Windows 10

Лабораторный практикум

Живые плитки и уведомления

Октябрь 2015 г.

Microsoft

Обзор

Windows 10 предоставляет уникальные и привлекательные способы для взаимодействия с пользователями, которые выходят за рамки привычных приложений. Теперь вы можете задать контент для «Живой плитки» посредством использования новой адаптивной разметки, которая позволит максимально эффективно учитывать размер плитки и плотность экрана. Всплывающие уведомления (Toast) могут включать интерактивные и адаптивные элементы, изображения, а также способны синхронизироваться с живыми плитками.

Цели

Данная лабораторная работа научит вас:

- Задавать графические элементы и фоновый цвет для плиток по умолчанию
- Использовать BadgeUpdateManager для обновления счетчика плитки
- Создавать адаптивные шаблоны для живых плиток
- Обновлять живые плитки
- Поддерживать плитки разного размера
- Создавать интерактивные уведомления с возможностью для пользователя выбирать действие
- Обрабатывать действия уведомлений с помощью фоновой активации приложения

Системные требования

Для прохождения данной лабораторной работы вам понадобятся:

- Microsoft Windows 10
- Microsoft Visual Studio 2015

Настройка

Вы должны выполнить следующие действия для подготовки компьютера:

1. Установить Microsoft Windows 10.
2. Установить Microsoft Visual Studio 2015.

Упражнения

Данный практикум включает следующие упражнения:

1. Настройка плитки по умолчанию
2. Управление обновлениями плиток
3. Запуск интерактивного всплывающего уведомления

Расчетное время для завершения лабораторной работы: **45-60 минут**.

Упражнение 1: Настройка плитки по умолчанию

Приложения Windows 10 используют выбранный вами логотип (изображение) для отображения плитки по умолчанию. В данном упражнении мы добавим новый логотип (изображение) с целью создания простых плиток и обновления счетчика (бейджа) плитки при использовании BadgeUpdateManager.

Задача 1 – Создание пустого универсального Windows-приложения

Мы начнем с создания проекта на основе шаблона Blank App (Пустого приложения).

1. В новой версии Visual Studio 2015 выберите **File (Файл)** -> **New (Новый)** -> **Project (Проект)**, чтобы открыть диалоговое окно New Project (Новый проект). Далее **Installed (Установленное)** > **Templates (Шаблоны)** > **Visual C#** > **Windows** > **Universal**, а затем выберите шаблон **Blank App** (Universal Windows).
2. Назовите свой проект **"TilesAndNotifications"** и выберите место в файловой системе, в котором будет храниться ваш проект для данного Лабораторного практикума. На диске **C:** создана папка под именем **"HOL"**, мы отсылаемся на нее в примерах снимков экрана.

Оставьте без изменения выбранные опции для **Create new solution** (Создание нового решения) и **Create directory for solution** (Создание папки для решения). Вы можете снять галочки с **Add to source control** (Добавить в систему контроля версий) и **Show telemetry in the Windows Dev Center** (Отобразить телеметрию в Windows Dev Center), если не хотите управлять версиями своего проекта или использовать инструмент Application Insights. Нажмите **OK** для создания проекта.

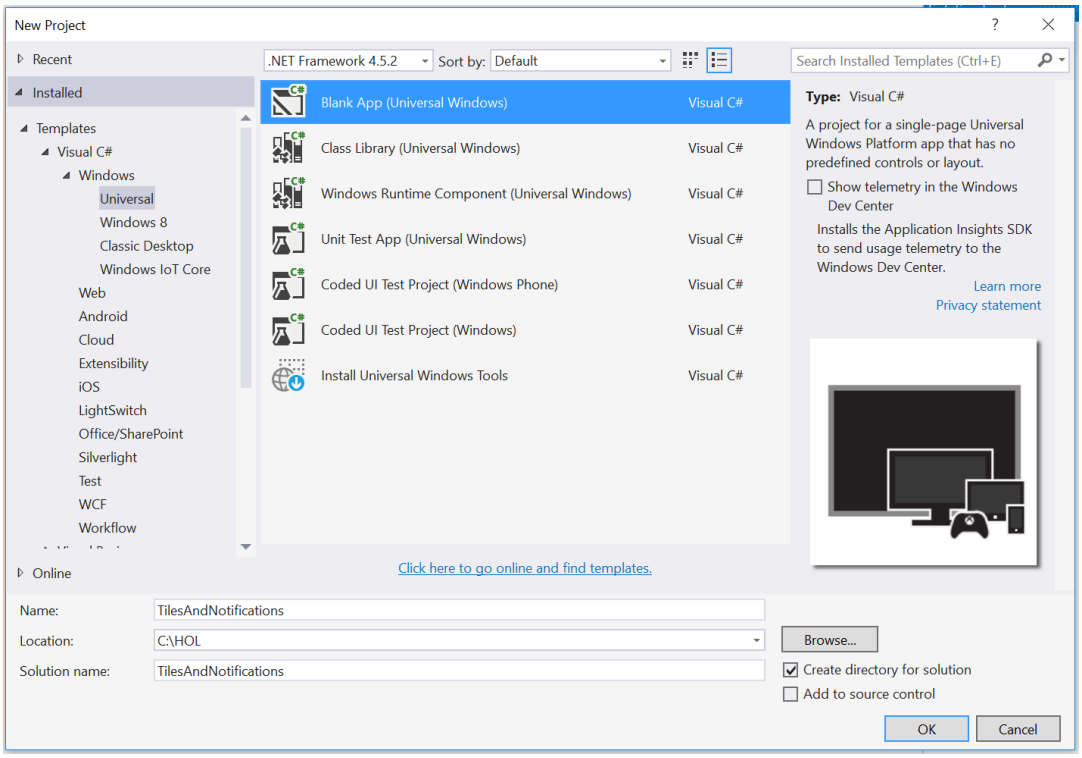

Рисунок 1

Создайте в Visual Studio 2015 новый проект Blank App.

3. Настройте Solution Configuration (Текущую конфигурацию решения) на **Debug** (Отладку) и Solution Platform (Платформу решений) в соответствии с **x86**. Выберите **Local Machine** (Локальный компьютер) из выпадающего меню Debug Target (Цели отладки).

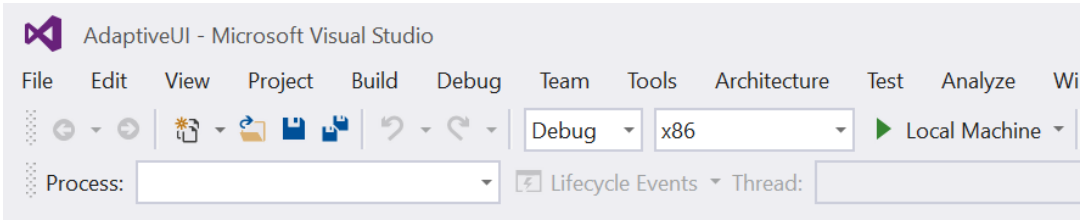

Рисунок 2

Настройте свое приложение таким образом, чтобы оно запускалось на Local Machine (Локальном компьютере).

4. Создайте и запустите свое приложение. Вы увидите окно Blank App со счетчиком частоты кадров, активированным по умолчанию для отладки.

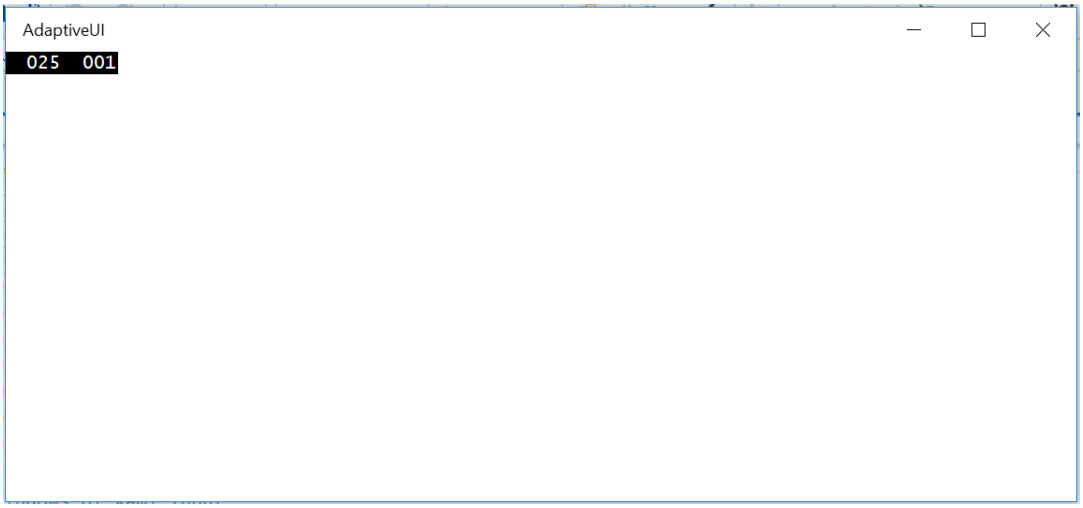

Рисунок 3

Пустое универсальное приложение в десктопном режиме.

Примечание: Счетчик частоты кадров является инструментом отладки, который помогает следить за производительностью вашего приложения. Он полезен для тех приложений, которые требуют интенсивной графической обработки, однако не подходит для простых приложений, которые будут создаваться вами на данный момент.

В шаблоне Blank App директива препроцессора, которая активирует или отключает счетчик частоты кадров, указана в **App.xaml.cs**. Счетчик частоты кадров может перекрывать или скрывать контент вашего приложения, если его не отключить. При выполнении данной работы вы можете отключить его, отметив **DebugSettings.EnableFrameRateCounter** как **false**.

Вернитесь к Visual Studio и остановите отладку.

Задача 2 – Импорт визуальных элементов

Когда вы создаете новый проект, он включает папку Assets с шаблонными изображениями плиток. Чтобы выделить свое приложение, вам нужно заменить шаблонные изображения, представленные в шаблоне Blank App, на собственные логотипы. Визуальные элементы можно указывать с учетом различных коэффициентов масштаба для экранов с разной плотностью пикселей. Если вы указываете изображения только для одного масштаба, то рекомендуется выбирать изображение в масштабе 200 (200% от исходного размера для типичного устройства с 96 dpi), что должно привести к хорошим результатам, поскольку Windows масштабирует изображения, подстраиваясь под плотности пикселей экранов различных устройств. Для достижения наилучших результатов при различной плотности экрана вам стоит предоставить изображения, созданные для нескольких масштабных коэффициентов.

В рамках данной задачи мы добавим в приложение изображения для масштаба 200 с целью их отображения на плитках по умолчанию.

1. Откройте **Package.appxmanifest** в редакторе манифеста и перейдите на вкладку **Visual Assets**. Выберите пункт **Tile Images and Logos** на панели слева.

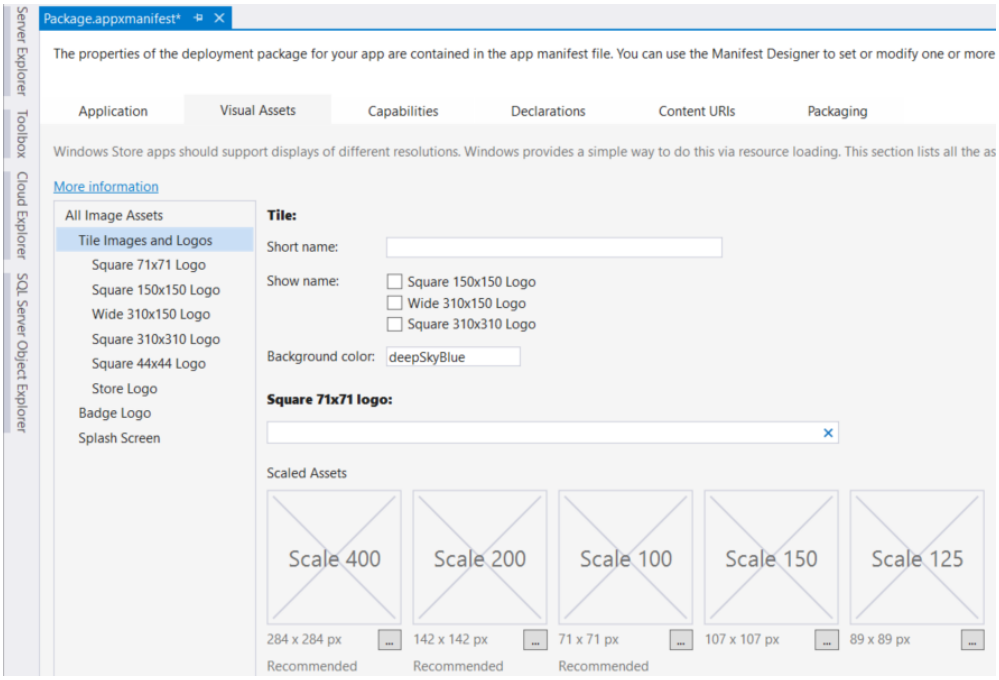

Рисунок 4

Выбор пункта Tile Images and Logos на вкладке Visual Assets в редакторе манифеста.

2. В поле Background color (Цвет фона) введите **"deepskyblue"**.

Примечание: Если вы оставите поле Background прозрачным, ваши плитки унаследую акцентный цвет, выбранный пользователем для системы в настройках **Windows 10 Settings (Настройки Windows 10) > Personalization (Персонализация) > Colors (Цвета)**. В этом упражнении мы зададим конкретный цвет фона, чтобы было легче видеть изображения плиток в редакторе манифеста.

3. В разделе **Square71x71Logo** используйте символ многоточия под изображением для масштаба 200 (**scale 200)**, чтобы открыть диалоговое окно **Select Image** (Выбор изображения). Перейдите в папку **Lab Assets** и выберите файл **Square71x71Logo.scale-200.png**. Нажмите **Open**, чтобы добавить изображение в качестве логотипа.

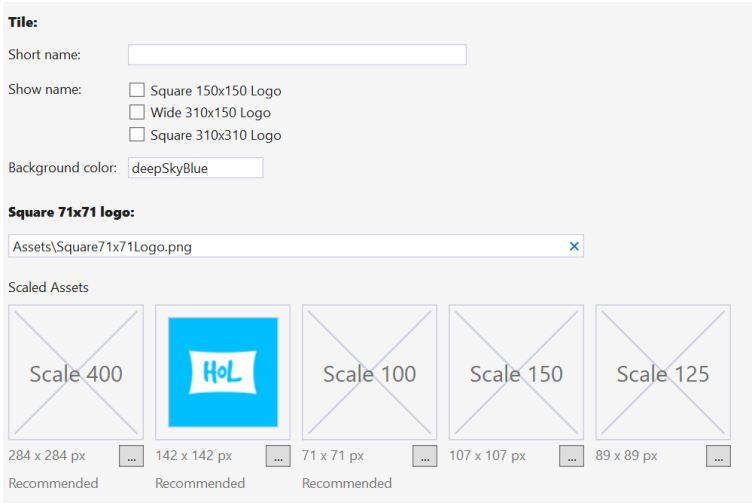

Рисунок 5

Изображение для Square71x71Logo в редакторе манифеста.

4. Повторите **шаг 3**, чтобы добавить изображения **Square150x150Logo.scale-200.png**, **Wide310x150Logo.scale-200.png, Square310x310Logo.scale-200.png** и **Square44x44Logo.scale-200.png**. Когда дойдете до **StoreLogo.png,** добавьте изображение в **масштабе 100**. В каждом случае, если вы заменяете одно из шаблонных изображений, существующих в проекте, Visual Studio уточнит у вас, действительно ли вы хотите заменить. Нажмите Yes.

5. Выберите пункт **Splash Screen** в панели слева. В настройках экрана загрузки установите цвет фона также в значение **deepskyblue**. Используйте кнопку с многоточием под масштабом 200, чтобы выбрать **SplashScreen.scale-200.png** и папки Lab Assets.

6. Соберите и запустите приложение. После разворачивания, найдите приложение в меню Пуск (Start). Щелкните правой кнопкой по имени приложения и выберите **Pin to Start (закрепить на экране Пуск)**.

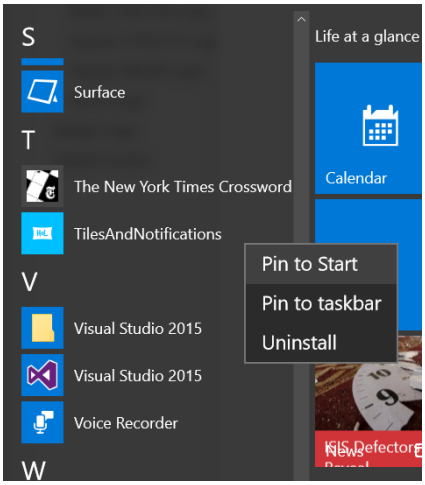

Рисунок 6

Закрепление приложения в меню Пуск.

7. Ваша плитка по умолчанию появится в меню Пуск уже с ваши логотипом и соответствующим цветом фона, указанными в манифесте. Нажмите правой кнопкой мыши на плитке и поменяйте размер, чтобы посмотреть, как плитка масштабируется.

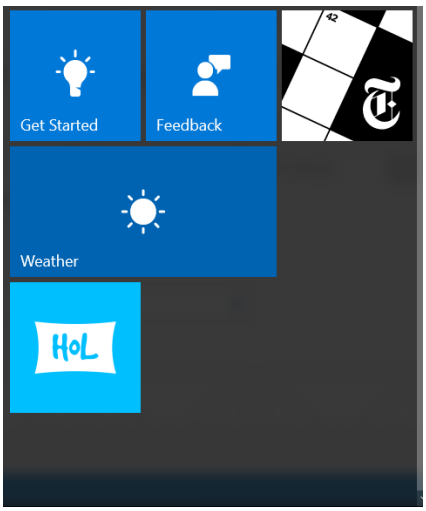

Рисунок 7
Плитка закрепленного приложения

8. Остановите отладку и вернитесь в Visual Studio.

Задача 3 – Обновление счетчика на плитке

Счетчик на плитке – это отличный способ для отображения информации, которую можно было бы легко считать взглядом, например, количество новых элементов в приложении. Мы посмотрим, как обновить бейдж со счетчиком для плитки по умолчанию, это быстрый и простой способ добавить дополнительную ценность присутствию вашего приложения на экране Пуск.

1. Щёлкните правой кнопкой мыши по имени проекта и выберите **Add (Добавить) > New Folder (Новая папка)**. Назовите папку **Services**.

2. Щелкните правой кнопкой мыши по папке и выберите **Add (Добавить) > Class (Класс)**. Назовите файл с классом **TileService.cs**.

3. Откройте файл **TileService.cs** и сделайте класс публичным (**public**).

```
C#
namespace TilesAndNotifications.Services
{
    public class TileService
    {
    }
}
```

4. Добавьте статичный метод для обновления счетчика на плитке. Добавьте с помощью `using` ссылки на `Windows.Data.Xml.Dom` и `Windows.UI.Notifications`.

C#

```
using Windows.Data.Xml.Dom;
using Windows.UI.Notifications;

namespace TilesAndNotifications.Services
{
    public class TileService
    {
        static public void SetBadgeCountOnTile(int count)
        {
            // Update the badge on the real tile
            XmlDocument badgeXml =
BadgeUpdateManager.GetTemplateContent(BadgeTemplateType.BadgeNumber);

            XmlElement badgeElement =
(XmlElement)badgeXml.SelectSingleNode("/badge");
            badgeElement.SetAttribute("value", count.ToString());

            BadgeNotification badge = new BadgeNotification(badgeXml);

BadgeUpdateManager.CreateBadgeUpdaterForApplication().Update(badge);
        }
    }
}
```

5. Вернитесь в файл MainPage.xaml. Создайте кнопку для обновления счетчика

XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <Button Click="UpdateBadge" VerticalAlignment="Top" Margin="12">Update
Badge Count</Button>
</Grid>
```

6. В коде для MainPage.xaml.cs добавьте свойство **_count** типа `int` и метод **UpdateBadge()** для вызова сервиса TileService. Добавьте пространство имен **TilesAndNotifications.Services**.

C#

```
public MainPage()
{
    this.InitializeComponent();
}
int _count;
private void UpdateBadge (object sender, RoutedEventArgs e)
{
    _count++;
    TileService.SetBadgeCountOnTile(_count);
}
```

7. Соберите и запустите приложение на локальной машине. После развертывания снова найдите ваше приложение в меню Пуск.

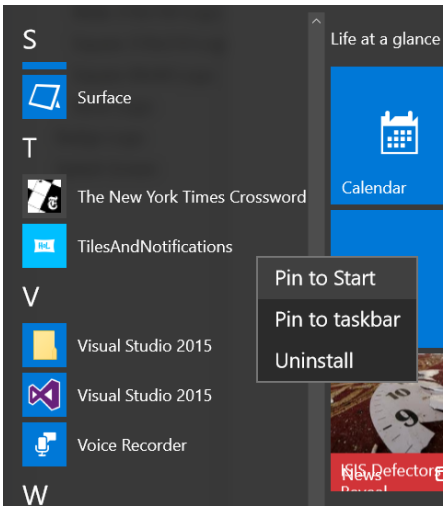

Рисунок 8

Закрепление приложения (если вы его открепили).

8. В запущенном приложении нажмите на кнопку **Update Badge Count**. Когда вы вернетесь к свое живой плитке на экране Пуск, вы увидите, что значение счетчика обновилось до 1. Этот счетчик будет увеличиваться каждый раз, когда вы вызываете метод **UpdateBadge()**.

9. Остановите отладку и вернитесь в Visual Studio.

Упражнение 2: Создание адаптивных живых плиток

Живые плитки в Windows 10 используют адаптивные шаблоны для показа контента с учетом параметров устройства и плотности экрана. Хотя шаблоны прежних версий все еще совместимы с живыми плитками, адаптивные шаблоны предоставляют вам большую свободу в плане выбора, как контент будет отображаться на разных устройствах. Группы и подгруппы позволяют вам семантически связывать контент в пределах плитки. В настоящем упражнении мы создадим адаптивный макет и используем его для показа статичных данных на плитке.

Задача 1 – Добавление модели

В типичном приложении живая плитка отображает имеющиеся в самом приложении данные. В рамках данной задачи мы создадим класс как набросок статических данных с целью их отображения на плитках.

1. Щелкните правой кнопкой мыши по имени проекта и создайте папку **Models**.

2. Добавьте класс **PrimaryTile.cs** в папку Models.

3. Откройте класс **PrimaryTile** и сделайте его публичным. Добавьте статичные данные в виде строковых полей, чтобы задать данные, которыми мы будем заполнять плитки.

C#

```
namespace TilesAndNotifications.Models
{
    public class PrimaryTile
    {
        public string time { get; set; } = "8:15 AM, Saturday";
        public string message { get; set; } = "Lorem ipsum dolor sit amet,
consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore.";
        public string message2 { get; set; } = " At vero eos et accusamus
et iusto odio dignissimos ducimus qui blanditiis praesentium voluptatum
deleniti atque corrupti quos dolores et quas molestias excepturi sint
occaecati cupiditate non provident.";
        public string branding { get; set; } = "name";
        public string appName { get; set; } = "HoL";
    }
}
```

Задача 2 – Построение XML-описания плитки

Адаптивное описание плитки задается в XML. В рамках настоящей задачи мы сгенерируем XML, необходимый для отображения текстового контента из PrimaryTile на маленькой и средней живых плитках.

1. В файле **TileService.cs** добавьте пространства имен **TilesAndNotifications.Models** и **System.Xml.Linq**.

C#

```
using TilesAndNotifications.Models;
using System.Xml.Linq;
```

2. Добавьте метод **CreateTiles** для генерации XML под маленькую и среднюю плитки.

C#

```
public static Windows.Data.Xml.Dom.XmlDocument CreateTiles (PrimaryTile
primaryTile)
{
    XDocument xDoc = new XDocument(
        new XElement("tile", new XAttribute("version", 3),
            new XElement("visual",
                // Small Tile
                new XElement("binding", new XAttribute("branding",
primaryTile.branding), new XAttribute("displayName", primaryTile.appName),
new XAttribute("template", "TileSmall"),
                    new XElement("group",
                        new XElement("subgroup",
```

```
                            new XElement("text", primaryTile.time, new
XAttribute("hint-style", "caption")),
                            new XElement("text", primaryTile.message, new
XAttribute("hint-style", "captionsubtle"), new XAttribute("hint-wrap",
true), new XAttribute("hint-maxLines", 3))
                        )
                    )
                ),

                // Medium Tile
                new XElement("binding", new XAttribute("branding",
primaryTile.branding), new XAttribute("displayName", primaryTile.appName),
new XAttribute("template", "TileMedium"),
                    new XElement("group",
                        new XElement("subgroup",
                            new XElement("text", primaryTile.time, new
XAttribute("hint-style", "caption")),
                            new XElement("text", primaryTile.message, new
XAttribute("hint-style", "captionsubtle"), new XAttribute("hint-wrap",
true), new XAttribute("hint-maxLines", 3))
                        )
                    )
                )
            )
        );

    Windows.Data.Xml.Dom.XmlDocument xmlDoc = new
Windows.Data.Xml.Dom.XmlDocument();
    xmlDoc.LoadXml(xDoc.ToString());
return xmlDoc;
}
```

Примечание: Существует несколько элементов, которые вы можете включить в свою схему адаптивной плитки. Для каждого типа элементов вы можете выбрать один из предопределенных стилей. Дополнительная информация доступна в документации https://msdn.microsoft.com/en-us/library/windows/apps/Mt186446.aspx.

Дополнительные инструкции и примеры можно найти в статье: Схема адаптивных плиток и документация.

3. Откройте MainPage.xaml. Добавьте новую кнопку после кнопки обновления счетчика, она будет обновлять основную плитку. Мы расположим кнопки в панели StackPanel, чтобы облегчить компоновку.

XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <StackPanel>
        <Button Click="UpdateBadge" VerticalAlignment="Top"
Margin="12">Update Badge Count</Button>
```

```
        <Button Click="UpdatePrimaryTile" VerticalAlignment="Top"
Margin="12">Update Primary Tile</Button>
    </StackPanel>
</Grid>
```

4. В файле кода для MainPage добавьте ссылки на пространства имен **TilesAndNotifications.Models** и **Windows.UI.Notificiations**.

C#

```
using TilesAndNotifications.Models;
using Windows.UI.Notifications;
```

5. Добавьте метод **UpdatePrimaryTile()** в файле MainPage.xaml.cs.

C#

```
private void UpdatePrimaryTile(object sender,
Windows.UI.Xaml.RoutedEventArgs e)
{
    var xmlDoc = TileService.CreateTiles(new PrimaryTile());

    var updater = TileUpdateManager.CreateTileUpdaterForApplication();
    TileNotification notification = new TileNotification(xmlDoc);
    updater.Update(notification);
}
```

Примечание: Каждый раз обновляя плитку, вы на самом деле создаете новый экземпляр такой плитки. При создании плитки в приложении с динамичными данными, вы можете передать наиболее актуальные сведения для отображения на плитке.

6. Соберите и запустите приложение. Закрепите приложение на экране Пуск, если это еще не сделано.

7. Нажмите кнопку **Update Primary Tile** в запущенном приложении, чтобы начать обновление плитки. Откройте меню Пуск на вашем устройстве и подождите, пока изображение плитки по умолчанию обновит свой вид. Измените размер плитки, чтобы убедиться, что обновление затрагивает как маленькую, так и среднюю плитки.

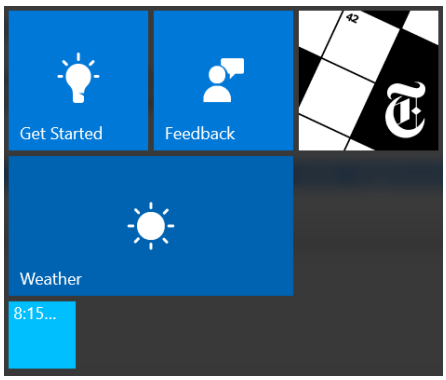

Рисунок 9

Маленькая живая плитка.

8. Измените размер плитки на **средний**. Обратите внимание, что теперь при обновлении плитки показывается больше информации из адаптивного шаблона.

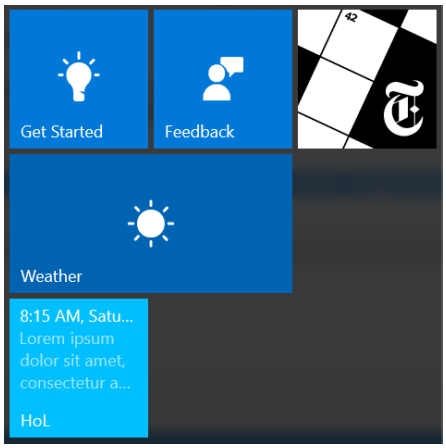

Рисунок 10
Средняя живая плитка.

9. Остановите отладку и вернитесь в Visual Studio.

Задача 3 – Создание адаптивного шаблона для широкой и большой плиток.

1. Добавьте широкую и большую плитки к методу **CreateTiles()**. Плитки большего размера имеют дополнительное пространство для отображения помимо текста также и изображений. В нашем случае мы покажем изображение, уже присутствующее в папке Assets. Убедитесь, что добавили запятую после кода для создания XML-описания средней плитки, чтобы продолжить список.

C#

```
),

// Wide Tile
new XElement("binding", new XAttribute("branding", primaryTile.branding),
new XAttribute("displayName", primaryTile.appName), new
XAttribute("template", "TileWide"),
    new XElement("group",
        new XElement("subgroup",
            new XElement("text", primaryTile.time, new XAttribute("hint-
style", "caption")),
            new XElement("text", primaryTile.message, new XAttribute("hint-
style", "captionsubtle"), new XAttribute("hint-wrap", true), new
XAttribute("hint-maxLines", 3)),
            new XElement("text", primaryTile.message2, new
XAttribute("hint-style", "captionsubtle"), new XAttribute("hint-wrap",
true), new XAttribute("hint-maxLines", 3))
            ),
        new XElement("subgroup", new XAttribute("hint-weight", 15),
            new XElement("image", new XAttribute("placement", "inline"),
new XAttribute("src", "Assets/StoreLogo.png"))
        )
```

```
    )
),

//Large Tile
new XElement("binding", new XAttribute("branding", primaryTile.branding),
new XAttribute("displayName", primaryTile.appName), new
XAttribute("template", "TileLarge"),
    new XElement("group",
        new XElement("subgroup",
            new XElement("text", primaryTile.time, new XAttribute("hint-
style", "caption")),
            new XElement("text", primaryTile.message, new XAttribute("hint-
style", "captionsubtle"), new XAttribute("hint-wrap", true), new
XAttribute("hint-maxLines", 3)),
            new XElement("text", primaryTile.message2, new
XAttribute("hint-style", "captionsubtle"), new XAttribute("hint-wrap",
true), new XAttribute("hint-maxLines", 3))
            ),
        new XElement("subgroup", new XAttribute("hint-weight", 15),
            new XElement("image", new XAttribute("placement", "inline"),
new XAttribute("src", "Assets/StoreLogo.png"))
        )
    )
)
```

Примечание: Чтобы отобразить широкую и большую живые плитки, вы должны иметь изображения WideLogo и Square310x310Logo, определяемые в вашем манифесте приложения. Мы добавили данные изображения в Упражнении 1.

2. Соберите и запустите приложение. Нажмите на кнопку Update Primary Tile. Измените размер закрепленной плитки приложения на **Wide (Широкая)** и **Large (Большая)**. Обратите внимание, что, хотя в обоих случаях есть потенциальная возможность отобразить как **message**, так и **message2**, большая плитка с большой вероятностью будет иметь место для дополнительного текста.

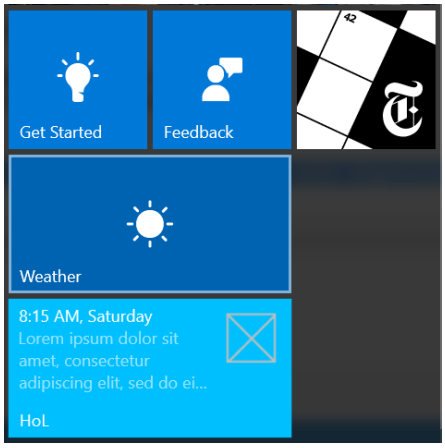

Рисунок 11

Широкая живая плитка

Рисунок 12

Большая живая плитка имеет место для большего количества контента.

Примечание: Большие плитки не доступны на устройствах с Windows 10 Mobile.

3. Остановите отладку и вернитесь в Visual Studio.

Упражнение 3: Запуск интерактивного уведомления (Toast)

Помимо адаптивных плиток, в Windows 10 также обновились уведомления – теперь они тоже адаптивные, а также интерактивные. Всплывающие уведомления могут включать контент, встроенные изображения и возможность ввода данных пользователем. Уведомления могут открыть приложение для выполнения задачи или запустить фоновый сервис без запуска основного приложения.

В данном упражнении мы создадим уведомление с управляющими кнопками, которое обновит данные в приложении через фоновый сервис. Приложение будет содержать в качестве примера to-do элемент, состоящий из чек-бокса и описания. Уведомление позволит отметить элемент как выполненный или незавершенный, и это проявится при обновлении основной страницы приложения.

Задача 1 – Создание сервиса для уведомления

XML для уведомления устроен очень похоже на XML плитки, который мы создавали в прошлом упражнении. Мы создадим класс сервиса для создания XML-описания всплывающего уведомления и используем его из кода MainPage.

1. Щелкните правой кнопкой мыши на папку **Services** и выберите **Add > Class**. Назовите класс **ToastService.cs**.

2. Сделайте класс публичным (**public)** и статичным (**static)**, также добавьте пространства имен **System.Xml.Linq** и **Windows.Data.Xml.Dom**.

C#

```
using System.Xml.Linq;
using Windows.Data.Xml.Dom;

namespace TilesAndNotifications.Services
{
    public static class ToastService
```

3. Добавьте метод **CreateToast()** для создания и загрузки XmlDocument.

C#

```
public static class ToastService
{
    public static XmlDocument CreateToast()
    {
        var xDoc = new XDocument(
            new XElement("toast",
                new XElement("visual",
                    new XElement("binding", new XAttribute("template",
"ToastGeneric"),
                        new XElement("text", "To Do List"),
                        new XElement("text", "Is the task complete?")
                        ) // binding
                    ), // visual
                new XElement("actions",
                    new XElement("action", new XAttribute("activationType",
"background"),
                        new XAttribute("content", "Yes"), new
XAttribute("arguments", "yes")),
                    new XElement("action", new XAttribute("activationType",
"background"),
                        new XAttribute("content", "No"), new
XAttribute("arguments", "no"))
                    ) // actions
                )
            );
        var xmlDoc = new XmlDocument();
        xmlDoc.LoadXml(xDoc.ToString());
        return xmlDoc;
    }
}
```

Данный код создает следующее описание всплывающего уведомления с помощью адаптивной XML-схемы:

```
<toast>
  <visual>
    <binding template="ToastGeneric">
      <text>To Do List</text>
      <text>Is the task complete?</text>
    </binding>
  </visual>
  <actions>
    <action activationType="background" content="Yes" arguments="yes" />
    <action activationType="background" content="No" arguments="no" />
  </actions>
</toast>
```

Примечание: Кнопки **action** пока ни с чем не связаны но они понадобятся нам далее в упражнении. Значение **background** для **activationType** означает, что действие не запустит приложение при активации. Подробнее XML-схема для уведомлений описана в документации: https://msdn.microsoft.com/en-us/library/windows/apps/br230849.aspx

4. Сохраните и закройте ToastService.

5. Откройте **MainPage.xaml**. Добавьте определение строки в сетку (Grid) и второй StackPanel в нижней строке.

XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <Grid.RowDefinitions>
        <RowDefinition />
        <RowDefinition />
    </Grid.RowDefinitions>
    <StackPanel Grid.Row="0" Margin="12">
        <TextBlock Text="Adaptive Tiles" FontSize="20" FontWeight="Light"
/>
        <Button Click="UpdateBadge" VerticalAlignment="Top"
Margin="12">Update Badge Count</Button>
        <Button Click="UpdatePrimaryTile" VerticalAlignment="Top"
Margin="12">Update Primary Tile</Button>
    </StackPanel>
    <StackPanel Grid.Row="1" Margin="12">

    </StackPanel>
</Grid>
```

6. Добавьте название секции и кнопку Notify в новую StackPanel. Метод Notify() мы создадим на следующем шаге.

XAML

```
<StackPanel Grid.Row="1" Margin="12">
    <TextBlock Text="Interactive Toast" FontSize="20" FontWeight="Light" />
    <Button Click="Notify" Margin="12">Notify</Button>
</StackPanel>
```

7. Добавьте пространства имен **Windows.UI.Notifications** и **TilesAndNotifications.Services** в код MainPage.

C#

```
using TilesAndNotifications.Services;
```

8. Создайте метод **Notify()** в коде страницы.

C#

```
private void Notify(object sender, RoutedEventArgs e)
{
    var xmlDoc = ToastService.CreateToast();
    var notifier = ToastNotificationManager.CreateToastNotifier();
    var toast = new ToastNotification(xmlDoc);
    notifier.Show(toast);
}
```

Соберите и запустите приложение на локальной машине. Используйте кнопку **Notify** для отправки всплывающего уведомления. При появлении всплывающего уведомления вы можете щелкнуть по управляющим кнопкам, но ничего не произойдет. Мы реализуем данные действия в последующей задаче.

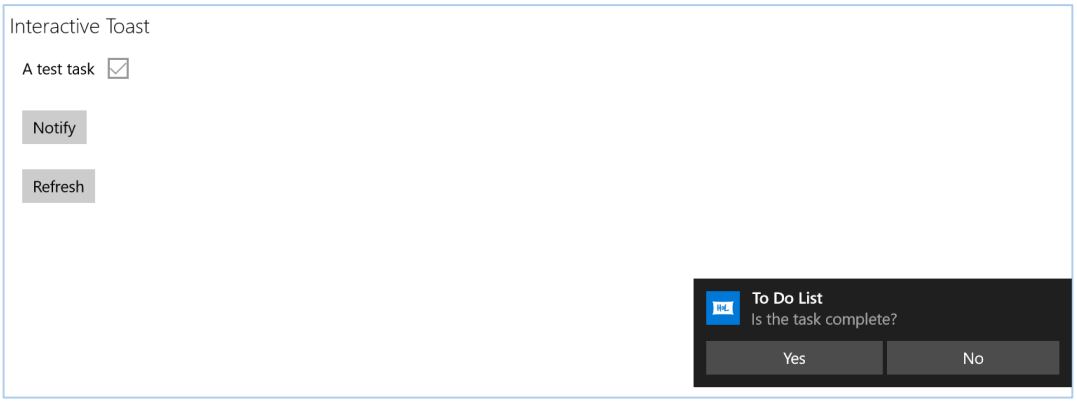

Рисунок 13

Уведомление с кнопками действия.

9. Остановите отладку и вернитесь в Visual Studio.

Задача 2 – Создание модели данных и вспомогательных методов

Прежде, чем мы сможем реализовать действия в уведомлениях, нам понадобится простой класс и вспомогательная конструкция, чтобы сэмулировать to-do элемент. Мы создадим их в библиотеке классов, чтобы сделать доступными из любого проекта в решении. Как приложение, так и фоновый сервис, определяемы в Windows Runtime компоненте, будут требовать доступа к модели.

1. Щёлкнитесь правой кнопкой мыши на Solution Explorer (Обозреватель решений). Затем Add (Добавить) > New Project (Новый проект). Выберите тип проекта: **Visual C# >**

Windows > Universal > Class Library (Universal Windows). Назовите его **TasksAndNotifications.Library**.

2. Удалите **Class1.cs**, используя контекстное меню обозревателя. Если необходимо, подтвердите, что действительно хотите удалить.

3. Щелкните правой кнопкой мыши на проект **TilesAndNotifications.Library** и выберите **Add (Добавить) > Existing Item (Существующий элемент)**. Найдите папку **Lab Assets** в директории практикума и выберите файлы **ToDoTask.cs** и **ToDoTaskFileHelper.cs**. Добавьте в вашу библиотеку классов.

4. Откройте **ToDoTask.cs**. В файле вы увидите, что простой ToDo-элемент состоит из **Id**, **Description** и флага **IsComplete**. Также в классе есть два метода для управления сериализацией в и из JSON.

Примечание: Мы будем хранить данные ToDo в виде файла в формате JSON, чтобы сделать их доступными для фоновой задачи, которая может иметь доступ к файлам в пакете приложений. Позднее в рамках данного упражнения вы создадите фоновую задачу.

5. Откройте **ToDoTaskFileHelper.cs**. Данный вспомогательный код сохраняет сериализованный JSON в файл и считывает его назад из файла при необходимости.

6. Вернитесь к своему приложению в Solution Explorer. Щелкните правой кнопкой мыши на папку **References** и добавьте **TilesAndNotifications.Library** как ссылку.

7. Щелкните правой кнопкой мыши на папку **Assets** и выберите **Add > Existing Item**. Найдите папку Lab Assets и добавьте стартовый файл **task.json**.

8. Щелкните правой кнопкой мыши на файл **task.json** и откройте его свойства (**Properties)**. Установите его свойство **Build Action** в значение **Content**, и свойство **Copy to Output Directory** в значение **Copy Always**. Закройте панель свойств.

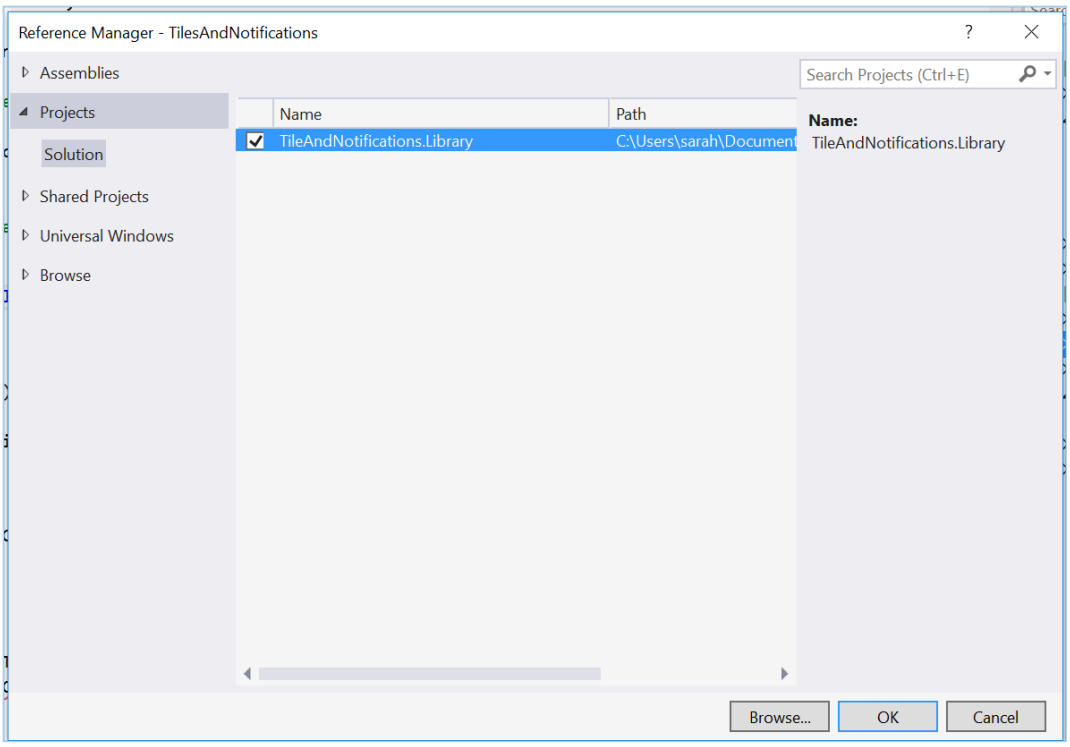

Рисунок 14

Добавление библиотеки классов по ссылке.

Задача 3 – Добавление элемента ToDo и реализация INotifyPropertyChanged

Мы отобразим элемент ToDo на странице MainPage с использование чек-бокса для отображения статуса IsComplete. Задача ToDo будет связана со свойством в коде. Чтобы наш интерфейс понимал, когда наша фоновая задача вносит изменения в значение ToDo (позднее в данном упражнении), мы реализуем интерфейс **INotifyPropertyChanged** для CurrentToDoTask.

1. Откройте **MainPage.xaml.cs**. Добавьте пространства имен **TilesAndNotifications.Library** и **System.ComponentModel**.

C#

```
using TilesAndNotifications.Library;
using System.ComponentModel;
```

2. Добавьте приватное своейство для ToDoTask и публичное свойство для доступа к нему.

C#

```
private int _count;
private ToDoTask _currentToDoTask;

public MainPage()
{
    InitializeComponent();
    Loaded += MainPage_Loaded;
}
```

```
public ToDoTask CurrentToDoTask
{
    get { return _currentToDoTask; }
    set
    {
        _currentToDoTask = value;
    }
}
```

3. Реализуйте **INotifyPropertyChanged** для сообщения в UI, когда **CurrentToDoTask** изменяется. Мы свяжем свойство с UI далее.

C#

```
public sealed partial class MainPage : Page, INotifyPropertyChanged
{
…
    public ToDoTask CurrentToDoTask
    {
        get { return _currentToDoTask; }
        set
        {
            _currentToDoTask = value;
            PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(nameof(CurrentToDoTask)));
        }
    }

    public event PropertyChangedEventHandler PropertyChanged;
}
```

Примечание: InotifyPropertyChanged предоставляет обобщенные уведомления об изменениях свойств подписчикам, которые обычно свящаны с изменяющимся значением. В данном примере наш UI должен знать, когда вызванный всплывающим уведомлением фоновый сервис обновляет элемент ToDo. Подробнее InotifyPropertyChanged описан тут: https://msdn.microsoft.com/en-us/library/windows/apps/windows.ui.xaml.data.inotifypropertychanged

4. Откройте MainPage.xaml добавьте элементы управления TextBlock и CheckBox для отображения ToDo-задачи и ее статуса. Добавьте кнопку для обновления данных. Мы добавим метод обработки нажатия в следующем шаге.

XAML

```
<StackPanel Grid.Row="1" Margin="12">
    <TextBlock Text="Interactive Toast" FontSize="20" FontWeight="Light" />
    <StackPanel Orientation="Horizontal" Margin="12">
        <TextBlock x:Name="Description" VerticalAlignment="Center"
Text="{x:Bind CurrentToDoTask.Description, Mode=OneWay}"/>
        <CheckBox Margin="12,0,0,0" IsChecked="{x:Bind
CurrentToDoTask.IsComplete, Mode=OneWay}" IsEnabled="False" />
```

```
    </StackPanel>
    <Button Click="Notify" Margin="12">Notify</Button>
    <Button Click="{x:Bind Refresh}" Margin="12">Refresh</Button>
</StackPanel>
```

5. Вернитесь к коду MainPage. Добавьте асинхронный метод **Refresh()** для чтения последнего значения задачи из данных в JSON-файле.

C#

```
private async void Refresh()
{
    var json = await ToDoTaskFileHelper.ReadToDoTaskJsonAsync();
    CurrentToDoTask = ToDoTask.FromJson(json);
}
```

6. Подпишитесь на Loaded-событие и вызовите метод Refresh() при загрузке страницы.

C#

```
public MainPage()
{
    InitializeComponent();
    Loaded += MainPage_Loaded;
}

…

private void MainPage_Loaded(object sender, RoutedEventArgs e)
{
    Refresh();
}

private async void Refresh()
{
    var json = await ToDoTaskFileHelper.ReadToDoTaskJsonAsync();
    CurrentToDoTask = ToDoTask.FromJson(json);
}
```

7. Соберите и запустите приложение. Вы увидите тестовую задачу, с по умолчанию выделенным чек-боксом IsComplete.

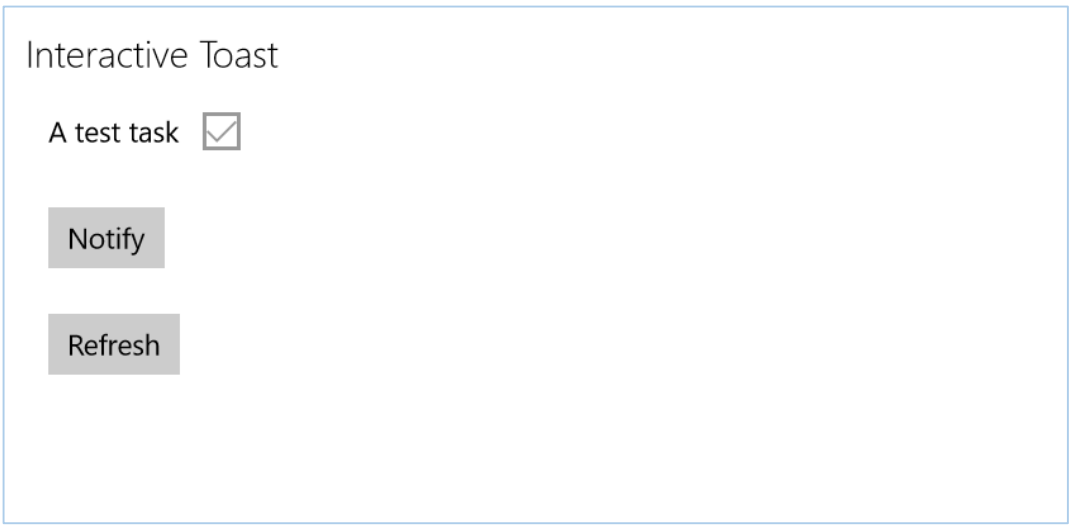

Рисунок 15
Элемент ToDo-задачи.

8. Остановите отладку и вернитесь в Visual Studio.

Задача 4 – Создание фонового сервиса

В этой задаче мы создадим фоновый сервис как часть компонента для Windows Runtime.

1. Щелкните правой кнопкой мыши на Solution Explorer и выберите Add > New Project. Добавьте проект типа **Visual C# > Windows > Universal > Windows Runtime Component (Universal Windows)**. Назовите его **BackgroundTasks**.

2. Щелкните правой кнопкой мыши на папку **References** в проекте BackgroundTasks и выберите **Add > Reference**. Добавте проект **TilesAndNotifications.Library** по ссылке. Библиотека даст доступ к ToDoTask и вспомогательному коду.

3. Щелкните правой кнопкой мыши и **переименуйте** Class1.cs в **ToastUpdateTask.cs**. При запросе согласитесь выполнить переименование всех ссылок с **Class1** на **ToastUpdateTask**.

4. Откройте **ToastUpdateTask.cs** и добавьте пространства имен **Windows.ApplicationModel.Background**, **Windows.UI.Notifications** и **TilesAndNotifications.Library.**

```
C#
using Windows.ApplicationModel.Background;
using Windows.UI.Notifications;
using TilesAndNotifications.Library;
```

5. Реализуйте интерфейс **IBackgroundTask**. Создайте и завершите deferral внутри метода Run.

C#

```
namespace BackgroundTasks
{
    public sealed class ToastUpdateTask : IBackgroundTask
    {
        public async void Run(IBackgroundTaskInstance taskInstance)
        {
            var deferral = taskInstance.GetDeferral();

            deferral.Complete();
        }
    }
}
```

6. Преобразуйте **taskInstance.TriggerDetails** в **ToastNotificationActionTriggerDetails**. Данные детали передаются из действия **Yes** в уведомлении. Если детали не пусты, прочитайте JSON-задачу и установите ее флаг **IsComplete** в true.

C#

```
namespace BackgroundTasks
{
    public sealed class ToastUpdateTask : IBackgroundTask
    {
        public async void Run(IBackgroundTaskInstance taskInstance)
        {
            var deferral = taskInstance.GetDeferral();

            var details = taskInstance.TriggerDetails as
ToastNotificationActionTriggerDetail;
            if (details != null)
            {
                string arguments = details.Argument;
                // this is where you would retrieve any user input
                var userInput = details.UserInput;

                var json = await
ToDoTaskFileHelper.ReadToDoTaskJsonAsync();
                var task = ToDoTask.FromJson(json);

                task.IsComplete = arguments == "yes";

                await ToDoTaskFileHelper.SaveToDoTaskJson(task.ToJson());
            }

            deferral.Complete();
        }
    }
}
```

Примечание: Дополнительную информацию об аргументах и деталях триггеров уведомлений можно найти тут: https://msdn.microsoft.com/en-us/library/windows/apps/windows.ui.notifications

Задача 5 – Регистрация фонового сервиса (задачи)

До запуска приложения необходимо зарегистрировать фоновый сервис в приложении TilesAndNotifications и объявить его в манифесте.

1. Вернитесь к своему проекту приложения. Откройте пакет манифеста в редакторе и перейдите на вкладку **Declarations**. Выберите **BackgroundTasks** из списка **Available Declarations (Доступные объявления)** и нажмите для **Add** добавления объявления.

2. Установите тип **task type** в значение **System Event** и **Entry point** в значение **BackgroundTasks.ToastUpdateTask**. Сохранение и закройте манифест.

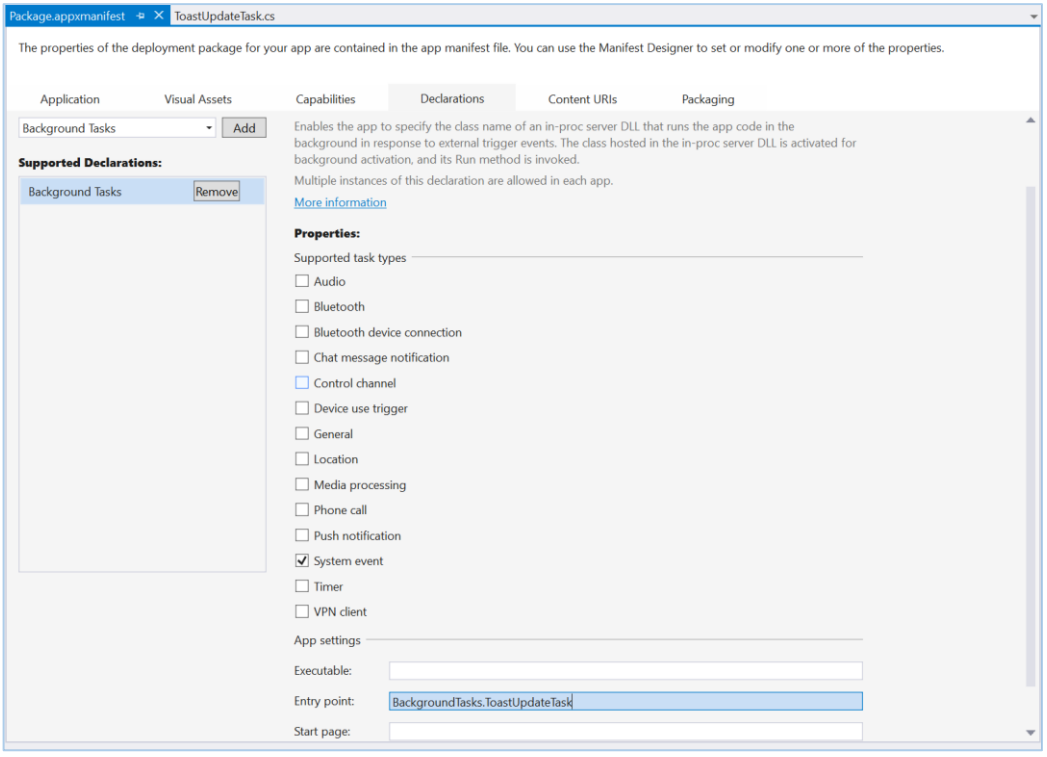

Рисунок 16

Объявление фоновой задачи в манифесте приложения.

3. В вашем проекте **TilesAndNotifications** щелкните правой кнопкой мыши на папку **References** и выберите **Add > Reference**. Добавьте проект **BackgroundTasks** по ссылке.

4. Откройте **App.xaml.cs**. Добавьте пространства имен **Windows.ApplicationModel.Background** и **BackgroundTasks**.

C#

```
using Windows.ApplicationModel.Background;
using BackgroundTasks;
```

5. Добавьте **async**-метод **RegisterBgTask** для управления регистрацией сервиса.

C#

```
private async void RegisterBgTask(string taskName, Type taskType, bool
isInternetRequired = false)
        {
            var backgroundAccessStatus = await
BackgroundExecutionManager.RequestAccessAsync();

            if (backgroundAccessStatus ==
BackgroundAccessStatus.AllowedMayUseActiveRealTimeConnectivity ||
                backgroundAccessStatus ==
BackgroundAccessStatus.AllowedWithAlwaysOnRealTimeConnectivity)
            {
                // Check to see if the task has already been registered
                if (BackgroundTaskRegistration.AllTasks.Any(t =>
t.Value.Name == taskName))
                {
                    return;
                }

                // Register the toast update task
                var taskBuilder = new BackgroundTaskBuilder
                {
                    Name = taskName,
                    TaskEntryPoint = taskType.FullName
                };

                taskBuilder.SetTrigger(new
ToastNotificationActionTrigger());
                var registration = taskBuilder.Register();
            }
        }
```

6. Выховите метод **RegisterBgTask** из конструктора приложения и передайте ему тип (typeof) **ToastUpdateTask**.

C#

```
public App()
{
    this.InitializeComponent();
    this.Suspending += OnSuspending;
    RegisterBgTask("ToastUpdateTask", typeof(ToastUpdateTask));
}
```

7. Соберите и запустите приложение. Используйте кнопку Notify отправки всплывающего уведомления. Если ваша задача была уже завершена, выберите **No** в качестве

действия. Используйте кнопку Refresh, чтобы обновить статус задачи. Чек-бокс будет отображать выбор, сделанный в уведомлении.

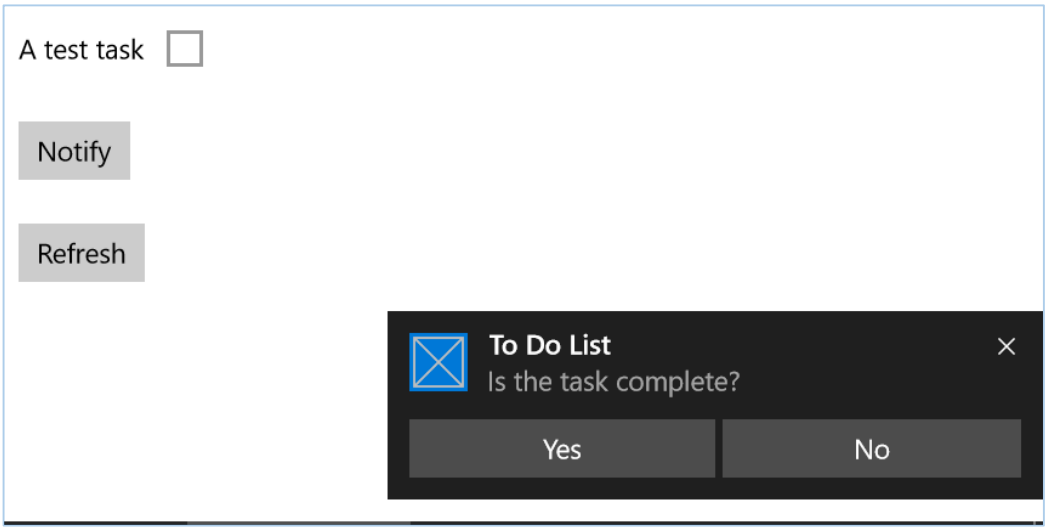

Рисунок 17
Изменение статуса задачи из уведомления.

8. Остановите отладку и вернитесь в Visual Studio.

Резюме

В рамках данной лабораторной работы вы сумели добавить в свое приложение собственные изображения в качестве графических ресурсов и использовать их для создания разнообразных плиток по умолчанию, которые могут отображать счетчик уведомлений. Вы узнали о новой адаптивной схеме плитки и научились создавать живые плитки (маленькие, средние, широкие и большие), а затем поработали над интерактивными всплывающими уведомлениями (toast) с фоновой активацией.