Свойство `ActualWidth` не генерирует уведомления об изменении значения, о чем написано в документации по `FrameworkElement.ActualWidth` на странице https://docs.microsoft.com/en-us/uwp/api/windows.ui.xaml.frameworkelement.actualwidth (рис. 1.6).

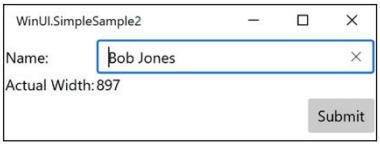


Рис. 1.6 ❖ Привязка другого элемента к данным

Кнопка **Submit** пока не работает. Как заставить ее работать с помощью **событий** и **команд** MVVM, мы узнаем в главе 5.

Стилизация UI с помощью XAML

В XAML можно определить и применить стили почти в любой области видимости: глобально для всего приложения в файле `App.xaml`, на текущей странице в объявлении `Page.Resources` или внутри любого уровня или вложенного элемента управления на странице. В элементе `Style` задается свойство `TargetType`, содержащее тип данных элементов, на которые распространяется данный стиль. Можно также задать необязательное свойство `Key`, играющее роль уникального идентификатора по аналогии с идентификатором класса в **каскадных таблицах стилей** (**CSS**). Свойство `Key` позволяет применить стиль только к избранным элементам указанного типа. В отличие от классов CSS элементу можно назначить только одно свойство `Key`.

В следующем примере мы определим свойство `Style` для всех кнопок на странице.

1. Сначала переместим кнопку **Submit**, вложив ее в элемент `StackPanel`, который размещает свои дочерние элементы в виде горизонтальной или вертикальной (по умолчанию) стопки. Некоторые свойства кнопки придется перенести в элемент `StackPanel`, поскольку теперь именно он является непосредственным потомком `Grid`. Добавим в `StackPanel` еще одну кнопку **Cancel** для отмены, после чего разметка будет иметь такой вид:

```
<StackPanel Grid.Row="1" Grid.Column="1"
            Margin="0,4,0,0"
            HorizontalAlignment="Right"
            VerticalAlignment="Top"
            Orientation="Horizontal">
    <Button Content="Submit" Margin="0,0,4,0"/>
```

```
    <Button Content="Cancel"/>
</StackPanel>
```

Для первой кнопки задано поле, чтобы увеличить промежуток между элементами.

2. Затем добавим в секцию `Page.Resources` блок `Style` для стилизации кнопок. Поскольку этому блоку не сопоставлен ключ `Key`, он применяется ко всем элементам `Button`, для которых стиль не переопределен во вложенной области видимости. Такой стиль называется *неявным*. Ниже показан соответствующий код:

```
<Page.Resources>
    <Style TargetType="Button">
        <Setter Property="BorderThickness"
                Value="2" />
        <Setter Property="Foreground"
                Value="LightGray" />
        <Setter Property="BorderBrush"
                Value="GhostWhite"/>
        <Setter Property="Background"
                Value="DarkBlue" />
    </Style>
</Page.Resources>
```

3. Если теперь запустить приложение, то мы увидим, что новый стиль применен к кнопкам **Submit** и **Cancel**, хотя напрямую ни к той, ни к другой кнопке мы его не применяли (рис. 1.7).

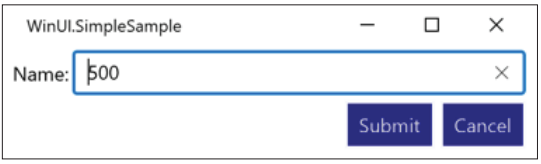


Рис. 1.7 ❖ Стилизованные кнопки

Если бы мы поместили этот блок `Style` в секцию `Application.Resources`, то определенный в нем стиль применялся бы ко всем кнопкам в приложении, для которых разработчик не переопределил какие-то свойства стиля. Например, если бы в кнопке **Submit** было задано свойство `Background`, равное `DarkGreen`, то темно-синей (`DarkBlue`) осталась бы только кнопка **Cancel**.

Мы вернемся к стилям и вопросам дизайна в главе 7.

Отделение представления от бизнес-логики

В предыдущем разделе, посвященном привязке к данным, мы кратко рассмотрели паттерн MVVM. Этот паттерн – ключ к отделению логики представления от бизнес-логики в UWP-приложении. XAML-элементу нужно только знать, что где-то в его контексте данных существует свойство с конкретным

именем. Классы `ViewModel` ничего не знают о представлении `View` (нашем XAML-файле).

У такого разделения есть несколько преимуществ. Во-первых, классы `ViewModel` можно автономно тестировать независимо от UI. Если тестируемая система обращается к каким-то элементам UWP, то необходим поток, в котором выполняется пользовательский интерфейс. Из-за этого тесты не будут проходить, когда выполняются в фоновых потоках локально или на сервере **непрерывной интеграции** (Continuous Integration – **CI**). Подробнее об автономном тестировании WinUI-приложений см. главу 3.

Еще одно преимущество разделения `View` и `ViewModel` заключается в том, что в компаниях, где имеются отдельные специалисты по **впечатлению от использования** (user experience – **UX**), проектированием XAML-разметки приложения занимаются именно они, тогда как другие разработчики сконцентрированы на моделях представлений (классах `ViewModel`). Когда приходит время синхронизировать то и другое, разработчик может добавить в XAML нужные свойства привязки к данным, или, быть может, дизайнер UX и разработчик заранее согласовали имена свойств в общем контексте данных. В Visual Studio имеется еще один инструмент, ориентированный на дизайнеров при таком технологическом процессе, – Blend for Visual Studio. Впервые программа Blend была выпущена Microsoft в 2006 году под названием Microsoft Expression Blend как инструмент для создания пользовательских интерфейсов в WPF. Впоследствии была добавлена поддержка для других диалектов XAML, в частности Silverlight и UWP. Blend по-прежнему включается в состав рабочей нагрузки для разработки UWP при установке Visual Studio.

И последнее преимущество, которое мы здесь обсудим, – тот факт, что удачное разделение обязанностей между уровнями приложения всегда упрощает сопровождение. Когда за одно и то же отвечают несколько компонентов или логика дублируется в нескольких местах, повышается риск внести в код ошибки и получить ненадежное приложение. Следуйте проверенным паттернам проектирования – так вы избавите себя от лишней работы.

Теперь, когда вы познакомились с историей UWP-приложений, самое время взглянуть на WinUI: что это, и зачем оно было создано?

Что такое WinUI?

Библиотека WinUI – это набор элементов управления и компонентов пользовательского интерфейса, «выдернутых» из Windows SDK. После такого разделения многие элементы управления были улучшены и добавились новые. Исходный код библиотек был выложен на GitHub и сопровождается как компанией Microsoft, так и сообществом разработчиков.

Но раз библиотеки WinUI произошли от библиотек UWP в составе Windows SDK, то спрашивается, зачем выбирать в качестве каркаса для построения UI именно WinUI, а не UWP. Ведь UWP существует с момента выхода Windows 10 и считается вполне надежной и стабильной технологией. Но существует несколько веских причин предпочесть WinUI.

Выбор WinUI несет с собой все выгоды ПО с открытым исходным кодом, которое обычно очень надежно. Когда программа разрабатывается открыто активным сообществом, ошибки находят и исправляют очень быстро. На самом деле, обнаружив ошибку, вы сами можете исправить ее и отправить запрос на включение своего исправления в основную ветвь, тем самым сделав ее достоянием всего сообщества. Проекты с открытым исходным кодом могут быстро развиваться, не нуждаясь в синхронизации с другими группами, существующими в большой компании, например с командой разработки Windows. Сейчас обновления Windows выходят регулярно, но все равно реже, чем типичная библиотека элементов управления.

Главная причина использовать WinUI – обратная совместимость. При использовании элемента управления UWP дополнения и исправления, внесенные в его конкретную версию, не могут быть использованы в приложениях, ориентированных на предыдущие версии Windows. В случае WinUI, если вы указываете минимальную версию Windows, поддерживаемую WinUI в целом, можете использовать новые элементы управления и новые возможности в нескольких версиях Windows. Элементы управления, которые ранее не были доступны разработчикам UWP в некоторой версии Windows, теперь доступны как WinUI-элементы.

Например, Microsoft не включала текучий UI в Windows до осени 2017 года (версия 16299). Однако элементы управления WinUI можно включить в приложения, ориентированные на минимальную версию Windows 10.0.15063.0, вышедшую весной 2017 года. Элементы управления WinUI поддерживают стили текучего UI. WinUI добавляет элементы управления и другие средства, которых вообще нет ни в UWP, ни в Windows SDK.

Первый выпуск WinUI

Первая версия WinUI вышла в июле 2018 года как предварительная для Windows-разработчиков. Она состояла из двух NuGet-пакетов:

- `Microsoft.UI.Xaml`: элементы управления WinUI и стили текучего UI;
- `Microsoft.UI.Xaml.Core.Direct`: компоненты для разработчиков ПО промежуточного уровня, необходимые для доступа к XamlDirect API.

Спустя три месяца была выпущена версия WinUI 2.0. Несмотря на номер, это была первая производственная версия WinUI. В нее вошло более 20 элементов управления и кистей. Перечислим наиболее примечательные элементы:

- `TreeView`: неизменная принадлежность любой библиотеки UI;
- `ColorPicker`: визуальное средство выбора цвета с непрерывным спектром цветов;
- `DropDownButton`: кнопка с возможностью открывания меню;
- `PersonPicture`: элемент для отображения графического аватара. Обладает возможностью отображать инициалы или обобщенное изображение в случае отсутствия нужной картинки;
- `RatingControl`: позволяет задавать рейтинг предмета в виде количества звездочек.

Добавим некоторые из этих элементов управления в проект WinUI, чтобы посмотреть, как они выглядят. Измените содержимое панели `StackPanel` следующим образом:

```
<StackPanel Grid.Row="1" Grid.Column="1" Margin="0,4,0,0"
            HorizontalAlignment="Right"
            VerticalAlignment="Top"
            Orientation="Horizontal">
    <PersonPicture Initials="MS" Margin="0,0,8,0"/>
    <DropDownButton Content="Submit" Margin="0,0,4,0">
        <DropDownButton.Flyout>
            <MenuFlyout Placement="Bottom">
                <MenuFlyoutItem Text="Submit + Print"/>
                <MenuFlyoutItem Text="Submit + Email"/>
            </MenuFlyout>
        </DropDownButton.Flyout>
    </DropDownButton>
    <Button Content="Cancel"/>
</StackPanel>
```

Элемент `PersonPicture` с инициалами `MS` добавлен в `StackPanel` первым, а первая из двух кнопок заменена элементом `DropDownButton`. Раскрывающаяся кнопка **Submit** содержит меню `FlyoutMenu`, работающее как выпадающий список, в котором есть два элемента `MenuFlyoutItem`. Теперь пользователь может либо просто нажать кнопку **Submit**, либо выбрать из выпадающего списка пункт **Submit + Print** или **Submit + Email**.

Элемент **DropDownButton** имеется только в Windows 10, начиная с версии 1809. Если вы собираетесь использовать его в производственном приложении, то должны задать эту версию как минимальную.

Вот как выглядит новое окно с раскрытой кнопкой `DropDownButton`:

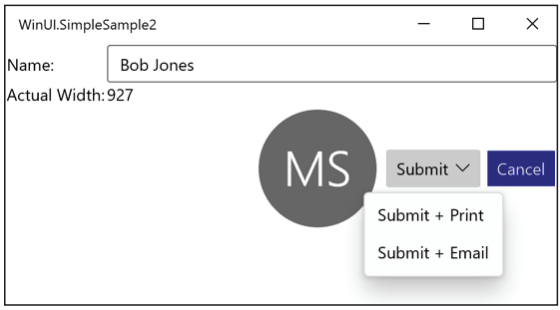


Рис. 1.8 ❖ Добавление элементов управления `PersonPicture` и `DropDownButton`

Мы лишь пробежались по самым верхам возможностей, которые предоставляла разработчикам первая версия. Но не расстраивайтесь, у нас еще будет шанс наверстать упущенное в последующих главах. А пока кратко рас-

смотрим, что было добавлено в последующие версии, увенчавшиеся выпуском WinUI 3.0.

По дороге к WinUI 3.0

После версии 2.0 было выпущено пять дополнительных версий WinUI, не считая многочисленных исправлений и предвыпускных версий.

WinUI 2.1

В версию WinUI 2.1 было включено несколько новых элементов управления и возможностей. Отметим некоторые из них.

- `TeachingTip`: этот элемент можно назвать обогащенной контекстно-зависимой всплывающей подсказкой. Он связывается с другим элементом на странице и отображает подробные сведения о целевом элементе, помогая пользователю ориентироваться, но только по его просьбе.
- `AnimatedVisualPlayer`: содержит анимации в формате Lottie. Это популярный формат, созданный в **Adobe After Effects** и используемый дизайнерами в Windows, вебе и на мобильных платформах. Для большинства современных систем разработки существуют библиотеки, поддерживающие Lottie.

Дополнительные сведения о Lottie-файлах можно найти на сайте https://airbnb.design/lottie/, а большой репозиторий анимаций в этом формате имеется по адресу https://lottiefiles.com/.

- `CompactDensity`: добавление этого словаря ресурсов в приложение наделяет его способностью переключаться между *компактным* и *обычным* режимами отображения. В режиме `CompactDensity` расстояние внутри элементов и между элементами на странице уменьшается, так что пользователь видит до 33 % больше содержимого. Эта идея текучего UI была представлена разработчикам на конференции Microsoft Build 2018.

WinUI 2.2

Эта версия принесла много улучшений уже существующей функциональности. Однако единственный новый элемент управления, добавленный в библиотеку, был с радостью встречен многими разработчиками.

Элемент `TabView` выглядит как хорошо знакомый пользователям контейнер с вкладками. Каждая вкладка может содержать страницу проекта WinUI.

Улучшения, включенные в WinUI 2.2

Ниже перечислены наиболее заметные обновленные элементы управления и библиотеки в версии 2.2:

- `NavigationView`: кнопка «Назад» остается видимой, когда панель свернута. Другие изменения увеличивают количество видимого содержимого;

- **визуальные стили**: были обновлены многие визуальные стили WinUI, включая `CornerRadius`, `BorderThickness`, `CheckBox` и `RadioButton`. Все обновления имели целью сделать визуальный облик WinUI более согласованным с рекомендациями по дизайну текучего UI.

WinUI 2.3

В версии WinUI 2.3 изменениям подвергся элемент `ProgressBar` и в библиотеку было добавлено несколько новых элементов.

Теперь при создании `ProgressBar` в приложении WinUI можно задать один из двух режимов: `Determinate` (Определенный) и `Indeterminate` (Неопределенный). Определенная полоса прогресса знает объем задачи и ее текущее состояние, а неопределенная показывает, что задача выполняется, но время до завершения неизвестно. Последняя играет роль индикатора занятости.

Новые элементы управления в WinUI 2.3

Были добавлены следующие элементы управления:

- `NumberBox`: это редактор входных данных, который упрощает поддержку форматирования чисел, имеет кнопки увеличения и уменьшения и выполняет математические вычисления. На первый взгляд, простой, но практичный элемент с довольно развитой функциональностью;
- `RadioButtons`: вы, наверное, подумали: «Но такие переключатели (радиокнопки) всегда были. Что же здесь нового?» На самом деле элемент управления `RadioButtons` группирует несколько элементов `RadioButton`, упрощая работу с ними как с единым целым.

WinUI 2.4

Версия WinUI 2.4 вышла в мае 2020 года и включала два дополнения: визуальный объект `RadialGradientBrush` и элемент управления `ProgressRing`.

Кисть похожа на класс `RadialGradientBrush`, знакомый разработчикам на платформе WPF. Он позволяет легко добавлять визуальным элементам градиент, распространяющийся из центральной точки.

Элемент `ProgressRing` – аналог полосы прогресса, только имеет форму кольца. В версии 2.4 для него доступно два режима: определенный и неопределенный. Неопределенный элемент `ProgressRing` повторяет одну и ту же анимацию, это режим по умолчанию.

В версии 2.4 несколько элементов было обновлено. Элемент `TabView` теперь позволяет точнее управлять отрисовкой вкладок и поддерживает режимы `Compact`, `Equal` и `Size to Content`. Элемент `TextBox` получил *темный режим*, в котором фон области темный, а текст по умолчанию отображается белым цветом. Наконец, элемент `NavigationView` стал поддерживать иерархическую навигацию благодаря режимам `Left`, `Top` и `LeftCompact`.

WinUI 2.5

Версия WinUI 2.5 вышла в декабре 2020 года и включала элемент управления `InfoBar`. Кроме того, в нее вошли некоторые улучшения и исправления ошибок.

Элемент `InfoBar` позволяет отображать пользователям важные информационные сообщения о состоянии. Он может показать значок предупреждения или информации, сообщение о состоянии и ссылку или кнопку, которая дает возможность предпринять некоторое действие. По умолчанию элемент включает значок, сообщение и кнопку закрытия. В документации приведены рекомендации по использованию этого элемента: https://docs.microsoft.com/en-us/windows/uwp/design/controls-and-patterns/infobar.

В версию 2.5 было включено также несколько обновлений. Элемент `ProgressRing` стал лучше вести себя в определенном режиме. Элемент `NavigationView` теперь предоставляет допускающие настройку пункты подвального меню `FooterMenuItem`. В предыдущих версиях подвальную область можно было показать или скрыть, но возможности настройки не было.

Мы видели, что было доступно UWP-разработчикам в WinUI 2.x. Теперь посмотрим, что мы получили в WinUI 3.0.

Что нового в WinUI 3.0?

В отличие от WinUI 2.0 и последовавших за ней дополнительных версий, WinUI 3.0 – крупное обновление, включающее куда больше, чем новые и улучшенные элементы управления и библиотеки, которые можно использовать совместно с приложениями UWP и .NET 5. На самом деле основной целью WinUI 3.0 было не добавление новых элементов управления и возможностей сверх того, что есть в UWP. Команда разработчиков сделала WinUI полноценным каркасом создания UI, который может работать поверх платформ UWP или Win32.

Прощай, UWP?

Так что же случилось с UWP? Неужели мои UWP-приложения перестанут работать?

Как уже было сказано, для библиотек UWP планируется и дальше выпускать важные обновления безопасности, но никакое развитие не предусмотрено. Все новые возможности и обновления будут разрабатываться только для WinUI. Новые WinUI-проекты будут поддерживать все типы клиентов Windows с пользовательским интерфейсом. Для существующих Win32-приложений разработчики могут постепенно переходить на WinUI с помощью **островков Xaml**, обеспечивающих интероперабельность. Новые WinUI-приложения будут создаваться либо с помощью .NET Core на языке C# или VB, либо непосредственно на C++. Эти клиенты будут работать поверх платформы Win32 или UWP. Все это возможно, потому что WinUI целиком написана на C++.

Некоторые возможности, рассматриваемые в этой книге, не появятся в первой стабильной производственной (RTM) версии WinUI 3.0, планируемой к выходу в марте 2021 года в составе Project Reunion v0.5. Островков XAML не будет, а UWP-клиенты

поддерживаются не полностью. Полный список всего, что планируется включить в первую стабильную версию WinUI 3.0, можно найти на дорожной карте по адресу https://github.com/microsoft/microsoft-ui-xaml/blob/master/docs/roadmap.md#winui-30-feature-roadmap.

Тот факт, что WinUI написана на C++, позволяет клиентским приложениям React Native for Windows взаимодействовать с платформой WinUI. WinUI обладает большим кросс-платформенным потенциалом для работы с React Native и платформой Uno.

Новая модель приложений показана на рис. 1.9.

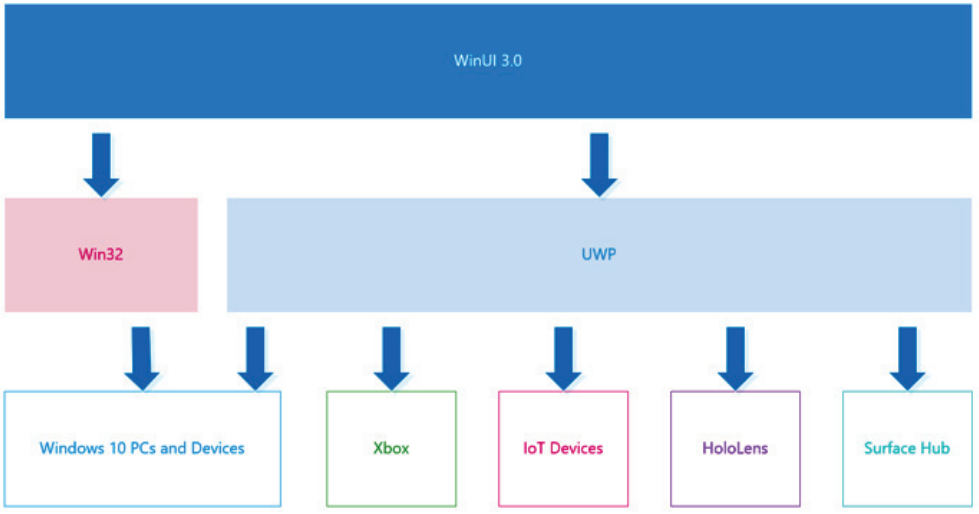


Рис. 1.9 ❖ Модель приложений WinUI 3.0

Как видим, перед разработчиками открыто много путей для создания приложений для ПК и планшетных устройств под управлением Windows, в частности Surface Duo с двумя экранами. Другие Windows-устройства, в т. ч. Xbox и HoloLens, должны будут следовать модели приложений UWP, расположенной ниже уровня WinUI.

Новые возможности в WinUI 3.0

Есть ли в WinUI 3.0 какие-нибудь новые возможности?

Хотя команда была очень занята созданием каркаса UI, призванного заменить библиотеки UWP UI, она нашла время для добавления нескольких новшеств. Главный элемент управления, появившийся в версии 3.0, – `WebView2`, основанный на браузере Microsoft Edge Chromium. Совместимость тоже можно считать новой возможностью. Все средства XAML и Composition, имеющиеся в версии Windows SDK, вышедшей весной 2019 года, будут обратно совместимы вплоть до обновления Windows Creators.

Project Reunion и WinUI

WinUI 3.0 объединяет разработчиков на платформах UWP и Win32, предоставляя им единый набор библиотек UI, но это только начало. На конференции Microsoft Build 2020 команда Windows анонсировала **Project Reunion**, долгосрочный план перевода всех разработчиков для Windows на единую платформу. В фокусе WinUI 3.0 находится уровень пользовательского интерфейса, тогда как Project Reunion охватывает WinUI и всю платформу разработки для Windows. В 2021 году Microsoft выпустит три версии Project Reunion и WinUI 3.x, а именно:

- *март 2021*: первый выпуск Project Reunion, версия 0.5, будет содержать первую поддерживаемую версию WinUI 3;
- *май 2021*: версия Project Reunion 0.8 будет включать несколько улучшений WinUI 3;
- *октябрь 2021*: это будет версия Project Reunion 1.0, содержащая дополнительные улучшения WinUI 3.

Чтобы больше узнать о Project Reunion и следить за новостями, заходите в репозиторий проекта на GitHub по адресу https://github.com/microsoft/ProjectReunion. А теперь сравним WinUI с другими каркасами разработки для Windows.

Сравнение WinUI с другими каркасами разработки для Windows

Каково место WinUI в общей картине каркасов разработки для Microsoft Windows? Для ответа на этот вопрос сравним несколько продуктов и начнем с наиболее похожего на WinUI.

WinUI и UWP

Сравнивать трудно, потому что большинство сегодняшних WinUI-приложений в основе своей являются UWP-приложениями. На самом деле WinUI 2.x – это элементы управления для UWP-приложений. При рассмотрении WinUI 3.0 разумно считать WinUI каркасом UI, а UWP – уровнем платформы для разработки приложений. Они разделяют общую схему XAML, базовые визуальные объекты и набор Windows API. Любое UWP-приложение, для которого заданы одинаковые минимальная и целевая версии Windows, может добавить библиотеки WinUI 2.x, чтобы воспользоваться новой и обновленной функциональностью.

Ключевое различие между приложениями, использующими WinUI, и традиционными UWP-приложениями – доступ к новым и обновленным элементам управления и другим визуальным объектам, не требуя при этом обнов-

ленного Windows SDK. Это позволяет разработчикам предлагать приложения с одинаковым внешним обликом и функциональными возможностями большему количеству пользователей на нескольких версиях Windows 10. Данное отличие облегчает жизнь как разработчиков, так и пользователей.

WinUI и WPF

У WinUI и WPF много общего. То и другое – каркасы разработки приложений, и оба типа приложений полагаются на XAML как на средство определения элементов UI. Это означает, что та и другая технологии предлагают одинаковое разделение UI и бизнес-логики при реализации паттерна MVVM. Для WPF XAML характерны те же идеи стилей, ресурсов, привязки к данным и адаптивности макета UI.

Преимущества WinUI

Важное преимущество WinUI – и UWP-приложений вообще – доступность **откомпилированных привязок**. Мы будем подробно обсуждать откомпилированные привязки ниже. А пока достаточно знать, что их применение резко повышает производительность по сравнению с традиционной привязкой к данным в UWP и WPF. Откомпилированные привязки выделяются использованием синтаксиса `x:Bind` в XAML, пришедшего на смену `Binding`.

Еще одно преимущество WinUI по сравнению с WPF – их большая безопасность по умолчанию. Доступ к пользовательским файловым системам и устройствам из песочницы вселяет в пользователя чувство облегчения, поскольку он знает, что доступ вредоносной программы к оборудованию и данным ограничен. Доступ WPF-приложений ограничен только конфигурацией **контроля учетных записей пользователей** (**UAC**) на ПК.

Преимущества WPF

Главное преимущество WPF-приложений – отсутствие прямой привязки к минимальной версии Windows 10. WPF-приложения привязаны к версии .NET Framework или .NET Core. Любая версия Windows, поддерживающая целевую версию .NET, может исполнять WPF-приложение. Это значительно расширяет состав пользователей WPF-приложений. WPF-приложения можно даже развернуть и выполнить в Windows 7, что для UWP или WinUI абсолютно невозможно.

Существует проект Uno Platform, который позволяет выполнять WinUI XAML на iOS и Android, пользуясь Xamarin, и в вебе с помощью технологии WebAssembly. Эти веб-приложения WinUI можно выполнять в браузере на старых версиях Windows, включая Windows 7. Цель и девиз проекта Uno Platform – WinUI всюду.

Подробнее о проекте Uno Platform можно прочитать на сайте https://platform.uno/.

Подробнее о технологии WebAssembly можно прочитать на сайте https://webassembly.org/.

WinUI-приложения по умолчанию безопасны, и по сравнению с ними WPF-приложения обладают большей гибкостью и функциональностью. Для многих типов приложений необходим полный доступ к файловой системе Windows или к конкретным устройствам и драйверам – например, создать специальный диспетчер файлов гораздо проще, имея полный доступ к файловой системе компьютера. Правда, в последних версиях Windows 10 WinUI-приложения могут запросить такой доступ, но в WPF это режим по умолчанию.

Еще одно преимущество WPF выявилось с выходом в свет .NET Core 3.x и .NET 5. Теперь разработчики для .NET могут создавать WPF-приложения с помощью .NET Core, что дает в руки WPF-разработчиков производительность и широкие возможности развертывания .NET Core. Например, приложения, ориентированные на разные версии .NET Core, могут быть развернуты на одной машине без риска конфликта версий.

Из-за различия в моделях развертывания могут возникнуть споры – какой каркас лучше. Развернуть WinUI-приложение проще всего из Windows Store, а WPF-приложение для .NET Framework – с помощью пакета установки. WPF-приложения могут быть развернуты из Store посредством технологии Windows Containers, WinUI-приложения могут обойтись без Store, если воспользоваться MSIX-установщиками. Развертывание WinUI-приложений будет подробно рассмотрено в главах 12 и 13.

WinUI и Windows Forms (WinForms)

WinForms – каркас разработки пользовательских интерфейсов для .NET, появившийся еще во времена .NET Framework 1.0. Для дизайна интерфейса в WinForms имеется визуальная среда разработки в Visual Studio, которая генерирует код на C# или VB, создающий UI во время выполнения. Преимущества и недостатки WPF по большей части свойственны и WinForms: безопасность, развертывание и .NET Core – приложения WinForms тоже можно создавать в .NET Core 3 и более поздних версиях.

Преимущества WinUI

Общие черты WinUI и WPF являются основным их преимуществом по сравнению с WinForms: привязка к данным, адаптивный макет и гибкая модель стилизации. Все эти преимущества проистекают из использования XAML для макетирования UI. Еще одно достоинство XAML – перенос отрисовки с **центрального процессора** (**CPU**) на графический процессор (**GPU**). Элементы управления WinUI по умолчанию наследуют стили Windows 10 и выглядят более современно, чем элементы WinForms. WinUI-приложения также обрабатывают масштабирование в **точках на дюйм** (dots per inch – **DPI**) и сенсорный ввод. Каркас WinForms принял окончательную форму еще до появления сенсорного ввода и DPI-масштабирования.

Основное преимущество WinUI перед WPF относится также и к WinForms: безопасность по умолчанию.

Преимущества WinForms

Помимо преимуществ перед WinUI, общих для WinForms и WPF, – более полный доступ к Windows, приложения для .NET Core и совместимость с версиями Windows, – WinForms пользуется заслуженной репутацией за быстроту разработки UI. Если требуется создать простое приложение для Windows за минимальное время, то с простотой и интуитивной понятностью перетаскивания в конструкторе WinForms ничто не сравнится. Многие опытные разработчики до сих пор рассматривают WinForms как основного кандидата при создании простых утилит или тестовой оснастки UI для .NET-библиотеки.

Резюме

В этой главе мы рассмотрели историю развития разработки приложений для Windows. Мы узнали о том, что UWP уходит корнями в приложения для Windows 8, и разобрались в преимуществах XAML при построении пользовательских интерфейсов в Windows. Мы видели, как выглядит простой код и UI в WinUI-приложении. Наконец, мы поговорили о недавней истории версий WinUI и о том, что новая версия 3.0 является полной заменой библиотек UWP и перспективным путем вперед для WPF-разработчиков.

Это станет хорошим фундаментом для построения WinUI-приложения в последующих главах. В следующей главе мы подготовим среду разработки, опишем приложение, которым будем заниматься на протяжении всей книги, и создадим свой первый проект для WinUI 3.0. В главе 3 мы переработаем приложение под паттерн MVVM и тем самым заложим прочную, удобную для сопровождения основу для последующих добавлений и расширений.

Вопросы

1. В какой версии Windows впервые появились UWP-приложения для разработчиков?
2. Как называется паттерн, которым часто пользуются разработчики на платформе WinUI и других платформах на основе XAML, для разделения UI и бизнес-логики?
3. Приложения WinUI и WPF могут использовать один и тот же XAML-код. Это правда или ложь?
4. Как назывался первый каркас Microsoft для создания UI, в котором использовался язык XAML?
5. Каким был номер первой выпускной версии WinUI?
6. В чем преимущества разработки на WinUI по сравнению с WinForms?
7. Правда ли, что WinUI-приложения можно писать только на NET-совместимых языках?
8. Задача: создайте стиль, применимый к элементам `Button`.