К. Брокшмидт

Пользовательский интерфейс приложений для Windows 8, созданных с использованием HTML, CSS и JavaScript

Пользовательский интерфейс приложений для Windows 8, созданных с использованием HTML, CSS и JavaScript

2-е издание, исправленное

Брокшмидт К.

Национальный Открытый Университет "ИНТУИТ"
2016

Пользовательский интерфейс приложений для Windows 8, созданных с использованием HTML, CSS и JavaScript/ К. Брокшмидт - М.: Национальный Открытый Университет "ИНТУИТ", 2016

Учебный курс посвящен особенностям проектирования и разработки пользовательского интерфейса приложений для Windows 8 с использованием технологий HTML, CSS и JavaScript.

Курс знакомит читателя с особенностями работы с командными элементами пользовательского интерфейса, с подходами к обработке ввода данных пользователем. Он подробно исследует целенаправленный подход к анимации, соответствующей стилю Windows. Кроме того, читатель узнает о реализации мультимедийных возможностей приложений для Магазина Windows и о работе с пользовательскими данными и данными приложений для организации единой рабочей среды, ориентированной на работу пользователя с несколькими устройствами.

(c) ООО "ИНТУИТ.РУ", 2013-2016
(c) Брокшмидт К., 2013-2016

Командный пользовательский интерфейс

Здесь читателю предлагается подробное описание особенностей обработки состояний приложения, методика работы с настраиваемыми пользователем параметрами приложения и с пользовательскими данными

Файлы к данной лекции Вы можете скачать здесь скачать: http://old.intuit.ru/department/se/uiwin8htmljs/1/1.zip.

Одной из первых реакций у новых пользователей, не знакомых с Windows 8 и с приложениями для Магазина Windows, может быть следующая: "Где же меню? А лента? Как я могу указать приложению, чтобы оно сделало что-то с элементами, которые я выбрал из списка?". Это – вполне естественная реакция, которая возникает до тех пор, пока пользователи не привыкнут к расположению командных элементов управления, что дает еще одно значение, хотя и мирское, изречению: "Блаженны не видевшие и уверовавшие".

Следуя принципу дизайна "прежде всего содержимое, и лишь затем внешнее оформление", элементы пользовательского интерфейса, которые существуют исключительно для вызова некоторых действий, и, таким образом, не содержат значимого содержимого, попадают в категорию "внешнего оформления". Как таковые, они обычно скрыты до тех пор, пока не понадобятся, как команды системного уровня наподобие панели чудо-кнопок. Пользователь показывает желание воспользоваться подобными командами посредством соответствующего жеста. Это – скольжение от верхнего или нижнего краев экрана, щелчок правой кнопки мыши, или комбинация клавиш Win+Z, которые вызывают командные панели приложения, расположенные сверху или снизу. Скольжение от левого края экрана, щлелчок мышью по верхнему левому углу экрана, или сочетание клавиш Win+Tab позволяет переключаться между приложениями. И скольжение от правого края экрана, щелчок мышью по верхнему правому или нижнему правому углам экрана, или сочетание клавиш Win+C открывает панель чудо-кнопок. (Сочетания клавиш Win+Q, Win+H, Win+i активируют, соответственно, чудо кнопки Поиск, Общий доступ и Параметры напрямую). Приложение реагирует на различные чудо-кнопки посредством реализации отдельных контрактов, как мы увидим в

следующих лекциях.

Команды приложения, в свою очередь, обычно реализуются посредством панели приложения – элемента управления `WinJS.UI.AppBar`. Во многом, панель приложения – это эквивалент меню и ленты в приложениях для Магазина Windows, так как вы можете создать в них любые виды пользовательского интерфейса и даже отобразить элементы меню. Меню, которые поддерживает элемент управления `WinJS.UI.Menu`, могут, так же, всплывать из заданного места на основном экране приложения, как, например, меню прикрепленное к заголовку.

Панель приложения и меню – это конкретные экземпляры более общего элемента управления `WinJS.UI.Flyout`, который используется напрямую для вывода сообщений, или для действий, которые пользователь может отменить или проигнорировать. Подобные всплывающие элементы можно закрыть щелчком мыши или прикосновением за их пределами. (Это – что-то вроде нажатия на кнопку "Отмена"). Для показа важных сообщений, которые требуют какого-либо действий, то есть – тех, где пользователь должен выбрать из некоторого набора параметров, приложения могут использовать WinJS.UI.MessageDialog. Диалоговые окна – это всем известная концепция из мира настольных приложений, она уже долгое время используется для сбора самой разной информации и настройки параметров приложения. В дизайне приложений для Магазина Windows, однако, диалоговые окна используются лишь для задавания вопросов и получения простых ответов, или для того, чтобы сообщить пользователю некие важные сведения. Настройка параметров приложений реализуется посредством чудо-кнопки Параметры, как мы увидим в лекции 2.

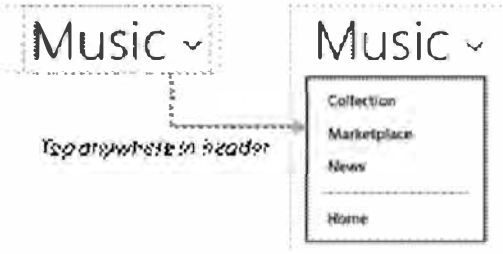
В том, что касается всех этих команд, важно то, что они не включаются в макет страницы: они, вместо этого "всплывают" и располагаются над текущей страницей. Это означает, что мы, к счастью, можем не беспокоиться об их воздействии на макет… с одним маленьким исключением, которое я пока буду держать в секрете.

Для начала, тем не менее, вернемся назад и подумаем о командах приложения в целом, и о том, где лучше всего эти команды размещать.

Где размещать командные элементы управления

Размещение команд имеет важное значение в дизайне приложений. В отличие от руководства, или, скорее, их отсутствия – для командных элементов управления настольных приложений, что приводило к беспорядку, центр разработчиков Windows предлагает два довольно обширных материала на эту тему: "Проектирование командного интерфейса для приложений Магазина Windows" (ссылка: http://msdn.microsoft.com/library/windows/apps/hh761499 - http://msdn.microsoft.com/library/windows/apps/hh761499) и "Создание макета пользовательского интерфейса" (ссылка: http://msdn.microsoft.com/library/windows/apps/hh465304.aspx). Эти материалы обязательны для прочтения для дизайнеров, работающих над приложением, так как они описывают различные виды командного интерфейса и рассказывают о том, как получить одобрение у профессионалов в области дизайна для Windows 8. Это так же отличные материалы для разработчиков, так как они могут дать вам некоторое понимание того, чего вы можете ожидать от ваших дизайнеров. Рассмотрим эти руководства, в качестве вводного тура по различным возможностям:

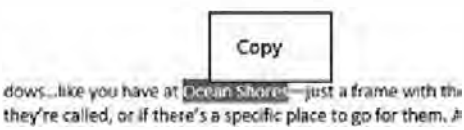
- Пользователь должен иметь возможность выполнять ключевые сценарии, используя только полотно приложения (app canvas), таким образом, команды, необходимые в процессе работы, появляются прямо на экране. Общая цель подобной рекомендации заключается в уменьшении числа командных элементов, которые в данный момент не нужны пользователю, но отвлекают его от задачи. Команды, в которых нет постоянной необходимости, следует скрывать, за исключением навигационных команд, которые можно расположить в выпадающем меню заголовка, как показано ниже:



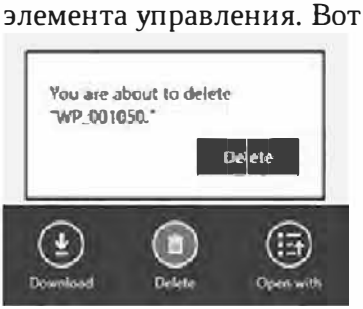
- Всегда, везде, где это возможно, используйте панель чудо-кнопок (Charms) для реализации общих команд приложения. Таким образом, вместо поддержки вашего собственного элемента управления для осуществления поиска, используйте чудо-кнопку Поиск (за исключением тех случаев, когда приложение имеет гораздо более богатый интерфейс поиска, использует, помимо строки поискового запроса, дополнительные параметры). Вместо поддержки отдельных команд для предоставления общего доступа к содержимому различным целевым приложениям, таким, как почтовые клиенты, приложения, поддерживающие списки контактов, приложения социальных сетей, и так далее, используйте чудо-кнопку Общий доступ. Вместо собственной команды Печать, положитесь на возможности чудо-кнопки Устройства. И вместо того, чтобы создавать внутри навигационной иерархии приложения страницы с его настройками, справочной информацией, заявлениями о конфиденциальности, информацией о программе, страницы с информацией об учетных записях и страницы с данными о разрешениях, которые имеет приложение, упростите себе жизнь, воспользовавшись чудо-кнопкой Параметры! (Взгляните, так же, на врезку: "Пользовательские данные для входа и лицензионные соглашения"). Примеры показаны на изображении ниже, где так же показано, что многие команды приложения могут использовать панель чудо-кнопок, что означает меньший беспорядок в остальной части их пользовательского интерфейса. Как приложение может реагировать на события панели чудо-кнопок, мы рассмотрим в следующих лекциях.

- Команды, которые не могут быть размещены на панели чудо-кнопок, и в которых нет необходимости на полотне приложения, размещаются на панели приложения (app bar), как показано ниже для приложения Travel. Это – близкая аналогия традиционным меню:
    - Верхняя панель приложения зарезервирована для навигационных команд
    - Нижняя панель приложения содержит все остальные команды, коточе чувствительны к контексту или выделению, а так же глобальные команды (не зависящие от выделения). Контекстные и глобальные команды размещают на разных сторонах панели.
    - Команды панели приложения могут отображать меню для группировки связанных команд, которые, таким образом, не загромождают панель приложения.

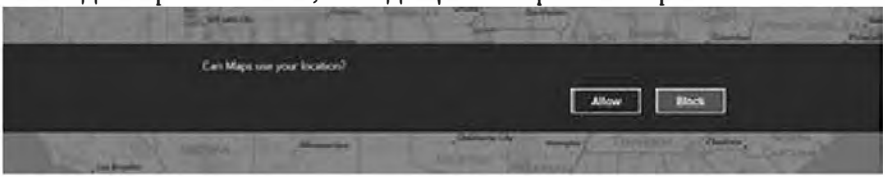
- Контекстные меню (context menu) могут предоставлять специальные команды для содержимого или выделения. Например, выделенный текст обычно предоставляет контекстное меню для работы с буфером обмена, как показано ниже для приложения Mail.



- Если в ответ на действие пользователя вам нужно отобразить запрос на подтверждение или вопрос (включая сбор информации), воспользуйтесь всплывающим элементом управления (flyout). Посмотрите материал "Руководство и контрольный список для всплывающих элементов" (ссылка: http://msdn.microsoft.com/library/windows/apps/hh465341). Щелчок мышью или прикосновение за пределами элемента управления (или нажатие клавиши ESC на клавиатуре) равносильно закрытию элемента управления. Вот пример из приложения SkyDrive:



- Для событий, блокирующих интерфейс, которые не относятся к командам пользователя, но воздействуют на все приложение, используйте диалоговые окна сообщений (message dialogs). Диалоговое окно блокирует остальное приложение до тех пор, пока пользователь не уделит ему внимание! Хороший пример их использования – разрыв сетевого соединения, когда пользователю нужно сообщить, что некоторые возможности будут недоступны до тех пор, пока соединение не восстановится. Пользователь подтверждает свое согласие на использование возможностей, таких, как определение местоположения, как показано ниже для приложения Maps, это – еще одна ситуация, в которой можно увидеть диалоговые окна. Обратите внимание на то, что диалоговое окно используется лишь тогда, когда приложение находится на переднем плане. Всплывающие уведомления (toast notifications), как мы увидим в лекции 2 курса "Программная логика приложений для Windows 8, созданных с использованием HTML, CSS и JavaScript и их взаимодействие с системой", применимы лишь для приложений, находящихся в фоновом режиме.



И, наконец, другие сообщения об ошибках, которые не требуют реакции пользователя, могут быть отображены либо на полотне приложения, либо посредством всплывающих элементов. Смотрите материал "Создание макета пользовательского интерфейса: Ошибки" (ссылка: http://msdn.microsoft.com/library/windows/apps/hh465304.aspx#errors) для того, чтобы узнать дальнейшие подробности. Позже мы рассмотрим некоторые примеры.

Если говорить о нижней панели приложения, то важно организовать команды в наборы, так как это упрощает их реализацию, как мы увидим в следущем разделе. Для того, чтобы получить полную информацию по этому вопросу, я рекомендую обратиться к материалам "Руководство по панелям приложения" (ссылка: http://msdn.microsoft.com/library/windows/apps/hh465302.aspx) и "Проектирование командного интерфейса для приложений Магазина Windows" (ссылка:

http://msdn.microsoft.com/library/windows/apps/hh761499.aspx). Они содержат множество подробностей о размещении, разделении и группировке. Вот выдержки из этих руководств:

- Для начала, создайте две группы команд: одну с теми командами, которые актуальные для всего приложения, независимо от контекста, и другую – с командами, которые отображаются лишь на определенных страницах. Панель приложения очень легко перенастроить во время выполнения программы для отображения различных групп команд.
- Далее, создайте наборы команд, например, для функционально связанных команд, таких, как кнопки переключения режимов, и таких, которые связаны с выделением. Помните, что команды панели приложения могут отображать всплывающие меню, как показано ниже, для вывода списка параметров и/или дополнительных элементов управления, включая более длинные подписи, выпадающие списки, флаги, переключатели и тумблеры. Таким способом вы можете объединить близко связанные команды в одну, которая позволяет использовать для их отображения больше места, чем доступно на панели приложения.

- Что касается порядка размещения команд, поместите постоянно присутствующие команды на правой стороне панели приложения, а наиболее часто используемые контекстно-зависимые команды – на левой. После этого размещайте дополнительные команды, двигаясь к центру панели. Эта рекомендация получена благодаря анализу эргономичности устройств: пальцы и большие пальцы – даже на крупных руках баскетболистов – способны достигать лишь определенного пространства на экране без необходимости перемещения рук. Наиболее часто используемые команды лучше всего располагать ближе к тем местам, где находятся большие

пальцы человека, который держит устройство, как показано на рисунке ниже (из материала "Проектирование взаимодействия с сенсорным экраном" (ссылка: http://msdn.microsoft.com/library/windows/apps/hh465415.aspx#touch_pos Эти места легко достижимы (особенно теми из нас, кто не может охватить большой мяч одной рукой) и, таким образом, учет этого повышает комфортность использования приложения.

Панель приложения доступна всегда, в любых состояниях просмотра, в том числе – в прикрепленном. Рекомендована в прикрепленном режиме (и иногда в портретном) ограничивать число команд до 10, чтобы они уместились в один или два ряда.

Знайте, так же, что панель приложения не ограничена круглыми командными кнопками. Вы можете создать все, что захотите, обычно именно так реализуются верхние панели приложений. Работая с любым собственным макетом, убедитесь, что ваши элементы имеют размер, подходящий для сенсорного взаимодействия. Больше об этом можно найти в материалах "Руководство по панелям приложения" (ссылка: http://msdn.microsoft.com/library/windows/apps/hh465302.aspx) и "Проектирование взаимодействия с сенсорным экраном" (ссылка: http://msdn.microsoft.com/library/windows/apps/hh465415.aspx) в разделе "Целевые объекты сенсорного ввода Windows 8". Там можно увидеть и небольшую картинку с изображением вышеупомянутых пальцев баскетболиста.

Врезка: Пользовательские данные для входа и лицензионные соглашения

Как отмечено выше, Microsoft рекомендует, чтобы ввод пользовательских даных для входа в программу, управление учетными данными, лицензионные соглашения, условия использования программы были доступны посредством чудо-кнопки Параметры. При этом программа добавляет необходимые команды на панель чудо-кнопки Параметры, которая появляется при активации этой чудо-кнопки. Эти команды затем вызывают вспомогательные страницы с необходимыми элементами управления для каждой функции. Конечно, иногда вход в систему или лицензионные соглашения требуют специальной обработки. Например, если ваше приложение требует ввода данных пользователя или принятия лицензионного соглашения при старте, подобные элементы управления можно отобразить на первой странице приложения или предоставить их посредством интерфейса средства выбора учетных даных (Credential Picker UI) (подробнее об этом – в лекции 3 курса "Программная логика приложений для Windows 8, созданных с использованием HTML, CSS и JavaScript и их взаимодействие с системой"). Если пользователь вводит учетные данные и/или принимает лицензионное соглашение, приложение может продолжить работу. В противном случае приложению следует показать страницу, которая указывает на то, что учетные данные или принятие соглашения необходимы для того, чтобы пользователь мог с его помощью сделать что-нибудь более интересное, чем вывод сообщений об ошибках.

Если ввод учетных данных рекомендован, но не является необходимым, возможно, нужен лишь для включения дополнительных возможностей, вы можете разместить соответствующие элементы управления прямо на полотне. Когда пользователь вводит учетные данные, вы можете заменить эти элементы управления на сведения об учетной записи (имя пользователя и изображение, например, как на Начальном экране Windows). Если, с другой стороны, ввод учетных данных необязателен, соответствующие элементы управления лучше всего размещать в Параметрах приложения.

Во всех случаях, команды для просмотра лицензионного соглашения, для управления учетными записями или профилями, для входа в программу или выхода из нее, следует размещать среди средств чудо-кнопки Параметры. Другие команды на панели приложения или на полотне могут активировать Параметры программным образом, как мы

увидим в лекции 2.

Панель приложения

После размещения самых важных команд на полотне приложения, большинство команд будет размещено в панели приложения. Повторюсь, панель приложения можно вызвать разными способами – жестом скольжения от верхнего или нижнего края экрана, комбинацией клавиш Win+Z, щелчком правой кнопки мыши. Когда вы выполните одно из этих действий, Windows ищет элемент управления панели приложения на текущей странице и запускает его – вам не нужно самостоятельно обрабатывать события ввода.

Совет. Для того чтобы предотвратить появление панели приложения, вы можете использовать один из двух подходов. Первый заключается в том, чтобы полностью отключить появление панели приложения в ответ на любой из жестов, для этого нужно установить свойство `winControl.disabled` элемента панели в значение `true`. Второй применим тогда, когда вы, например, не хотите, чтобы панель приложения появлялась при щелчке правой кнопкой мыши по конкретному элементу (например, по полотну). Для этого можно прослушивать событие `contextmenu` (то есть – щелчок правой кнопки мыши) для этого элемента и вызвать `eventArgs.preventDefault()` в обработчике события.

В приложениях, написанных на HTML и JavaScript, элемент управления панели приложения реализован как элемент управления WinJS: `WinJS.UI.AppBar` (ссылка: http://msdn.microsoft.com/library/windows/apps/br229670.aspx). Для первой демонстрации нам достаточно тех возможностей, которые представлены в шаблоне проекта Приложение таблицы в Visual Studio/Blend, где стандартная панель приложения включена в default.html (изначально закомментирована):

```
<div id="appbar" data-win-control="WinJS.UI.AppBar">
<button data-win-control="WinJS.UI.AppBarCommand"
data-win-options="{id:'cmd', label:'Command', icon:'placeholder'}">
</button>
</div>
```

Восхитительный результат этой разметки, с использованием таблицы стилей ui-dark.css, показан ниже:

Так как панель приложения объявлена в default.html, которая является контейнером для всех других элементов управления страниц, та же панель приложения будет использована для всех остальных страниц приложения. Используя такой подход, вы можете объявить все команды на одной панели приложения и присваивать командам различные классы, что позволит вам легко показывать и скрывать наборы команд для соответствующих страниц. Это так же позволяет централизованно управлять теми командами, которые актуальны для нескольких страниц, и вы можете подключить к ним обработчики событий в основном коде активации приложения (например, в default.js).

Кроме того, вы можете объявить панель приложения в разметке отдельных элементов управления страниц. До тех пор, пока панель приложения находится в DOM, жесты Windows активируют ее на каждой странице. В проекте Приложение таблицы, например, вы можете переместить вышеприведенную разметку из default.html в groupedItems.html, groupDetail.html и itemDetail.html с любыми модификациями для каждой страницы. Это может быть особенно полезным, если у страниц приложения немного общих команд.

В подобных случаях, метод `ready` каждой страницы должен позаботиться о том, чтобы привязать команды на панелях приложения этих страниц к обработчикам. Обратите внимание на то, что вы можете добавлять обрабочтики внутри метода страницы ready даже для общей панели приложения. Это – вопрос вызова `addEventListener` для нужного дочернего элемента в панели приложения.

Посмотрим теперь как все это работает с помощью примера "HTML-элемен управления AppBar" (ссылка: http://code.msdn.microsoft.com/windowsapps/App-bar-sample-a57eeae9). (Дополнительные материалы к этой лекции так же содержат модифицированную версию этого примера). Начнем с основ панели приложения и со стандартной конфигурации, ориентированной на команды. Посмотрим, как показывать меню для некоторых из этих

команд, и затем посомтрим, как можно создать собственный макет, как это делается для верхней панели приложения.

Подсказка. Чисто технически, вы можете объявить столько панелей приложения, сколько вам нужно, на любой странице, и все они будут представлены в DOM. Однако, последней из них, которая будет обрабатываться, будет та, что расположена на самом верху по оси Z по умолчанию, и, таким образом, она одна будет получать события. Windows не делает попыток комбинирования панелей приложения, в итоге, потому что элементы управления страниц вставляются в середину главной страницы наподобие default.html, панель приложения в default.html, которая объявляется после элемента управления страницы, появится над панелью приложения, объявленной на странице. В то же время, если панель приложения элемента управления страницы больше, чем в default.html, ее часть может быть видимой. В итоге: объявляйте панели приложения или на главной странице, или на элементах управления страниц, но не и там и там одновременно.

Основы панели приложения и стандартные команды

Как я уже говорил, может быть объявлена одна панель приложениян на все приложение, например, на странице-контейнере default.html, или собственную панель приложения можно объявить для каждого элемента управления страницы. Пример, посвященный элементу управления HTML AppBar, реализует последний вариант, так как он в нем показаны очень разные панели приложения, реализующие различные сценарии.

Сценарий 1 в примере (html/create-appbar.html) объявляет панель приложения с четырьмя командами и разделителем:

```
<div id="createAppBar" data-win-control="WinJS.UI.AppBar" data-win-option
<button data-win-control="WinJS.UI.AppBarCommand" data-win-options="{id
label:'Add', icon:'add', section:'global', tooltip:'Add item'}">
</button>
<button data-win-control="WinJS.UI.AppBarCommand" data-win-options="{id
label:'Remove', icon:'remove', section:'global', tooltip:'Remove item'}">
</button>
```

```
<hr data-win-control="WinJS.UI.AppBarCommand"
data-win-options="{type:'separator', section:'global'}" />
<button data-win-control="WinJS.UI.AppBarCommand" data-win-options="{id
label:'Delete', icon:'delete', section:'global', tooltip:'Delete item'}">
</button>
<button data-win-control="WinJS.UI.AppBarCommand" data-win-options="{id
 label:'Camera', icon:'camera', section:'selection', tooltip:'Take a picture'}">
</button>
</div>
```

В приложении это выглядит так, как показано ниже, с использованием таблицы стилей ui-light.css. Здесь мы так же можем видеть всплывающую подсказку, прямоугольник фокуса ввода и эффект, применяемый при зависании указателя мыши над командой Add (Добавить) (для того, чтобы все это увидеть, я разместил указатель мыши над этой командой):

В разметке, элемент управления панель приложения объявлен так же, как другие элементы управления WinJS (это становится привычкой!), с использованием элемента-контейнера (это `div`) с `data-win-control="WinJS.UI.AppBar"`. Каждая страница в этом примере загружается с помощью `WinJS.UI.Pages.render`, которая удобна тем, что вызывает `WinJS.UI.processAll` для создания экземпляра панели приложения. (Можно, как и для других элементов управления, программно создавать панель приложения, используя оператор `new`).

Этот пример не предусматривает каких-либо специальных параметров для панели приложения в `data-win-options`, но имеется ряд возможностей:

- `disabled`, если установлено в значение `true`, создается изначально отключенная панель приложения. Значение по умолчанию – `false`.
- `layout` может принимать значение `"commands"` (по

умолчанию) или "custom", как мы увидим в разделе "Пользовательские панели приложений и панели навигации" ниже.

- placement может быть либо "top", либо "bottom" (по умолчанию). Далее мы используем значение "top" для навигационной панели
- sticky изменяет поведение панели приложения, когда она исчезает с экрана, если в ней нет необходимости. Со значением по умолчанию, установленным в false, панель приложения исчезает, когда вы щелкнете мышью или коснетесь экрана за ее пределами. Если параметр установлен в true, панель приложения будет оставаться видимой до тех пор, пока вы не измените параметр на false, и пользователь не коснется экрана за ее пределами, или вы программно не скроете ее с помощью метода hide.

Если вам нужна постоянно видимая панель приложения с пользовательским макетом, отображаемая в верхней части экрана, вы можете воспользоваться размеркой, подобной этой:

```
<div id="navBar" data-win-control="WinJS.UI.AppBar"
data-win-options="{layout:'custom', placement:'top', sticky: true}">
```

Обратите внимание на то, что две панели приложения на странице, с различными значениями параметра placement не мешают друг другу. Кроме того, свойство sticky для каждого из расположений панелей действует независимо. Таким образом, если вам нужно реализовать верхнюю навигационную панель в масштабах всего приложения, вы можете объявить ее в default.htm (или в другой подобной странице) и объявить нижние панели приложения на каждом элементе управления страницы. Опять же, это всего лишь элементы в DOM!

Как вы можете видеть, элемент управления панели приложения может содержать любое количество дочерних элементов для своих команд, каждый из которых должен быть элементом управления WinJS.UI.AppBarCommand (ссылка: http://msdn.microsoft.com/library/windows/apps/hh700497.aspx) внутри элемента button или hr, или экземпляр панели приложения не будет

создан.

Свойства и параметры команд панели приложения перечислены ниже:

- `id` Идентификатор элемента, который вы можете использовать с командой `document.getElementById` или с методом панели приложения `getCommandById` для подключения обработчиков события `click`.
- `type` Может принимать несколько значений: `"button"` (по умолчанию), `"separator"`(с его помощью создаются вертикальные разделители), `"flyout"` (используется для создания всплывающих меню, которые описывает свойство `flyout`; смотрите дальше раздел "Командные меню"), и `"toggle"` (создающий кнопку, которая может быть во включенном либо выключенном состоянии). В последнем случае, свойство `selected` команды может быть использовано для установки начального значения и получения состояния кнопки во время выполнения программы.
- `label` Задает текст, который отображается ниже командной кнопки. Его используют всегда вместо того, чтобы задавать текстовое содержимое самой кнопки, так как подобный текст не выравнивается соответствующим образом в элементе управления. (Попробуйте это сделать и увидите сами!). Кроме того, обратите внимание на то, что это свойство, наряду со свойством `tooltip` , которое описано ниже, часто локализуют, используя атрибут `data-win-res` . Мы рассмотрим это в лекции 6 курса "Программная логика приложений для Windows 8, созданных с использованием HTML, CSS и JavaScript и их взаимодействие с системой", но уже сейчас вы можете посмотреть файл html/localize-appbar.html в примере Сценарий 8 для того, чтобы увидеть, как это работает.
- `tooltip` Содержит текст всплывающей подсказки для команды (обычно локализованный), в качестве значения по умолчанию используется текст, заданный в `label`. Обратите внимание на то, что это – обычный текст. Использование полнофункционального HTML- элемента управления `WinJS.UI.Tooltip` не поддерживается.
- `icon` Задает графический элемент, который отображается в

команде. Обычно это одна из строк из перечисления `WinJS.UI.AppBarIcon` (ссылка: http://msdn.microsoft.com/library/windows/apps/hh770557.aspx), которое содержит 150 различных символов из шрифта Segoe UI Symbol. Если вы посмотрите в файл ресурсов WinJS ui.strings.js, вы увидите, как заданы эти символы, с использованием кодов наподобие `\uE109` . Перечисление, фактически, просто предоставляет понятные имена для кодов символов – от `\uE100` до `\uE1E9` . Но вы не ограничены этими символами. Вы можете использовать любые escape-коды в формате `'\uXXXX'` из шрифта Segoe UI Symbol (Обратите внимание на одиночные кавычки). Так же вы можете воспользоваться другими шрифтами или использовать собственные изображения, как показано в разделе "Пользовательские значки" далее[1] .

- `section` Управляет расположением команд. Для языков, где тексты расположены слева направо (таких, как английский), значение по умолчанию `"selection"` располагает команды (контекстно-зависимые) в левой части панели приложения. Свойство `"global"` располагает команды (глобальные для приложения) с правой стороны. В языках, где пишут справа налево (таких, как еврейский, арабский), стороны меняются местами. Эти простые варианты выбора приводят к единообразному расположению этих двух категорий команд (значение по умолчанию – `"selection"`). Это притягивает взгляд пользователей к одной стороне панели, если они ищут постоянные команды, и к другой, если ищут команды, зависящие от контекста. Обратите внимание на то, что команды в каждом из разделов располагаются слева направо (или справа налево) в том порядке, в котором они появляются в разметке.
- `onClick` Может быть использовано для декларативного задания обработчика события `click` . Помните, что любая функция, упомянутая в разметке, должна быть помечена как безопасная для обработки (смотрите Главу 4 курса "Введение в разработку приложений для Windows 8 с использованием HTML, CSS и JavaScript"). Обработчик события `click` может быть так же задан программно, с помощью `addEventListener` , в таком случае маркировка не нужна.
- `disabled` Отключает команду, если установлено в `true` .

Значение по умолчанию – `false` .

- `extraClass` Задает один или большее количество классов CSS, которые прикреплены к команде. Они могут использоваться для индивидуальной стилизации командных элементов управления и для создания наборов команд, которые можно легко отображать и скрывать, как показано в разделе "Отображение, скрытие, включение и обновление команд" ниже.
- `section` Управляет расположением команд. Для языков, где тексты расположены слева направо (таких, как английский), значение по умолчанию `"selection"` располагает команды (контекстно-зависимые) в левой части панели приложения. Свойство `"global"` располагает команды (глобальные для приложения) с правой стороны. В языках, где пишут справа налево (таких, как еврейский, арабский), стороны меняются местами. Эти простые варианты выбора приводят к единообразному расположению этих двух категорий команд (значение по умолчанию – `"selection"`). Это притягивает взгляд пользователей к одной стороне панели, если они ищут постоянные команды, и к другой, если ищут команды, зависящие от контекста. Обратите внимание на то, что команды в каждом из разделов располагаются слева направо (или справа налево) в том порядке, в котором они появляются в разметке.
- `onClick` Может быть использовано для декларативного задания обработчика события `click` . Помните, что любая функция, упомянутая в разметке, должна быть помечена как безопасная для обработки (смотрите Главу 4 курса "Введение в разработку приложений для Windows 8 с использованием HTML, CSS и JavaScript"). Обработчик события `click` может быть так же задан программно, с помощью `addEventListener` , в таком случае маркировка не нужна.
- `disabled` Отключает команду, если установлено в `true` . Значение по умолчанию – `false` .
- `extraClass` Задает один или большее количество классов CSS, которые прикреплены к команде. Они могут использоваться для индивидуальной стилизации командных элементов управления и для создания наборов команд, которые можно легко отображать и скрывать, как показано в разделе "Отображение, скрытие, включение и обновление команд" ниже.

Если вы хотите создавать команды во время выполнения приложения, вы можете сделать это, присвоив свойству панели приложения `commands` JSON-массив дескрипторов (descriptors) `AppBarCommand` в любое время, когда панель приложения не видна (то есть, когда ее свойство `hidden` имеет значение `true`). Массив подобных дескрипторов из Сценария 1 нашего примера может выглядеть так (вы можете найти этот код в модифицированном примере к этой лекции; смотрите файл js/create_appbar.js):

```
var appbar = document.getElementById("createAppBar").winControl;

//Set the app bar commands property to populate it
var commands = [
{ id: 'cmdAdd', label: 'Add', icon: 'add', section: 'global', tooltip: 'Add item' },
{ id: 'cmdRemove', label: 'Remove', icon: 'remove', section: 'global', tooltip: 'Rem
{ type: 'separator', section: 'global' },
{ id: 'cmdDelete', label: 'Delete', icon: 'delete', section: 'global', tooltip: 'Delete iter
{ id: 'cmdCamera', label: 'Camera', icon: 'camera', section: 'selection',tooltip: 'Tak
];

appbar.commands = commands;
```

Когда создается панель приложения, происходит перебор массива `commands` и создание элементов управления `WinJS.UI.AppBarCommand` для каждого элемента. Если параметр `type` не задан или если он установлен в `"button"`, `"flyout"` или `"toggle"`, тогда создается командный элемент `button`. При параметре `type`, установленном в значение `"separator"` создается элемент `hr`. Обратите внимание на то, что вам следует локализовать поля `label`, `tooltip`, и, возможно, `icon` в каждом описании команды, вместо того, чтобы использовать встроенный текст, как показано здесь, в примерах кода.

Так же вы можете использовать подобный массив непосредственно в декларативной разметке, но подобная конструкция не поддается локализации и, таким образом, не рекомендуется (хотя я, в модифицированном примере, включил комментарии, которые показывают, как это делается). В то же время, так как значение

`commands` в разметке – это обычная строка, вы можете присваивать данное значение посредством механизмов привязки данных к атрибутам, наподобие этого в элементе панели приложения:

```
data-win-bind="{ winControl.commands: myData.commands }"
```

Однако подобный подход не работает для атрибута `data-win-res` (как мы увидим в лекции 6 курса "Программная логика приложений для Windows 8, созданных с использованием HTML, CSS и JavaScript и их взаимодействие с системой", и как, кроме того, показано в Сценарии 8 текущего примера), так как строка ресурса не может быть конвертирована в JSON как часть поиска ресурса. Попытка выполнить подобный трюк, в любом случае, принесет больше сложностей, чем преимуществ, поэтому лучше использовать либо декларативную форму описания в HTML, либо программное подключение локализованных массивов команд во время выполнения приложения.

Так же следует помнить о том, что `commands` – это редкий пример свойства только для записи (write-only property). Вы можете записать в него данные, но не можете получить массив из панели приложения. Панель приложения использует этот массив только для задания собственной конфигурирации и массив отбрасывается сразу после того, как все элементы созданы в DOM. Во время выполнения приложения, однако, вы можете использовать метод панели приложения `getCommandById` для получения конкретного командного элемента.

События команд

Говоря о командных элементах, об элементах управления `AppbarCommand` панели приложения (о тех, которые не являются разделителями), все это – лишь элементы `button` , и, таким образом, они реагируют на обычные события. Так как каждому командному элементу присвоен заданный вами `id`, вы можете использовать `getElementById` , как обычно, перед добавлением `addEventListener` . В Сценарии 1 примера, посвященного элементу управления HTML AppBar, например, подобный код можно увидеть в методе страницы `ready` :

```
document.getElementById("cmdAdd").addEventListener("click", doClickAdd, fa
document.getElementById("cmdRemove").addEventListener("click", doClickRen
document.getElementById("cmdDelete").addEventListener("click", doClickDelet
document.getElementById("cmdCamera").addEventListener("click", doClickCan
```

Хотя такой подход работает, он относительно неэффективен, так как каждый вызов `document.getElementById` обходит всю DOM. Я рекомендую, вместо этого, использовать метод панели приложения `getCommandById`. Вот изменения, которые я внес в модифицированный пример, который можно найти в дополнительных материалах к этой лекции:

```
//Использование метода getCommandById панели приложения избавляе
var appbar = document.getElementById("createAppBar").winControl;
appbar.getCommandById("cmdAdd").addEventListener("click", doClickAdd, fa
appbar.getCommandById("cmdRemove").addEventListener("click", doClickRen
appbar.getCommandById("cmdDelete").addEventListener("click", doClickDelete
appbar.getCommandById("cmdCamera").addEventListener("click", doClickCam
```

Конечно, если вы зададите обработчик для свойства onclick каждой команды в разметке (эти обработчики должны иметь свойство `supportedForProcessing`, установленное в значение `true`), вы полностью устраните необходимость в подобных действиях!

Так же должно быть очевидно то, что можно подключать обработчики событий, подобные этим, в любом месте приложения, и вы можете, конечно, прослушивать любые другие события, особенно, когда задаете собственный макет панели приложения с иным пользовательским интфрейсом. Кроме того, знайте, что событие `click` удобно объединяет обработку касаний, нажатий мыши, событий ввода данных с клавиатуры, таким образом, в данной области вам не нужно выполнять дополнительные действия. В случае с клавиатурой, кстати, панель приложения позволяет вам перемещаться между командами с помощью клавиши Tab. Нажатие на Enter или на клавишу пробела активирует обработчик `click`.

События и методы панели приложения

В дополнение к методу панели приложения `getCommandById`, который мы только что рассмотрели, панель приложения имеет несколько других методов и событий. Во-первых, вот методы:

- `show` отображает панель приложения, если ее свойство `disabled` установлено в значение `false`. В противном случае вызов игнорируется.
- `hide` скрывает панель приложения.
- `showCommands`, `hideCommands` и `showOnlyCommands` используются для управления наборами команд, как описано в следующем разделе "Показ, скрытие и обновление команд".

Что касается событий, то их всего четыре, что обычно для элементов управления WinJS, которые накладываются на пользовательский интерфейс (то есть, для тех, которые не включаются в состав макета приложения):

- `beforeshow` происходит перед тем, как всплывающий элемент станет видимым. Для панели приложения, это время, когда вы можете задать свойство `commands` в зависимости от состояния приложения в данный момент, активировать или деактивировать отдельные команды.
- `aftershow` происходит сразу после того, как всплывающий элемент станет видимым. Для панели приложения, если ее свойство `sticky` установлено в значение `true`, вы можете использовать это приложение для подстройки макета приложения, если у вас есть элементы для прокрутки содержимого, которые могут быть, в противном случае, частично перекрыты (смотрите ниже).
- `beforehide` происходит перед тем, как всплывающий элемент будет скрыт. Для панели приложения, вы можете использовать это событие для скрытия вспомогательныхз элементов пользовательского интерфейса, созданных вместе с панелью приложения и для перенастройки макета вокруг панели приложения с включенным режимом `sticky`.
- `afterhide` происходит сразу после того, как всплывающий элемент будет скрыт. Для панели приложения, это может быть временем перенастройки макета приложения, если в этом есть

необходимость.

Вы можете найти пример использования метода `show` вместе с событиями `aftershow` и `beforeshow` в Сценарии 4 примера, посвященного элементу управления HTML AppBar.

Проблемы с макетом приложения, названные выше (и то, что я держал в секрете во введении к этой лекции) возникают потому, что панель приложения накладывается на нижную часть страницы и перекрывает ее. Если эта страница содержит элемент управления для прокрутки содержимого, панель приложения со свойством `sticky`, установленным в `true`, для тех пользователей, которые применяют мышь, частично перекрывает вертикальную полосу прокрутки, а горизонтальную полосу прокрутки делает полностью недоступной. Если вы используете залипающую (sticky) панель приложения на подобных страницах, тогда, тогда – так как политика Магазина Windows не поддерживают ущемление прав пользователей мышей, вам следует использовать aftershow для уменьшения высоты элементов прокрутки с помощью значения панели приложения `offsetHeight` или `clientHeight`, таким образом, оставляя полосы прокрутки доступными. Когда панель приложения скрывается и происходит событие `afterhide`, вы можете снова перенастроить макет. Всегда используйте то значение, которое имеется во время выполнения приложения, такое, как `clientHeight` в данных вычислениях, так как оно учитывает масштабирование разрешения и потому, что высота панели приложения может меняться в зависимости от содержащихся на ней команд и режима просмотра.

Для того чтобы показать это, в Сценарии 6 имеется горизонтально прокручиваемый элемент управления ListView, который, в обычном состоянии, занимает почти всю страницу. Полоса прокрутки отображается вдоль самой нижней части элемента при использовании мыши. Если вы выделяете элемент, панель приложения переводится в режим залипания и отображается (смотрите функцию `doSelectItem` в файле js/appbar-listview.js):

```
appBar.sticky = true;
appBar.show();
```

Вызов метода show приводит к вызову событий beforeshow и aftershow . Для подстройки макета подходит событие aftershow , которое позволяет работать с реальным значением высоты панели приложения. В примере это событие обрабатывается в функции doAppBarShow (так же в файле js/appbar-listview.js):

```
function doAppBarShow() {
var listView = document.getElementById("scenarioListView");
var appBar = document.getElementById("scenarioAppBar");
var appBarHeight = appBar.offsetHeight;
// Сделать полосу прокрутки видиой, если панель приложения в режим
if (appBar.winControl.sticky) {
var listViewTargetHeight = "calc(100% - " + appBarHeight + "px)";
var transition = { property: 'height', duration: 367,
timing: "cubic-bezier(0.1, 0.9, 0.2, 0.1)",
to: listViewTargetHeight
};
WinJS.UI.executeTransition(listView, transition);
}
}
```

Примечание. Пример из Центра разработчиков Windows использует beforeshow вместо aftershow , в результате, иногда полученная высота панели приложения равняется нулю и макет не подстраивается соответствующим образом. Для того чтобы гарантировать то, что панель приложения имеет подходящую высоту для подобных вычислений, используйте событие aftershow , как показано в модифицированном примере, включенном в состав дополнительных материалов к этой лекции.

Здесь вы можете видеть, что значение appBar.offsetHeight просто вычитается из значения height элемента управления ListView с применением анимированного перехода. (Смотрите Главу 5). Операция выполняется в обратном порядке в doAppBarHide , где высота ListView просто сбрасывается до 100% с использованием похожей анимации. В этом случае обработчик события совсем не зависит от высоты панели приложения, поэтому мы можем использовать либо событие beforehide, либо – afterhide . Если, с другой стороны,

вам нужно знать размер панели приложения для настройки вашего макета, используйте событие `beforehide` .

В качестве эксперимента, запустите Сценарий 7 из SDK-примера. Обратите внимание на то, как нижняя часть полосы прокрутки текстовой области перекрывается залипающей панелью приложения. Попытайтесь взять код из Сценария 6 для того, чтобы используя события `aftershow` и `beforehide` настроить высоту текстовой области и сохранить видимость полосы прокрутки. И не считайте это оценочным испытанием ваших возможностей: в дополнительных материалах к этой лекции есть решение этой задачи.

Отображение, скрытие, включение и обновление команд

В предыдущем разделе я упоминал об использовании события `beforeshow` для настройки свойства `commands` панели приложения таким образом, чтобы панель содержала команды, соответствующие текущей странице и состоянию страницы. Это может включать установку свойства `disabled` для некоторых команд, которые, например, зависят от состояния выделения. Это можно реализовать с помощью массива `commands`, в разметке, или, опять же, используя метод панели приложения `getCommandById`:

```
appbar.getCommandById("cmdAdd").disabled = true;
```

Позвольте мне повторить, что команды, которые появляются на панели приложения, уникальны для каждой страницы. Нет необходимости пытаться поддерживать неизменную структуру панели приложения для всех страниц. Таким образом, если команды не нужны на какой-либо странице, не беспокойтесь о том, чтобы их отображать. Что важнее, так это то, постоянный состав панели приложения для отдельно взятой страницы, так как это по-настоящему неудачная идея – появляющиеся и исчезающие команды, зависящие от состояния страницы. Это приведет к тому, что пользователи будут размышлять о том, как привести страницу в нужное состояние для того, чтобы появились какие-либо команды!

Говоря об изменениях, вполне допустимо изменять или обновлять команды во время выполнения программы, что может устранить необходимость создания множества команд, которые вы можете поочередно показывать или скрывать. Так как каждая команда в панели приложения – это лишь DOM-элемент, вы можете, на самом деле, выполнять любые изменения, какие хотите, в любое время. Пример этого показан в Сценарии 3 примера, где панель приложения изначально создается с кнопкой Play (html/custom-icons.html):

```
<button data-win-control="WinJS.UI.AppBarCommand"
data-win-options="{id:'cmdPlay', label:'Play', icon:'play', tooltip:'Play this song'}"
</button>
```

Обработчик события click этой кнопки использует функцию `doCluckPlay` в js/custom-icons.js для переключения между состояниями:

```
var isPaused = true;

function doClickPlay() {
var cmd = document.getElementById('cmdPlay');

if (!isPaused) {
isPaused = true; // поставлено на паузу cmd.winControl.icon = 'play';
cmd.winControl.label = 'Play'; cmd.winControl.tooltip = 'Play this song';
} else {
isPaused = false; // проигрывается cmd.winControl.icon = 'pause';
cmd.winControl.label = 'Pause';
cmd.winControl.tooltip = 'Pause this song';
}
}
```

Вы можете использовать что-то подобное для закрепления или открепления дополнительных плиток (secondary tiles), как мы увидим в лекции 2 курса "Программная логика приложений для Windows 8, созданных с использованием HTML, CSS и JavaScript и их взаимодействие с системой". И, снова, кнопки – это лишь элементы в

DOM и обновление любых их свойств, включая стили, обновит элемент на экране, как только вы вернете управление в поток пользовательского интерфейса.

Использование `beforeshow` для целей настройки команд, безусловно, эффективно, но вы можете достичь тех же результатов другими способами. Стратегия, которой вы пользуетесь, зависит от архитектуры вашего приложения и от ваших собственных предпочтений. С точки зрения пользователя, до тех пор, пока нужные команды доступны ему в нужное время, неважно, как это реализовано в программе!

Размышления в рамках выбранного подхода особенно важны при работе с режимом прикрепленного просмотра, так как рекомендовано, чтобы в подобном режиме использовалось десять или меньшее количество команд, таким образом, панель приложения вместит их в одном или в двух рядах. Это означает, что вам нужно подумать о том, как настроить панель приложения для различных состояний просмотра, возможно, комбинируя несколько команд во всплывающем меню отдельных кнопок.

Один из подходов заключается в том, чтобы каждая страница приложения объявляла и поддерживала собственную панель приложения, что подразумевает создание панелей приложения "на лету" в методе страниц `ready`. Это делает взаимодействие между содержимым страницы и панелью приложения весьма четким и ограниченным рамками страницы. Недостаток такого подхода заключается в том, что общие команды – те, которые появляются более чем на одной странице – приходится объявлять множество раз, что ухудшает возможности по управлению ими и способствует привлечению небольшой неоднородности, которая приходит как муравьи на сахар. Тем не менее, если содержимое разных страниц сильно отличается и у вас есть лишь несколько общих команд, такой подход может быть правильным выбором. Это, так же, необходимо, если ваша страница использует несколько страниц верхнего уровня, вместо одной страницы с элементом управления страницы, как мы обсуждали в лекции 3курса "Введение в разработку приложений для Windows 8 с использованием HTML, CSS и JavaScript", так как каждая HTML-страница верхнего уровня, в любом случае, должна объявить свою собственную панель приложения.

Для приложений, использующих элементы управления страниц, применяется другой подход, который заключается в объявлении одной панели приложения на странице верхнего уровня и в установке ее свойства `command` внутри метода `ready` каждого из подобных элементов управления. Недостатком здесь является то, что так как `command` – это свойство только для записи, вы не можете объявить общие команды в HTML и позже добавить команды, характерные для отдельной страницы, только если вы не идете сложным путем создания каждого отдельного дочернего элемента AppBar в каждом методе `ready` . Подобный код утомительно писать и поддерживать.

К счастью, есть и третий подход, который позволяет вам объявить одну панель приложения на странице верхнего уровня, которая содержит все необходимые команды, для всех страниц, и затем выборочно показывать определенные наборы таких команд в каждом из методов `ready` отдельных страниц. Это – цель существования методов панели приложения `showCommands`, `hideCommands` и `showOnlyCommands` .

Все эти методы прпнимают массив командных элементов, которые могут быть либо объектами `AppBarCommand` , либо `id` команд. Метод `showCommands` делает эти командные элементы управления видимыми и может быть вызван несколько раз с различными установками для получения итогового набора. С другой стороны, метод `hideCommands` скрывает заданные команды в панели приложения, опять же, с возможностью многократного вызова. Основы использования этих методов показаны в Сценарии 4 нашего примера.

Метод `showOnlyCommands` объединяет два вышеописанных метода, делая заданные команды видимыми и скрывая остальные. Если вы задаете панель приложения со всеми командами, вы можете использовать метод `showOnlyCommands` в методе `ready` каждой страницы для того, чтобы быстро и просто настроить вид панели приложения. Хитрость заключается в получении массива, подходящего для передачи в метод. Вы можете, конечно, вручную написать код для команд, занести данные в специальный массив, как в Сценарии 4 примера сделано для `showCommands` и `hideCommands` . Однако, если вы думаете, что это хороший пример Плохой идеи, то знайте, что я тоже так думаю! Использование подобного массива подразумевает, что

любые изменения, которые вы вносите в панель приложения, должны произойти и в HTML-файле, и в JS-файле, а это означает, что любой, кто будет поддерживать ваш код в будущем, без сомнения, станет осыпать вас проклятиями!

Лучший путь к счастью и долгой жизни, таким образом – программно получать необходимые массивы из DOM, используя свойство `extraClass` для каждой команды для того, чтобы эффективно задавать наборы команд. Это позволяет вам вызывать `querySelectorAll` для получения тех команд, которые принадлежат опреденному набору.

Рассмотрим нижеприведенное определение панели приложения, где, для краткости, я опустил свойства вроде `label, icon` и `section` , так же, как и любые другие классы стилей:

```
<div id="appbar" data-win-control="WinJS.UI.AppBar" data-win-options="{
commands:[
{id:'home', extraClass: 'menuView gameView scoreView'},
{id:'play', extraClass: 'menuView gameView scoreView'},
{id:'rules', extraClass: 'menuView gameView scoreView'},
{id:'scores', extraClass: 'menuView gameView scoreView'},
{id:'newgame', extraClass: 'gameView gameSnapView'},
{id:'resetgame', extraClass: 'gameView gameSnapView'},
{id:'loadgame', extraClass: 'gameView gameSnapView'},
{id:'savegame', extraClass: 'gameView gameSnapView'},
{id:'hint', extraClass: 'gameView gameSnapView'},
{id:'timer', extraClass: 'gameView gameSnapView'},
{id:'pause', extraClass: 'gameView gameSnapView'},
{id:'home2', extraClass: 'gameSnapView'},
{id:'replaygame', extraClass: 'scoreView'},
{id:'resetscores', extraClass: 'scoreView'}
]}" >
</div >
```

В свойствах `extraClass` мы задаем четыре различных набора: menuView, gameView, gameSnapView, и scoreView. Когда все находится там, где нужно, простой вызов querySelectorAll предоставляет тот массив,

который нам нужен для `showOnlyCommands`. Общая функция, наподобие следующей, может быть, затем, использована из метода ready каждой страницы (или из любого другого места) для активации командных элементов для нужного экрана программы:

```
function updateAppBar(view) {
var appbar = document.getElementById("appbar").winControl;
var commands = appbar.element.querySelectorAll(view);
appbar.showOnlyCommands(commands);
}
```

При таком подходе, заслуга создания которого принадлежит моему колеге Джесси МакГата, панель приложения полностью определяется в едином месте, что серьезно упрощает управление ей и ее поддержку.

Стилизация панели приложения

Свойство команд `extraClass` может, конечно, быть использовано для целей стилизации, так же, как и для управления наборами команд. Это очень просто: любые классы, которые заданы в `extraClass`, добавляются к элементам `AppBarCommand`, созданных для панели приложения.

Существуют семь классов WinJS для стилизации, которые используются панелью приложения, как описано в нижеприведенной таблице, где первые два применяются к панели приложения в целом, а другие пять – к отдельным командным элементам:

CSS-класс (панель приложения)	Описание
`win-appbar`	Задает стиль контейнера панели приложения. Обычно этот стиль используется в качестве корневого для более специфических селекторов.
`win-commandlayout`	Задает стиль макета команд панели приложения. Обычно приложения не модифицируют этот стиль
CSS-класс	

(команды)	
win-command	Задает стиль всех элементов AppBarCommand.
win-commandicon	Задает стиль контейнера элемента для AppBarCommand.
win-commandimage	Задает стиль изображения для AppBarCommand.
win-commandring	Задает стиль круглой рамки, которая выводится вокруг изображения на AppBarCommand.
win-label	Стилизует подпись AppBarCommand.

Подсказка. Для того чтобы помочь себе в настройке стилей панели приложения в Blend сделайте ее залипающей (sticky), или добавьте вызов метода show в методе ready страницы, или в обработчике события activated . Это позволит сделать панель приложения видимой в Blend, работать с ней, в противном случае возникают сложностью с открытием панели в процессе разработки.

Обычно нет необходимости в прямом переопределении стилей win-appbar или win-commandlayout. Вместо этого следует создавать селекторы для пользовательских классов, связанных с этими стилями и затем стилизовать то, что вам нужно. Здесь можно использовать и псевдоселекторы, наподобие button:hover, button:active и так далее.

В Сценарии 2 примера, посвященного элементу управления HTML AppBar, показано много подобных селекторов в действии, в данном случае – для установки синего фона панели приложения и команд и зеленого цвета переднего плана (несколько отвратительное сочетание, однако, показывающее все, что требуется от примера).

В качестве основы, Сценарии 2 (html/custom-color.html) добавляет CSS-класс customColor к панели приложения:

```
<div id="customColorAppBar" data-win-control="WinJS.UI.AppBar" class="cu
```

В css/custim-color.css он затем стилизует селекторы на основе .win-appbar.customColor. Следующие правила, например, задают фоновый цвет, цвет текста подписи и цвет круглой рамки вокруг

фоновый цвет, цвет текста подписи и цвет круглой рамки вокруг команды для состояний `:hover` и `:active`:

```
.win-appbar.customColor {
background-color: rgb(20, 20, 90);
}
.win-appbar.customColor .win-label {
color: rgb(90, 200, 90);
}
.win-appbar.customColor button:hover .win-commandring,
.win-appbar.customColor button:active .win-commandring {
background-color: rgba(90, 200, 90, 0.13);
border-color: rgb(90, 200, 90);
}
```

Все эти настройки стилей, кстати, применимы только к стандартному макету, основанному на командах. Если вы используете пользовательский макет, панель приложения просто содержит в себе все элементы, которые вы хотите на нее добавить, с любыми классами стилей, таким образом, вы можете управлять стилизацией как при работе с любыми другими HTML-элементами

Пользовательские значки

Ранее мы видели, что свойство `icon` элемента типа `AppBarCommand` обычно является одним из символов шрифта Segoe UI Symbol. Хотя для большинства нужд это подходит, время от времени может возникнуть необходимость в использовании символов из разных шрифтов (некоторые из нас просто не могут не пользоваться Wingdings!) или в отображении пользовательских изображений. Панель приложения поддерживает и то и другое.

Для использования различных шрифтов для всей панели приложения, просто добавьте класс к панели приложения и создайте правило, основанное на `win-appbar`:

```
win-appbar.customFont {
font-family: "Wingdings";
```

Для изменения шрифта определенной командной кнопки, добавьте класс к ее свойству `extraClass` (такому, как `customButtonFont`) и создать правило со следующим селектором (как в Сценарии 1 модифицированного примера):

```
button.customButtonFont .win-commandimage {
font-family: "Wingdings";
}
```

Для использования собственных изображений, выполните следующие шаги для масштаба разрешения 100%:

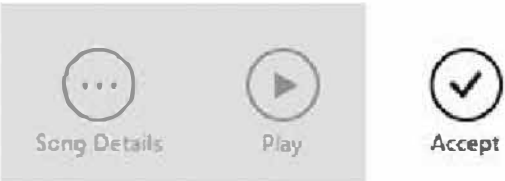
- Создайте спрайт в формате png с размером 160x80 пикселей с прозрачным фоном, сохранив файл с именем, имеющим суффикс .scale-100.
- Этот спрайт разделен на две строки по два столбца, то есть, на ячейки размером 40x40 пикселей. Верхняя строка предназначена для светлой темы интерфейса, нижняя – для темной.
- Каждая строка содержит четыре иконки для следующих состояний кнопок, по поярдку слева направо: кнопка по умолчанию (не нажатая), кнопка, над которой находится указатель мыши, нажатая кнопка (активная) и неактивная кнопка
- Каждое изображение отцентровано в соответствующей ему ячейке, но помните, что вокруг значка будет нарисована круглая рамка, поэтому помещайте изображение в диапазоне 20-30 пикселей вертикально и горизонтально. Оно, конечно, может быть шире или выше в центре – там, где больше места в круглой рамке.

Для других масштабов разрешений, умножьте размеры на 1.4 (140%) и 1.8 (180%) и используйте в именах файлов суффиксы .scale-140 и .scale-180.

Для использования пользовательского значка, установите свойство `icon` командного элемента в значение базового URI изображения (без суффиксов .scale-1x0). Например, `icon:`

`'url(images/icon.png)'` .

Сценарий 3 примера, посвященного элементу управления HTML AppBar показывает использование пользовательских значков для кнопки Accept:



Значок хранится в файле, который называется accept.png, который выглядит примерно так, как показано ниже. Я настроил яркость и контраст, добавил рамку, чтобы вы могли четко увидеть каждую ячейку:

Объявление кнопок панели приложения, таким образом, выглядит так, как показано ниже (некоторые свойства опущены для краткости):

```
<button data-win-control="WinJS.UI.AppBarCommand"
data-win-options="{id:'cmdAccept', label:'Accept', icon:'url(images/accept.png)'
```

Обратите внимание на то, что хотя в примере нет различных значков для разных масштабов разрешения, он содержит варианты для тем с высокой контрастностью, это важное условие обеспечения соглашений по специальным возможностям, о которых мы поговорим в лекции 6 курса "Программная логика приложений для Windows 8, созданных с использованием HTML, CSS и JavaScript и их взаимодействие с системой". По этой причине элемент `button` включает `style="-ms-high-contrast-adjust:none"` для того, чтобы переопределить автоматическую настройку для тем с высокой контрастностью.

контрастностью.

Командные меню

Следующий аспект панели приложения, который нуждается в пояснениях, касается тех команд, свойство `type` который установлено в значение `flyout`. В данном случае свойство `flyout` командного элемента должно идентифицировать объект `WinJS.UI.Flyout` или на элемент управления `WinJS.UI.Menu` (он является всплывающим элементом). Как мы упоминали ранее, всплывающие элементы или всплывающие меню наподобие этих используются тогда, когда в панели приложения нужно расположить слишком много связанных команд, или когда вам нужны другие типы элементов управления, которые не вполне подходят для их непосредственного расположения в панели приложения. Если вы пытаетесь воспользоваться кнопками с подписями "Еще", "Дополнительно", "Другие возможности", так как вы не можете решить, как организовать команды, это верный признак того, что приложение получается слишком сложным. Изыщите способы упрощения целей приложения, в итоге панель приложения не превратится в хранилище случайных элементов управления.

Мы подробнее поговорим о всплывающих элементах ниже в этой лекции, а сейчас посмотрим, как использовать их в панели приложения, как показано в Сценарии 6 примера "Всплывающие элементы управления HTML" (ссылка: http://code.msdn.microsoft.com/windowsapps/Flyout-sample-258757b3) (обратите внимание, это не пример про панель приложения!):

```
<button data-win-control="WinJS.UI.AppBarCommand"
data-win-options="{id:'respondButton', label:'Respond', icon:'edit', type:'flyout', f
>
```

Элемент respondFlyout, который здесь идентифицируется, определен ранее, в html/appbar-flyout.html. Обратите внимание на то, что подобный элемент должен быть объявлен раньше, чем панель приложения для того, чтобы его экземпляр был создан до создания панели приложения:

```
<div id="respondFlyout" data-win-control="WinJS.UI.Menu"
<button data-win-control="WinJS.UI.MenuCommand"
data-win-options="{id:'alwaysSaveMenuItem',
label:'Always save drafts', type:'toggle', selected:'true'}">
</button>
<hr data-win-control="WinJS.UI.MenuCommand"
data-win-options="{id:'separator', type:'separator'}" />
<button data-win-control="WinJS.UI.MenuCommand"
data-win-options="{id:'replyMenuItem', label:'Reply'}">
</button>
<button data-win-control="WinJS.UI.MenuCommand"
data-win-options="{id:'replyAllMenuItem', label:'Reply All'}">
</button>
<button data-win-control="WinJS.UI.MenuCommand"
data-win-options="{id:'forwardMenuItem', label:'Forward'}">
</button>
</div>
```

Вас не должно уже удивить то, что меню – это лишь еще один элемент управления WinJS, `WinJS.UI.Menu`, дочерние элементы которого определяютс содержимое меню. Все эти элементы, опять же, лишь элементы в DOM. Их события `click` подключаются к обработчикам в js/appbar-flyout.js с уже известным вам `addEventListener`. (В примере использована команда `document.getElementById` для получения элементов при вызове `addEventListener`. Эффективнее будет использовать метод панели приложения `getCommandById`, как в модифицированном примере работы с панелью приложения).

Каждый элемент меню, как вы можете видеть, это объект `WinJS.UI.MenuCommand`, мы так же рассмотрим подробности позднее, а сейчас вы можете видеть, что эти элементы имеют свойства `id`, `label`, `type` – это очень похоже на объекты `WinJS.UI.AppBarCommand`.

Вот и почти все, что нужно сделать, добавлю лишь, что когда элемент меню выбирают, нужно скрыть меню и, возможно, так же панель приложения (если она не находится в режиме залипания). Это показано в примере, в файле hs/appbar-flyout.js, в функции, которая называется `hideFlyoutAndAppBar`:

```
function hideFlyoutAndAppBar() {
document.getElementById("respondFlyout").winControl.hide();
document.getElementById("appBar").winControl.hide();
```

Пользовательские панели приложений и панели навигации

Все это время мы изучали стандартный макет команд панели приложения, что, конечно, является самым простым способом использования этого элемента управления. Однако, стандартных возможностей хватает не всегда. Возможно, вы захотите расположить на панели приложения больше интересных элементов управления, в особенности это касается пользовательских элементов управления (таких, как средство для выбора цвета). Для подобных целей вы можете установить свойство панели приложения `layout` в значение `'custom'` и задать любой HTML-макет для панели приложения, стилизовать его с помощью CSS и подключить обработчики любых необходимых событий в JavaScript.

Пользовательские макеты обычно используют для создании верхней панели навигации – то есть, панели приложения со свойством `placement` установленным в значение `'top'` – так как командные кнопки – это обычно не тот пользовательский интерфейс, который вам нужен. Мы видели пример подобной панели приложения ранее, в

`placement` установленным в значение `'top'` – так как командные кнопки – это обычно не тот пользовательский интерфейс, который вам нужен. Мы видели пример подобной панели приложения ранее, в приложении Weather, и навигационная панель в Internet Explorer устроена по тому же принципу:

Наш старый друг – пример элемента управления HTML AppBar (ссылка: http://code.msdn.microsoft.com/windowsapps/App-bar-sample-a57eeae9) предоставляет пример пользовательского макета в Сценарии 5. В html/custom-layout.html мы видим разметку для пользовательской верхней панели приложения, содержащей произвольные элементы:

```
<div id="customLayoutAppBar" data-win-control="WinJS.UI.AppBar" aria-lab
data-win-options="{layout:'custom', placement:'top'}">
<header aria-label="Navigation bar" role="banner">
<button id="cmdBack" class="win-backbutton" aria-label="Back"
</button>
<div class="titleArea">
<h1 class="win-type-xx-large" tabindex="0">
Page Title </h1>
</div>
</header>
</div>
```

Правда, результат этого примера странноват – он создает навигационную панель с обычным заголовком страницы и кнопкой "Назад", где каждый элемент управления может иметь прямоугольник выделения. Я не советую следовать подобным образцам дизайна!

Как я упоминал в подразделе "Некоторые советы и секреты" (в разделе

помощью правила `<selector>:focus { outline: none; }` в CSS. Для того, чтобы убрать его с кнопки "Назад", например, вы можете добавить следующее правило стиля в css/custom-layout.css

```
.win-appbar header .win-backbutton {
margin-left: 39px;
margin-top: 59px;
outline: none;
        }
```

Снова обратите внимание на то, как это правило и другие правила в css/custom-layout.ss используют класс `win-appbar` как базовый селектор, но только потому, что он используется для стилизации других обычных классов, таких, как `header` или `win-backbutton`. Если вы используете особые классы в панели приложения или в навигационной панели, вы можете совсем не пользоваться селектором `win-appbar`.

Для того, чтобы реализовать навигационную панель, напоминающую таковую в Internet Explorer или в приложении Weather, вы можете использовать элемент управления ListView вместе с шаблонами элементов или пользовательскими функциями вывода элементов, где вы подключаете события `itemInvoked` к `WinJS.Navigation.navigate` и так далее. Опять же, здесь нет ничего особенно сложного: панель приложения с пользовательским макетом – это лишь всплывающий контейнер для других HTML-элементов.

Всплывающие элементы и меню

Возвращаясь к дискуссии о том, где размещать командные элементы управления, напомним, что всплывающий элемент управления `WinJS.UI.Flyout` – используется для целей подтверждения действий, сбора информации и получения ответов на вопросы при реакции на действия пользователя. Меню – элемент управления `WinJS.UI.Menu` – это, в свою очередь, особый вид всплывающего элемента, который содержит элементы управления `WinJS.UI.MenuCommand` вместо произвольного HTML-кода. На

самом деле, `WinJS.UI.Menu` является, благодаря использованию `WinJS.Class.define`, прямым наследником `WinJS.UI.Flyout`, поэтому у них много общего. Что касается всплывающих элементов, они так же имеют общие черты с панелью приложения. (Классы панели приложения и всплывающего элемента являются наследниками базового класса `WinJS.UI._Overlay`, который является внутренним классом WinJS).

Совет. В дополнение к объекту `WinJS.UI.Flyout`, который вы можете использовать из приложения, существует и системный всплывающий элемент, который отображается в ответ на некоторые вызовы API, такие, как создание или удаление дополнительных плиток (смотрите Главу 2 курса "Программная логика приложений для Windows 8, созданных с использованием HTML, CSS и JavaScript и их взаимодействие с системой"). Хотя выглядят эти элементы одинаково, системный всплывающий элемент вызывает для приложения событие `blur`, в то время, как всплывающий элемент WinJS является частью приложения и не делает этого. В результате, показ системного всплывающего элемента приводит к скрытию панели приложения, которая не находится в режиме залипания. Для того, чтобы это предотвратить, необходимо устанавливать свойство панели приложения sticky в значение true перед вызовом API, использующими системные всплывающие элементы. Это показано в Сценарии 7 примера "Дополнительные плитки" (ссылка: http://code.msdn.microsoft.com/windowsapps/Secondary-Tiles-Sample-edf2a178).

Прежде чем мы рассмотрим подробности, взглянем на некоторые изображения из примера "Всплывающий элемент управления HTML" (ссылка: http://code.msdn.microsoft.com/windowsapps/Flyout-sample-258757b3), где мы уже видели меню, которое всплывает по команде элемента управления панели приложения. Элементы управления WinJS.UI.Flyout, использованные в сценариях 1-4 показаны на рис. 1.1. Обратите внимание на использование разного содержимого во всплывающих элементах и на то, что всплывающий элемент всегда расположен около элемента управления, который вызвал его, например, у кнопок Buy, Login, Format output text и около текста гиперссылки Lorem ipsum. Эти примеры показывают, что всплывающий элемент может содержать простое сообщение с кнопкой (Сцераний 1, для

у кнопок Buy, Login, Format output text и около текста гиперссылки Lorem ipsum. Эти примеры показывают, что всплывающий элемент может содержать простое сообщение с кнопкой (Сцераний 1, для предупреждений и подтверждений), может содержать поля для ввода информации или изменения параметров (Сценарии 2 и 3), и может иметь заголовок (Сценарий 4). Сценарий 5, в свою очередь, содержит пример всплывающего меню заголовка `WinJS.UI.Menu`, которое мы увидим немного позже.

Существуют две ключевые характеристики всплывающего элемента управления, включая меню. Первая заключается в том, что всплывающий элемент может быть скрыт программно, как панель приложения, когда активирован соответствующий элемент управления внутри него. Подобное поведение характерно для кнопки Complete order в Сценарии 1, для кнопки Login в Сценарии 2.

Вторая характеристика, так же актуальная для панели приложения, это закрытие неактивных всплывающих окон (light dismiss): щелчок мышью или прикосновение к области вне элемента управления скрывает его, как при нажатии клавиши Escape, что означает, что такое закрытие аналогично нажатию на кнопку Cancel (Отмена) или на кнопку закрытия в традиционном диалоговом окне. Преимущество такого подхода заключается в том, что нам не нужны видимые кнопки для этой цели, что помогает упрощению пользовательского интерфейса. В то же время, обратите внимание на то, что в Сценарии 3 рис. 1.1 нет кнопки OK или другого элемента управления, подтверждающего изменения, которые вы могли внести во всплывающем элементе. При рассматриваемой модели взаимодействия, изменения немедленно вступают в силу, таким образом ,закрытие всплывающего элемента не возвращает их к исходному состоянию и не отменяет. Если вас такое поведение не устраивает, вы можете разместить что-то вроде кнопки "Применить изменения" во всплывающем элементе и не вносить изменения до тех пор, пока кнопка не будет нажата. В таком случае, закрытие всплывающего элемента отменит изменения.

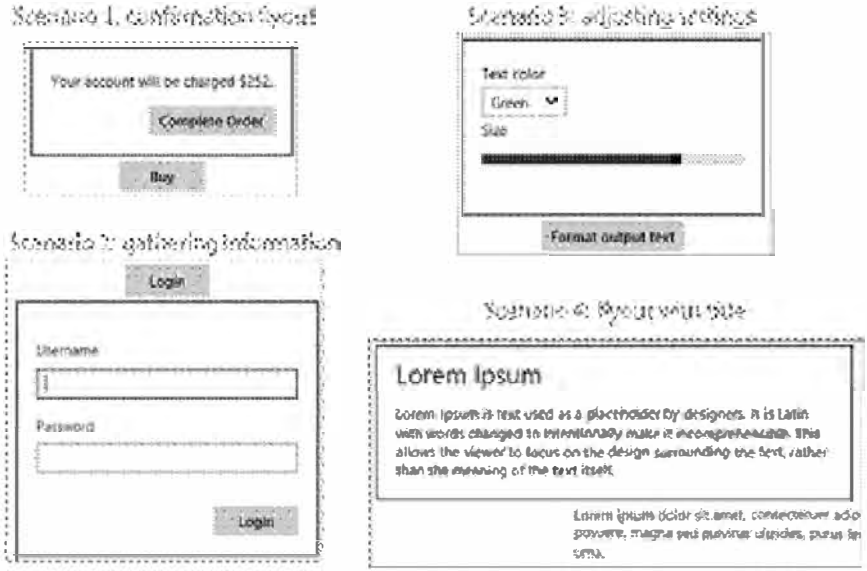


Рис. 1.1.

Я снова предлагаю вам почитать "Руководство и контрольный список для всплывающих элементов" (ссылка: http://msdn.microsoft.com/library/windows/apps/hh465341.aspx), который в подробностях рассматривает разные варианты дизайна и размещения, которые возможны для этого элемента управления. В нем так же говорится о том, где не следует использовать этот элемент управления: например, для вывода сведений об ошибках, не связанных с действиями пользователя (вместо этого используйте окна сообщений), для основных команд (используйте панель приложения), для контекстных меню, связанных с выделением текста, и для элементов пользовательского интерфейса, которые являются частью рабочего процесса и которые следует размещать на полотне приложения. Описываемые принципы так же предполагают использование всплывающих элементов небольших размеров, целенаправленное их использование (без лишних элементов управления), и их расположение около объекта, который их вызывает. Посмотрим теперь, как все это работает в коде.

Примечание. В дополнение к приложениям, которые могут отображать собственные элементы `WinJS.UI.Flyout`, некоторые системные API (например, при создании дополнительных плиток) создают

системные всплывающие элементы. В подобных случаях приложение получает событие blur, что ведет к закрытию панелей приложения. Для предотвращения подобного поведения, переведите панель приложения в режим залипания (sticky) при вызове этих API.

Свойства, методы и события WinJS.UI.Flyout

Большинство свойств, методов и событий элемента управления WinJS.UI.Flyout – это те, что мы уже рассматривали для панели приложения. Методы show и hide управляют его видимостью, свойство hidden показывает состояние видимости, в соответствующие моменты вызываются события beforeshow, aftershow, beforehide, и afterhide. Событие afterhide обычно используют для определения момента закрытия элемента.

Как и панель приложения, всплывающий элемент имеет свойство placement, но оно имеет другие значения, которые имеют смысл лишь в контексте свойств всплывающего элемента alignment и anchor. На самом деле, все три свойства – это необязательные параметры для метода show, так как они определяют позицию, в которой всплывающий элемент отобразится на экране. Значения placement и alignment могут быть установлены на сам элемент управления, так как они не обязательны при вызове show. (Заметим, что если вы не установите anchor в методе show, это свойство уже должно быть установлено на элемент управления, иначе show выдаст исключение.)

Свойство anchor идентифицирует элемент управления, который активирует всплывающий элемент, или любую другую операцию, которая может открыть всплывающий элемент (как в случае с подтверждением некоторого действия). Свойство placement, указывает, как всплывающий элемент должен располагаться по отношению к положению, заданному свойством anchor: 'top', 'bottom', 'left', 'right', или 'auto' (по умолчанию). Обычно placement задают только тогда, когда всплывающий элемент может перекрыть важное содержимое. В противном случае вы

рискуете тем, что всплывающий элемент будет уменьшен для того, чтобы занять доступное ему место. Содержимое всплывающего элемента при этом останется того же размера, что означает, - вот незадача! – что появятся полосы прокругки! Таким образом, если только у вас нет достойной причины и справки от врача, используйте тип размещения `'auto'` , так всплывающий элемент будет размещен там, где он может быть показан в полном размере. В том же духе, помните, что в прикрепленном режиме просмотра у вас есть лишь 320 пикселей по горизонтали, что означает, что всплывающий элемент, который вы показываете в данном режиме просмотра следует делать такого или меньшего размера.

Свойство `alignment`, в свою очередь, когда используется со свойством `placement` , установленным в `'top'` или `'bottom'` , определяет, как всплывающий элемент выравнивается с `anchor`: `'left'`, `'right'` , или `'center'` (по умолчанию). Само содержимое всплывающего элемента выравнивается посредством CSS как и любые HTML-элементы.

Если вам нужно стилизовать сам всплывающий элемент управления, вы можете задать стили в классе `win-flyout` , наподобшие шрифтов, выравнивания по умолчанию, полей и так далее. Что касается других классов стилей WinJS, родственных этому, используйте `win-flyout` как базу для более конкретных селекторов, если только вы не хотите стилизовать каждый всплывающий элемент в приложении. Обычно, на самом деле, вы так же исключаете `win-menu` из правила, таким образом, на всплывающие элементы меню такая стилизация не действует. Например, большинство сценариев в примере, который посвящен всплывающим элементам управления HTML, которые мы рассмотрим в дальнейшем, имеют правила наподобие такого:

```
.win-flyout:not(.win-menu) button,
.win-flyout:not(.win-menu) input[type="button"] {
margin-top: 16px;
margin-left: 20px;
float: right;
```

И, наконец, если по какой-либо причине вам нужно знать, когда

загружается всплывающий элемент, прослушивайте `DOMNodeInserted` в `document.body`:

```
document.body.addEventListener("DOMNodeInserted", insertionHandler, false);
```

Примеры всплывающих элементов

Всплывающий элемент управления создается так же, как любой другой элемент управления WinJS, с помощью атрибутов `data-win-control` и `data-win-options`, и обрабатывается `WinJS.UI.process/processAll`. Всплывающие элементы с относительно постоянным содержимым элементов обычно объявляют в разметке, где вы можете использовать привязку данных для некоторых свойств элементов во всплывающем элементе. Всплывающие элементы с динамическим содержимым, с другой стороны, могут быть созданы прямо из кода с использованием команды вида `new WinJS.UI.Flyout(<element>, <options>)` , и вы можете, безусловно, менять его дочерние элементы в любое время. Все это – лишь часть DOM! (Не повторяю ли я собственные слова?)

Как я говорил раньше (видимо, я повторяюсь) элемент управления `WinJS.UI.Flyout` может содержать произвольный HTML, стилизованный, как всегда, с помощью CSS. Всплывающий элемент для Сценария 1 в примере выглядит следующим образом в html/confirm-action.html (слегка сокращенный вид):

```
<div id="confirmFlyout" data-win-control="WinJS.UI.Flyout" aria-label="{Confi
<div>Your account will be charged $252. </div>
<button id="confirmButton">Complete Order</button>
</div>
```

Всплывающий элемент для входа в систему из Сценария 2 похож, и он даже использует HTML-форму для прикрепления к кнопке Login события нажатия на кнопку Enter на клавиатуре:

```
<div id="loginFlyout" data-win-control="WinJS.UI.Flyout" aria-label="{Login fly
<form onsubmit="return false;" >
```

```
<p >
<label for="username" gt;Username <br / ></label >
<span id="usernameError" class="error" > </span >
<input type="text" id="username" / >
</p >
<p >
<label for="password">Password<br / ></label >
<span id="passwordError" class="error" ></span >
<input type="password" id="password" / >
</p >
<button id="submitLoginButton" >Login</button >
</form >
</div >
```

Всплывающий элемент отображается путем вызова его метода `show` . В Сценарии 1, например, событие кнопки `click` прикреплено к функции `showConfirmFlyout` (js/confirm-action.js), где кнопка Buy задается как элемент прикрепления. Обработка нажатия на кнопку Complete Order так же производится посредством обработчика события click, прикрепленного к этому элементу, и здесь мы вызовем метод `hide` для программного закрытия всплывающего элемента. Наконец, событие `afterhide` используется для выявления факта закрытия элемента:

```
var bought;

var page = WinJS.UI.Pages.define("/html/confirm-action.html", {
ready: function (element, options) {
document.getElementById("buyButton").addEventListener("click", showConfirmI
document.getElementById("confirmButton").addEventListener("click",
confirmOrder, false);
document.getElementById("confirmFlyout").addEventListener("afterhide",
onDismiss, false);
}

function showConfirmFlyout() {
bought = false;
var buyButton = document.getElementById("buyButton");
```

```
document.getElementById("confirmFlyout").winControl.show(buyButton);


// Когда кнопка Buy нажата, скрыть всплывающий элемент, так как поль
function confirmOrder() {
bought = true;
document.getElementById("confirmFlyout").winControl.hide();
}
// При закрытии элемента, определяет, был ли он закрыт потому, что по
// Кнопку Byu. Если нет, элемент был закрыт автоматически.
function onDismiss() {
if (!bought) {
// (В примере сообщение о закрытии отображается на полотне)
}
```

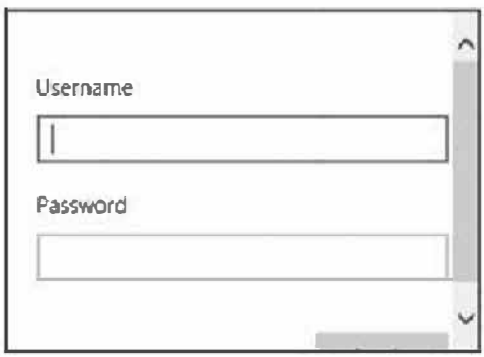
Работа с элементами управления для входа в систему в Сценарии 2, в основном – это то же самое, с некоторыми дополнениями в виде кода, который позволяет проверить, были ли заполнены поля для ввода имени пользователя и пароля. Если это не так, обработчик нажатия кнопки Login показывает встроенное в элемент сообщение об ошибке и устанавливает фокус ввода на соответствующее поле:

Так как всплывающий элемент в Сценарии 2 немного больше, значение по умолчанию `'auto'` для свойства `placement` (как в имитаторе) приводит к появлению элемента ниже кнопки, которая его вызывает. Выше этой кнопки не так много места. Поэтому попытаемся установить

свойство placement в значение `'top'` в вызове `show`:

```
function showLoginFlyout() {
// ...
document.getElementById("loginFlyout").winControl.show(loginButton, "top");
}
```

После этого вы можете увидеть, как во всплывающем элементе появляются полосы прокрутки, так как его размер слишком мал:



Каким словом я раньше это называл? "Незадача"?

Двигаемся дальше, к Сценарию 3, в котором всплывающий элемент определен в разметке, где он содержит элементы управления `label`, `select`, `input`. В JavaSctipt, однако, он прослушивает события изменения этих элеменов и применяет полученные значения к элементам вывода данных на полотне приложения:

```
var page = WinJS.UI.Pages.define("/html/change-settings.html", {
ready: function (element, options) {
// ...
document.getElementById("textColor").addEventListener("change", changeColor
document.getElementById("textSize").addEventListener("change", changeSize, fa
}
});

// Изменение цвета текста
function changeColor() {
document.getElementById("outputText").style.color =
```

```
document.getElementById("textColor").value;

// Изменение размера текста
function changeSize() {
document.getElementById("outputText").style.fontSize =
document.getElementById("textSize").value + "pt";
```

Если бы у этого всплывающего элемента была бы кнопка Применить (Apply) вместо его немедленной реакции на изменения, ее обработчик `click` получал бы текущее выделение и значение из ползунка и использовал бы их для того, чтобы выполнить действия, аналогичные тем, что производятся в функциях `changeColor` и `changeSize`.

И, наконец, в Сценарии 4 мы видим всплывающий элемент с заголовком, который является просто элементом в разметке, выводящим крупный текст. Сам по себе всплывающий элемент не имеет выделенного описания для заголовков:

```
<div id="moreInfoFlyout" data-win-control="WinJS.UI.Flyout" aria-label="{Mor
<div class="win-type-x-large">Lorem Ipsum</div>
<div>
Lorem Ipsum is text used as a placeholder by designers...
</div>
</div>
```

Смысл этого последнего примера заключается в том, чтобы показать, что в отличие от традиционных диалоговых окон настольных приложения, всплывающие элементы не часто нуждаются в заголовках, так как у них уже есть контекст внутри приложения. Диалоговые окна в настольных приложениях нуждаются в заголовках, так как они могут относиться к разным приложениям.

Меню и команды меню

Элемент управления `WinJS.UI.Menu` отличает от более общего элемента `WinJS.UI.Flyout` то, что меню ожидает, что все его

дочерние элементы будут объектами типа `WinJS.UI.MenuCommand`, так же, как стандартный макет панели приложения ожидает объекты `AppBarCommand` (и его экземпляр не будет создан, если вы объявите у него какие-нибудь другие дочерние элементы). На самом деле, у меню есть общие характеристики и с панелью приложения, и со всплывающим элементом:

- Методы `show` и `hide`.
- Методы `getCommandById`, `showCommands`, `hideCommands`, и `showOnlyCommands`, вместе со свойством `commands`, что означает, что вы можете использовать те же подходы для управления командами меню, которые обсуждались в подразделе "Отображение, скрытие, включение и обновление команд" в разделе, посвященном панели приложения, в том числе – задание команд меню используя JSON-массив вместо описания отдельных элементов.
- События `beforeshow`, `aftershow`, `beforehide`, и `afterhide`.
- Свойства `anchor`, `alignment`, и `placement`.

Меню поддерживает два стиля, влияющих на его отображение – win-menu и win-command – который можно использовать для создания более конкретных селекторов, как мы видели, для всего меню или для отдельных текстовых команд.

Объекты MenuCommand так же очень похожи на объекты AppBarCommand. У них много одинаковых свойств: `id`, `label`, `type` (`'button'`, `'toggle'`, `'flyout'`, и `'separator'`), `disabled`, `extraClass`, `flyout`, `hidden`, `onclick`, и `selected`. У команд меню нет значков, разделов и всплывающих подсказок, но из `type` вы можете видеть, что элементы меню могут быть кнопками (включая полностью текстовые элементы), элементами с возможностью установки флагов, разделителями и, кроме того, другими всплывающими элементами. В последнем случае, вторичное меню заменит первое вместо того, чтобы отображаться рядом с ним, а чтобы быть полностью честным, я должен все же видеть вторичное меню, используемое в реальном приложении. Тем не менее, такая возможность поддерживается.

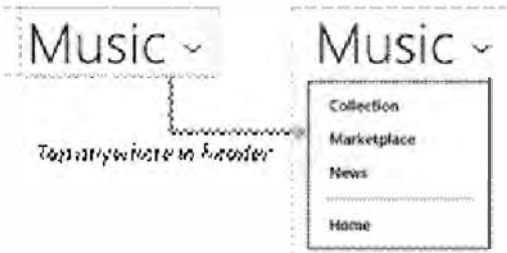
Мы уже видели, как использовать всплывающие меню с помощью команд панели приложения, что показано в Сценарии 6 примера, посвященного всплывающему элементу управления HTML (посмотрите раздел "Командные меню" выше). Другая типичная сфера использования – реализация того, что выглядит как выпадающее меню заголовочного элемента, что показано в Сценарии 5. Здесь (html/header-menu.html), показан стандартный дизайн помещения символа перевернутой угловой скобки (&#xe099) в конце заголовка:

```
<header aria-label="Header content" role="banner">
<button class="win-backbutton" aria-label="Back"></button>
<div class="titlearea win-type-ellipsis">

<button class="titlecontainer">
<h1>
<span class="pagetitle">Music</span>
<span class="chevron win-type-x-large" > & #xe099 </span>
</h1>

</button>
</div>
</header>
```

Обратите внимание на то, что весь заголовок помещен в элемент управления `button` (кнопка), ее обработчик `click` может отображать меню с помощью метода `show`:

```
document.querySelector(".titlearea").addEventListener("click", showHeaderMen

function showHeaderMenu() {
var title = document.querySelector("header .titlearea");
var menu = document.getElementById("headerMenu").winControl;
menu.anchor = title; menu.placement = "bottom";
menu.alignment = "left";
menu.show();
}
```

Всплывающий элемент (определенный как headerMenu в html/header-menu.html) появляется, когда вы щелкаете в любом месте заголовка (а не только на угловой скобке, которая является обычным символом в тексте заголовка):



Отдельные команды меню – это простые элементы button, поэтому вы можете присоединить к ним обработчики click если вам это нужно. Как и в случае с панелью приложения, лучше всего использовать метод элемента управления меню getCommandById для обнаружения этих элементов, так как это гораздо эффективнее, чем document.getElementById (что, к сожалению, используется в примере из SDK).

Для того чтобы увидеть в действии вторичное меню, попытайтесь добавить следующий элемент secondaryMenu в html/header-menu.thml перед элементом headerMenu и добавить элемент `button` в headerMenu, свойство flyout которого ссылается на secondaryMenu:

```
<div id="secondaryMenu" data-win-control="WinJS.UI.Menu">
<button data-win-control="WinJS.UI.MenuCommand"
data-win-options="{id:'command1', label:'Command 1'}"></button>
<button data-win-control="WinJS.UI.MenuCommand"
data-win-options="{id:'command2', label:'Command 2'}"></button>
<button data-win-control="WinJS.UI.MenuCommand"
data-win-options="{id:'command3', label:'Command 3'}"></button>
</div>
<div id="headerMenu" data-win-control="WinJS.UI.Menu">
<!-- ... -->
<button data-win-control="WinJS.UI.MenuCommand"
data-win-options="{id:'showFlyout', label:'Show secondary menu',
type:'flyout', flyout:'secondaryMenu'}">
</button>
</div>
```

Кроме того, перейдите в css/header-menu.css и установите стиль width элемента #headerMeny в значение 200px. С этими изменениями первое меню будет выглядеть так, как показано ниже, изменение цвета заголовка – результат действия эффекта зависания над ним указателя мыши:

Когда вы выберете пункт Show secondary meny (Показать вторичное меню), первое меню будет закрыто и откроется меню второго уровня:

Другой пример использования всплывающих элементов в заголовке можно найти в примере "Адаптивный макет с использованием CSS" (ссылка: http://code.msdn.microsoft.com/windowsapps/Adaptive-layout-with-sample-062e7fe2), который мы видели в лекции 6 курса "Введение в разработку приложений для Windows 8 с использованием HTML, CSS и JavaScript". Он реализован так же, как показано выше, с добавлением механизмов, которые изменяют содержимое страницы в ответ на выделение.

Контекстные меню

Помимо всплывающих меню, которые мы уже видели, существуют и контекстные меню, описанные в материале "Руководство и

контрольный список для контекстных меню" (ссылка: http://msdn.microsoft.com/library/windows/apps/hh465308.aspx). Они используются для команд, которые напрямую связаны с выделением разного рода, таким, как команды буфера обмена для текста, они вызываются щелчком правой кнопки мыши по элементу, прикосновением или клавишей вызова контекстного меню на клавиатуре. Текстовые элементы управления и гиперссылки обеспечивают данную возможность по умолчанию. Контекстные меню так же хорошо подходят для предоставления команд для объектов, которые не могут быть выделены (таких, как части беседы в программе мгновенного обмена сообщениями), команды панели приложения не могут быть контекстно чувствительными к подобным элементам. Они так же рекомендованы для совершения действий, которые нельзя выполнить путем непосредственного взаимодействия с объектом. Однако, не используйте их для фона страницы – именно для этого существует панель приложения, она будет появляться автоматически при щелчке правой кнопкой мыши или соответствующем жесте.

Совет. Если вы обрабатываете событие нажатий правой кнопки мыши на элемент, убедитесь в том, что для этого элемента отключено стандартное поведение, которое подразумевает отображение панели приложения. Таким образом, используйте событие щелчка правой кнопки мыши разумно, так как пользователи привыкли пользоваться правой кнопкой для вызова панели приложения. Обратите так же внимание на то, что вы можете программно открыть панель приложения, используя ее метод `show`.

Пример "Контекстное меню" (ссылка: http://code.msdn.microsoft.com/windowsapps/Context-menu-sample-40840351) предоставляет нам некоторый контекст – я знаю, что это неудачная игра слов! В любом случае, вам нужно лишь прослушивать событие HTML `contextmenu` для соответствующего элемента. Вам не нужно заботиться особым образом об обработке касаний или о событиях клавиатуры. Сценарий 1 примера, например, имеет невыделяемый элемент `attachment`, для которого прослушивается вышеупомянутое событие (html/scenario1.html):

```
document.getElementById("attachment").addEventListener("contextmenu", attach
```

В обработчике события, вы создаете объект `Windows.UI.Popups.PopupMenu` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.ui.popups.popupmenu.a (это объект из WinRT, не из WinJS!), заполняет его объектами `Windows.UI.Popups.UICommand` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.ui.popups.uicommand.a (они содержат подпись элемента и обработчик события) или объектами `UICommandSeparator` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.ui.popups.uicommandse и затем вызывает метод меню `showAsync` (js/scenario1.js):

```
function attachmentHandler(e) {
var menu = new Windows.UI.Popups.PopupMenu();
menu.commands.append(new Windows.UI.Popups.UICommand("Open with", c
menu.commands.append(new Windows.UI.Popups.UICommand("Save attachm
onSaveAttachment));

menu.showAsync({ x: e.clientX, y: e.clientY }).done(function (invokedCommand
if (invokedCommand === null) {
// Команда - это null если ни одна команда не была активирована.
}
});
```

Обратите внимание на то, что результат метода [2] `showAsync` – это активация объекта `UICommand`. Вы можете проверить его свойство `id` для того, чтобы предпринять дальшейшие действия. Кроме того, параметр, который передают в `showAsync` – это объект `Windows.Foundation.Point` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.foundation.point.aspx), который показывает, где следует появиться меню по отношению к указателю мыши или точке касания. Меню размещается выше данной точки и выровнено по центру.

Объект PopupMenu так же поддерживает метод, который называется showForSelectionAsync, первым аргументом которого является объект `Windows.Foundation.Rect` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.foundation.rect.aspx), который описывает область выделения. Опять же, меню размещается

выше прямоугольника выделения с выравниванием по центру. Это показано в Сценарии 2 примера в js/scenario2.js:

```
//В обработчике contextmenu
menu.showForSelectionAsync(getSelectionRect()).then(function (invokedComma
// ...

function getSelectionRect() {
var selectionRect = document.selection.createRange().getBoundingClientRect();

var rect = {
x: getClientCoordinates(selectionRect.left),
y: getClientCoordinates(selectionRect.top),
width: getClientCoordinates(selectionRect.width),
height: getClientCoordinates(selectionRect.height)
};
return rect;
};
```

Сценарий так же показывает, что вы можете использовать обработчик события `contextmenu` с текстом для переопределения команд по умолчанию, которые предоставляют соответствующие элементы управления.

Последнее замечание, касающееся контекстных меню: так как они создаются с помощью API WinRT, а не с помощью элементов управления WinJS, меню не существуют в DOM, они не известны DOM, что объясняет использование других конструкций WinRT – как `point` (Точка) и `Rect` (Прямоугольник). Подобное утверждение так же справедливо для окон сообщений, которые станут последней темой этой лекции.

Диалоговые окна сообщений

Последняя тема этой лекции, касающаяся командного интерфейса, посвящена диалоговым окнам сообщений. Как и контекстные меню, эти всплывающие элементы относятся к WinRT а не к WinJS, для работы с ними используется API Windows.UI.MessageDialog (ссылка:

http://msdn.microsoft.com/library/windows/apps/windows.ui.popups.messagedialo
Это, опять же, означает, что окна сообщений просто появляются поверх текущей страницы и не присутствуют в DOM. Диалоговые окна автоматически затеняют текущую страницу приложения и блокируют события ввода от приложения до тех пор, пока пользователь не отреагирует на окно сообщения.

В материале "Руководство и контрольный список для окон сообщений" (ссылка: http://msdn.microsoft.com/library/windows/apps/hh738363.aspx) показаны варианты их использования:

- Для отображения срочной важной информации, которую пользователь должен знать для продолжения работы, особенно для ситуаций, которые не связаны с пользовательскими командами различного рода.
- Для вывода сведений об ошибках, которые применимы ко всему приложению, что отличается от ошибок, возникающих в ходе рабочего процесса, которые лучше отображать внутри приложения. Хороший пример – потеря сетевого соединения.
- Вопросы, которые требуют реакци пользователя и окна, отображающие которые, не могут быть закрыты как неактивные всплывающие окна. Таким образом, используйте диалоговые окна для того, чтобы прервать работу пользователя в ситуациях, когда ответ пользователя необходим для продолжения работы.

Интерфейс диалоговых окон очень прост. Вы создаете объект диалогового окна сообщения командой `new Windows.UI.Popups.MessageDialog`. Конструктор принимает в качесве обязательного параметра строку с содержимым сообщения и с необязательным вторым строковым параметром, содержащим заголовок окна. Окно сообщения имеет свойства `content` и `title`, которые вы можете использовать независимо друг от друга. В любом случае, строковые параметры поддерживают лишь обычный текст.

После этого вы конфигурируете диалоговое окно с помощью его свойств `commands`, `options`, `defaultCommandIndex` (команда, связанная с клавишей Enter) и `cancelCommandIndex` (команда, связанная с клавишей Escape).

Значения параметра options берутся из перечисления `Windows.UI.Popups.MessageDialogOptions`, которое содержит лишь два члена: `none` (значение по умолчанию, для случаев, когда особое поведение не требуется), и `acceptUserInputAfterDelay` (что приводит к тому, что окно сообщения некоторое время не воспринимает пользовательский ввод для того, чтобы предотвратить кликджекинг (clickjaking); это существенно, преимущественно, для интернет-браузеров, которые загружают произвольное веб-содержимое и практически не требуется большинству приложений).

Свойство command содержит до трех объектов типа `Windows.UI.Popups.UICommand` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.ui.popups.uicommand.a тех же самых, которые используются в контекстных меню. Каждая из команд содержит свойства `id`, `label` и `invoked`, к которому вы подключаете обработчик для команды. Обратите внимание на то, работа свойств `defaultCommandIndex` и `cancelCommandIndex` основана на индексах массива `commands`, а не на `id` этих команд. Кроме того, если вы не добавите собственные команды, диалоговое окно сообщения отобразится с единственной командой по умолчанию – Закрыть (Close).

И, наконец, когда диалоговое окно настроено, вы отображаете его, вызывая его метод `showAsync`. Как и в случае с контекстным меню, в результате выбранный пользователем объект `UICommand`, который передается обработчику завершения, который вы предоставили для метода `done` promise-объекта. Обычно вам не нужно получать этот результат, так как выбранная команда имеет связанный с ней обработчик `invoked`, где вы и выполняете действия, соответствующие команде.

Примечание. Если в то время, когда активно окно сообщения, активируются чудо-кнопки Поиск, Общий доступ, Устройства или Параметры, или если приложение активируется для обслуживания контракта, диалоговое окно сообщения будет закрыто и без выбора какой-либо команды в нем. Будет осуществлен вызов обработчика завершения для `showAsync`, с результатом, установленным на команду по умолчанию. Знайте об этом, если вы используете обработчик завершения для обработки подобных команд.

Пример "Диалоговое окно сообщения" (ссылка: http://code.msdn.microsoft.com/windowsapps/Message-dialog-sample-00c928f5) – это один из самых простых примеров во всем SDK! Он показывает разные варианты использования API окон сообщений. Сценарий 1 показывает окно сообщения с заголовком и двумя командными кнопками, вторая командная кнопка (с индексом 1) установлена как кнопка по умолчанию. Вот как это выглядит:

Сценарий 2 показывает вывод команды по умолчанию – Закрыть (Close) с сообщением и без заголовка:

Сценарий 3 похож на Сценарий 1, но использует обработчик завершения метода `showAsync().done` для обработки команды, выбранной пользователем. Вы можете использовать этот пример для того, чтобы увидеть эффект активации чудо-кнопок при открытом диалоговом окне.

И, наконец, Сценарий 4 делает командой по умолчанию первую команду и отмечает вторую команду в качестве команды отмены, то есть, той команды, которая закрывает окно либо при ее выборе, либо при нажатии на клавишу клавиатуры Escape.

И это действительно все, что относится к окнам сообщений!

Улучшение обработки ошибок в "Here My Am!"

Для того, чтобы завершить эту лекцию и свести воедино большую часть того, о чем мы говорили, внесем некоторые изменения в "Here My Am!". В последний раз мы работали над этой программой, улучшая обработку различных ошибок. В ее теперешнем состоянии программа ведет себя не очень хорошо. А именно, есть сложности в следующих областях:

- Если загрузка скрипта элемента управления карт Bing из удаленного источника не удается, код в html/map.html просто выдает исключение и программа завершается.
- Если мы используем приложение на мобильном устройстве и изменили наше местоположение, нет способа обновить отметку на карте другим способом кроме ручного перетаскивания маркера. Таким образом, API для определения местоположения используется лишь при старте программы.
- Когда API определения местоположения WinRT пытается получить сведения о местоположении и при этом нет сетевого соединения, возникают паузы в несколько секунд, когда пользователь не может знать, что происходит.
- Если при попытке воспользоваться API определения местоположения вызов не удастся, обычно из-за тайм-ауга или проблем с сетевым соединением, но так же – возможно из за-того, что пользователь это запретил, нет способа попробовать выполнить то же действие еще раз.

Приложение "Here My Am!" (7) для этой лекции учитывает эти вопросы. Во-первых, я добавил изображение, которое должно выводиться при ошибке, в html/map.html (изображение находится по адресу html/maperror.png), таким образом, если не удается загрузить скрипт карт Bing, вместо карты будет выведено изображение (в этом случае стиль отображения `none` будет удален):

```
<img id="errorImage" style="display: none; width: 100%; height: 100%;" src="ma
```

Кроме того, я добавил к изображению обработчик события `click`, который перезагружает содержимое `iframe` с `document.location.reload(true)`. Когда это сделано, я могу убрать исключения, которые возникли ранее, когда карта не могла быть создана, предотвращая аварийное завершение приложения. Вот как выглядит ситуация, когда карта не может быть создана:

Для того, чтобы протестировать такой вариант развития событий, вам нужно отключиться от Интернет, деинсталлировать приложение (для очистки кэшированных скриптов карты; в противном случае карта, все равно, будет загружаться!) и снова запустить приложение. Ошибка должна произойти в начале метода `init` в файле html/map.html, что приведет к появлению изображения путем удаления стиля по умолчанию `display: none` и подключения обработчика `click` . Затем снова подключитесь к Интернету и нажмите на изображение. Карта должна перезагрузиться, но если проблема с соединением сохранилась, снова появится изображение, сигнализирующее об ошибке.

Вторая проблема – добавление возможности обновлять наше местоположение – просто решаема с помощью панели приложения. Я добавил такой элемент управления на страницу default.html с одной командой:

```
<div id="appbar" data-win-control="WinJS.UI.AppBar" data-win-options="">
<button data-win-control="WinJS.UI.AppBarCommand"
data-win-options="{id:'cmdRefreshLocation', label:'Refresh location',
icon:'globe', section:'global', tooltip:'Refresh your location'}">
</button>
</div>
```

Эта команда подключена внутри pages/home/home.js в методе `ready`

элемента управления страницы:

```
var appbar = document.getElementById("appbar").winControl;
appbar.getCommandById("cmdRefreshLocation").addEventListener("click", this.
```

Здесь обработчик tryRefresh, так же в элементе управления страницы, скрывает панель приложения и вызывает другой новый метод, refreshPosition, куда я поместил код, который получает сведения о местоположении и обновляет карту:

```
tryRefresh: function () {
//Скрыть панель приложения и попробовать снова
var appbar = document.getElementById("appbar").winControl.hide();
this.refreshPosition();
},
```

Мне так же понадобилось настроить функцию `pinLocation` в html/map.html. Без команды обновления местоположения, эта функция вызывалась лишь однажды, при запуске приложения. Так как теперь она может быть вызвана много раз, нам нужно удалить любые существующие метки на карте, прежде чем добавлять метку для нового местоположения. Это делается путем вызова `map.entities.pop` перед уже существующим вызовом `map.entities.push`, который отмечает новое местоположение.

Панель приложения теперь выглядит так, как показано ниже, и мы можем, при необходимости, обновить местоположение. (Если вы запускаете программу не на мобильном устройстве, попытайтесь перетащить первую метку на новое место и затем запустить обновление для того, чтобы увидеть, как метка вернулась на прежнее место).

Что касается третьей проблемы – дать пользователю знать о том, что приложение пытается получить данные о местоположении – мы можем

показать маленькое всплывающее сообщение сразу же после попытки вызова `getGeopositionAsync` из WinRT. Всплывающий элемент, определенный в pages/home/home.html (в нашем элементе управления страницы) центрирован вдоль нижней части области карты:

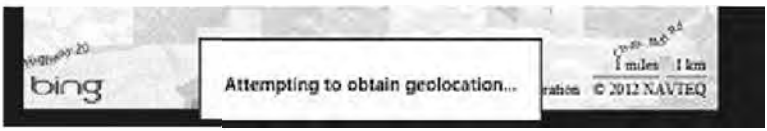
```
<div id="retryFlyout" data-win-control="WinJS.UI.Flyout" aria-label="{Trying ge
data-win-options="{anchor: 'map', placement: 'bottom', alignment: 'center'}">
<div class="win-type-large">Attempting to obtain geolocation...</div>
</div>
```

Функция `refreshFunction` которую мы только что добавили в pages/home/home.js является отличным местом для отображения всплывающего окна сразу перед вызовом `getGeopositionAsync`:

```
 refreshPosition: function () {
document.getElementById("retryFlyout").winControl.show();
var gl = new Windows.Devices.Geolocation.Geolocator();

gl.getGeopositionAsync().done(function (position) {
//...

//Всегда скрывать всплывающий элемент document.getElementById("retryF

//...
}, function (error) {
//...

//Всегда скрывать всплывающий элемент
document.getElementById("retryFlyout").winControl.hide();
});
},
```

Обратите внимание на то, что мы скрываем всплывающий элемент внутри обработчиков ошибки и завершения, поэтому сообщение остается видимым во время проведения асинхронной операции. Если мы разместим одну команду, скрывающую всплывающий элемент, за пределами этих обработчиков, сообщение будет отображаться лишь очень короткое время перед закрытием, а это – не то, что нам нужно.

Когда мы это написали, у пользователя будет достаточно времени для того, чтобы увидеть сообщение вдоль нижней части карты (если только он не решит закрыть его):



Последняя задача заключается в том, чтобы сообщить пользователю о том, что при попытке получить сведения о местоположении произошла ошибка. Мы можем сделать это с помощью еще одного всплывающего элемента с кнопкой Retry (Повторить), или во встроенном сообщении, как показано ниже. Нам не следует пользоваться в этом случае диалоговым окном сообщения, так как сообщение появляется в ответ на действие пользователя, которым он запускает процесс обновления данных. Окно сообщения допустимо использовать при запуске программы, но с помощью встроенного сообщения, скомбинированного со всплывающим элементом мы уже выполнили все, что нужно.

Для встроенного сообщения, я добавил плавающий элемент `div`, который позиционирован примерно на трети расстояния от верхнего края карты. Он определен в defined/home/home.html, как показано, в качестве элемента одного уровня с `iframe`:

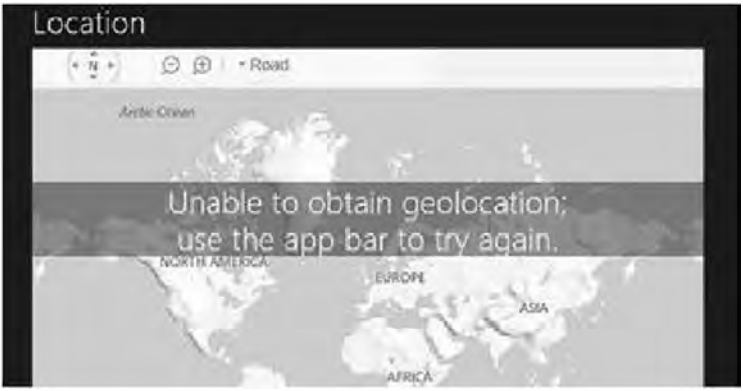
```
<div id="locationSection" class="subsection" aria-label="Location section">
<h2 class="group-title" role="heading">Location</h2>
<iframe id="map" class="graphic" src="ms-appx-web:///html/map.html"
aria-label="Map"></iframe>
<div id="floatingError" class="win-type-x-large">Unable to obtain geolocation;<
use the app bar to try again.</div>
</div>
```

Стиль для правила `#floatingError` в pages/home/home.css задает внешний вид и расположение элемента:

```
#floatingError {
display: none; float: left;
-ms-grid-column: 1;
```

```
-ms-grid-row: 2;
-ms-grid-row-align: start;
width: 100%;
text-align: center;
background-color: rgba(128, 0, 0, 0.5);
margin-top: 20%;
}
```

Так как этот элемент размещен в той же ячейке сетки, что и карта со стилем float, это выглядит как хорошо выглядящее наложение:



Это сообщение появится, если пользователь отклонил запрос на доступ приложения к информации о местоположении, который ему задавали при запуске программы, или позволил доступ, но позже, используя чудо-кнопку Параметры, запретил пользоваться этой возможностью. Вы можете использовать эти варианты для тестирования появления сообщения. Возможно так же, если вы запускаете приложение в первый раз и при этом нет соединения с Интернетом, чтобы это сообщение выводилось поверх изображения, сообщающего об ошибке загрузки карты. Именно поэтому я расположил сообщение об ошибке сервиса получения местоположения ближе к верхней части, в итоге, оно не перекрывает сообщение об ошибке загрузке карты. Но, если вы успешно запустили приложение и затем связь оборвалась, карта должна создаваться, так как скрипты карт Bing кэшируются.

С параметром `display: none` в CSS, сообщение об ошибке первоначально скрыто, как это и должно быть. Если мы попадаем в

обработчик ошибки для `getGeolocationAsync`, мы устанавливаем `style.display` в значение `block`, что отображает элемент:

```
document.getElementById("floatingError").style.display = "block";
```

Мы снова скрываем сообщение внутри функции tryRefresh, которая вызывается посредством команды на панели приложения, таким образом, сообщение остается скрытым, если ошибка повторяется:

```
tryRefresh: function () {
document.getElementById("floatingError").style.display = "none";
//...
},
```

Так же мы можем добавить окно сообщения, если мы обнаружим, что сетевое соединение потеряно, и, таким образом, не можем обновить местоположение. Как мы увидим в лекции 3 курса "Программная логика приложений для Windows 8, созданных с использованием HTML, CSS и JavaScript и их взаимодействие с системой", это можно сделать с помощью события `Windows.Networking.NetworkInformation.onnetworksta` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.networking.connectivity. В подобном случае вполне можно воспользоваться окном сообщения, так как происходящее не связано с непосредственными действиями пользователя.

Кроме того, стоит отметить, что если бы мы использовали элемент управления из Bing Maps SDK в данном приложении, скрипт, который мы загружаем из удаленного источника, хранился бы в пакете нашего приложения, таким образом, устраняя причину первой ошибки. Мы внесем это изменение в следующую версию программы

Что мы только что изучили

- В дизайне приложений для Магазина Windows, команды, которые имеют отношение к рабочему процессу, следует располагать на

полотне приложения или во всплывающих меню элементов, например – заголовка. Те команды, которые могут быть размещены в интерфейсе чудо-кнопки Параметры, нужно размещать там; это значительно упрощает реализацию приложения. Те команды, которые остались, обычно отображаются на панели приложения или на панели навигации. Панели могут содержать всплывающие меню для некоторых командных элементов. Контекстные меню (`Windows.UI.Popups.PopupMenu`) так же можно использовать для команд, связанных с содержимым.

- Элемент управления `WinJS.UI.Flyout` используется для подтверждений и других вопросов, которые отображаются в результате действий пользователя. Они так же могут просто выводить сообщения, собирать дополнительную информацию или предоставлять элементы управления для изменения настроек некоторых частей страниц. Неактивные всплывающие элементы могут быть скрыты, то есть, щелчок или прикосновение за пределами элемента или нажатие на клавишу Escape приводит к их скрытию, что равносильно нажатию на кнопку Отмена (Cancel) в ответ на вопрос, который может быть задан в подобном элементе.
- Диалоговые окна сообщений (`Windows.UI.Popups.MessageDialog`) используются для задавания вопросов, на которые пользователь должен ответить, или вывода информации, с которой он обязательно должен ознакомиться, прежде чем приложение сможет продолжить работу. Окно сообщения блокирует приложение. Окна сообщений лучше всего использовать для вывода сведений об ошибках или условиях, которые воздействуют на приложение в целом. Сообщения об ошибках, которые относятся к содержимому странице, следует показывать на странице.
- Панель приложения – это элемент управления WinJS, на котором можно располагать стандартные команды, используя командный макет, или любой HTML-код, который вам нужен, с использованием пользовательского макета. Возможно использование пользовательских значков, можно использовать различные шрифты или пользовательские изображения. Приложение может иметь и верхнюю и нижнюю панели, где верхняя обычно используется для целей навигации и применяет

пользовательский макет. Панели приложения могут отображаться в режиме залипания для того, чтобы оставаться видимыми, а не скрываться, если они не активны.

- Методы панели приложения `showCommand`, `hideCommand` и `showOnlyCommand`, вместе со свойством команд `extraClass`, упрощают описание панели приложения в одном месте приложения и выборочный показ конкретных наборов команд с использованием `querySelectorAll` с классом, который представляет данный набор.
- Командные меню, появляющиеся у команд панели приложения, или вызываемые различными элементами управления, расположенными на полотне, реализованы с помощью элемента управления `WinJS.UI.Menu`.
- В качестве примера использования многих из этих возможностей, в данной лекции обновлено приложение "Here My Am!", что привело к значительному улучшению обработки ошибок в этом приложении.

1) Хочу сделать три примечания. Во-первых, внутри data-win-options escape-последовательности Unicode могут быть заданы в HTML-формате, вида &#xNNNN; я предпочитаю формат JSON, так как он имеет более простую запись и менее подвержен ошибкам. Во-вторых, вы можете воспользоваться настольным приложением Таблица символов (Char Map, charmap.exe) для того, чтобы просмотреть все символы в каком-либо конкретном шрифте. В-третьих, если вам нужно локализовать значок, вы можете задать свойство значка в строке data-win-res, так как свойство icon, в конечном счете, разрешается в строку.

2) Пример, на самом деле, вызывает здесь не done, а then. Если вам интересно, почему существует подобная последовательность, это потому, что метод done был представлен не в начале разработки Windows 8, а тогда, когда стало ясно, что нам нужен более совершенный механизм для обработки ошибок в promise-вызовах, объединенных в цепочку. В результате, множество SDK-примеров и кода в документации все еще используют then вместо done, когда обрабатывают последний promise-вызов в цепочке. Такой подход все еще работает, это лишь приводит к тому, что сведения об ошибках, возникающих в цепочке, теряются, что скрывает возможные ошибки

Состояния, параметры, файлы и документы

Здесь читателю предлагается подробное описание особенностей обработки состояний приложения, методика работы с настраиваемыми пользователем параметрами приложения и с пользовательскими данными

Файлы к данной лекции Вы можете скачать здесь скачать: http://old.intuit.ru/department/se/uiwin8htmljs/2/2.zip.

Было бы очень интересно, если бы, когда вы путешествуете, каждый гостиничный номер, в котором вы останавливаетесь, автоматически подстраивался бы под ваши предпочтения. Чтобы там были подушки и простыни, которые вам нравятся, правильные стулья, ваша любимая еда в минибаре – вместо непристойно дорогих и невероятно маленьких банок с закусками. Если вы достаточно богаты, конечно, вы можете заранее отправить своих людей для того, чтобы они все подготовили, но подобное для множества из нас – лишь мечта.

Программное обеспечение, к счастью, не связано подобными ограничениями. Отправить своих людей в гостиницу перед своим приездом не означает необходимость бронирования для них авиабилетов, не нужно платить им зарплату и обеспечивать медицинскую страховку, не нужно делать вклады в их пенсионное обеспечение. Все, что нужно – подключение к Интернету, облачные сервисы, и вуаля! Все ваши предпочтения могут автоматически путешествовать вместе с вами. То есть – перемещаться между различными устройствами, которыми вы пользуетесь.

Этот опыт работы с перемещаемыми данными (roaming experience), как он называется, встроен в Windows 8 для параметров, оказывающих влияние на систему, таких, как изображение вашего профиля, установки начального экрана, избранные интернет-страницы, тема рабочего стола, сохраненные учетные данные и так далее. Когда вы используете учетную запись Microsoft для входа в Windows на доверенном компьютере, эти параметры безопасно сохраняются в облаке и автоматически перемещаются на другие доверенные устройства под управлением Windows 8, на которых вы используете ту же учетную запись. Я был приятно удивлен, когда в ходе разработки Windows 8 мне

не приходилось вручную переносить все эти данные, когда я обновлял предварительные релизы системы на своем компьютере!

Если подобное возможно для системных параметров, пользователи будут ожидать похожего поведения и от приложений. Они будут ожидать, что параметры приложения будут надлежащим образом перемещаться с одного устройства на другое. Я сказал "надлежащим образом", так как некоторые настройки не имеет смысла перемещать, особенно те, которые специфичны для аппаратного обеспечения устройства. С другой стороны, если я настраиваю почтовые учетные записи в приложении на одном компьютере, я, конечно, надеюсь на то, что они окажутся и на других! (Не могу точно сказать вам, сколько раз я настраивал четыре своих активных почтовых учетных записи в Outlook). В общем, как пользователь, я ожидаю, что мой переход между устройствами – как на уровне системы, так и на уровне приложений – будет прозрачным и гладким.

Таким образом, это означает, что каждое приложение должно выполнять свою работу по управлению собственным состоянием (включая – перемещение документов и других пользовательских данных посредством служб наподобие SkyDrive), и даже обеспечивать те виды кэширования, которые помогут увеличить производительность и сформировать хороший опыт взаимодействия пользователя и программы в условиях, когда устройство не подключено к Интернету. Как я уже говорил о многих подобных функциональных деталях, усилия, которые вы вкладываете в них, могут сыграть важную роль в том, как пользователи воспринимают ваше приложение и в том, какие оценки и отзывы о приложени они оставят в Магазине Windows.

Многие подобные параметры будут полностью внутренними по отношению к приложению, а другие пользователь может настраивать и обычно так и поступает. В прошлом это порождало сложные наборы из вложенных диалоговых окон с множеством закладок, каждая из котоых была украшена кнопками, выпадающими меню и большими взаимосвязаными группами флажков и переключателей. Как результат, в деле настройки приложений не было единообразия. Не было и постоянного места, где располагались настройки приложений. Это могли быть, кроме прочих, команды меню Инструменты > Параметры, или Правка > Предустановки, или Файл > Сведения

К счастью, те, кто проектировал Windows 8 поняли, что у большинства приложений есть какие-либо параметры, поэтому они включили чудо-кнопку Параметры в панель чудо-кнопок, вместе с другими, практически так же распространенными, кнопками для доступа к функциям поиска, для огранизации общего доступа к данным, для работы с устройствами. С одной стороны, это избавляет пользователей от необходимости помнить о том, где находятся настройки конкретного приложения. Разработчиков это избавляет от необходимости решать, как именно встроить подсистему настройки параметров в содержимое и навигационную иерархию приложения. Таким образом, помещенные в интерфейс, обеспечиваемый чудо-кнопкой Параметры, параметры эффективно устраняются из структуры содержимого приложения, что упрощает общую структуру приложения. Приложению лишь нужно предоставить различные страницы, которые отображаются, когда пользователь нажимает на чудо-кнопку Параметры.

Очевидно, данные приложения – параметры и внутренние состояния – лишь часть рассказа. Пользовательские данные (user data), такие, как документы, картинки, музыкальные и видео-файлы – тоже весьма важны. Для работы с ними мы ищем соответствующие возможности в манифесте приложения, позволяющие приложению работать с документами и медиа-библиотеками, со сменными носителями информации, позволяющие ему получать доступ к содержимому папок с помощью запросов. Мы решаем, как средство выбора файлов (file picker) позволит пользователю принять решение о доступе к другим безопасным областям файловой системы (но не к системным областям и папкам данных других приложений).

И здесь Windows 8, на самом деле, выводит нас за пределы локальной файловой системы. Подавляющее большинство данных, к которым имеет доступ современный пользователь, обычно расположены в сетевых хранилищах данных, а не в локальной файловой системе. Проблема здесь в том, что подобные данные обычно скрыты за специализированными API или веб-сервисами, что означает, что пользователь должен самостоятельно работать с веб-приложениями для просмотра данных, загружать и сохранять их в локальную файловую систему, и затем импортировать их в другие приложения. Видя такую модель взаимодействия, разрабочтики Windows 8 нашли другую возможность, которая представляет новый уровень интеграции и

единообразия, которая позволяет приложениям представлять данные, хранящиеся в различных сервисах так, что они выглядят для других приложений как часть локальной файловой системы. Подобное реализуется посредством контракта средства выбора файлов (file picker contract), предоставляя пользователям возможность бесшовного опыта взаимодействия с локальными и сетевыми данными. Здесь мы рассмотрим ситуацию с точки зрения пользователя, а в лекции 1 курса "Программная логика приложений для Windows 8, созданных с использованием HTML, CSS и JavaScript и их взаимодействие с системой", поговорим о провайдерах данных.

Короче говоря, управление состояниями и обеспечение доступа к данным пользователя, где бы они ни находились, это одна из самых ценных возможностей, которую может предоставить приложение, и она имеет большое значение в том, чтобы помочь пользователям почувствовать, что ваше приложение обращается с ними, считая их по-настоящему важными.

Рассказ о состояниях приложения

Продолжая сравнение, о котором мы говорили в начале лекции, когда мы приезжаем в новые места и останавливаемся в гостинице, большинство из нас признает тот факт, что им, по прибытии, придется потратить некоторое время на то, чтобы распаковать вещи и расставить все в комнате так, как нам нравится. С другой стороны, мы ожидаем полной противоположености от наших домов: мы ожидаем преемственности или сохранения состояния (statefulness). За последний год я дважды переезжал (один раз – во временное жилище, пока наш дом не был достроен), и я способен оценить достоинства сохранения состояния. Представьте себе, что все в вашем доме упаковывается в коробки каждый раз, когда вы уезжаете. В итоге вам приходится тратить несколько часов, дней или недель на то, чтобы все распаковать! Нет, дом – это место, в котором, по нашим ожиданиям, вещи остаются на своих местах, даже если мы на некоторое время уезжаем. Думаю, именно поэтому многие люди любят путешествовать в домах на колесах!

Приложения для Магазина Windows рассчитаны на такую вот неизменность состояния, что означает, что они поддерживают у

пользователя ощущение преемственности между сеансами работы, даже если приложение было приостановлено или остановлено между сеансами. Таким образом, приложения воспринимаются больше как дом, а не как временное пристанище. Они становятся местом, в которое пользователь приходит, чтобы отдохнуть среди данных, которые ему нужны. В итоге, чем меньше усилий ему потребуется, чтобы начать наслаждаться подобным опытом взаимодействия с программой – тем лучше.

Состояние приложения занимает центральное место в подобном опыте взаимодействия, так как срок их жизни гораздо дольше, чем срок жизни приложений. Состояния сохраняются, как и должны, когда приложение не исполняется, они так же могут сохраняться между различными версиями приложений. На самом деле, управление версиями состояния приложения осуществляется отдельно от управления версиями приложения.

Чтобы четко осознавать понятие "состояние приложения", вернемся ненадолго к пользовательским данным. Пользовательские данные, наподобие документов, изображений, музыкальных записей, видеофайлов, плейлистов и других подобных данных, создаются и используются приложением, но не зависят от приложения. Пользовательские данные подразумевают возможность работы с ними из разных приложений, и такие данные всегда остаются в системе, вне зависимости от существования приложения. Поэтому данные пользователя не являются частью состояния приложения. То есть, хотя приложение может запоминать пути (paths) к документам и другим файлам в составе списков избранных материалов или среди недавно использованных объектов реальное содержимое (content) этих файлов не является частью состояния приложения. Пользовательские данные, таким образом, не имеют прочной взаимосвязи с событиями жизненного цикла приложения. Их обычно сохраняют либо по непосредственной команде пользователя, либо, неявно, по событию наподобие `visibilitychange`, вместо использования такого события, как `suspending`. Опять же, приложение может запомнить список загруженных файлов как часть состояния сеанса работы при обработке события `suspending`, но содержимое файлов следует сохранить вне этого события, так как у вас есть лишь пять секунд на то, чтобы совершить все необходимые действия.

В отличие от пользовательских даных, данные приложения включают в себя все, что нужно для исполнения приложения и для поддержания состояния приложения между его запусками. Данными приложения, кроме того, управляют на уровне пользователя, они полностью привязаны к существованию конкретного приложения, и доступны лишь ему. Как мы уже видели ранее в этом курсе, данные приложения сохраняются в специальной папке пользователя, которая полностью удаляется из файловой системы при деинсталляции приложения. По этой причине никогда не храните в папке данных приложения что-либо, что может понадобиться пользователю за пределами приложения. Так же имеет смысл избегать хранения содержимого состояний приложения в библиотеках документов или медиаданных, так как эти данные не будут иметь для пользователя значения, если приложение будет удалено.

Данные приложения используются для управления следующими видами состояний:

- Состояние сеанса (Session state). Это состояние, которое приложение сохраняет при приостановке для восстановления его после возможной остановки. Сюда включаются данные форм, история навигации по страницам и так далее. Как мы видели в лекции 3 курса "Введение в разработку приложений для Windows 8 с использованием HTML, CSS и JavaScript", перезапуск после приостановки и последующей остановки – это единственный случай, в котором приложение восстанавливает состояние сеанса. Состояние сеанса обычно сохраняют инкрементно (при изменении состояния), либо при обработке событий `suspending` или `checkpoint`.
- Локальное состояние приложения (Local app state). Это – параметры, которые обычно загружаются при запуске приложения. Состояние приложения включает в себя кэшированные данные, сохраненные результаты поиска, списки недавно просмотренных элементов и различные параметры поведения приложения, которые отображаются в панели Параметры в виде настроек отображаемых элементов, предпочитаемых видеоформатов, настроек, зависящих от устройства и так далее. Локальное состояние приложения обычно сохраняется при его изменении, а не при обработке событий

жизненного цикла приложения.

- Перемещаемое состояния приложения (Roaming app state). Это состояние приложения, которое перемещается между экземплярами одного и того же приложения, исполняющимися на различных Windows 8 – устройствах, на которых пользователь выполнил вход. Это – списки избранного, позиция воспроизведения видео, параметры учетной записи, URI для важных файлов в облачном хранилище данных, возможно, некоторые сохраненные результаты поиска или запросы, и так далее. Так же, как локальном состоянием приложения, этим состоянием ожно управлять посредством панели Параметры. Перемещаемое состояние так же лучше всего сохранять при изменении значений. Больше об этом мы узнаем далее в этой лекции.

Есть два других компонента состояния приложени, которыми, на самом деле, управляют вне папки данных приложения или контейнеров параметров. Один из них – это список файлов, которые изначально получены посредство средства выбора файлов, к которым приложению может понадобиться программный доступ в будущем. Для подобных файлов недостаточно сохранить лишь путь – нужно сохранить и сведения о факте предоставления пользователем разрешения на доступ к ним с помощью средства выбора файлов. Это – цель использования API `Windows.Storage.AccessCache` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.storage.accesscache.asp и, в целом, является частью локального состояния приложения.

Второй компонент – это учетные данные, которые вы получили от пользователя и хотите использовать в будущем. Так как эти данные имеют критическую важность, в плане безопасности, приложению никогда не следует хранить их в составе собственных данных. Вместо этого используйте API хранилища учетных данных `Windows.Security.Credentials.PasswordVault` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.security.credentials.pass Содержимое хранилища будет перемещаться между доверенными ПК пользователя, и, таким образом, составлять часть перемещаемого состояния приложения. Больше об этом вы узнаете в лекции 3 курса "Программная логика приложений для Windows 8, созданных с использованием HTML, CSS и JavaScript и их взаимодействие с

системой".

Параметры и состояние приложения

Состояние приложения может не быть непосредственно представлено пользователю. Многие данные состояния приложения используются внутри него, или, наподобие истории навигации, могут отражать деятельность пользователя, но не быть представлены ему напрямую. Другие части состояния, наподобие предпочтений пользователя, учетных записей, изображений профиля и так далее, могут и должны быть представлены пользователю непосредственно, что является целью использования чудо-кнопки Параметры.

Состояние приложения может не быть непосредственно представлено пользователю. Многие данные состояния приложения используются внутри него, или, наподобие истории навигации, могут отражать деятельность пользователя, но не быть представлены ему напрямую. Другие части состояния, наподобие предпочтений пользователя, учетных записей, изображений профиля и так далее, могут и должны быть представлены пользователю непосредственно, что является целью использования чудо-кнопки Параметры.

В панели чудо-кнопки Параметры следует отображать те параметры приложения, которые влияют на приложение в целом и изменяются лишь изредка. Состояния, которые применимы только к отдельным страницам или рабочим процессам, не следует отображать в интерфейсе чудо-кнопки Параметры: их следует разместить прямо на странице (на полотне приложения) или в панели приложения, как мы видели в лекции 1. Все это по-прежнему входит в состав состояния приложения, подлежит соответствующему управлению, но не все подходит для чудо-кнопки Параметры.

Вот некоторые примеры хороших кандидатов на размещение в интерфейсе чудо-кнопки.

- Отображение предустановок, наподобие единиц измерения, цветовых тем, выравнивающих сеток и предустановок.
- Предпочтения, касающихся перемещаемых данных, которые

позволяют пользователю настроить перемещение данных приложения, такое, как, например, раздельное хранение настроек для рабочего и домашнего компьютеров.

- Настройки учетной записи и профиля, а так же – команды для входа в аккаунт и выхода от него, и для управления аккаунтами и профилями. Никогда не следует хранить пароль в составе локальных или перемещаемых состояний, используйте вместо этого Хранилище учетных данных.
- Параметры, влияющие на поведение приложения, наподобие оффлайнового и онлайнового режима, авто-обновления данных, интервалов обновления данных, предпочитаемого качества потокового аудио и видео. Здесь же находятс параметры передачи данных в сетях с лимитными тарифными планами, расположения, из которых приложению следует загружать данные и так далее.
- Форма или ссылка для обратной связи, с помощью которой вы можете получить от пользователя какую-то особую информацию.
- Дополнительные сведения о приложении, такие, как Справка, О программе, страница с информацией об авторских правах, заявление о конфиденциальности, лицензионное ограничение, условия использования. Часто подобные команды перенаправляют пользователя на специальный сайт, такой подход отлично подходит для реализации.

Я настоятельно рекомендую вам ознакомиться с приложениями, которые встроены в Windows и исследовать особенности использования ими чудо-кнопки Параметры. Так же вы можете изучить и то, как чудо-кнопка Параметры используется другими приложениями в Магазине Windows, однако, эти приложения не всегда следуют руководствам по проектированию приложений, которые декларируют единообразие интерфейса, а это очень важно для настройки параметров.

Если говорить об этом, то Windows автоматически предоставляет всем приложениям команды, которые называются Разрешения (Permissions) и Отзывы и оценки (Rate and Review). Команда Отзывы и оценки ведет пользователя к странице приложения в Магазине Windows, где пользователь может поставить приложению оценку и написать отзыв. Команда Разрешения, в свою очередь, позволяет пользователю контролировать доступ к критически важным ресурсам, таким, как определение местоположения, камера, микрофон и так далее. То, что

появляется в данном разделе, зависит от набора возможностей, объявленных в манифесте приложения, и это то место, где пользователь может отозвать то или иное разрешение или предоставить его приложению. Конечно, если приложение не использует подобных возможностей, команда Разрешения не отображается.

Вы можете обратить внимание на то, что я не упоинал отображение сведений об обновлениях приложения внутри интерфейса чудо-кнопки Параметры. В этом нет необходимости, так как уведомления об обновлениях обеспечиваются напрямую Магазином Windows. Это – еще один способ уменьшения количества ненужного содержимого, с которым ранее сталкивались пользователи, когда каждое приложение по-своему представляло информацию об обновлениях (и иногда это происходило слишком часто!).

Расположение данных приложения

Теперь, когда мы понимаем, какик виды информации составляют состояние приложения, следующий вопрос заключается в том, где все это хранится. Вы можете помнить, из Главы 1 курса "Введение в разработку приложений для Windows 8 с использованием HTML, CSS и JavaScript", что когда Windows устанавливает приложение для пользователя (и все приложения для Магазина Windows доступны только тому пользователю, который их устанавливает), она автоматически создает папки LocalState, TempState и RoamingState внутри папки AppData текущего пользователя. Те же папки удаляются при деинсталляции приложения. В файловой системе, если вы, с помощью Проводника, перейдете в папку %localappdata%\packages, вы увидите множество папок для различных приложений в вашей системе. Если вы зайдете в любую из них, вы увидите вышеописанные папки вместе с еще одной, которая называется "Settings", как показано на рис. 2.1 для встроенного в систему приложения Sports (Спорт). На рисунке так же показано различное содержимое этих папок.

Рис. 2.1. Папка AppData приложения Sports (Спорт) и ее содержимое

В папке LocalState на рис. 2.1 вы можете видеть файл с именем `_sessionState.json`. Это файл, где WinJS хранит, и откуда загружает содержимое объекта WinJS.Application.sessionState, как мы видели в лекции 3 курса "Введение в разработку приложений для Windows 8 с использованием HTML, CSS и JavaScript". Так как это обычный текстовый файл в формате JSON, вы можете просто открыть его в приложении Notepad (Блокнот) или в любом другом просмотрщике JSON для того, чтобы просмотреть его содержимое. Если вы взглянете на открытый файл приложения Sports (Спорт), который показан на рисунке, вы увидите что-то вроде {"lastSuspendTime":1340057531501}. Приложение Sports (Спорт) (вместе с приложениями News (Новости), Weather (Погода), и так далее), отображает содержимое, зависящее от времени, таким образом, это и другие приложения сохраняют время, когда они были приостановлены и проверяют прошедшее время при возобновлении. Если это время превышает интервал обновления данных, они могут запросить новые данные с сервиса, с которым связаны. В случае приложения Sports (Спорт), один из его параметров позволяет пользователю настроить интервал обновления данных.

Если ваше приложение использует любое API хранения данных HTML5, такое, как локальное хранилище, IndexedDB, и кэш приложения, их данные так же появятся в папке LocalState.

Примечание. Если вы внимательно посмотрите на рис. 2.1 вы увидите, что все папки с данными приложения, включая перемещаемые данные, находятся в папке пользователя AppData/Local. Есть и папка, родственная ей, AppData/Roaming, но она применяется лишь для параметров перемещаемых учетных записей пользователей в интранет-сети, как в случае, когда пользователь из присоединенного домена входит в систему с другого компьютера в корпоративной сети. Эта папка AppData/Roaming никак не связана с папкой AppData/Local…/RoamingState приложений для Магазина Windows.

Вы можете обращаться к этим расположениям различными программными способами. Во-первых, вы можете использовать URI-схему `ms-appdata:///`, как мы видели в лекции 3 курса "Введение в разработку приложений для Windows 8 с использованием HTML, CSS и JavaScript". Здесь `ms-appdata:///local`, `ms-appdata:///roaming`, и `ms-appdata:///temp` позволяют обратиться к отдельным папкам и их содержимому. (Обратите внимание на тройной слэш, что является сокращением, позволяющим вам опускать имя пакета приложения). Так же вы можете использовать объект, который возвращает метод `Windows.Storage.ApplicationData.current`, который содержит все необходимые API, которые нужны вам для работы с состоянием приложения.

Кстати, у вас могут быть данные только для чтения прямо в пакете приложения. С помощи URI вы можете просто использовать относительные пути, которые начинаются с /. Если вы хотите открыть и прочитать содержимое файла напрямую, вы можете использовать объект `StorageFolder` из свойства `Windows.ApplicationModel.Package.current.installec` Скоро мы вернемся к классу `StorageFolder`.

API AppData (WinRT и WinJS)

Когда вы запрашиваете у Windows свойство `Windows.Storage.ApplicationData.current`, вы получаете объект `Windows.Storage.ApplicationData` (ссылка:

http://msdn.microsoft.com/library/windows/apps/br241587.aspx), который полностью настроен для вашего приложения. Вот, что содержит этот объект:

- Свойства `localFolder`, `temporaryFolder`, и `roamingFolder`, каждое из которых является объектом `Windows.Storage.StorageFolder`, который позволяет вам создавать любые файлы и дополнительные структуры папок в соответствующих расположениях (но помните о параметре `roamingStorageQuota`, который описан ниже).
- Свойства `localSettings` и `roamingSettings`, которые являются объектами `Windows.Storage.ApplicationDataContainer` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.storage.applicatio и позволяют управлять иерархией контейнеров, состоящих из пар ключ-значение, или составными группами подобных пар значений. Все эти установки хранятся среди других данных приложения в папке Settings, в файле settings.dat.
- Свойство `roamingStorageQuota`, которое содержит сведения об объеме данных, в килобайтах, которые Windows автоматически перемещает для целей приложения (обычно – 100). Если общий объем данных, сохраненных в папках `roamingFolder` и `roamingSettings`, превышает указанный лимит, перемещение будет приостановлено до тех пор, пока объем данных не упадет ниже квоты. Вы должны контролировать объем хранимых данных, если вы полагаете, что их объем близок к лимиту.
- Событие `dataChanged` указывает на синхронизацию папок `roamingFolder` или `roamingSettings` с данными, хранящимися в облаке. По данному событию приложению следует повторно прочесть данные перемещаемого состояния. Это событие WinRT, для которого нужно использовать `removeEventListener`, как было описано в лекции 3 курса "Введение в разработку приложений для Windows 8 с использованием HTML, CSS и JavaScript".
- Метод `signalDataChanged` вызывает событие `dataChanged`. Это позволяет вам объединить обновления локальных и перемещаемых состояний в одном обработчике для

события dataChanged.

- Свойство version и метод setVersionAsync предназначены для управления отметками версий данных приложения. Версии применимы к полному объему данных – к локальным, временным и перемещаемым данным. Отдельные версии для разных данных не предусмотрены.
- Метод clearAsync позволяет очищать состояние всех папок AppData и контейнеров параметров. Используйте его, когда вы хотите повторно инициализировать состояние по умолчанию, что особенно полезно, когда вы перезапускаете приложение из-за нарушения данных его состояния.
- Метод clearAsync(<locality>) это вариант clearAsync, который ограничен одним расположением (локальным, временным, перемещаемым). Расположение идентифицируется с помощью значения из Windows.Storage.ApplicationDataLocality, такое, как Windows.Storage.ApplicationDataLocality.local. В случае с локальными и перемещаемыми данными, содержимое обеих папок и содержимого контейнеров очищается. Параметр temp воздействует только на папку TempState.

Теперь давайте посмотрим на то, как использовать это API для управления различными видами состояния приложения, что включает в себя некоторые вспомогательные функции WinRT, используемые для тех же целей.

Подсказка. API, которые работают с состоянием приложения, вызывают события в Средстве просмотра событий (Event Viewer), если вы включили эту возможность, как было описано в лекции 3 курса "Введение в разработку приложений для Windows 8 с использованием HTML, CSS и JavaScript". Убедитесь в том, что функция Вид > Отобразить аналитический и отладочный журналы (View > Show Analytic and Debug logs) включена. Затем пройдите в раздел Application and Services Log (Журналы приложения и служб), разверните раздел Microsoft/Windows/AppModelState, где вы найдете группы Debug (Отладка) и Diagnostic (Диагностика).

Контейнеры параметров

Для начинающих, давайте посмотрим на свойства `localSettings` и `roamingSettings`, которые обычно называют контейнерами параметров (settings containers). Вы работаете с ними с помощью API `ApplicationDataContainer`, которое устроено довольно просто. Каждый контейнер имеет четыре свойства только для чтения: `name` (строка), `locality` (значение из `Windows.Storage.ApplicationDataLocality`, здесь есть лишь значения `local` и `roaming`), и коллекции, которые называются `values` и `containers`.

Параметры контейнеров верхнего уровня имеют пустые имена. Свойство устанавливается на дочерние контейнеры, которые вы создаете с помощью метода `createContainer` (и удаляете с помощью `deleteContainer`). Эти дочерние контейнеры могут содержать другие контейнеры, что позволяет вам создавать иерархию настроек. Таким образом, эти контейнеры настроек предназначены для хранения небольших объемов данных, обычно – параметров пользователя, и отдельных параметров, объем которых ограничен 8 Кб, а так же любые наборы взаимосвязанных параметров (смотрите ниже) размером до 64 Кб. Учитывая эти ограничения, превышение объема в один мегабайт предусматривает довольно сложную иерархию настроек, которой сложно управлять, и доступ к которой осуществляется сравнительно медленно. Поэтому не поддавайтесь искушению воспринимать эти настройки как своего рода базу данных. Для этих целей гораздо лучше подходят иные механизмы, такие, как IndexedDB и SQLite, и с их помощью вы можете хранить любые объемы данных, так же, как и в папках AppData (помните об ограничении на объем перемещаемых данных, когда вы записываете данные в `roamingFolder`).

Для любого контейнера, который у вас есть, его коллекция `containers` – это объект `IMapView` (ссылка: http://msdn.microsoft.com/library/windows/apps/br226037.aspx), с помощью которого вы можете просматривать содержимое контейнера. Коллекция `values`, с другой стороны, это обычный массив (с технической точки зрения – объект `IPropertySet` в WinRT, который проецируется в

WinJS в виде массива с методами `IPropertySet`). Таким образом, хотя свойство `values` в любом контейнере – это свойство только для чтения, что означает, что вы не можете назначить ему какой-либо произвольный массив, вы можете управлять содержимым данного массива так, как вам нужно.

Мы можем увидеть это в примере "Данные приложения" (ссылка: http://code.msdn.microsoft.com/windowsapps/ApplicationData-sample-fb043eb2), в котором есть много примеров базовых операций с данными приложения. Сценарий 2, например, (js/settings.js) показывает простоту использование массива localSettings.value:

```
var localSettings = Windows.Storage.ApplicationData.current.localSettings;
var settingName = "exampleSetting";
var settingValue = "Hello World";

function settingsWriteSetting() {
localSettings.values[settingName] = settingValue;
}

function settingsDeleteSetting() {
localSettings.values.remove(settingName);
}
```

Многие параметры, похожие на те, которые показаны выше, это простые пары ключ-значение, но другие параметры могут быть объектами с множеством свойств. Это являет собой особую проблему: хотя вы можете, конечно, записывать и считывать отдельные свойства данного объекта внутри массива `values`, что случится, если в одном из них произойдет сбой? Это приведет к тому, что состояние приложения окажется поврежденным.

Для защиты от этого API для работы с данными приложения предоставляет возможность работы с взаимосвязанными параметрами (composite settings) как с неделимой единицей. (Повторюсь, каждый набор взаимосвязанных параметров ограничен размером в 64 Кб). Это похоже на идеальное групповое сознание: либо все преуспеют, либо всех ждет неудача и ничего кроме этих двух вариантов! Таким образом,

если возникает ошибка при чтении или записи любой части взаимосвязанных параметров, вся операция признается несостоявшейся. В случае с перемещаемыми данными, либо перемещается вся группа взаимосвязанных параметров, либо они не перемещаются вовсе.

Объект взаимосвязанных параметров создается с помощью `Windows.Storage.ApplicationDataCompositeValue` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.storage.applicationdatac как показано в Сценарии 4 примера работы с данными приложения (js/compositeSetting.js) нет такого сценария:

```
var roamingSettings = Windows.Storage.ApplicationData.current.roamingSetting
var settingName = "exampleCompositeSetting";
var settingName1 = "one";
var settingName2 = "hello";
function compositeSettingsWriteCompositeSetting() {
var composite = new Windows.Storage.ApplicationDataCompositeValue();
composite[settingName1] = 1; // пример значения
composite[settingName2] = "world"; // пример значения
roamingSettings.values[settingName] = composite;
}


function compositeSettingsDeleteCompositeSetting() {
roamingSettings.values.remove(settingName);
}


function compositeSettingsDisplayOutput() {
var composite = roamingSettings.values[settingName];
// ...
 }
```

Объект `ApplicationDataCompositeValue` имеет, как вы можете увидеть в документации, некоторые дополнительные методы и события для того, чтобы помочь в управлении им, это такие методы, как `clear`,

`insert` и `mapchanged`.

Взаимосвязанные параметры, во многом, напоминают контейнер параметров, однако, они не могут содержать другие контейнеры. Важно не путать их с обычными контейнерами. Дочерние контейнеры внутри параметров используются лишь для создания иерархии (посмотрите Сценарий 3 примера). Взаимосвязанные параметры, с другой стороны, специально созданы для создания более сложных наборов параметров, которые работают как неделимая единица, подобное поведение не гарантировано для обычных контейнеров параметров.

Как упомянуто выше, все эти параметры записываются в файл settings.dat в папке приложения Settings. Так же полезно знать, что изменения, которые вы вносите в контейнеры параметров, автоматически сохраняются, хотя здесь и присутствуют встроенные функции пакетной обработки данных для предотвращения чрезмерной дисковой активности, когда вы изменяете множество значений подряд. В любом случае, вам не нужно беспокоиться о том, когда вы сохраняете значения. Система берет управление этим процессом на себя.

Управление версиями состояния приложения

С точки зрения Windows, локальное, временное и перемещаемое состояние – это части единого целого, и все они имеют одну и ту же версию. Номер версии устанавливается посредством `Windows.Storage.ApplicationData.setVersionAsync` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.storage.applicationdata.s прочитать номер версии можно, воспользовавшись свойством только для чтения `Windows.Storage.ApplicationData.version`. Если хотите, вы можете поддерживать собственную систему назначения версий, с помощью отдельного файла или параметра приложения. Однако, я рекомендую, чтобы вы избегали подобного с перемещаемым состоянием, так как сложно предсказать, как Windows будет управлять синхронизацией слегка отличающихся структур. Даже в случае с локальным состоянием, попытка играть в сложные игры версий, на самом деле, может оказаться гораздо сложнее, чем кажется, и, вероятно, подобного лучше избегать вообще.

Версия данных вашего приложения и версия приложения – это разные вещи. На самом деле, они никак не связаны. В то время, как версия данных приложения устанавливается с помощью `setVersionAsync`, версией приложения управляют в разделе Упаковка (Packaging). У вас может быть приложение, которое от версии 1.0.0.0 до версии 4.3.9.3 использует данные приложения версии 1.0.0.0, или, может быть, версия приложения 1.2.1.9 переходит на версию 1.0.1.0. данных приложения, и версия 2.1.1.3 использует версию 1.2.0.0 данных приложения. На самом деле, это неважно, до тех пор, пока вы контролируете этот процесс и старые данные приложения могут переходить к новым версиями приложений!

Переход (migration) происходит при вызове `setVersionAsync`, где второй параметр является функцией, которая конвертирует данные. Эта функция принимает объект `SetVersionRequest` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.storage.setversionreques который содержит свойства `currentVersion` и `desiredVersion`, таким образом, показывая вашей функции, какого рода конверсия, на самом деле, нужна. В ответ, вам следует просмотреть все данные приложения и соответствующим образом осуществить перенос отдельных параметров и файлов. Как только вы вернетесь из обработчика конверсии, Windows решит, что переход завершен. Конечно, так как процесс часто задействует асинхронные файловые операции ввода-вывода, вы можете использовать отложенный механизм, наподобие того, который мы видели при активации приложения. Вызовите метод `SetVersionRequest.getDeferral` для того, чтобы получить объект для выполнения отложенных действий (`SetVersionDeferral`), и вызовите его метод complete, когда все асинхронные операции будут завершены. Примеры подобного можно найти в Сценарии 9 примера "Данные приложения" (ссылка: http://code.msdn.microsoft.com/windowsapps/ApplicationData-sample-fb043eb2).

Можно осуществлять перенос данных приложения и тогда, когда устанавливается новое обновление приложения. Для этого пользуются фоновой задачей для вызова `sevicingComplete`. Смотрите Главу 2 курса "Программная логика приложений для Windows 8, созданных с использованием HTML, CSS и JavaScript и их взаимодействие с системой", в особенности – раздел "Фоновые задачи и приложения на

экране блокировки" ближе к концу.

Управление папками и файлами

Так как часто очень удобно сохранять данные приложения напрямую в файл, настало время для того, чтобы внимательно рассмотреть файловые API приложений для Магазина Windows.

Для начала, однако, взглянем на другие API, такие, как URL.createObjectURL – работающее с тем, что мы называем большим двоичным объектом (blob), делая возможным выполнять множество действий в приложения без необходимости спускаться на уровень файлового ввода/вывода. Мы уже видели, как этим пользоваться, устанавливая свойство `src` для элемента `img`, тот же подход работает и для других элементов, таких, как `audio`, `video` и `iframe`. Файловые операции ввода-вывода, связанные с подобными действиями, инкапсулируются в `createObjectURL`. Есть и другие способы использовать большие двоичные объекты, такой, как конверсия элемента `canvas` с помощью команды `canvas.msToBlob` в то, что можно присвоить элементу `img`, и получение двоичных данных большого размера от `WinJS.xhr`, сохраняя их в файл и затем задавая этот файл в качестве источника для `img`. Больше об этом вы можете узнать в лекции 5, так же вы можете рассмотреть пример "Использование больших двоичных объектов для сохранения и загрузки содержимого" (ссылка: http://code.msdn.microsoft.com/windowsapps/Blob-Sample-0e35889e).

Для того чтобы работать напрямую с файлами, рассмотрим то, что имеется в нашем распоряжении, использовав конкретные образцы из примера "Доступ к файлам" (ссылка: http://code.msdn.microsoft.com/windowsapps/File-access-sample-d723e597).

Основные API WinRT для работы с файлами расположены в пространстве имен `Windows.Storage`. Ключевую роль играют классы StorageFolder (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.storage.storagefolder.as и StorageFile (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.storage.storagefile.aspx)

И на то и на другое иногда ссылаются как на "элементы хранения", так как и тот и другой класс унаследованы от IStorageItem (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.storage.istorageitem.asp и имеют одинаковые свойства, такие, как `name`, `path`, `dateCreated` и `attributes`, а так же методы `deleteAsync` и `renameAsync`.

Файловые операции ввода-вывода в WinRT практически всегда начинаются с получения объекта `StorageFolder` посредством одного из нижеперечисленных способов. В некоторых случаях вы так же можете получить объект `StorageFile` напрямую:

- Свойство `Windows.ApplicationModel.Package.current.insta` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.applicationmodel. получает `StorageFolder`, посредством которого вы можете загружать данные из файлов, которые находятся в пакете приложения (все файлы – только для чтения).
- Свойства `Windows.Storage.ApplicationData.current.localF roamingFolder`, или `temporaryFolder` обеспечивают объекты `StorageFolder` для различных расположений данных приложения (для чтения и записи).
- Приложение может позволить пользователю выбирать папку или файл напрямую, используя средство выбора файлов, запущенное посредством `Windows.Storage.Pickers.FolderPicker` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.storage.pickers.fc а так же `FileOpenPicker` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.storage.pickers.fi и `FileSavePicker` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.storage.pickers.fi Для приложений это – предпочтительный способ, используя который нет необходимости в просмотре содержимого библиотеки (смотрите следующий пункт списка). Это так же единственное средство, с помощью которого приложение может получить доступ к безопасным (несистемным) областям файловой системы без необходимости объявления дополнительных возможностей в

манифесте.

- Объект `Windows.Storage.KnownFolders` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.storage.knownfol предоставляет объекты `StorageFolder` для библиотек "Документы", "Музыка", "Видео", так же, как и для доступа к съемным устройствам. Задавая соответствующие возможности в манифесте, вы можете работать с содержимых этих папок. (Попытка получить доступ к папке без объявления соответствующих возможностей приведет к выдаче исключения "Доступ запрещен").
- Объект `Windows.Storage.DownloadsFolder` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.storage.downloac предоставляет метод `createFolderAsync`, посредством которого вы можете получить `StorageFolder` для папки "Загрузки". Он так же предоставляет метод `createFileAsync` для непосредственного создания объектов `StorageFile`. Данное API следует использовать, если ваше приложение управляет загруженными файлами напрямую. Обратите внимание на то, что `DownloadFolder` предоставляет лишь эти два метода, она не является `StorageFolder`.
- Статический метод `Windows.Storage.StorageFolder.getFolderFromPat` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.storage.storagefo возвращает `StorageFolder` для заданного пути, если и только если у вашего приложения уже есть разрешение на доступ к данному расположению. В противном случае вы получите исключение "Доступ запрещен". Похожий статический метод существует и для работы с файлами, он называется `Windows.Storage.StorageFile.getFileFromPathAsy` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.storage.storagefil к нему применимы те же ограничения. `Windows.Storage.StorageFile.getFileFromApplica` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.storage.storagefil открывает файлы с помощью URI `ms-appx:// (package)` и `ms-appdata:///` . Другие схемы не поддерживаются.

- Как только объект, соответствующий папке или файлу, получен, его можно сохранить в кэше `Windows.Storage.AccessCache` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.storage.accessca который позволяет приложению получать эти объекты позже, с теми же программными разрешениями. Это нужно, преимущественно, для файлов или папок, выбранным с помощью средств выбора, так как разрешения на доступ к элементу хранения выдаются лишь на период существования данного объекта в памяти. Всегда следует использовать данное API, как показано в Сценарии 6 примера о доступе к файлам, где вы принимаете решение о сохранении пути к файлу. Опять же, `StorageFolder.getFolderFromPathAsync` и `StorageFile.getFileFromPathAsync` выдадут исключение "Доступ запрещен", если они указывают на любое расположение, для доступа к которому у вас уже нет разрешения. Пути к файлам, так же, не будут работать для файлов, предоставленных другими приложениями через средство выбора файлов, так как объект `StorageFile`, на самом деле, может не указывать на файл, который реально существует в файловой системе.

Как только у вас есть `StorageFolder`, вы можете выполнять с ним вполне ожидаемые действия: получать свойства папки (в том числе – миниатюру), создавать и/или открывать файлы и папки, просматривать содержимое папки. В случае с последним действием, существует, конечно, API для получения списка содержимого папки (смотрите метод `getItemsAsync`), но чаще нужно получить ограниченный список содержимого, отобранного на основе определенных критериев, вместе с миниатюрами и другими индексированными метаданными файлов (музыкальные альбомы и сведения о записях, подписи к изображениям, теги и так далее), которые вы можете использовать для группировки и организации этих файлов. Это – цель запросов (queries), направленных на файлы, папки или элементы хранения (файлы + папки), которыми можно управлять посредством методов `createFileQuery[WithOptions]`, `createFolderQuery[WithOptions]`, и `createItemQuery[WithOptions]`. Мы уже видели некоторые

примеры этих механизмов в приложении FlipView, которые мы построили с использованием Библиотеки "Изображения" в лекции 5 курса "Введение в разработку приложений для Windows 8 с использованием HTML, CSS и JavaScript", и мы вернемся к этой теме в контексте пользовательских данных, основного сценария для использования запросов, в конце этой лекции[1].

Совет. Существуют некоторые расширения имен файлов, таких, как .lnk, .url и другие, которые зарезервированы системой и не могут быть перечислены. Их полный список можно найти в материале "Обработка активации файлов" (ссылка: http://msdn.microsoft.com/library/windows/apps/hh452684.aspx). Кроме того обратите внимание на то, что доступ к UNC-путям требует объявления возможностей Частные сети (клиент и сервер) (Private Networks (ClientServer)) и Корпоративная аутентификация (Enterprise Authentication) в манифесте вместе с объявлением типов файлов, к которым вы хотите получить доступ.

С помощью любого `StorageFolder`, в особенности для расположений, соответствующих данным приложения, вы можете создать любую необходимую структуру папок с помощью методов `createFolderAsync/getFolderAsync`, которые дают вам больше объектов `StorageFolder`. В любой из этих папок вы затем используете методы `createFileAsync/getFileAsync` для доступа к отдельным файлам, каждый из которых будет представлен в виде объекта `StorageFile`.

Каждый `StorageFile` (ссылка: http://msdn.microsoft.com/library/windows/apps/br227171.aspx) обеспечивает соответствующие свойства наподобие `name`, `path`, `dateCreated`, `fileType`, `contentType`, и `attributes`, конечно, вместе с методами наподобие `getThumbnailAsync`, `copyAsync`, `deleteAsync`, `moveAsync`, `moveAndReplaceAsync`, и `renameAsync` для управления файлами. Файл можно открыть разными способами, зависящими от того, доступ какого рода вам нужен:

- Методы `openAsync` и `openReadAsync` предоставляют потоки

для произвольного побайтового доступа для чтения и записи. Потоки – это объекты IRandomAccessStream (ссылка: http://msdn.microsoft.com/library/windows/apps/hh438400.aspx) и IRandomAccessStreamWithContentType (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.storage.streams.ir соответственно, оба находящиеся в пространстве имен Windows.Storage.Streams. Первый из них работает с исходным двоичным потоком. Второй работает с данными, используя их тип, что может быть нужно для работы с HTTP-ответом, который добавляет в поток данных сведения о типе содержимого.

- Метод openSequentialReadAsync предоставляет объект только для чтения Windows.Storage.Streams.IInputStream (ссылка: http://msdn.microsoft.com/library/windows/apps/br241718.aspx), посредством которого можно читать содержимое файла блоками байтов, но нельзя возвращаться к ранее прочитанной области. Данный метод следует использовать во всех случаях, когда вам просто нужно прочесть данные из потока, так как он обладает лучшей производительностью, чем поток для произвольного доступа (источник можно оптимизировать для последовательного чтения данных).
- Метод openTransactedWriteAsync предоставляет объект Windows.Storage.StorageStreamTransaction (ссылка: http://msdn.microsoft.com/library/windows/apps/hh996767.aspx), который является вспомогательным объектом, построенным на основе IRandomAccessStream , который имеет методы commitAsync и close для обработки транзакций. Это необходимо при сохранении данных со сложной структурой, для того, чтобы быть уверенным в том, что вся операция записи произошла атомарным образом, и файлы не были повреждены при ее прерывании. Сценарий 4 в примере о доступе к файлам показывает этот подход.

Класс StorageFile так же предоставляет два статических метода: createStreamedFileAsync и createStreamedFileFromUriAsync. Результатом их работы является объект StorageFile, который обычно передают другим

приложениям посредством контракта, как мы увидим в лекции 1 курса "Программная логика приложений для Windows 8, созданных с использованием HTML, CSS и JavaScript и их взаимодействие с системой". Полезность этих методов заключается в том, что доступ к файлу, лежащему в их основе, не осуществляется до тех пор, пока его данные не будут запрошены в первый раз, если такой запрос вообще произойдет.

Соберем теперь все это воедино. Вот небольшой фрагмент кода, использующий исходные API, которые мы рассматривали, для создания и открытия файла "data.tmp" в папке временных данных приложения, расположенной в папке AppData, и для записи в него заданной строки. Это – код из упражнения RawWriteFile к этой лекции. Позвольте мне пояснить, что то, что здесь показано, использует низший уровень API в WinRT для этих целей, и это не то, чем обычно пользуются, как мы увидим в следующем разделе. Тем не менее, это полезно знать, так как бывают случаи, когда нужно использовать нечто подобное:

```
var fileContents = "Congratulations, you're written data to a temp file!";
writeTempFileRaw("data.tmp", fileContents);



function writeTempFileRaw(filename, contents) {
var tempFolder = Windows.Storage.ApplicationData.current.temporaryFolder;
var outputStream;

//Ей-богу!
tempFolder.createFileAsync(filename, Windows.Storage.CreationCollisionOptio
.then(function (file) {
return file.openAsync(Windows.Storage.FileAccessMode.readWrite);
}).then(function (stream) {
outputStream = stream.getOutputStreamAt(0);
var writer = new Windows.Storage.Streams.DataWriter(outputStream);
writer.writeString(contents);
return writer.storeAsync();
}).done();
}
```

Хорошо, что недавно мы узнали об асинхронных операциях, объединенных в цепочку! Для начала мы создаем или открываем заданный файл в папке `temporaryFolder` среди данных приложения (`createFileAsync`), затем получаем выходной поток для файла (`openAsync` и `getOutputStream`). Затем мы создаем на основе потока объект `DataWriter`, записываем в файл данные (`writeString`) и убеждаемся, что информация сохранена в файле (`storeAsync`).

Но вы скажете: "Вы, должно быть, шутите! Четыре объединенных в цепочку асинхронных операции только для того, чтобы записать обычную строку в файл! Кто разрабатывал это API?" Действительно, когда мы начинали создавать самое первое приложение для Магазина Windows в Microsoft, у нас было лишь это, и мы задавали себе те же вопросы. В конце концов, выполнение некоторых ожидаемо простых файловых операций ввода-вывода – это обычно первое, что добавляют в приложение "Hello World", и это в данном случае совсем не просто. Что еще хуже, тогда у нас не было promise-объектов для асинхронных операций в JavaScript, поэтому мы должны были написать это на основе исходных вложенных операций. Такие были времена.

К счастью, более простые API уже были доступны, и многое было сделано уже после этого. Это те API, которые вы обычно будете использовать, при работе с файлами, и это мы рассмотрим в следующем разделе. Тем не менее, важно понимать структуру вышеприведенного низкоуровневого кода, так как класс `Window.Storage.Streams.DataWriter` (ссылка: http://msdn.microsoft.com/library/windows/apps/br208154.aspx) и родственный ему `DataReader` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.storage.streams.datarea – это очень важные механизмы для работы с множеством различных потоков ввода-вывода, и они необходимы для процессов кодирования данных. Контроль над мелкими деталями, кроме того, позволяет реализовывать сценарии, когда различные компоненты вашего приложения вносят собственный вклад в структуру файла. Поэтому хорошо будет взглянуть на документацию по ним, и на пример "Чтение и запись данных" (ссылка: http://code.msdn.microsoft.com/windowsapps/Reading-and-writing-data-75ea10a3) лишь для того, чтобы вы были знакомы с их возможностями.

Врезка: Закрытие потоков против закрытия файлов

Разработчики, которые работали ранее с API ввода-вывода иногда спрашивают, почему у объекта `StorageFile` нет чего-то вроде метода `close`. Причина в том, что `StorageFile` представляет собой объект, соответствующий файлу, но не поток данных, посредством которого можно получать доступ к содержимому файла. Таким образом, когда вы вызываете методы вроде `StorageFile.openAsync` для получения потока, при этом файл открывается, и файл закрывается только при закрытии потока посредством его метода `close`.

В вышеприведенном коде вы не увидите подобного вызова, так как и `DataReader`, и `DataWriter` заботятся о подобных деталях для вас, когда такие вызовы опущены. Однако если вы отделите поток от этих объектов посредством их методов `detachStream`, вы ответственны за вызов метода потока `close` .

При разработке приложений, которые осуществляют запись данных в файлы, если вы сталкиваетесь с ошибками, указывающими на то, что файл все еще открыт, проверьте, закрыли ли вы соответствующие потоки.

FileIO, PathIO и вспомогательные классы WinJS (а так же FileReader)

Простота – это очень хорошо, если речь идет о файловых операциях ввода-вывода, и дизайнеры WinRT убедились в том, что наиболее распространенные сценарии не нуждаются в длинных цепочках асинхронных операций, которые мы видели в предыдущем разделе. Классы `Windows.Storage.FileIO` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.storage.fileio.aspx) и `PathIO` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.storage.pathio.aspx) предоставляют усовершенствованный интерфейс, а разница между ними заключается в том, что методы `FileIO` принимают параметр типа `StorageFile`, а методы `PathIO` принимают строковое имя файла. В остальном они имеют одинаковые методы, в частности,

`[read | write]BufferAsyn` с (работают с байтовыми массивами), `[append | read | write]LinesAsync` (работают со строковыми массивами),и `[append | read | write]TextAsync` (работают с отдельными строками). В последнем случае класс `WinJS.IOHelper` предоставляет даже более простой интерфейс посредством своих методов `readText` и `writeText`.

Посмотрим, как все это работает, начнем с нескольких образцов кода из примера "Доступ к файлу" (ссылка: http://code.msdn.microsoft.com/windowsapps/File-access-sample-d723e597). Сценарий 2 демонстрирует запись текстовой строки из элемента управления в файл (этот код упрощен, для большей понятности):

```
var userContent = textArea.innerText;

//sampleFile создается при старте с помощью Windows.Storage.KnownFold
Windows.Storage.FileIO.writeTextAsync(sampleFile, userContent).done(functio
outputDiv.innerHTML = "The following text was written to '" + sampleFile.name
+ "':<br /><br />" + userContent;
});
```

Для сравнения с кодом из предыдущего раздела, мы можем заменить все, что касается потоков и объекта DataWriter, одной строкой кода:

```
tempFolder.createFileAsync(filename, Windows.Storage.CreationCollisionOptio
.then(function (file) {
Windows.Storage.FileIO.writeTextAsync(file, contents).done();
})
```

Чтобы сделать это еще проще, объект `WinJS.Application.temp` (`WinJS.Application.IOHelper` (ссылка: http://msdn.microsoft.com/library/windows/apps/br211764.aspx)) сокращает все до одной строки (которая является асинхронной операцией и возвращает promise-объект:

```
WinJS.Application.temp.writeText(file, contents);
```

Чтение текста посредством асинхронного метода `readText` так же

просто, и WinJS предоставляет тот же интерфейс для папок `local` и `roaming`, вместе с двумя другими методами – `exists` и `remove`[2] . Тем не менее, эти вспомогательные функции WinJS доступны лишь для папок в AppData, но не для других областей файловой системы. Для работы с другими областями следует использовать классы `FileIO` и `PathIO`.

Кроме того, вы можете пользоваться классом HTML5 `FileReader`, который доступен приложениям для Магазина Windows, и который является частью спецификации "W3C File API" (ссылка: http://www.w3.org/TR/FileAPI/#dfn-filereader). Как следует из его названия (reader), он подходит только для чтения файлов и не обладает возможностью записи в них, но одно из его преимущества заключается в том, что он может работать не только с файлами, но и с большими двоичными объектами. Некоторые образцы подобных действий можно найти в примере "Использование больших двоичных объектов для сохранения и загрузки содержимого" (ссылка: http://code.msdn.microsoft.com/windowsapps/Blob-Sample-0e35889e).

Шифрование и сжатие

WinRT предоставляет две возможности, которые может быть весьма полезны для управления состоянием приложения: шифрование и сжатие данных.

Возможности шифрования предоставляют API `Windows.Security.Cryptography` (ссылка: http://msdn.microsoft.com/library/windows/apps/br241404.aspx) и `Windows.Security.Cryptography.Core` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.security.cryptography.cc Первое содержит методы для базовых операций кодирования и декодирования (для форматов base64, hex, text). Второе занимается шифрованием в соответствии с различными алгоритмами. Как показано в примере "Сохранение секретных данных с помощью шифрования" (ссылка: http://code.msdn.microsoft.com/windowsapps/Secret-Saver-f8a69623), данные обычно кодируют каким-либо образом с использованием метода `Windows.Security.Cryptography.CryptographicBuffer.`

а затем создают или получают алгоритм и передают его вместе с буфером данных методу `Windows.Security.Cryptography.Core.CryptographicEr` Другие методы, такие как `decrypt` и `convertBinaryToString`, выполняют обратные операции.

Механизмы сжатия выглядят немного проще, их единственная цель – предоставить встроенное API, с помощью которого можно уменьшить объем данных (например, для уменьшения размера перемещаемых данных). API для этих целей находится в `Windows.Storage.Compression` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.storage.compression.asp оно включает в себя классы `Compressor` и `Decompressor`, и тот и другой показаны в примере "Сжатие данных" (ссылка: http://code.msdn.microsoft.com/windowsapps/Compression-sample-9d57900f). Хотя это API может использовать различные алгоритмы сжатия, включая алгоритм под названием MSZIP, оно не предоставляет средств для управления .ZIP-файлами и их содержимым. Для подобных целей вам понадобится либо использовать JavaScript-библиотеки сторонних производителей, либо написать WinRT-компонент на C# или VisualBasic, который может использовать API `System.IO.Compression` (смотрите Главу 5 курса "Программная логика приложений для Windows 8, созданных с использованием HTML, CSS и JavaScript и их взаимодействие с системой").

Использование API данных приложения для управления состоянием

Сейчас, когда мы ознакомились с сущностью API, посмотрим, как их можно использовать для работы с различными видами данных приложения, а так же рассмотрим специальные соглашения, которые применимы в данной работе

Состояние сеанса

Как было описано выше, состояние сеанса это то, что приложение сохраняет при приостановке, и, таким образом, может восстановить

свое состояние после того, как было остановлено системой и запущено снова. Единственный вариант развития подобных событий – это остановка приложения системой, поэтому все, что включают в состояние сеанса, всегда должно быть рассчитано на то, чтобы давать пользователю иллюзию непрерывной работы приложения. В некоторых случаях, как описано в лекции курса "Введение в разработку приложений для Windows 8 с использованием HTML, CSS и JavaScript", вы можете, на самом деле, не восстанавливать это состояние, особенно, если приложение было остановлено в течение длительного времени и маловероятно, чтобы пользователь помнил, как все выглядело во время последнего сеанса работы. Это решение вам нужно будет принять для собственного приложения, от него зависит опыт взаимодействия пользователя и приложения.

Состояние сеанса следует сохранять в папке данных приложения `localFolder`, или в объекте `localSettings`. Не следует сохранять его в области временных данных, так как пользователь может запустить средство очистки диска в то время, когда приложение приостановлено или остановлено, и в таком случае данне состояния сеанса будут удалены (смотрите следующий раздел).

Объект WinJS `sessionState` самостоятельно создает файл, который называется _sessionState.json внутри `localFolder`. В этом файле хранится обычный текст в формате JSON, поэтому вы можете в любое время его просмотреть. Вы можете записать переменные сеанса работы в объект `sessionState`, в любое время, когда они изменяются, используя `sessionState` как пространство имен для этих переменных, и именно так следует поступать. При таком подходе эти значения сохраняются и перезагружаются автоматически, без необходимости управлять этими переменными где-либо еще.

Если вам нужно сохранить дополнительные значения внутри `sessionState`, прежде чем данные будут записаны, сделайте это в обработчике `WinJS.Application.oncheckpoint`. Хороший пример подобных данных – это навигационный стек элементов управления страниц, который доступен посредством `WinJS.Navigation.history`. Так же вы можете скопировать эти данные в `sessionState` внутри метода `PageControlNavigation.navigated` (он находится в

navigation.js и предоставляется шаблонами проектов). В любом случае, у WinJS есть собственный обработчик `checkpoint`, который всегда вызывается последним (после вашего обработчика) для того, чтобы убедиться, что любые изменения, которые вы внесли в `sessionState` после этого события, были сохранены.

Если вы не используете объект WinJS `sessionState` , а просто используете API WinRT для работы с данными приложения, вы можете сохранять состояние сеанса в любое время (в том числе – в `checkpoint`), и вам понадобится восстанавливать их в событии активации при выполнении условия `previousExecutionState == terminated` .

Кроме того, хорошей практикой является обеспечение некоторой устойчивости в обработке состояния сеанса: если загруженные данные кажутся неподходящими или имеются другие проблемы, можно вернуться к данным сеанса по умолчанию. Помните так же, что вы можете использовать контейнер `localSettings` с взаимосвязанными параметрами, что гарантирует то, что группа значений будет сохраняться и извлекаться как единое целое. Так же может оказаться полезным при разработке иметь команду, которая дает вам возможность очищать состояние сеанса в том случае, если оно заполняется ненужными данными, но, в то же время, обычная деинсталляция приложения так же очищает состояние сеанса. В то же время, нет необходимости предоставлять пользователям команду для очищения состояния приложения: если ваше приложение не может запуститься после завершения, флаг `previousExecutionState` будет иметь состояние `notRunning` , когда пользователь в следующий раз попытается запустить приложение, в подобной ситуации вы не будете пытаться восстановить состояние.

Кроме того, нет необходимости включать в состояние сеанса номер версии. Если пользователь установил обновление в то время, когда ваше приложение было приостановлено и остановлено, и версия данных приложения была изменена, значение previousExecutionState будет сброшено. Если по каким-либо причинам вы не изменяете версию данных приложения – например, если обновления весьма незначительны, тогда ранее сохраненные данные состояния приложения могут использоваться дальше. Но в данном случае перед

нами – то же самое приложение, поэтому управление версиями состояния сеанса не требуется.

Врезка: использование sessionStorage и localStorage HTML5

Если хотите, вы можете использовать объект HTML5 `localStorage` для хранения данных состояния сеанса приложения и других данных приложения. Содержимое объекта хранится в папке `localFolder` . Содержимое `localStorage` не загружается до первой попытки доступа к нему и ограничено объемом в 10 Мб на приложение. API WinRT и WinJS, с другой стороны, ограничены лишь емкостью файловой системы.

Что касается объекта HTML5 `sessionStorage` , он, на самом деле, не нужен, когда вы используете элементы управления страниц и поддерживаете общий контекст скрипта для этих страниц, так как переменные, находящиеся в памяти уже выполняют нужную функцию. Однако, если вы изменяете контекст страниц, используете ссылки `<a>` или `document.location` для целей навигации, `sessionStorage` способен быть полезным. Кроме того, вы можете кодировать информацию в URI, как обычно делается в веб-приложениях.

И `sessionStorage` , и `localStorage` полезны внутри страниц `iframe` , исполняющихся в веб-контексте, так как API WinRT недоступны. В то же время, вы можете загрузить WinJS в веб-контекст (такая возможность поддерживается) и объекты `WinJS.Application.local` , `roaming` , и `temp` будут работать, используя буферы для данных, расположенные в памяти, вместо файловой системы.

Локальное и временное состояние

В отличие от состояния сеанса, которое восстанавливается лишь в особых обстоятельствах, локальное состояние приложения, которое состоит из этих параметров и других данных, всегда применяется при запуске приложения. Сюда попадает все, что пользователь может

настроить, несмотря на то, что это так же часть перемещаемых данных, в таком случае эти данные так же загружаются при запуске приложения. Любые другие кэшированные данные, сохраненные результаты поиска, недавно использованные элементы, экранные элементы, предпочитаемые медиа-форматы, настройки, зависящие от устройства, так же попадают сюда. Коротко говоря, если какие-то данные не являются чистым состоянием сеанса, либо – частью перемещаемого состояния, это – либо локальное, либо временное состояние приложения. (Помните, что учетные данные следует хранить в Хранилище учетных данных вместо хранения их среди данных приложения).

Те же API, которые мы видели, работают и для этих видов состояния, включая все API WinRT, объекты `WinJS.Application.local` и `temp` , и HTML `localStorage` . Так же вы можете использовать HTML5 API IndexedDB, SQLite, и HML App Cashe – это лишь другие формы локальных данных приложения.

При работе с локальными и временными данными очень важно назначать им номер версии, так как они сохраняются и при обновлении приложения (хотя временное состояние в это время очищается). В процессе любого обновления приложения будьте готовы к тому, чтобы загрузить старую версию состояния приложения и выполнить необходимые обновления, или просто решить, что версия слишком стара и очистить данные (`Windows.Storage.ApplicationData.current.clearAsyn`), прежде чем задавать новые значения по умолчанию. Как упоминалось выше, возможно осуществить перенос состояния с использованием фоновой задачи (смотрите Главу 2 курса "Программная логика приложений для Windows 8, созданных с использованием HTML, CSS и JavaScript и их взаимодействие с системой").

Вообще говоря, локальные и временные данные приложения – это одно и то же – они имеют одинаковые API, и хранятся в папках, расположенных в одном и том же месте. Временные данные, однако, не поддерживают параметры и контейнеры параметров. Другое различие заключается в том, что содержимое в папке временных данных (вместе с HTML5-кэшем приложения) могут подвергнуться воздействию средства очистки диска Windows. Это означает, что временные данные

могут исчезнуть в любой момент, когда пользователь захочет освободить немного дискового пространства. Так же вы можете использовать фоновую задачу с триггером обслуживания для самостоятельного выполнения очистки (опять же, смотрите Главу 2 курса "Программная логика приложений для Windows 8, созданных с использованием HTML, CSS и JavaScript и их взаимодействие с системой").

По этим причинам, временные данные следует использовать для хранения информации, которая оптимизирует производительность приложения, но не для данных, которые критически важны для работы приложения. Например, если в пакете приложения имеется JSON-файл, который вы обрабатываете или распаковываете при первом запуске программы, чтобы в дальнейшем приложение работало быстрее, и вы не выполняете никаких изменений этих данных из приложения, вы можете решить хранить это среди временных данных. То же самое справедливо для графических ресурсов, которые вы можете обработать для применения на конкретном устройстве, на котором исполняется приложение. Вы всегда можете повторить этот процесс, воспользовавшись исходным источником, таким образом, эти данные – так же хороший кандидат для хранения среди временных данных. Похожим образом, если вы получили данные из онлайнового сервиса для целей оптимизации производительности (таким образом, вы можете обновлять локальные данные инкрементно), вы всегда можете запросить их снова. Это особенно полезно для обеспечения возможности работы без подключения к сети, хотя в некоторых случаях может понадобиться предоставить пользователям выбрать, следует ли сохранять эту информацию среди локальных данных приложения или среди временных (параметр, который может появиться в интерфейсе чудо-кнопки Параметры вместе с командой, которая позволяет очистить кэш).

Врезка: AppCache HTML5

Приложения для Магазина Windows могут использовать кэш приложения HTML5 как часть стратегии кэширования и работы в оффлайновом режиме. Это особенно полезно в элементах веб-контекста iframe, где этот механизм можно использовать для любого содержимого. Например, приложение для чтения книг в онлайн-режиме может

показывать содержимое в iframe, и если страницы включают теги кэша приложения, они будут сохраняться и будут доступны в режиме, когда сетевое соединение недоступно. В локальном контексте, кэш приложения работает для неисполняемых ресурсов, таких, как изображения, аудио, видео, но не для HTML или JavaScript.

IndexedDB и другие технологии баз данных

Многие виды локальных данных приложения прекрасно подходят для хранения в базе данных. В приложениях для Магазина Windows API IndexedDB доступно посредством объектов `window.indexedDB` и `worker.indexedDB` . Подробности об этом вы можете узнать в спецификациях W3C (ссылка: https://dvcs.w3.org/hg/IndexedDB/), в справочных материалах по Indexed Database API (ссылка: http://msdn.microsoft.com/library/windows/apps/hh466139.aspx) и в примере "IndexedDB" (ссылка: http://code.msdn.microsoft.com/windowsapps/IndexedDB-sample-eb1e95af).

Хотя база данных IndexedDB хранится среди локальных данных приложения, знайте, что существуют некоторые ограничения, так как нет средств, с помощью которых приложение или операционная система могут уменьшить размер файла базы данных и очистить неиспользуемое пространство:

- У IndexedDB есть ограничение в 250 Мб на приложение и общий системный лимит в 375 Мб на жестких дисках меньше 32 Гб, или 4% (максимум – 20 Гб) на дисках, больше 32 Гб. Таким образом возможна ситуация, что ваше приложение может не иметь достаточно места для хранения данных, в подобном случае вам нужно иметь некий запасной механизм. (При превышении лимита API будет выдавать исключение "Quota Exceeded").
- IndexedDB в Windows 8 не поддерживает составные ключи – таким образом, сейчас не поддерживается несколько значений для ключа или индекса (multientry).
- По умолчанию, доступ к IndexedDB предоставляется только HTML-страницам, которые являются частью пакета приложения, и тем, которые объявлены как URI содержимого (смотрите раздел "Локальный и веб-контексты внутри хост-процесса приложения"

Главы 3 курса "Введение в разработку приложений для Windows 8 с использованием HTML, CSS и JavaScript"). Произвольные веб-страницы, которые могут быть загружены в `iframe`, не имеют доступа к IndexeDB, преимущественно для того, чтобы не занимать место в базе данных, объем которой ограничен 250 Мб и сохранить его для страниц приложения. Однако, вы можете предоставить произвольным страница доступ, включив следующий тег в домашнюю страницу и не устанавливая атрибут `src` элемента `iframe` до тех пор, пока не будет запущено событие `DOMContentLoaded` или `activated`:

```
<meta name="ms-enable-external-database-usage" content="true"/>
```

Помимо IndexedDB существуют и другие технологии баз данных, доступные для приложений Магазина Windows. Для создания локальных реляционных баз данных попробуйте SQLite. Это API, которое хорошо подходит для приложений, написанных на языке наподобие C#, как описано в блоге Тима Хейера (ссылка: http://timheuer.com/blog/archive/2012/08/07/updated-how-to-using-sqlite-from-windows-store-apps.aspx), но, к счастью, есть и версия, которая называется SQL.js, где SQLite скомпилирован для JavaScript с использованием Emscripten (ссылка: http://badassjs.com/post/18857332551/sql-js-sqlite-compiled-to-javascript-via-emscripten). Очень хорошо! В сообществе разработчиков можно найти и другие решения для JavaScript.

Если вас беспокоят ограничения IndexedDB, вы можете воспользоваться API Win32 "Jet" или Extensible Storage Engine (ESE) (ссылка: http://msdn.microsoft.com/library/windows/apps/br205753.aspx) (на них построена реализация IndexedDB). Для этого вам понадобится написать оболочку в виде компонента WinRT на C# или C++ (этому посвящена Глава 5 курса "Программная логика приложений для Windows 8, созданных с использованием HTML, CSS и JavaScript и их взаимодействие с системой"), так как из JavaScript нельзя получить прямой доступ к этим API.

То же самое справедливо для API баз данных сторонних разработчиков. До тех пор, пока они используют лишь API Win32, использование которых разрешено приложениям для Магазина Windows (они

перечислены на странице "Win32 и COM для приложения Магазина Windows" (ссылка: http://msdn.microsoft.com/library/windows/apps/br205757.aspx)), они отлично работают.

Следует так же отметить, что Библиотека OData для JavaScript (ссылка: http://www.odata.org/libraries#JavaScript) так же отлично работает с приложениями для Магазина Windows, реализуя доступ к онлайновым SQL-серверам, так как протокол OData работает посредством REST.

И, наконец, еще одна возможность для организации данных, хранящихся в файлах, по которым можно организовать поиск, заключается в использовании системного индекса (system index) путем создания папки с именем "Indexed" в локальной папке данных приложения. Содержимое файлов в этой папке будет проиндексировано системной службой индексирования и к этому содержимому можно обращаться посредством запросов, использующих Advanced Query Syntax (AQS) с помощью API, описанного ниже в разделе "Расширенные возможности запросов к файлам". Так же вы можете осуществлять поиск, основанный на свойствах Windows (ссылка: http://msdn.microsoft.com/library/windows/desktop/dd561977.aspx), делая такой подход простой альтернативой использования баз данных.

Перемещаемое состояние

Автоматическое перемещение данных состояния приложения между устройствами пользователя – это одна из наиболее интересных и дающих новые возможности функций Windows 8. Есть несколько областей, где небольшие технологические разработки, наподобие этой, значительно снижают нагрузку на разработчиков приложений.

Этот механизм работает очень просто. Во-первых,папка `roamingFolder` и контейнер `roamingSettings` ведуг себя точно так же, как соответствующие локальные объекты. До тех пор, пока их общий размер не превышает `Windows.Storage.ApplicationData.current.roamingSto`, Windows будет копировать эти данные на другие устройства, на которых авторизовался тот же самый пользователь, и на которых

установлено то же приложение. На самом деле, когда приложение установлено, Windows пытается скопировать перемещаемые данные, таким образом, они уже присутствуют при первом запуске приложения.

Если приложение исполняется одновременно на нескольких устройствах, выигрывают последние записанные файлы или параметры. Когда данные перемещаются, другие приложения получают событие `Windows.Storage.ApplicationData.ondatachanged` . Таким образом, приложение всегда загружает подходящее перемещаемое состояние при запуске и обновляет это состояние при необходимости внутри обработчика `datachanged` . Вам следует всегда задействовать эту стратегию, так как Windows может не загрузить сразу перемещаемое состояние для вновь установленного приложения (например, когда пользователь устанавливает приложение и связь с сетью теряется). Как только перемещаемое состояние будет доступно, приложение получит событие `datachanged` . Это показано в Сценарии 5 примера "Данные приложения" (ссылка: http://code.msdn.microsoft.com/windowsapps/ApplicationData-sample-fb043eb2).

Принятие решения о том, как должен выглядеть опыт взаимодействия пользователя с перемещаемым состоянием, это больше вопрос дизайна, чем разработки. Для этого нужно взять все параметры приложения, которые не зависят от аппаратного обеспечения устройства (такие, как параметры, связанные с размером экрана, с возможностями видео, с наличием периферийного оборудования или сенсоров) и подумать о том, имеет ли смысл перемещать тот или иной параметр. Списки избранного пользователя, например, подходят для перемещения, если они относятся к данным, которые не хранятся локально. Таким образом, избранные URI или элементы, расположенные в облачных хранилищах данных, таких, как SkyDrive, Facebook или Flickr подходят для перемещения, а списки избранного и недавно использованные файлы из локальных библиотек пользователя – не подходят. Позиция просмотра в видеофайле из облачного сервиса, наподобие потокового видео, так же может подойти для перемещения, так же, как и информация о месте, где пользователь остановился, читая журнал или книгу. Но, опять же, если данное содержимое локально, тогда, возможно – нет. Конфигурации учетных записей, наподобие параметров

почтового ящика, так же часто хорошие кандидаты, таким образом пользователю не придется настраивать приложение снова на другом устройстве.

В то же время, вы не можете предсказать, захочет ли пользователь перемещать те или иные параметры. В этом случае, правильным шагом будет предоставить пользователю выбор! Для этого включите в пользовательский интерфейс чудо-кнопки Параметры команду, которая пользяет пользователю настраивать перемещение данных по своему усмотрению, особенно если у пользователя есть устройства и дома и на работе, и ему захочется, чтобы одно и то же приложение вело себя по-разному на этих устройствах. Например, в случае с RSS-ридером, пользователь может не захотеть получать оповещения о новых статьях на рабочем компьютере, но хотел бы, чтобы данные обновлялись в реальном времени на домашнем. Набор RSS-лент, с другой стороны, весьма вероятно всегда будет перемещаться, но может случиться так, что пользователь захочет иметь отдельные их списки дома и на работе.

Для того чтобы минимизировать размер перемещаемого состояния и не превышать квоту, вы можете задействовать API `Windows.Storage.Compression` для данных, хранящихся в файлах. По той же причине, никогда не используйте перемещаемое состояние для пользовательских данных. Вместо этого используйте онлайновые сервисы, наподобие SkyDrive для хранения пользовательских данных в облаке и перемещайте URI для этих файлов как часть перемещаемого состояния. Больше об использовании SkyDrive посредством REST API вы сможете узнать из материала "Работа с папками и файлами Microsoft SkyDrive" (ссылка: http://msdn.microsoft.com/library/live/hh826531), из материала "Основные принципы работы SkyDrive" (ссылка: http://msdn.microsoft.com/library/live/hh826545.aspx) (здесь есть список поддерживаемых типов файлов), и из примера "PhotoSky" (ссылка: http://msdn.microsoft.com/library/live/hh826545.aspx). Разъяснения, касающиеся этого и других сервисов Windows Live вы можете найти в блоге "Создание Windows 8", в материале "Реализация возможностей по работе с облаком в приложениях Windows 8 с помощью SkyDrive" (ссылка: http://blogs.msdn.com/b/b8_ru/archive/2011/10/03/windows-8-skydrive.aspx).

Сейчас, возможно, у вас есть множество других вопросов о том, как, на самом деле, работает перемещение данных: "Как часто синхронизируются данные", "Как можно управлять различными версиями данных?", "Что еще нужно об этом знать?". Это хорошие вопросы, и, к счастью, на них есть достойные ответы!

- Учитывая наличие сетевого соединения, перемещаемое состояние на активном компьютере обновляется каждые 30 минут. Так же перемещение происходит немедленно, когда пользователь входит в систему или блокирует компьютер. Блокирование компьютера всегда – лучший способ инициировать синхронизацию с облаком. Обратите внимание на то, что если облачные сервисы знают лишь о пользователе (то есть, об учетной записи Microsoft), имеющем лишь одно устройство, синхронизация с облачным сервисом происходит лишь раз в день. Когда сервисы получают информацию о том, что у пользователя есть несколько устройств, синхронизация происходит каждые 30 минут. Если приложение деинсталлировано со всех компьютеров кроме одного, период синхронизации увеличивается.
- При сохранении перемещаемого состояния вы можете записывать данные в любое время, например, при их изменении. Вам не нужно заботиться о групповой записи параметров, так как у Windows есть встроенный механизм для комбинирования изменений и уменьшения общей нагрузки на сеть.
- Если у вас есть группа параметров, которая по объективным причинам должна перемещаться как единое целое, реализуйте ее в виде взаимосвязанных параметров в контейнере `roamingSettings` .
- В случае с файлами, созданными внутри `roamingFolder` , они не будут перемещаться до тех пор, пока они открыты для записи (то есть, до тех пор, пока открыты потоки, связанные с ними). Полезно проверить, закрыты ли все потоки при приостановке приложения.
- Windows позволяет всем приложениям иметь до 8 Кб параметров с "высоким приоритетом", которые будут перемещены в течение одной минуты, таким образом, экземпляры приложения на нескольких устройствах оказываются лучше синхронизированными. Для того, чтобы это использовать, создайте отдельный параметр или группу взаимосвязанных

параметров в корневой части `roamingSettings` и дайте этому объекту имя HighPriority, таким образом, мы получим `roamingSettings.values["HighPriority"]` (контейнер с этим именем будет перемещаться обычным образом). До тех пор, пока размер этого параметра не превышает 8 Кб, он будет перемещаться в течение минуты после изменения. Если размер будет превышен, он будет перемещаться с обычным приоритетом. Демонстрацию этого вы можете найти в Сценарии 6 примера "Данные приложения".

- На доверенном компьютере, параметры пользователя общеситемного значения, наподобие конфигурации начальной страницы, автоматически перемещаются независимо от приложений. Они так же включают в себя зашифрованные учетные данные, которые приложения записали в хранилище учетных данных. Приложениям никогда не следует пытаться перемещать пароли. Приложения, которые создают дополнительные плитки (как мы увидим в лекции 2 курса "Программная логика приложений для Windows 8, созданных с использованием HTML, CSS и JavaScript и их взаимодействие с системой") могут указать, следует ли копировать эти плитки на новое устройство при установке на него приложения.
- Когда существует несколько версий данных приложения, используемых одним и тем же приложением (с несколькими версиями приложения, конечно), Windows будет управлять каждой версией данных приложения раздельно, что означает, что более новые данные приложения не переместятся на устройство с более старой версией приложения. В свете этого, полезно не применять слишком агрессивное изменение версий данных приложения, так как это может прервать связь между приложениями.
- Облачные сервисы хранят несколько версий перемещаемых данных приложения до тех пор, пока множество версий приложения используются в рамках одной и той же учетной записи Microsoft. Только когда все экземпляры приложения будут обновлены, старая версия данных может быть удалена.
- Когда обновленное приложение воспринимает старую версию перемещаемого состояния, ему следует загрузить его как старую версию, но сохранить – как новую и вызвать `setVersionAsync` .
- Избегайте использования дополнительной схемы назначения

версий внутри перемещаемого состояния, как, например, внесение структурных изменений без изменения версии данных приложения посредством `setVersionAsync` . Так как облачный сервис управляет перемещаемыми состояниями, основываясь на номере их версии, и так как последние записанные данные выигрывают, некоторые версии приложения, которые ожидают некоторых дополнительных данных, и, на самом деле, сохраняют эти данные, могут обнаружить, что эти данные были удалены, так как немного более старые версии приложения их не записали.

- Даже если все экземпляры приложения были удалены с устройств пользователя, облачный сервис хранит перемещаемые данные в течение "разумного срока" (около 30 дней), таким образом, если пользователь переустановит приложение в течение этого периода, он обнаружит, что ранее заданные параметры сохранились. Для того чтобы избежать этого и явным образом удалить перемещаемое состояние из облака, воспользуйтесь методом `clearAsync` .

Панель параметров и пользовательский интерфейс

Сейчас мы уже рассмотрели, в том, что касается хранения данных, различные API, которые приложение может использовать для управления информацией о своем состоянии. Это все, что вам нужно для работы с параметрами и другими данными приложения, которыми вы управляете внутри приложения. Сейчас нас интересует вопрос о том, как управлять параметрами, которые настраивает пользователь, и поэтому мы обратимся к чудо-кнопке Параметры.

Когда пользователь нажимает на чудо-кнопку Параметры (это можно сделать и напрямую, использовав комбинацию клавиш Win + i), Windows отображает панель параметров – часть пользовательского интерфейса, которая заполнена различными командами для настройки параметров, а так же, в нижней части – системными настройками. Приложения могут добавлять собственные команды для настройки параметров, но не обязаны этого делать. Windows гарантирует, что какие-то настройки для приложения всегда будут присутствовать на этой панели: она автоматически показывает названия приложения и разработчика,

команду "Отзывы и оценки", которая ведет к странице приложения в Магазине Windows, команду "Обновить", если обновление для приложения доступно в Магазине Windows, и команду "Разрешения", если приложение объявило какие-либо возможности в манифесте. (Обратите внимание на то, что команда "Отзывы и оценки" не отображается, если приложение запущено из Visual Studio, так как эта возможность предоставляется Магазином Windows).

Чудо-кнопка Параметры доступна всегда, независимо от места в приложении, где вы находились, поэтому нет нужды думать о реализации подобной команды в панели приложения, не нужна подобная команда и на холсте приложения. Тем не менее, вы можете программно активировать чудо-кнопку Параметры, когда, например, вы обнаружили, что какая-то возможность выключена и оповещаете об этом пользователя. Вы можете задать ему вопрос наподобие: "Вы хотите включить функцию определения местоположения для этого приложения?", и если он ответит "Да", вы можете активировать чудо-кнопку Параметры. Это делается посредством объекта панели параметров, полученного из `Windows.UI.ApplicationSettings.SettingPane.getFor(` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.ui.applicationsettings.set метод `show` которого отображает пользовательский интерфейс (или выдает исключение, если приложение находится в прикрепленном режиме просмотра, или не является приложением переднего плана, поэтому не выполняйте подобный вызов при таких обстоятельствах!). Свойство `edge` объекта панели параметров сообщает вам о том, с левой или с правой стороны экрана находится панель, в зависимости от ориентации интерфейса системы слева направо или справа налево (зависит от региональных установок).

Теперь мы рассмотрели все методы и свойства этого объекта! Гораздо более интересная часть нашего рассказа посвящена тому, как можно добавлять собственные команды на панель параметров. Но сначала давайте посмотрим на руководство по использованию панели параметров.

Руководства по дизайну панели параметров

Помимо команд, которые Windows автоматически добавляет на панель параметров, приложение может самостоятельно добавить туда до восьми команд, обычно – около четырех. Все, что больше восьми, вызовет исключение. Так как параметры глобальны для приложения, команды, которые вы добавляете, никогда не меняются: они не чувствительны к контексту. Другими словами, на панели параметров располагают только те команды, которые воздействуют на приложение в целом. Команды, применимые лишь к определенным страницам или контексту внутри страниц, следует размещать на панели приложения или на полотне. Некоторые примеры команд верхнего уровня на панели приложения показаны на рис. 2.2.

Рис. 2.2. Примеры команд верхнего уровня на панели параметров. Обратите внимание на то, что нижняя часть панели всегда занята системными параметрами, а названия приложения и разработчика всегда расположены сверху. Команды "Разрешения" и "Оценки и отзывы" добавляются автоматически

Каждая команда, предоставленная приложением, может выполнять одно из двух действий. Во-первых, эта команда может быть обычной гиперссылкой, которая ведет к страницам приложения "Справка", "Соглашение о конфиденциальности", "Условия использования", "Лицензионное соглашение" и так далее, по нажатию на которую соответствующая страница будет открыта в браузере. Другая

возможность заключается в том, что команда вызывает дополнительную всплывающую панель со специализированными элементами управления, или просто iframe для отображения веб-содержимого. Вы можете реализовать страницы "Помощь", "Условия использования" и другое текстовое содержимое обоими способами вместо того, чтобы переключаться в браузер.

Примечание. Как указано в "Сертификационных требованиях к приложениям для Windows 8" (ссылка: http://msdn.microsoft.com/library/windows/apps/hh694083.aspx), в разделе 4.1., приложения, которые тем или иным образом собирает персональные данные, должны иметь политику конфиденциальности или соответствующее заявление. Это должно быть, как минимум, отражено на странице описания приложения в Магазине Windows. Хотя это и не требуется, подразумевается, что вы так же включите в панель приложения соответствующую команду.

Дополнительные всплывающие элементы создают с помощью элемента управления `WinJS.UI.SettingsFlyout` . На рис. 2.3. есть примеры. Обратите внимание на то, что дополнительные панели параметров бывают двух размеров: узкая (346 пикселей) и широкая (646 пикселей). Руководства по дизайну предлагают делать все дополнительные панели параметров приложения одного размера – то есть, не делать некоторые из них узкими, а некоторые – широкими. В любом случае, у вас будет лишь пара подобных панелей, так что это не должно превратиться в проблему. Кроме того, обратите внимание на о, что всплывающий элемент Разрешения (Permissions), показанный в левой части На рис. 2.3. предоставляется Windows автоматически и он настроен в соответствии с возможностями, заявленными в манифесте. Некоторыми возможностями, наподобие определения местоположения, можно управлять из этой панели. Другие, наподобие доступа к Интернету и к библиотекам, просто перечислены, так как пользователь не может включать или выключать их.

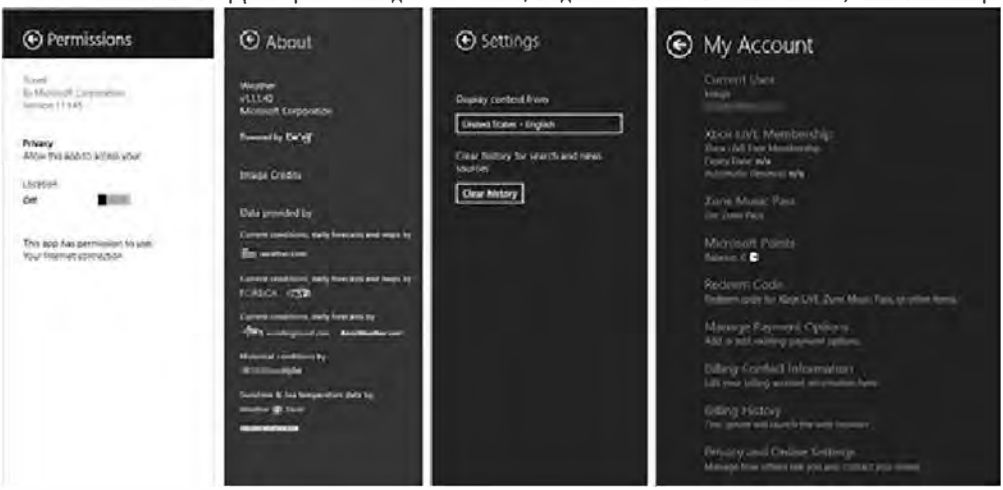


Рис. 2.3. Примеры дополнительных панелей параметров в приложениях Windows 8 Travel, Weather, News и Music. Первые три – узкие, четвертая – широкая. Обратите внимание на то, что каждая из панелей, предоставленная приложением, соответствующим образом брендирована и имеет кнопку "Назад", ведущую к основной панели параметров. Панель Разрешения (Permissions) предоставлена системой, и, таким образом, отражает системную тему. Ее нельзя настроить.

Обычно используемые здесь группы параметров – это те, которые позволяют пользователю настроить особенности перемещения данных приложения – то есть, группа параметров, которая определяет, какие данные состояния приложения перемещаются (это можно увидеть, выполнив команду Изменение параметров компьютера > Синхронизация параметров (PC Settings > Sync You Settings)). Кроме того, рекомендуется включать в панель параметров команды управления учетной записью/профилем, так же как и возможности по входу и выходу. Как отмечено в лекции 1, вход в приложение и лицензионное соглашение, которые необходимы для продолжения работы следует показать при запуске приложения. Для текущих возможностей, связанных со входом в приложение, для просмотра лицензионного соглашения и так далее, создайте соответствующие команды и панели в интерфейсе чудо-кнопки Параметры. Обратитесь к материалу "Руководство и контрольный список для элементов управления входом" (ссылка: http://msdn.microsoft.com/library/windows/apps/hh965453.aspx) для получения более подробных сведений по этому вопросу. Руководство по добавлению справки можно найти в материале "Добавление справки по

приложению" (ссылка: http://msdn.microsoft.com/library/windows/apps/hh465043.aspx).

В плане поведения, дополнительные панели автоматически скрываются, когда они неактивны, но так же имеют заголовок с кнопкой "Назад" для возврата на основную панель параметров со всеми командами. Из-за возможности автоматического скрытия, изменения, которые вносят на панелях, применяются немедленно: здесь нет кнопок "OK" или "Применить" или чего-то подобного. Если пользователь хочет вернуться к прежнему варианту настроек, ему следует просто восстановить исходные параметры.

По этой причине, хорошо использовать простые элементы управления, которые легко переключить в прежнее состояние, вместо сложных наборов элементов управления, которые сложнее вернуть к исходному виду. Рекомендовано использовать тумблеры для параметров, имеющих значения включено/выключено (вместо флагов), кнопки для выполнения некоторых действий (но без закрытия панели параметров), гиперссылки (для открытия браузера), текстовые поля ввода (следует задать им соответствующий тип – адрес электронной почты (email address), пароль (password) и так далее), переключатели для групп, включающих в себя до пяти взаимоисключающих элементов, и списки (элемент select) для четырех-шести текстовых элементов.

Смотрите на все параметры с точки зрения "меньше – значит больше". Избегайте большого количества параметров всех видов, так как пользователь, вероятно, никогда не соберется их искать, и вам, возможно, не нужно показывать их в панели настроек. Кроме того, хотя панель настроек может прокручиваться вертикально, постарайтесь ограничить общее количество настроек, так как пользователь воспользуется прокруткой один-два раза, если вообще станет это делать.

Вот еще некоторые вещи, которых надо избегать, работая с панелью параметров:

- Не используйте панель параметров для размещения команд, имеющих отношение к рабочему процессу. Они должны располагаться на панели приложения, или на полотне, как показано в лекции 1.

- Не используйте команды верхнего уровня в панели параметров для выполнения действий, которые не относятся к связи с другим приложениям (как в случае с браузером). Таким образом, команды верхнего уровня никогда не следует использовать для выполнения каких-либо действий внутри приложения.
- Не используйте команды панели параметров для навигации по приложению.
- Не используйте элемент управления `WinJS.UI.SettingsFlyout` как обычный элемент управления.

Теперь давайте посмотрим на то, как соответствующим образом использовать чудо-кнопку Параметры и элемент управления `SettingsFlyout`.

Расположение команд на панели параметров

Первая часть работы заключается в расположении команд приложения на панели параметров, которая отображается при активации чудо-кнопки Параметры. В отличие от команд панели приложения, это всегда должны быть одни и те же команды, не зависящие от текущего состояния приложения. Если у вас есть контекстно-зависимые параметры, поместите элементы управления для их настройки на панель приложения.

Есть два пути реализации этого в приложениях, написанных на HTML и JavaScript: использовать WinRT напрямую, или применить вспомогательные функции WinRT. Рассмотрим это на примере простой команды Справка (Help).

Для того чтобы знать, когда чудо-кнопка активируется посредством WinRT, получим объект панели параметров из `Windows.UI.ApplicationSettings.SettingsPane.getFor` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.ui.applicationsettings.set и добавим прослушиватель для его события `commandsrequested` (это событие WinRT, поэтому удостоверьтесь в том, что прослушиватель, если это необходимо, удален):

```
// Переменная n здесь – это удобное краткое имя
var n = Windows.UI.ApplicationSettings;
var settingsPane = n.SettingsPane.getForCurrentView();
settingsPane.addEventListener("commandsrequested", onCommandsRequested);
```

В обработчике события создадим объект `Windows.UI.ApplicationSettings.SettingsCommand` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.ui.applicationsettings.set для каждой команды, где каждая команда имеет свойства `id`, `label`, и функцию `invoked`, которая вызывается, когда пользователь коснулся элемента команды или щелкнул по нему мышью. Все это может быть задано в конструкторе, как показано ниже:

```
function onCommandsRequested(e) {
// n все еще является кратким именем для Windows.UI.ApplicationSettings
var commandHelp = new n.SettingsCommand("help", "Help", helpCommandInvo
e.request.applicationCommands.append(commandHelp);
}
```

Вторая строка кода – это место, где вы затем добавляете подобные команды на панель параметров. Это выполняется путем присоединения их к объекту `e.request.applicationCommands` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.ui.applicationsettings.set Этот объект – конструкция WinRT, которая называется вектор (vector) и используется для управления коллекцией элементов с помощью команд наподобие `append` и `insertAt`. В данном случае у нас есть вектор объектов `SettingsCommand`, как вы можете видеть выше, к нему довольно просто присоединять команды. Подобный вызов выполняется для каждой команды, можно и передать массив команд методу `replaceAll` вместо метода `append`. То, что потом происходит в обработчике `invoked` для каждой команды, весьма интересно, и мы вернемся к этому в следующем разделе.

Кроме того, вы можете заранее заполнить вектор `applicationCommand` за пределами события `commandrequested`. Это прекрасно, так как команды,

соответствующие параметрам, следует поддерживать в неизменном состоянии для всего приложения. Материал "Краткое руководство: добавление справки в приложение" (ссылка: http://msdn.microsoft.com/library/windows/apps/hh465062.aspx) показывает пример подобного подхода, который я модифицировал здесь для того, чтобы показать использование `replaceAll`:

```
var n = Windows.UI.ApplicationSettings;
var settingsPane = n.SettingsPane.getForCurrentView();
var vector = settingsPane.applicationCommands;

//Гарантирует отсутствие заданных команд в интерфейсе чудо-кнопки Г
vector.clear();

var commands = [ new settingsSample.SettingsCommand("Custom.Help", "Help'
new n.SettingsCommand("Custom.Parameters", "Parameters", OnParameters)];
vector.replaceAll(commands);
```

При таком подходе вам не нужно добавлять прослушиватель для `commandsrequested` или напрямую обрабатывать это событие.

Теперь, так как большинство приложений, вероятнее всего, будет использовать некоторое количество настроек, WinJS предоставляет несколько вспомогательных механизмов для всего процесса. Во-первых, вместо прослушивания события WinRT, можно просто назначить обработчик для `WinJS.Application.onsettings` (это – контейнер для `commandsrequested`):

```
WinJS.Application.onsettings = function (e) {
// ...
};
```

В вашем обработчике, создайте JSON-объект, описывающий ваши команды и сохраните этот объект в `e.detail.applicationcommands`. Имейте в виду, что он отличается от объекта WinRT – простая настройка этого свойства ни к чему не приведет. Далее происходит передача этого

модифицированного объекта события в `WinJS.UI.SettingsFlyout.populateSettings`, как показано ниже (взято из примера "Параметры приложения" (ссылка: http://code.msdn.microsoft.com/windowsapps/App-settings-sample-1f762f49)):

```
WinJS.Application.onsettings = function (e) {
e.detail.applicationcommands =
{ "help": { title: "Help", href: "/html/2-SettingsFlyout-Help.html" } };
WinJS.UI.SettingsFlyout.populateSettings(e);
};
```

Метод `populateSettings` обходит объект `e.details.applicationcommands` и вызывает метод WinRT `applicationCommands.append` для каждого элемента. Это предоставляет вам более компактный способ для выполнения тех же действий, что вы выполняли с помощью WinRT и так же упрощает реализацию команд параметров, как мы увидим ниже.

Примечание. Вспомогательные функции WinJS специально разработаны для создания элемента управления `SettingsFlyout`, заполненного из HTML-файла, который вы указываете в свойстве `href`. Это свойство должно ссылаться на содержимое, находящееся в пакете приложения; его нельзя использовать для создания элементов управления параметров, которые осуществляют переход по URI (что обычно используется для команд "Условия предоставления услуг" и "Заявление о конфиденциальности"). В подобных случаях вы должны использовать API WinRT напрямую вместе с `WinJS.UI.SettingsFlyout.populateSettings`. Опять же, это просто – поместить веб-содержимое прямо во всплывающий элемент параметров с помощью `iframe`, который поддерживает параметры в соответствии с опытом использования приложения.

Реализация команд: ссылки и всплывающие элементы параметров

С технической точки зрения внутри функции invoked команды

параметров вы можете делать все, что угодно. Правда! Конечно, как описано выше в руководстве по дизайну, есть рекомендации, касающиеся того, как следует и как не следует использовать эти команды. Например, команды в панели параметров не должны действовать как команды панели приложения, которые воздействуют на содержимое, их не следует использовать для навигации внутри приложения. В идеальном варианте эти команды выполняют одну из двух функций: либо открывают гиперссылки (в браузере), либо отображают дополнительные панели параметров.

На базе модели параметров WinRT, при открытии гиперссылки используется API `Windows.System.Launcher.launchUriAsync` (ссылка: http://msdn.microsoft.com/library/windows/apps/hh701480.aspx) как показано ниже:

```
function helpCommandInvoked(e) {
var uri = new Windows.Foundation.Uri("http://example.domain.com/help.html");
Windows.System.Launcher.launchUriAsync(uri).done();
}
```

Во втором случае дополнительные панели реализуются с помощью элемента управления `WinJS.UI.SettingsFlyout` (ссылка: http://msdn.microsoft.com/library/windows/apps/hh701253.aspx). Опять же, с технической точки зрения, вы не обязаны использовать этот элемент управления: вы можете отобразить в обработчике `invoked` любой пользовательский интерфейс, какой хотите. Элемент управления `SettingsFlyout`, однако, предоставлен для обеспечения рекомендованного узкого и широкого размера, поддерживает анимацию при появлении и исчезновении, запускает события наподобие `[before | after][show | hide]` [3] и имеет другие полезные возможности. И хотя вы можете поместить любой HTML-код в элемент управления, включая другие элементы управления, и обычный всплывающий элемент поддерживает вертикальную прокрутку, на самом деле нет причин не использовать его.

Так как он является элементом управления WinJS, вы можете объявить `SettingsFlyout` для каждой из ваших команд в разметке

(убедившись в том, что вызван `WinJS.UI.process/processAll`, благодаря чему будут обработаны другие элементы управления во всплывающем элементе). Например, Сценарий 2 примера "Параметры приложения" (ссылка: http://code.msdn.microsoft.com/windowsapps/App-settings-sample-1f762f49) содержит следующий всплывающий элемент, отображающий справку (здесь текстовое содержимое опущено, код немного переформатирован), в результате получается то, что показано на рис. 2.4.

```
<div data-win-control="WinJS.UI.SettingsFlyout" aria-label="Help settings flyout
 data-win-options="{settingsCommandId:'help', width:'wide'}">
<!-- Используем либо 'win-ui-light' либо 'win-ui-dark' в зависимости от ра
цвета заголовка и фона; фоновый цвет отражает индивидуальность при
<div class="win-ui-dark win-header" style="background-color:#00b2f0">
<button type="button" onclick="WinJS.UI.SettingsFlyout.show()" class="win-bac
<div class="win-label">Help</div>
<img src="../images/smallTile-sdk.png" style="position: absolute; right: 40px;"/>
</div>
<div class="win-content ">
<div class="win-settings-section">
<h3>Settings charm usage guidelines summary</h3>
<!-- Другое содержимое опущено -->
<li>For more in-depth usage guidance, refer to the
<a href="http://msdn.microsoft.com/en-us/library/windows/apps/hh770544">
 App settings UX guide</a>.</li>
</div>
</div>
</div>
```

Как всегда, у этого элемента управления есть параметры, так же, как и несколько применимых классов стилей `win-*`. Есть два параметра, это `settingCommandId`, идентификатор элемента, назначение которого очевидно, и `width`, который может принимать значения `'narrow'` или `'wide'`. И тот и другой присутствуют в вышеприведенном примере. Здесь применены стили win-settingsflyout, который задает стиль всего элемента управления (обычно не используется, за исключением основы для других правил стиля), а так же win-ui-light и win-ui-dark, которые применяют темную или светлую тему к частям всплывающего элемента. В данном примере использована темная тема для заголовка, в

то время как остальные части элемента управления стилизованы с помощью светлой темы по умолчанию.

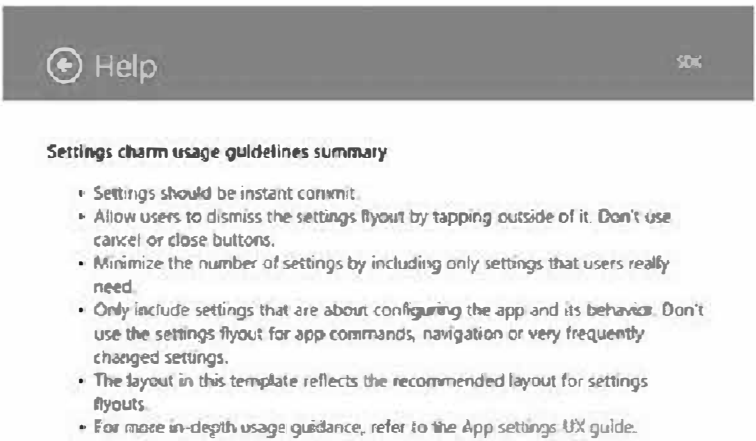


Рис. 2.4. Всплывающий элемент среди параметров, отображающий справку (обрезанный вертикально) из Сценария 2 примера о "Параметры приложения". Обратите внимание на гиперссылку в правом нижнем углу

В любом случае, вы можете видеть, что все внутри элемента управления – это обычная разметка для содержимого, ничего больше, и вы можете подключать события к элементам управления в разметке или в коде. Вы можете использовать здесь гиперссылки, например, для того, чтобы открыть полную версию справочной информации в браузере. Так же вы можете использовать iframe для непосредственной поддержки веб-содержимого внутри всплывающего элемента, как показано в Сценарии 3 того же примера.

Как мы показываем этот всплывающий элемент, когда активирована команда верхнего уровня на панели параметров? Самый простой способ – позволить WinJS позаботиться о деталях, используя сведения, которые вы предоставили для `WinJS.UI.SettingsFlyout.populateSettings`. Вот пример, снова из Сценария 2, который мы видели в предыдущем разделе:

```
e.detail.applicationcommands =
{ "help": { title: "Help", href: "/html/2-SettingsFlyout-Help.html" } };
```

```
WinJS.UI.SettingsFlyout.populateSettings(e);
};
```

В JSON-коде, который вы назначили `applicationCommand`, каждый объект определяет и команду и связанный с ней всплывающий элемент. Имя объекта – это `id` всплывающего элемента ("help"), его свойство `title` задает надпись для команды верхнего уровня на панели параметров ("Help"), свойство `href` задает HTML-страницу, где описан всплывающий элемент с заданным `id` ("/html/2-SettingsFlyout-Help.html").

Используя эту информацию, WinJS может и заполнить панель параметров верхнего уровня, и обеспечить автоматическую активацию нужного всплывающего элемента (вызывая `WinJS.UI.Process` для этого), и вам не нужно писать для этого какой-либо код. Поэтому в сценариях примера вы не увидите явного вызова `showSettings`, а лишь вызовы `populateSettings`.

Вызов всплывающего элемента параметров из программного кода

Посмотрим теперь на то, что происходит внутри описываемых процессов. В дополнении к элементу управления, который используется для определения конкретного всплывающего элемента, `WinJS.UI.SettingsFlyout` имеет пару других статических методов, помимо `populateSettings`: это `show` и `showSettings`. Метод `show` (ссылка: http://msdn.microsoft.com/library/windows/apps/hh701277.aspx) отображает панель параметров Windows верхнего уровня – то есть - Windows.UI.ApplicationSettings.SettingsPane. Поэтому вы можете видеть, что событие `click` кнопки "Назад" в вышеприведенной разметке привязано напрямую к методу show, так как нажатие на кнопку "Назад" должно возвратить нас к интерфейсу верхнего уровня.

Примечание. Хотя можно программно вызывать ваши собственные панели параметров, вы не можете сделать этого с командами, которые предоставляет система, наподобие команд "Разрешения" или "Оценки и

отзывы". Если вы столкнулись с ситуацией, в которой вам нужно, чтобы пользователь изменил разрешения, например, включил бы определение местоположения, рекомендовано отобразить сообщение об ошибке, предлагающее пользователю сделать это.

Метод showSettings (ссылка: http://msdn.microsoft.com/library/windows/apps/hh770581.aspx), с другой стороны, показывает конкретный всплывающий элемент настроек, который определен где-то в приложении. Сигнатура метода выглядит как showSettings(<id> [, <page>]) , где <id> идентифицирует всплывающий элемент, который вам нужен, и необязательный параметр <page> указывает на HTML-документ, в котором следует искать всплывающий элемент с заданным <id> , который не найден в текущем документе. Таким образом, showSettings всегда начинает с просмотра текущего объекта document на предмет элемента WinJS.UI.SettingsFlyout, который имеет совпадающее с заданным свойство settingsCommandId или HTML-атрибут id. Если подобный всплывающий элемент найден, он показывается.

Если разметка в предыдущем разделе (с рис. 2.4.) содержится в той же HTML-странице, которая загружена в приложении, следующая строка кода покажет этот всплывающий элемент:

```
WinJS.UI.SettingsFlyout.showSettings("help");
```

В подобном случае вы так же можете опустить часть href JSON-объекта, который передается в populateCommands, но только если всплывающий элемент уже содержится в текущем HTML-документе.

Параметр <page>, в свою очередь, позволяет вам разделить всплывающие элементы параметров и остальные страницы приложения. Его значение – это относительный URI, относящийся к пакету приложения. В примере "Параметры приложения" подобный подход используется для помещения всплывающих элементов для каждого сценария в отдельный HTML-файл. Вы так же можете поместить все ваши всплывающие элементы в один HTML-файл, до тех пор, пока они имеют уникальные id. В любом случае, если вы предоставляете параметр <page>, showSettings загрузит код

указанного HTML-файла в текущую страницу, используя `WinJS.UI.Page.load` (который вызовет `WinJS.UI.processAll`), просматривает полученное дерево DOM на предмет `id` всплывающего элемента, совпадающего с заданным `<id>` и показывает его. Если всплывающий элемент найти не удается, вызывается исключение.

Сценарий 5 примера показывает этот способ программной активации элемента. Это так же хороший пример (рис. 2.5.) всплывающего элемента с вертикальной прокруткой:

```
WinJS.UI.SettingsFlyout.showSettings("defaults", "/html/5-SettingsFlyout-Setting
```

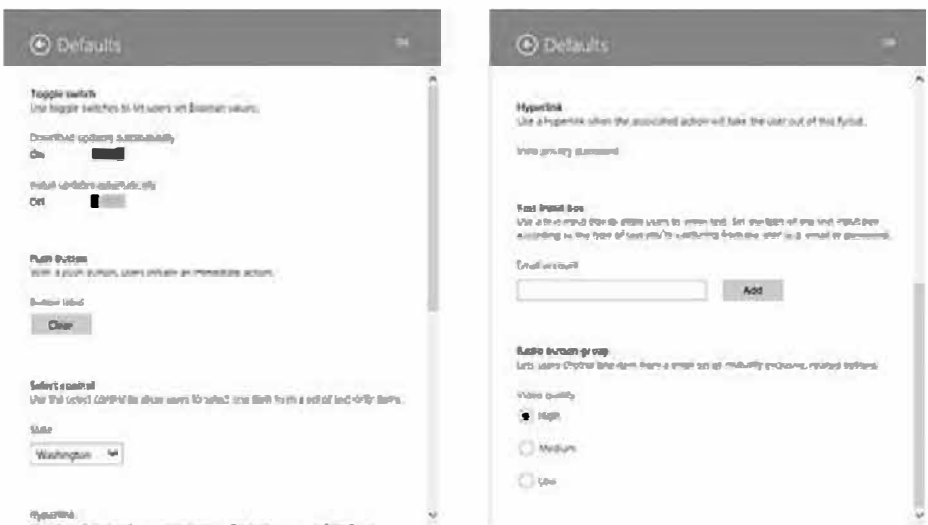


Рис. 2.5. Всплывающий элемент настроек из Сценария 5 примера "Параметры приложения", показывающий, как всплывающие элементы поддерживают вертикальную прокрутку. Обратите внимание на позицию полосы прокрутки при отображении верхней части элемента (слева), и нижней (справа)

Вызов `showSettings`, таким образом, это то, что используется в любом обработчике `invoked` команды, и то, что WinJS выполняет внутри `populateCommands`. Но это так же означает, что вы можете вызвать `showSettings` из любого места вашего кода, когда вам нужно показать конкретную панель параметров. Например, если вы столкнулись с ошибкой, с которой можно справиться, изменив

параметры приложения, вы можете предоставить в сообщении об этом кнопку, которая вызывает `showSettings` для открытия конкретной панели. И, если это нужно, метод `hide` всплывающего элемента может скрыть его. Это не действует на панель параметров верхнего уровня, для скрытия которой нужно использовать `Windows.UI.ApplicationSettings.SettingsPane.getFor`

Вы можете использовать `showSettings` и `hide` вместе, на самом деле, если вам нужно организовать перемещение к панелям параметров третьего уровня. Таким образом, одна из ваших собственных панелей параметров может содержать, команду, которая вызывает `hide` для текущего всплывающего элемента, и затем – `showSettings` для открытия другого. Кнопка "Назад" этого дополнительного элемента (у них всегда должна быть такая кнопка), похожим образом, вызывает `hide` для текущего элемента, и `showSettings` для того, чтобы всплывающий элемент второго уровня снова появился. Тем не менее, мы не рекомендуем делать ваши параметры настолько сложными, чтобы понадобились всплывающие элементы третьего уровня, но такая возможность есть, если у вас есть необходимость в подобном.

Зная о том, как `showSettings` ищет всплывающий элемент так же полезно, если вы хотите программно создать элемент `WinJS.UI.SettingsFlyout`. До тех пор, пока подобный элемент управления находится в DOM, когда вы вызываете `showSettings` с его `id`, WinJS может найти его и отобразить, как и другие элементы. Подобный подход так же работает, хотя я не пробовал это сделать и не подготовил соответствующего примера, использующего такой гибридный подход. Так как `showSettings` загружает заданную HTML-страницу как элемент управления страницы с помощью `WinJS.UI.Pages.load`, эта страница так же может включать собственный скрипт, в котором вы можете задать объект элемента управления страницы с помощью методов вроде `processed` и `ready`. В этих методах вы, затем, можете выполнить необходимые настройки всплывающих элементов настроек, заданных в разметке.

Врезка: Изменение разрешений

Обычный вопрос, который связан с параметрами, заключается в том, может ли приложение получить событие, когда пользователь меняет параметры на панели Разрешения. Ответ на этот вопрос отрицательный. Что означает, что вы можете понять, что доступ к какой-то возможности закрыт, либо обрабатывая исключение Access Denied (Доступ запрещен), когда пытаетесь воспользоваться той или иной возможностью. Вам всегда следует обрабатывать ошибки отказа в доступе, так как пользователь может запретить доступ к возможности при первой попытке использования соответствующего API. Когда это происходит, вы отображаете сообщение об отключенных разрешениях (как показано на примере приложения "Here My Am!" в лекции 1) и предоставляете некоторые интерфейсные элементы для выполнения нужной операции. Однако, пользователю все еще нужно самостоятельно активировать команду Разрешения. Дополнительные подробности вы можете узнать в материале "Руководство для устройств, осуществляющих доступ к персональным данным" (ссылка: http://msdn.microsoft.com/library/windows/apps/Hh768223.aspx).

Данные пользователя: библиотеки, средства выбора файлов и файловые запросы

Сейчас, когда мы тщательно изучили данные приложений и параметры приложений, мы готовы к тому, чтобы рассмотреть другую часть состояния приложения: пользовательские данные. Пользовательские данные, это все то полезное, не привязанное к приложению, что оно может использовать или создавать. Разные приложения могут работать с одними и теми же файлами, такими, как изображения и музыкальные записи, и пользовательские данные всегда остаются на устройстве, независимо от того, какие приложения на нем присутствуют.

Наша первая задача, касающаяся пользовательских данных, заключается в том, чтобы решить, где их разместить, и как получить к ним доступ. Реализация этой задачи включает в себя различные библиотеки пользовательских данных, съемные носители информации и средства выбора файлов. Использование кэша прав доступа так же важно, так как позволяет запоминать факты предоставления пользователем доступа к файлу или папке, к которым мы, в обычных условиях, не имеем программного доступа. Хорошая новость обо всех этих папках и файлах

заключается в том, что работа с ними происходит с помощью те же классов StorageFolder и StorageFile, с которыми мы уже знакомы. Другая важная тема, которую мы рассмотрим, касается использования файловых запросов – наиболее богатого возможностями способа для перечисления содержимого папок и библиотек, которое предоставляет возможности для качественного визуального представления данных в элементах управления наподобие ListView.

Как мы уже видели, приложения для Магазина Windows, по умолчанию, имеют доступ лишь к собственному пакету и к папкам данных приложения. Это означает, что по умолчанию у приложения нет доступа к типичным местам хранения пользовательских файлов! Есть два способа для предоставления такого доступа:

- Объявить возможность доступа к библиотеке в манифесте.
- Позволить пользователю выбрать расположение посредством средства выбора файлов (File Picker).

Посмотрим сначала на средство выбора файлов, так как во многих случаях это все, что, на самом деле, вам понадобится! Но есть и другие сценарии, такие, как приложения, ориентированные на работу с библиотеками – при реализации которых вам нужен прямой доступ к библиотекам. Для этих целей в манифесте предусмотрено пять функциональных возможностей, как показано слева на рис. 2.6. Три из них, Библиотека музыки (Music Library), Библиотека изображений (Pictures Library) и Библиотека видео (Video Library) – предоставляют полный доступ для чтения и записи к пользовательским папкам музыки, изображений и видео. Информация об этих возможностях на странице информации о приложении в Магазине Windows и на панели параметров Разрешения, но пользователь не может управлять ими при исполнении программы. Конечно, если не вполне очевидно, зачем вы объявили эти возможности, убедитесь в том, что вы пояснили это на странице сведений о приложении. Что касается возможностей Библиотека документов (Documents Library) и Съемные носители (Removable Storage), просто объявить эти возможности недостаточно: вам так же нужно задать сопоставление с конкретными типами файлов, которыми ограничено приложение. (Возможность Библиотека документов (Documents Library) предназначена только для приложений, которые нуждаются в открытии встроенного содержимого в другом

документе).

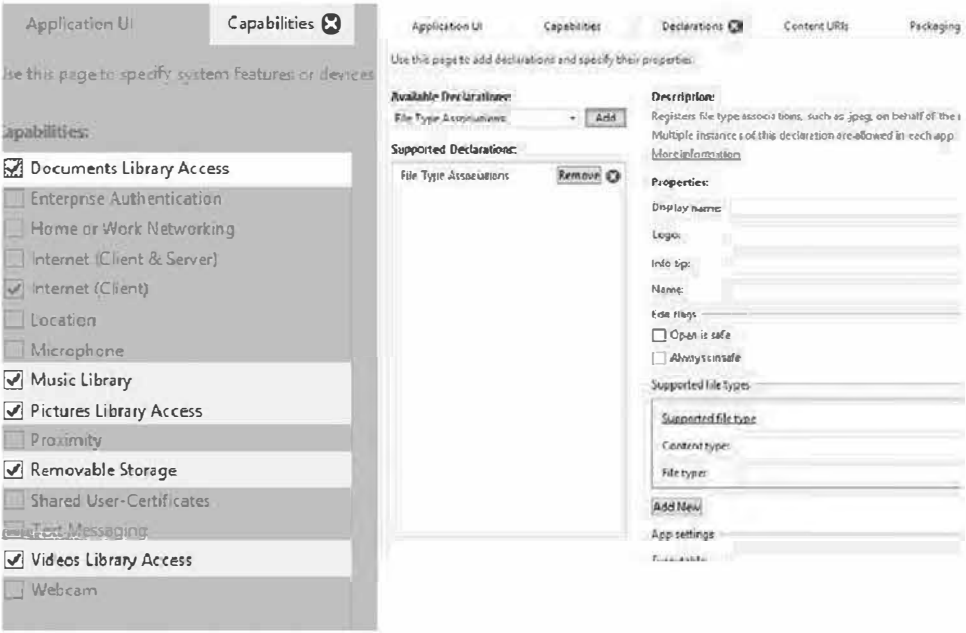


Рис. 2.6. Возможности, связанные с данными пользователя в редакторе манифеста (слева), и редакторе сопоставления типов файлов (справа). Обратите внимание на красный знак "X", который появляется на панели Возможности (Capabilities), когда необходимы дополнительные объявления, связанные с выбранной возможностью. Красный X на панели Объявления (Declarations) показывает, что введенная информация пока не полна.

Врезка: API фоновой передачи данных

Тема, которая относится к пользовательским данным, но которую мы не будем раскрывать в подробностях до Главы 3 курса "Программная логика приложений для Windows 8, созданных с использованием HTML, CSS и JavaScript и их взаимодействие с системой", это API WinRT `Windows.Networking.BackgroundTransfer` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.networking.backgroundt
Это API позволяет вам выполнять загрузку или отправку файлов независимо от жизненного цикла приложения – то есть – делать это,

когда приложение выполняется, приостановлено или полностью остановлено. Это API существует потому, что передача больших файлов на сетевые ресурсы и с них, является обычной потребностью приложений, но при подобной передаче само приложение может и не исполняться в фоновом режиме и не потреблять электроэнергию. Вместо этого приложение настраивает задачу передачи данных с помощью системных средств, и эта задача будет исполняться даже если приложение будет закрыто. Когда приложение будет запущено снова, оно может проверить статус выполнения этой задачи.

Использование средства выбора файлов

Хотя словосочетание "Средство выбора файлов" звучит не особенно шикарно, это, по-моему, одна из самых замечательных возможностей Windows 8. "Подождите минуту!", - скажете вы, "Как, элемент пользовательского интерфейса может выбирать папки и файлы. Да, это здорово!". Причина заключается в том, что используя это средство пользователь может просматривать и выбирать любые данные. Это не только то, что хранится в локальной файловой системе или в локальных сетевых хранилищах, но и те данные, которые доступны благодаря так называемым поставщикам выбора файлов (file picker providers). Эти приложения обычно содержат библиотеки данных, которые в противном случае были бы расположены на веб-сервисе, внутри собственной базы данных приложения, или даже предоставляют файлы, создаваемые по запросу, и предоставляют к ним доступ как к части локальной файловой системы.

Немного подумайте об этом (как я предлагал вам сделать в лекции 1 курса "Введение в разработку приложений для Windows 8 с использованием HTML, CSS и JavaScript"). Когда вы хотите поработать с изображением с фото-сервиса, наподобие Flickr или Picasa, что вы обычно делаете? Первым шагом является загрузка нужного файла и сохранение его в локальной файловой системе с помощью некоторого приложения, которое предоставляет вам интерфейс к веб-сервису (это может быть веб-приложение). Затем вы можете выполнить желаемые правки и модификации, после чего обычно нужно снова загрузить файл в сервис. На самом деле, все не так уж и плохо, за исключением того, что вы тратите время на переключение между разными приложениями,

и, очевидно, загрязняете систему множеством временных файлов, взаимоотношения которых с вашими файлами в Интернете скоро будут забыты.

Если у вас есть поставщик выбора файлов, который позволяет получать прямой доступ к подобным файлам, и для записи, и для чтения, все эти дополнительные шаги становятся ненужными, равно как и становится ненужным переключение между приложениями. Это означает, что поставщик выбора файлов фото-сервиса открывает другим приложениям возможность загружать, править и сохранять онлайновое содержимое так, как если бы оно находилось в локальной файловой системе. Приложениям, принимающим данные, не нужно ничего знать о подобных веб-сервисах, и они автоматически получают доступ к тем большему количеству сервисов, чем больше приложений-поставщиков будет установлено. Более того, поставщики могут представлять данные, которые обычно не хранятся в виде файлов так, если бы они были файлами. Например, приложение Camera (Камера) в Windows 8 – это поставщик выбора файлов, который позволяет вам активировать камеру, сделать снимок и затем возвращает этот снимок так, если бы он был взят из файловой системы. Все это дает пользователям естественные средства для работы с данными, и неважно, где эти файлы расположены. Как я и сказал, я думаю, что это – замечательная возможность!

Мы поговорим подробнее о поставщиках выбора файлов в лекции 1 курса "Программная логика приложений для Windows 8, созданных с использованием HTML, CSS и JavaScript и их взаимодействие с системой". Наша более близкая цель заключается в изучении средств выбора файлов для получения объектов `StorageFile` или `StorageFolder`.

Пользовательский интерфейс средства выбора файлов

Прежде чем смотреть на код, познакомимся поближе с пользовательским интерфейсом средства выбора файлов. Когда средство запущено, вы увидите его полноэкранное представление (рис. 2.7), которое представляет средство выбора файлов в режиме выбора одного элемента, данные отображаются в виде миниатюр. В подобном

режиме просмотра элементы отображаются в виде значков в элементе управления ListView, со всплывающим элементом управления, содержащим подробную информацию, который появляется, когда указатель мыши зависает над элементом. В некотором смысле, средство выбора файлов похож на приложение, запущенное для определенных целей, и оно создано для того, чтобы быть привлекательным и иммерсивным, как и прочие приложения для Магазина Windows.

На рис. 2.7. заголовок Pictures (Изображения) показывает текущее расположение, открытое в средстве выбора файлов. Выпадающий список Sort By Name (Сортировать по имени) позволяет вам выбирать другие критерии сортировки, так же, как выпадающий список в области заголовка Files (Файлы) позволяет вам выбирать другое расположение, как показано на рис. 2.8. Это расположения включают в себя другие области файловой системы (за исключением защищенных областей, наподобие папок Windows и Program Files), сетевые расположения и другие приложения – поставщики информации.

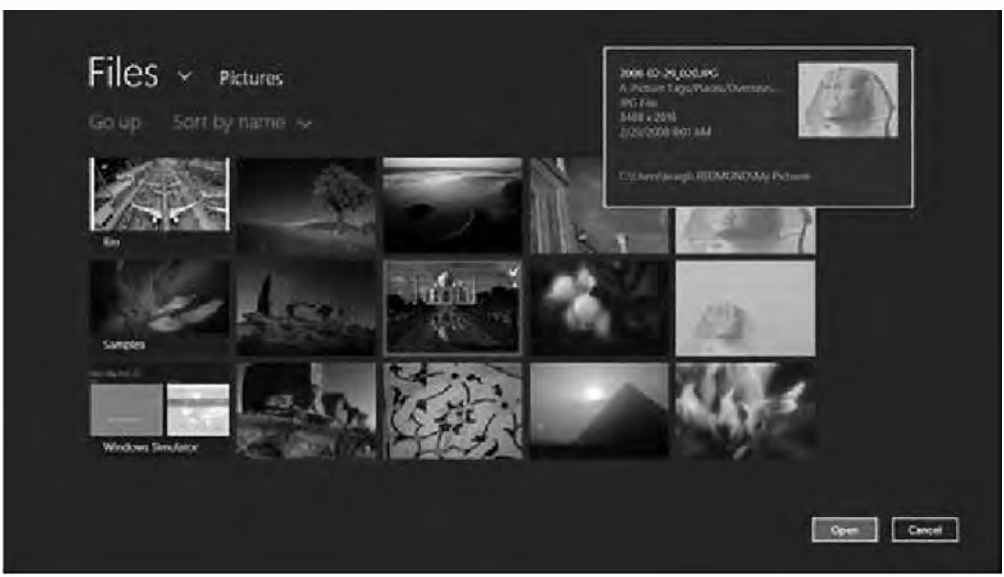


Рис. 2.7. Средство выбора файлов в режиме выбора одного элемента, открытое для Библиотеки изображений в режиме просмотра миниатюр, со всплывающим элементом подсказки, показанным для одного из элементов (для головы Сфинкса), в то же время рамка выделения присутствует у другого элемента (Тадж-Махал)

Рис. 2.8. Выбор другого расположения для просмотра файлов. Обратите внимание на то, что приложения-поставщики перечислены вместе с расположениями файловой системы

Выбор другого расположения в файловой системе переносит нас туда, конечно, из него можно перейти в другие папки. Выбор приложения, с другой стороны, запускает это приложение посредством контракта поставщика выбора файлов. В подобном случае появляется пользовательский интерфейс, предоставляемый системой (но настраиваемый приложением), наподобие того, который показан на рис. 2.9. Здесь выпадающий список около заголовка позволяет вам переключаться к другим расположениям средства выбора файлов, и кнопки Открыть (Open) и Отменить (Cancel) работают так, как и должны для того, что выделено в средстве выбора файлов. Коротко говоря, приложение-поставщик, на самом деле, лишь расширение пользовательского интерфейса средства выбора файлов, но очень мощное расширение. И в итоге, подобное приложение просто возвращает соответствующий объект `StorageFile`, который задает его связь с исходным приложением. Много всего происходит всего лишь при одном вызове API средства выбора файлов!

Рис. 2.9. Приложение для работы с камерой, запущенное с помощью средства выбора файлов. Откуда взялся этот поползень?

У средства выбора файлов есть пара других режимов. Один из них позволяет выделять сразу несколько файлов – даже из разных приложений! – как показано на рис. 2.10, где все выделенные файлы помещены в корзину элементов (basket) в нижней части экрана. Средство выбора так же можно использовать для выбора папок, как показано на рис. 2.11. (в данном случае приложения-поставщики не показаны), или место для сохранения файла и его имя (рис. 2.12.).

Рис. 2.10. Средство выбора файлов в режиме множественного выбора с корзиной выделенных элементов в нижней части. Здесь так же показан режим просмотра в виде "списка", что задается независимо от режима выбора файлов

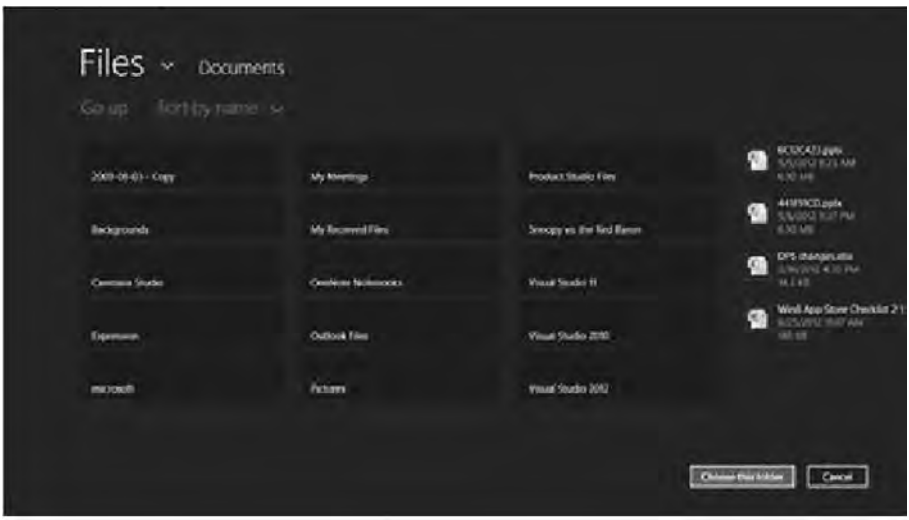


Рис. 2.11. Средство выбора файла используется для выбора папок – обратит внимание на то, что текст на кнопке изменился, и на область предварительного просмотра содержимого папки справа

Рис. 2.12. Средство выбора файлов используется для выбора места сохранения файла и задания его имени

API средства выбора файлов и его друзья

Теперь, когда мы видели визуальную часть средства выбора файлов, посмотрим, как можно вызвать его из кода приложения с использованием API Windows.Storage.Pickers (ссылка: http://msdn.microsoft.com/library/windows/apps/br207928.aspx). Все изображения, которые мы только что видели, взяты из примера "Средство выбора файлов" (ссылка: http://code.msdn.microsoft.com/windowsapps/File-picker-sample-9f294cba), и мы так же используем его как источник кода.

Для начинающих, Сценарий 1 в функции `pickSinglePhoto` (js/scenario1.js) использует средство выбора для получения одного объекта `StorageFile` для открытия (чтения и записи):

```
function pickSinglePhoto() {
// Удостоверяемся, что приложение не в прикрепленном режиме, или ч
// средства выбора файлов
var currentState = Windows.UI.ViewManagement.ApplicationView.value;
if (currentState === Windows.UI.ViewManagement.ApplicationViewState.snapp
!Windows.UI.ViewManagement.ApplicationView.tryUnsnap()) {
// Не выводим дополнительных сообщений,
 если приложение не вышло из прикрепленного режима
return;
}

// Создаем объект средства выбора файлов и настраиваем параметры
var openPicker = new Windows.Storage.Pickers.FileOpenPicker();
openPicker.viewMode = Windows.Storage.Pickers.PickerViewMode.thumbnail;
openPicker.suggestedStartLocation =
Windows.Storage.Pickers.PickerLocationId.picturesLibrary;

// Пользователи ожидают увидеть папки, к содержимому которых
применен фильтр, который зависит от сценария.
// Например, при выборе папки документов, ограничьте типы файлов д
openPicker.fileTypeFilter.replaceAll([".png", ".jpg", ".jpeg"]);
```

```
// Открываем средство выбора файлов, чтобы позволить пользователю
openPicker.pickSingleFileAsync().done(function (file) {
if (file) {
// Теперь у приложения есть доступ к выбранному файлу для чтения и з
} else {
// Средство выбора файлов было закрыто без выбора файла
}
});
}
```

Как вы можете видеть, вызывать средство выбора в прикрепленном режиме не следует. Это, как и попытка вызова панели параметров, приведет к исключению. Вы можете проверить это, как показано здесь, или добавить обработчик ошибки в метод `done` в конце[4]. В любом случае, для вызова средства выбора файлов мы создаем экземпляр `indows.Storage.Pickers.FileOpenPicker` (ссылка: http://msdn.microsoft.com/library/windows/apps/br207847.aspx), настраиваем его и затем вызываем его метод `pickSingleFileAsync`. Результат работы `pickSingleFileAsync` – это аргумент file, передаваемый в обработчик завершения, который может быть либо объектом типа `StorageFile`, либо содержать `null`, если пользователь не сделал выбор и закрыл интерфейс средства выбора файла. Именно поэтому всегда нужно проверять результат выбора файлов на `null`.

Настраивая средство выбора файлов, мы задали его `viewMode` как `thumbnail` (из перечисления `Windows.Storage.Pickers.PickerViewMode`), что привело к тому виду, который показан на рис. 2.7. Другая возможность – это `list`, что показано на рис. 2.10.

Кроме того, мы установили `suggestedStartLocation` в значение `picturesLibrary`, что является значением, взятым из перечисления `Windows.Storage.Pickers.PickerLocationId`. Другие возможности - `documentsLibrary`, `computerFolder`, `desktop`, `downloads`, `homeGroup`, `musicLibrary`, и `videosLibrary`, практически все остальные расположения вы

можете видеть на рис. 2.8. Обратите внимание на то, что использование этих расположений не требует от вас объявления возможностей в манифесте, так как пользователь, используя средство выбора файлов, дает разрешение на доступ к этим файлам. Если вы посмотрите манифест этого примера, вы увидите, что он не объявляет никаких возможостей.

Другой свойство, которое мы задаем, это `fileTypeFilter` (объект `FileExtensionVector` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.storage.pickers.fileexten который отражает типы файлов, которые нас интересуют (PNG и JPEG). Помимо этого, `FileOpenPicker` так же имеет свойство `commitButtonText`, которое задает подпись для основной кнопки в пользовательском интерфейсе (та из них, которая не является кнопкой Отмена (Cancel), и `settingsIndentifier`, средство для запоминания различных контекстов для средства выбора файлов. Например, приложение может использовать один идентификатор для выбора изображений, где начальное расположение установлено на библиотеку изображений и режим просмотра – на просмотр в виде миниатюр, и другой идентификатор для выбора документов с другим начальным расположением, и, возможно, с режимом просмотра в виде списка.

Этот пример, как вы можете видеть, помимо получения файлов, ничего не делает с ними, но очень легко увидеть, что мы можем сделать с ними. Мы можем, например, просто передать `StorageFile` в `URL.createObjectURL` и присвоить результат свойству `img.src` для вывода изображения. То же самое можно сделать с аудио и видео-файлами, эти возможности показаны в Сценарии 1 примера "Использование больших двоичных объектов для сохранения и загрузки содержимого" (ссылка: http://code.msdn.microsoft.com/windowsapps/Blob-Sample-0e35889e), о котором я говорил ранее. Этот пример так же показывает чтение содержимого файла посредством `FileReader` из API HTML, вместе с API WinRT и WinJS, которые мы уже видели. Так же вы можете перекодировать изображение (или другие данные), на которое ссылается `StorageFile`, в другой формат (мы увидим это в лекции 4), получить эскиз, как показано в примере "Эскизы файлов и папок" (ссылка: http://code.msdn.microsoft.com/windowsapps/File-thumbnails-sample-17575959), или использовать методы `StorageFile` для

создания копии файла в другом расположении, переименования файла, и так далее. Но, с точки зрения средства выбора файлов, его работа здесь сделана!

Возвращаясь к примеру о средстве выбора файлов, выбор нескольких файлов, во многом – это то же самое, как показано в функции `pickMultipleFiles` в js/scenario2.js. Здесь мы используем режим просмотра list и начинаем просмотр с `documentLibrary`. Опять же, это начальное расположение не требует объявления возможностей в манифесте.

```
function pickMultipleFiles() {
// Проверяем, не находимся ли мы в прикрепленном режиме, и так дале

// Создаем объект средства выбора файлов и настраиваем параметры
var openPicker = new Windows.Storage.Pickers.FileOpenPicker();
openPicker.viewMode = Windows.Storage.Pickers.PickerViewMode.list;
openPicker.suggestedStartLocation =
Windows.Storage.Pickers.PickerLocationId.documentsLibrary;
openPicker.fileTypeFilter.replaceAll(["*"]);

// Открываем средство выбора файлов, чтобы позволить пользователю
openPicker.pickMultipleFilesAsync().done(function (files) {
if (files.size > 0) {
// Теперь у приложения есть доступ к выбранному файлу (файлам) для ч
} else {
// Средство выбора файлов было закрыто без выбора файла
}
});
}
```

При выборе нескольких файлов, результат `pickMultipleFilesAsync` – это объект `FilePickerSelectedFilesArray` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.storage.pickers.filepicke доступ к которому осуществляется как к любому другому массиву, с использованием [ ] (хотя у него меньше методов, чем у обычного массива).

Сценарий 3 примера демонстрирует вызов `pickSingleFolderAsync`, результат этой операции – объект `StorageFolder`. Здесь вы должны задать fileTypeFilter, что поможет пользователям выбрать подходящее расположение, в котором существуют файлы нужного типа, или то, где они смогут создать новые подобные файлы (js/scenario3.js):

```
 function pickFolder() {
// Проверяем, не находимся ли мы в прикрепленном режиме, и так дале



// Создаем объект средства выбора файлов и настраиваем параметры
var folderPicker = new Windows.Storage.Pickers.FolderPicker;
folderPicker.suggestedStartLocation = Windows.Storage.Pickers.PickerLocatior
folderPicker.fileTypeFilter.replaceAll([".docx", ".xlsx", ".pptx"]);

folderPicker.pickSingleFolderAsync().then(function (folder) {
if (folder) {
// Кэшируем папку, таким образом позже мы сможем получить доступ к
Windows.Storage.AccessCache.StorageApplicationPermissions.futureAccessLis
.addOrReplace("PickedFolderToken", folder);
} else {
// Средство выбора файлов было закрыто без выбора объекта
}
});
}
```

В этом примере мы так же видим, как сохранить выбранный объект StorageFolder в `Windows.Storage.AccessCashe` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.storage.accesscache.asp для последующего использования. Опять же, выбирая эту папку, пользователь дает приложению программный доступ к ее содержимого, но только на время текущего сеанса работы. Для того, чтобы сохранить полученные права, приложение должно записать элемент хранения в `futureAccessList`, в кэш, откуда эти данные могут быть извлечены позже, с использованием методов `futureAccessList.getFolderAsync`, `getItemAsync`, или `getFileAsync`. Как и ранее, обратитесь к Сценарию 6 примера "Доступ к файлу" (ссылка: http://code.msdn.microsoft.com/windowsapps/File-

access-sample-d723e597) для того, чтобы больше узнать об этой возможности, и обратите внимание на то, что API `AccessCache` так же можно применять для недавно использованных элементов. Ключевая вещь, которую нужно здесь запомнить, заключается в том, что для любого расположения за пределами пакета приложения, данных приложения, или библиотек, доступ к которым вы объявили в манифесте, вы должны использовать AccessCashe для того, чтобы подобный доступ был у вас в будущем. Обычное сохранение пути к расположению и попытка открыть файлы из него позже не дадут результата.

Последний пример использования средства выбора файлов, в Сценарии 4 примера, создается объект `FileSavePicker` (ссылка: http://msdn.microsoft.com/library/windows/apps/br207871.aspx) и вызывается метод `pickSaveAsync`, что приводит к появлению интерфейса, показанного на рис. 2.12:

```
function saveFile() {
// Проверяем, не находимся ли мы в прикрепленном режиме, и так дале

// Создаем объект средства выбора файлов и настраиваем параметры
var savePicker = new Windows.Storage.Pickers.FileSavePicker();
savePicker.suggestedStartLocation =
Windows.Storage.Pickers.PickerLocationId.documentsLibrary;
// Выпадающий список типов файлов, применимый при сохранении
savePicker.fileTypeChoices.insert("Plain Text", [".txt"]);
savePicker.pickSaveFileAsync().done(function (file) {
if (file) {
// Предотвращает обновление удаленной версии файла до тех пор, пока
// изменения и не вызовем CompleteUpdatesAsync.
Windows.Storage.CachedFileManager.deferUpdates(file);



// записываем в файл
Windows.Storage.FileIO.writeTextAsync(file, file.name).done(function () {
// Дадим Windows знать, что мы завершили изменение файла и другие п
// могут обновлять удаленную версию файла.
// Завершение обновлений может потребовать у Windows запросить у по
Windows.Storage.CachedFileManager.completeUpdatesAsync(file)
```

```
.done(function (updateStatus) {
if (updateStatus === Windows.Storage.Provider.FileUpdateStatus.complete) {
} else {
// ...
}
}
});
});
} else {
// Интерфейс был закрыт
}
});
}
```

У объекта `FileSaverPicker` есть множество тех же свойств, что и у `FileOpenPicker`, но `fileTypeFilter` заменен на `fileTypeChoices` (для заполнения выпадающего списка), так же имеются свойства `suggestedFileName` (строка), `suggestedSaveFile` (объект `StorageFile`), и `defaultFileExtension` (строка). На что интересно взглянуть в вышеприведенном коде, так это на взаимодействие с `Windows.Storage.CachedFileManager` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.storage.cachedfilemanag Этот объект помогает поставщику средства выбора файлов узнать, следует ли ему синхронизировать локальные файлы и файлы, хранящиеся на удаленном сервисе, что необходимо, когда пользователь файла сохраняет новое содержимое, как мы видим здесь. Со стороны потребителя, то, что мы здесь видим – это обычный шаблон для работы с файлами, полученными от средства выбора файлов (или из кэша доступа, если эти данные были сохранены в предыдущих сеансах работы): нам просто нужно дать знать объекту CashedFileManager, что мы осуществляем запись в файл, и сообщить ему, когда сделаем это. Конечно, этого делать не нужно, когда вы работаете с файлами, о которых вы знаете, что они локальные, как с теми, которые находятся в папках, расположенных в AppData. Больше об этом механизме мы узнаем в лекции 1 курса "Программная логика приложений для Windows 8, созданных с использованием HTML, CSS и JavaScript и их взаимодействие с системой", посмотрим на него со стороны

поставщика.

Медиабиблиотеки

Теперь, когда мы познакомились с возможностями средства выбора файлов, мы можем направить внимание на библиотеки. Но прежде чем мы начнем проверять возможности, заявленные в манифесте, остановимся на мгновение для того, чтобы спросить себя: нужны ли эти возможности на самом деле. Средство выбора файлов предоставляет весьма широкий доступ ко всем библиотекам без необходимости объявления каких-либо возможностей. С помощью средства выбора файлов, мы можете позволить пользователю выбрать один или несколько файлов для открытия, управления и сохранения, а пользователь может предоставить вам доступ к папке. И пользователь может указать новое имя файла для сохранения пользовательских данных.

Вам нужен доступ к определенной библиотеке, если вы собираетесь работать с ней вне средства выбора файлов. Например, если вы хотите перебрать содержимое папок Изображения или Музыка для того, чтобы отобразить список файлов в элементе управления ListView или FlipView, как мы делали в лекции курса "Введение в разработку приложений для Windows 8 с использованием HTML, CSS и JavaScript", вам нужно объявлять возможности.

Чтобы быть более конкретным, без использования средства выбора файлов, это – единственный способ получить программный доступ к медиабиблиотекам: получения объекта `StorageFolder` из объекта `Windows.Storage.KnownFolders` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.storage.knownfolders.as В случае с медиаданными, применимые возможности здесь – `picturesLibrary`, `musicLibrary` и `videoLibrary`. Без объявления соответствующих возможностей, попытка получить одно из этих значения приведет к исключению отказа в доступе.

Если вы не собираетесь пользоваться `KnownFolders`, это означает, что вам не нужно объявлять возможности! Помните, что объявленные возможности перечислены на странице приложения в Магазине

Windows и могут вызвать у пользователя размышления о том, стоит ли устанавливать ваше приложение, поэтому, чем меньше возможностей будет объявлено – тем лучше.

Тем не менее, если вы решите, что вам нужен прямой доступ к медиа-библиотекам, работа с ними включает в себя объекты StorageFolder и StorageFile так же, как при работе с любым другим расположением. Единственная разница, однако, это то, что вы можете работать с метаданными, которые часто существуют вместе с медиафайлами, подробнее об этом мы поговорим в лекции 4.

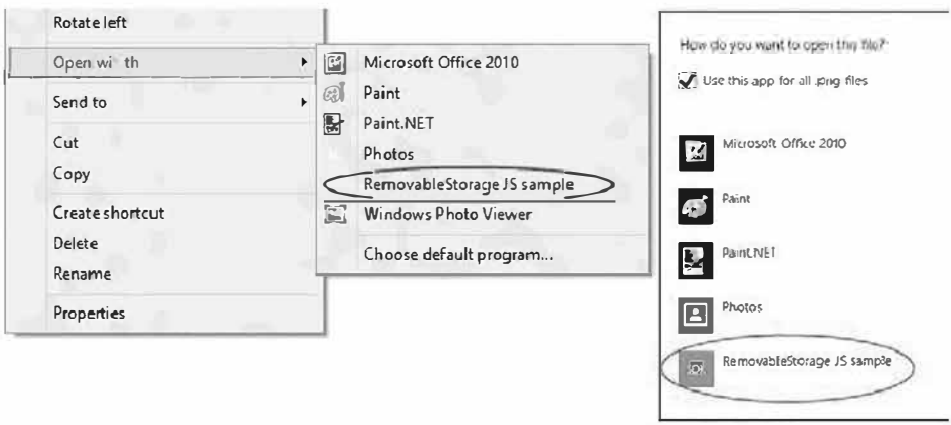
Документы и съемные носители

Как и в случае с доступом к папкам с медиаданными, программный доступ к папке с документами пользователя, так же, как и к съемным носителям, контролируется через объявление возможностей. Важно понимать и то, что обе эти возможности так же требуют, чтобы вы объявили сопоставление типов файлов, что означает, что вы не можете просто просмотреть содержимое папки напрямую, или записать в нее любой файл, какой захотите. Иными словами, получать непосредственный доступ к этим папкам с помощью `Windows.Storage.KnownFolders.documentsLibrary` и `removableDevices`, где и то и другое является объектом `StorageFolder` – просто для работы с ограниченным набором типов файлов, полезно лишь при реализации некоторых сценариев. Для библиотеки документов, на самом деле, документация говорит о том, что "единственное приемлемое использование [возможности] – это поддержка открытия содержимого, внедренного в другие документы". Магазин Windows так же требует оправданного использования объявления возможностей и так же требует, чтобы у вас была корпоративная учетная запись в Магазине Windows, а не только учетная запись индивидуального разработчика.

Более вероятно, что вы просто хотите позволить пользователю открывать или сохранять файлы в этих расположениях или просматривать их содержимое, а в подобных случаях все, что вам нужно – это средство выбора файлов. При таком подходе вам не нужно связывать ваше приложение с определенными типами файлов, что так

же говорит о том, что ваше приложение сможет работать с подобными файлами в любое время.

Например, в примере "Съемные носители информации" (ссылка: http://code.msdn.microsoft.com/windowsapps/Removable-Storage-52cc49f0), для демонстрации доступа к съемным носителям информации, объявляются сопоставления с .gif, .jpg и .png-файлами. В результате, приложение отображается в списке Открыть с помощью… (Open With) в контекстном меню, в Проводнике Windows на Рабочем столе, и в средстве выбора программ по умолчанию:



То же самое справедливо и для документов (снова посмотрите пример Доступ к файлам" (ссылка: http://code.msdn.microsoft.com/windowsapps/File-access-sample-d723e597)), таким образом, если ваше приложение не позиционируется как приложение для работы с подобными файлами, вам, возможно, не нужны эти возможности.

На самом деле, объявление сопоставления типов файлов, это разновидность контрактов, мы рассмотрим это подробно в лекции 1 курса "Программная логика приложений для Windows 8, созданных с использованием HTML, CSS и JavaScript и их взаимодействие с системой".

Расширенные возможности запросов к файлам

Для того чтобы перечислить файлы в определенном расположении

(помимо возможностей), вы используете то, что называется файловым запросом (file query), или просто, поиском. Файловый запрос – это то, что Проводник Windows на рабочем столе использует для поиска по файловой систем, и может включать в себя поиск по содержимому файла, так же, как любое количество других свойств и метаданных. Эти запросы используют то, что называется строками Advanced Query Syntax (AQS) (ссылка: http://msdn.microsoft.com/library/windows/desktop/bb266512.aspx) и могут идентифицировать и описать столько специфических критериев, сколько вам нужно. Вся эта тема несколько не вписывается в рамки курса, но мы можем, по крайней мере, взглянуть на то, как WinRT предоставляет богатые возможности файловых запросов приложениям посредством `Windows.Storage.Search` (ссылка: http://msdn.microsoft.com/library/windows/apps/br208106.aspx). Достаточно будет заглянуть в глубокую кроличью нору!

На самом деле, существуют буквально тысячи параметров, которые можно использовать в AQS-строках, так как они могут быть построены из любого количества свойств Windows (ссылка: http://msdn.microsoft.com/library/windows/desktop/dd561977.aspx), таких, как System.ItemDate, System.Author, System.Keywords, System.Photo.LightSource и так далее [5]. Каждое из свойств может содержать целевое значение, такое, как System.Author(Patrick or Bob) и System.ItemType: "mp3", и условия могут объединяться с помощью операторов AND, OR и NOT. Больше примеров мы увидим в лекции 4, где мы будем использовать запросы для получения коллекций файлов во множестве различных "форм", например, в виде одномерного списка, в иерархическом виде, в различных вариантах сортировки, включая те, которые основаны на медиа-свойствах. В дополнение, файловые запросы так же позволяют получать эскизы и автоматически получать альбомную графику для музыкальных произведений.

Сконцентрируемся теперь на основных принципах работы файловых запросов, начнем с основ, которые показаны в упражнении FileQuery, которое включено в материалы к этой лекции. Это упражнение – копия примера "Программный поиск файлов" (ссылка: http://code.msdn.microsoft.com/windowsapps/Programmatically-searching-25e1a56b) из Windows SDK, там присутствует лишь один сценарий, ориентированный на музыкальную библиотеку, который позволяет вам

непосредственно вводить AQS-строку. Однако, это не то, что вы всегда будете использовать в приложениях, поэтому я хочу показать вам и другие варианты.

Запросы всегда начинаются со `StorageFolder` (ссылка: http://msdn.microsoft.com/library/windows/apps/br227230.aspx), методы которого `createFileQuery[WithOptions]`, `createFolderQuery[WithOptions]`, и `createItemQuery[WithOptions]` (всего 6) используются для перечисления файлов, папок, или и того и другого в заданной папке `StorageFolder`. Простейшие запросы создают на основе методов `StorageFolder.create*` без параметров:

```
folder.createFileQuery();
folder.createFolderQuery();
folder.createItemQuery();
```

Здесь первые два метода – это лишь сокращения для варианта с одним параметром с одним и тем же именем, где параметр – это значение из перечисления `Windows.Storage.Search.CommonFileQuery` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.storage.search.common или `CommonFolderQuery` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.storage.search.common Эти короткие имена просто используют значение `defaultQuery` для обычного перечисления всего содержимого папки в алфавитном порядке. В свою очередь, `createItemQuery` имеет лишь эту единственную форму.

Само по себе создание запроса ничего не перечисляет до тех пор, пока вы не запросите перечисление посредством асинхронного метода: для файловых запросов это метод `getFilesAsync`, для папок – `getFoldersAsync`, а для элементов хранения – `getItemsAsync` (Видите здесь повторяющуюся последовательность?). Таким образом, в Сценарии 2 упражнения FileQuery, у меня есть все три функции, подключенные к кнопкам:

```
function fileQuery() {
var query = picturesLibrary.createFileQuery();
```

```
SdkSample.showResults(query.getFilesAsync());
}

function folderQuery() {
var query = picturesLibrary.createFolderQuery();
SdkSample.showResults(query.getFoldersAsync());
 }

function itemQuery() {
var query = picturesLibrary.createItemQuery();
SdkSample.showResults(query.getItemsAsync());
}
```

Здесь функция `SdkSample.showResults` в js/default.js просто создает список элементов в коллекции. Запустив этот пример, вы увидите список файлов или папок в вашей библиотеке изображений.

Совет. Реальный тип объектов, возвращаемый API `create*Query`, это `StorageFileQueryResult`, `StorageFolderQueryResult`, и `StorageItemQueryResult`, все они находятся в пространстве имен `Windows.Storage.Search`. Все они предоставляют некоторые дополнительные свойства, наподобие folder, методы наподобие `findStartIndexAsync` и `getItemCountAsync`, и события вроде `optionschanged` and `contentschanged` (оба - события WinRT). Последнее событие – это то, что вы можете использовать для отслеживания изменений в файловой системе, которые воздействуют на результаты запроса.

Помимо получения неглубокого (shallow) представления, запросы к файлам и папкам имеют множество других возможностей, которые отражены в перечислениях `CommonFileQuery` и `CommonFolderQuery`:

- В `CommonFileQuery` это: `orderByName` (порядок по имени), `orderByTitle` (порядок по заголовку), `orderByDate` (порядок по дате), `orderByMusicProperties` (порядок по свойствам аудиозаписи), и `orderBySearchRank` (порядок по

рейтингу поиска).

- В `CommonFolderQuery` это : `groupByType` (группировка по типу), `groupByTag` (группировка по тегу), `groupByAuthor` (группировка по автору), `groupByYear` (группировка по году), `groupByMonth` (группировка по месяцу), `groupByArtist`, `groupByComposer` (группировка по композитору), `groupByGenre` (группировка по жанру), `groupByPublishedYear` (группировка по жанру публикации), и `groupByRating` (группировка по рейтингу).

Очевидно, что эффект указания этих вариантов зависит от того, имеются ли у запрошенных элементов метаданные, которые поддерживают соответствующую группировку или упорядочение, но можно выполнить запрос ко всем папкам для всех типов файлов и папок. Для демонстрации этого Сценарий 3 упражнения FileQuery позволяет вам выбирать библиотеку музыки, изображений или видео. Затем вы можете выбрать, нужно ли строить запрос по файлам или папкам, указывать общепринятые параметры и запускать поиск для того, чтобы получить результат. Обратите внимание на то, что использование orderBySearchRank с файлами не имеет смысла в этом контексте, так как это предназначено для поисковых запросов, основанных на AQS. Мы увидим еще кое-что об этом позже (Кроме того – можете назвать меня бездельником! – результыты сгруппированного запроса к папке не особенно интересны, когда не выводятся в некие экранные элементы, но пример реализации подобного механизма вы можете увидеть в Сценарии 2 примера "Перечисление папок" (ссылка: http://code.msdn.microsoft.com/windowsapps/Folder-enumeration-sample-33ebd000)). Код в js/scenario3.js, в основном, реализует механизм отображения выделенных элементов пользовательского интерфейса в `createFileQuery` или `createFolderQuery` с верными параметрами, таким образом вам не нужно просматривать большую их часть. Один важный участок кода – это использование методов `isCommonFileQuerySupported` и `isCommonFolderQuerySupported` объекта `StorageFolder`. Они используются для проверки того, поддерживает ли текущая папка тот запрос, который вы пытаетесь выполнить:

```
if (folder.isCommonFileQuerySupported(selectedQuery)) {
query = folder.createFileQuery(selectedQuery);
if (query) {
promise = query.getFilesAsync();
}
}
```

Вы обнаружите, исполняя примеры в медиа-библиотеках, что все обычные запросы для папок и файлов поддерживаются ими, но это может быть не так для всех объектов `StorageFolder`, с которыми вы можете встретиться. Помните, например, что средство выбора папок может предоставить вам объект `StorageFolder` от поставщика, данные которого находятся на каком-нибудь онлайновом сервисе или в базе данных, в таком случае некоторые запросы могут не работать.

Похожий метод, `StorageFolder.areQueryOptionsSupported`, существует для проверки, поддерживаются ли пользовательские запросы, помимо стандартных. Пользовательский запросы описываются с помощью объекта `Windows.Storage.Search.QueryOptions` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.storage.search.queryopt (стандартные запросы – это лишь предварительно заполненные экземпляры этого объекта) и создаются путем передачи этого объекта в `createFileQueryWithOptions`, `createFolderQueryWithOptions`, и `createItemQueryWithOptions`.

Объект `QueryOptions` обычно создается с нуля, с использованием оператора `new`, после чего вы заполняете его свойства. Так же вы можете использовать `new QueryOptions(<CommonFolderQuery>)` для получения объекта для одного из обычных запросов к папкам, и `new QueryOptions(<CommonFileQuery> [, <file type filter>])` для того, чтобы сделать то же самое для файловых запросов. В последнем случае можно передать необязательный массив типов файлов. Это – короткое имя для часто настраиваемого стандартного запроса с особым набором типов файлов. Без него, фильтр установлен в"*" по умолчанию. Таким образом, если вам просто нужно

найти .mp3-файлы в вашей музыкальной библиотеке, упорядоченные по заголовку (title), вы можете использовать что-то вроде нижеприведенного кода (смотрите js/scenario4.js в упражнении FileQuery):

```
var musicLibrary = Windows.Storage.KnownFolders.musicLibrary;
var options = new Windows.Storage.Search.QueryOptions( Windows.Storage.S

if (musicLibrary.areQueryOptionsSupported(options)) {
var query = musicLibrary.createFileQueryWithOptions(options);
SdkSample.showResults(query.getFilesAsync());
}
```

Если вы создаете `QueryOptions` с нуля, вы можете настроить множество параметров. Вот основные параметры [6] :

- `fileTypeFilter` вектор из строк, который описывает желаемые расширения типов файлов, такие, как ".mp3". Значение по умолчанию – пустой список (фильтрация не применяется).
- `folderDepth` принимает либо значение `Windows.Storage.Search.FolderDepth.shallow` (по умолчанию, неглубокое представление содержимого), либо `deep` (глубокое рекурсивное представление всех вложенных папок и файлов.
- `indexerOption` значение из `Windows.Storage.Search.IndexerOption`, оно может быть одним из следующих: `useIndexerWhenAvailable`, `onlyUseIndexer` (ограничивает поиск индексированным содержимым), и `doNotUseIndexer` (запрос направлен напрямую на файловую систему, без использования индекса). Последний вариант по умолчанию, обычно данное свойство явно устанавливают в значение `useIndexerWhenAvailable`.
- `sortOrder` вектор из структур `Windows.Storage.Search.SortEntry`, каждая из которых содержит булево значение с именем `ascendingOrder` (значение false для убывающего порядка) и строку `propertyName`. Каждая запись в векторе определяет критерий

сортировки. Эти критерии применяются в том порядке, в котором они расположены в векторе. Пример этого будет дан немного позже.

Три свойства QueryOptions затем примняются к поиску с AQS-строками:

- `applicationSearchFilter` – AQS-строка.
- `userSearchFilter` – другая AQS-строка.
- `language` строка, содержащая идентификатор языка BCP-47, связанный с AQS-строками.

Когда запрос построен с помощью метода, наподобие `createFileQueryWithOptions`, строки пользовательского фильтра и фильтра приложения комбинируются. Это означает, что вы можете раздельно управлять любым фильтром, который вы хотите применять везде в вашем приложении (`applicationSearchFilter`) с помощью условий поиска, заданных пользователем (`userSearchFilter`). Таким образом, вы можете применить некоторые поисковые фильтры без необходимости ввода их пользователем, и без того, чтобы всегда выполнять объединение их со строками, заданными вами.

Как упоминалось выше, запрос `CommonFileQuery.orderBySearchRank` имеет смысл лишь тогда, когда он скомбинирован с AQS-строкой, что говорит о том, что поиск, основанный на ключевых словах, возвращает ранжированный результат, к которому может быть применен обычный файловый запрос. Возвращаясь к Сценарию 1 примера о программном поиске файлов, мы можем увидеть, как здесь используется такое упорядочение вместе со свойством `userSearchFilter`:

```
var musicLibrary = Windows.Storage.KnownFolders.musicLibrary;
var options = new Windows.Storage.Search.QueryOptions(
Windows.Storage.Search.CommonFileQuery.orderBySearchRank, ["*"]);
options.userSearchFilter = searchFilter;
var fileQuery = musicLibrary.createFileQueryWithOptions(options);
```

На моем компьютере, где у меня есть несколько музыкальных

композиций со словом "Nightingale" в заголовке, где есть и альбом, названный "Nightingale Lullaby", поиск с использованием строки "Nightingale" System.ItemType: "mp3" в вышеприведенном коде, дает мне следующий результат:

24 files found
04 Song Of The Nightingale2.mp3
16 Song Of The Nightingale (Instrumental).mp3
01 Song of the Nightingale.mp3
04 Song of the Nightingale.mp3
12 Song Of The Nightingale.mp3
- 18 - Twinkle Twinkle.mp3
- 17 - Tum-Balalayka.mp3
- 16 - Tina and Gina Bobina.mp3
- 15 - Shayna's Lullaby.mp3
- 14 - Rockabye Baby.mp3
- 13 - Lullaby for Carol.mp3

Здесь можно видеть, что при поисковом ранжировании отобраны предпочитаемые композиции с "Nightingale" в заголовке, но так же в поиск включены композиции из альбома, который имеет это слово в имени.

Моя поисковая строка, кстати, показывает, как вы можете использовать свойства `applicationSearchFilter` и `userSearchFilter` вместе. Если мое приложение может работать только с mp3 или с некоторыми другими форматами, я могу сохранить `"System.Item.Type: 'mp3'"` в `applicationSearchFilter`, а условия, которые задает пользователь, в `userSearchFilter`. Тем самым я избегаю самостоятельного объединения этих условий в коде.

Помимо свойств, которые устанавливают в объекте `QueryOptions` существуют некоторые данные и возможности этого объекта. Так, `groupPropertyName`, это строковое свойство, которое показывает тип свойства, по которому будет сгруппирован запрос. Так же вы можете получить параметры запроса в виде строки, используя метод `saveToString` и создать объект на основе строки, используя `loadFromString` (это – аналог `JSON.stringify` и `JSON.parse`).

Метод `setPropertyPrefetch` идет даже глубже, позволяя вам указать группу свойств файла, которую вы хотите оптимизировать для быстрого поиска. Доступ к ним осуществляется через те же API, что и к

другим свойствам, но результаты возвращаются быстрее. Это означает, что если вы отображаете коллекцию файлов в ListView, используя пользовательский источник данных с определенными свойствами из полученных файлов, вы можете установить для этих свойств выборку с упреждением (prefetch), и, в итоге, элемент управления быстрее выведет данные. (`WinJS.UI.StorageDataSource` уже это делает). Похожим образом, `setThumbnailPrefetch` сообщает Windows о том, эскизы (thumbnails) какого рода вы хотите включить в полученные результаты – опять же, вы можете запросить их, не устанавливая выборку с упреждением, но когда вы это делаете, вы получаете их быстрее. Это помогает вам оптимизировать визуализацию коллекции файлов[7] .

В лекции 5 курса "Введение в разработку приложений для Windows 8 с использованием HTML, CSS и JavaScript", мы уже видели пример похожего использования свойств, связанных с эскизами, когда пользовались коротким именем библиотеки изображений с `WinJS.UI.StorageDataSource` и могли задавать параметр, указывающий размер эскиза:

```
myFlipView.itemDataSource = new WinJS.UI.StorageDataSource("Pictures",
{ requestedThumbnailSize: 480 });
```

Более общий пример, который так же включает вектор QueryOptions.sortOrder, можно найти в примере "Использование StorageDataSource и GetVirtualizedFilesVector" (ссылка: http://code.msdn.microsoft.com/windowsapps/Data-source-adapter-sample-3d32e535), который упомянут в лекции 5 курса "Введение в разработку приложений для Windows 8 с использованием HTML, CSS и JavaScript". В примере, в js/scenario2.js, мы можем видеть создание QueryOptions с нуля, установку двух критериев sortOrder, и задание параметров эскизов в источнике данных:

```
function loadListViewControl() {
// Создаем источник данных из библиотеки изображений
var library = Windows.Storage.KnownFolders.picturesLibrary;
var queryOptions = new Windows.Storage.Search.QueryOptions;
// Неглубокий запрос для получения иерархии файлов
```

```
queryOptions.folderDepth = Windows.Storage.Search.FolderDepth.shallow;
queryOptions.sortOrder.clear();
// Упорядочиваем элементы по типу, таким образом, первыми идут пап
queryOptions.sortOrder.append({ascendingOrder: false, propertyName: "Systen
queryOptions.sortOrder.append({ascendingOrder: true, propertyName: "System
queryOptions.indexerOption =
Windows.Storage.Search.IndexerOption.useIndexerWhenAvailable;

var fileQuery = library.createItemQueryWithOptions(queryOptions);
var dataSourceOptions = {
mode: Windows.Storage.FileProperties.ThumbnailMode.picturesView,
requestedThumbnailSize: 190,
thumbnailOptions: Windows.Storage.FileProperties.ThumbnailOptions.none
};

var dataSource = new WinJS.UI.StorageDataSource(fileQuery, dataSourceOpti

// Создаем ListView...
};
```

Если вам хочется разобраться с этим глубже, вы можете посмотреть, как заданы файловые запросы в `StorageDataFolder`. Просто найдите этот класс в файле ui.js в WinJS. По пути вы столкнетесь с другим набором API WinRT – возможно, это – дно кроличьей норы. Я хочу упомянуть о них, прежде чем рассматривать эту тему: `Windows.Storage.BulkAccess`. Они действительно существуют исключительно для использования в `StorageDataSource` и не предназначены для прямого использования в приложениях. Даже если вы создадите собственный источник данных или элемент управления коллекцией, лучше всего – использовать средства перечисления и API для упреждающей выборки данных, о которых мы уже говорили, так как они дают такой же уровень производительности.

Обновление "Here My Am!"

Для того, чтобы собрать воедино некоторые темы, изученные в этой лекции, дополнительные материалы включают в себя новую версию

приложения "Here My Am!", в которую внесены следующие изменения и дополнения (в основном, в страницу pages/home/home.js, если не указано иное):

- Теперь программа включает в себя Bing Maps SDK (ссылка: http://msdn.microsoft.com/library/hh846481.aspx), таким образом, соответствующий элемент управления является частью пакета приложения, а не загружается с удаленного ресурса. Это устраняет необходимость в `iframe`, который мы использовали для поддержки карты, теперь весь код из html/map.html можно переместить в js/default.js. Обратите внимание на то, что для того, чтобы запустить этот пример в Visual Studio, вам нужно самостоятельно загрузить и установить этот SDK.
- Вместо копирования изображения, полученного с камеры, в область данных приложения, теперь оно копируется в папку HereMyAm в библиотеке изображений. Объявлена возможность Библиотека изображений (Pictures Library).
- Вместо сохранения пути к последнему полученному изображению, который используется, когда приложение было остановлено и перезапущено, `StorageFile` сохраняется в `Windows.Storage.AccessCache` для того, чтобы гарантировать возможность программного доступа к изображению в будущем.
- На панель приложения добавлена команда, которая позволяет вам использовать средство выбора файлов для выбора изображения, вместо того, чтобы полностью полагаться на возможности камеры. Это так же позволяет вам использовать приложение для работы с камерой, если хотите. Обратите внимание, что в этом случае мы используем собственный `settingsIdentifier` со средством выбора файлов, что отличает его от обычного средства выбора файлов для существующих изображений.
- Другая команда панели приложения позволяет вам выбирать из последних снимков, полученных с камеры. По умолчанию команда направлена на нашу папку в библиотеке изображений, она использует другой `settingdIdentifier`.
- Дополнительные команды – О программе (About), Помощь (Help) и Заявление о конфиденциальности (Privacy Stating) включены в панель параметров с использованием события

`WinJS.Application.onsettings` (смотрите js/default.js). Первые две отображают содержимое из приложения, в то время, как третья команда загружает веб-содержимое в `iframe`. Все страницы параметров можно найти в папке проекта html.

Что мы только что изучили

- Преемственность или сохранение состояния важны в приложениях для Магазина Windows для поддержания состояния непрерывности работы приложения между сеансами работы, даже если приложение приостановлено или остановлено.
- Данные приложения – это данные состояния сеанса, локальные, временные и перемещаемые состояния, которые привязаны к существованию приложения. Доступ к ним может получить только приложение.
- Пользовательские данных хранятся в местах, отличных от мест хранения данных приложения (таких, как библиотеки музыки, изображений, видео, документов, вместе со съемными носителями информации) и существуют независимо от любого отдельно взятого приложения. Открывать пользовательские файлы и манипулировать ими могут различные приложения.
- Доступ к данным приложения можно получить посредством API `Windows.Storage.ApplicationData`, что включает в себя и доступ к структурированным контейнерам параметров, и к данных, основанных на файлах. Так же доступны дополнительные API, наподобие IndexedDB и HTML5 `localStorage`.
- Важно задавать версии состоянию приложения, особенно если используется перемещаемое состояние, от управления версиями зависит то, как служба перемещения данных будет направлять данные на устройства, на которых установлены определенные версии приложений.
- Размер перемещаемого состояния ограничен квотой (она предоставляется API), при превышении которой Windows не будет перемещать данные. Для перемещения больших объемов данных, в том числе – пользовательских, можно использовать сервисы наподобие SkyDrive.
- Обычный период перемещения данных – 30 минут или менее. Отдельный или составной параметр, имеющий имя "HighPriority",

до тех пор, пока его размер не будет превышать 8 Кб, может перемещаться в течение минуты.

- Классы WinRT `StorageFolder` и `StorageFile` – это основные объекты для работы с папками и файлами. Все виды программного доступа к файловой системе, начинаются, на самом деле, со `StorageFolder`. В противном случае, пользователь может указать на файл или папку с использованием API средства выбора файлов, который, на самом деле, является первым вариантом выбора для доступа к файлов.
- Большие двоичные объекты (blob) – это удобные средства для работы с файлами, как и WinRT API в `WindowsStorage.FileIO` и `PathIO`. WinJS предлагает некоторые упрощенные методы для чтения и записи текстовых файлов (особенно они полезны при работе с состоянием приложения), так же поддерживается `FileReader` из HTML5.
- WinRT предлагает средства шифрования с помощью `Windows.Security.Cruptography`, а так же встроенный механизм сжатия данных в `Windows.Storage.Compression`.
- Для использования панели параметров, приложение заполняет панель верхнего уровня, которая предоставлена Windows, собственными командами. Эти команды связаны с обработчиками, которые либо открывают гиперссылки (в браузере), либо отображают всплывающие элементы настроек с использованием элемента управления `WinJS.UI.SettingsFlyout`. Эти всплывающие элементы могут включать в себя любой необходимый HTML-код, в том числе – элементы `iframe` для загрузки удаленного содержимого.
- Доступ к папкам пользовательских данных, таким, как медиа-библиотеки, библиотека документов, съемные носители, контролируется возможностями, объявленными в манифесте. Подобные возможности нужно объявлять лишь тогда, если приложению нужен доступ к файловой системе таким способом, для которого не подходит применение средства выбора файлов.
- Средство выбора файлов – это инструмент, с помощью которого пользователи могут выбирать файлы из любого безопасного расположения в файловой системе, а так же – файлы, предоставляемые другими приложениями (это могут быть

удаленные файлы, файлы, хранящиеся в базах данных, даже те, которые не существуют в виде файловых записей в локальной файловой системе). Одна из наиболее удобных и мощных функций приложений для Магазина Windows – это возможность выбора файлов непосредственно из других приложений, в том числе – файлов, которые приложения могут генерировать по запросу.

- Объекты типа `StorageFolder` обеспечивают обширные возможности по поиску в их содержимом и по доступу к нему с помощью файловых запросов. Запросы могут быть простыми и сложными и могут задействовать строки поиска Advanced Query Syntax (AQS).

1) Получение свойств папки происходит посредством метода элемента хранения `getBasicPropertiesAsync` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.storage.storagefolder.ge (подключенного для `StorageFolder` , но доступного и `StorageFile` .) Этот вызов предоставляет объект `Windows.Storage.FileProperties.BasicPropertiesClas` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.storage.fileproperties.ba который, в свою очередь, имеет метод `retrievePropertiesAsync` (ссылка: http://msdn.microsoft.com/ru-ru/library/windows/apps/br212124.aspx). С помощью этого метода вы можете получить любое количество свойств Windows (ссылка: http://msdn.microsoft.com/library/windows/desktop/dd561977.aspx). Свойство наподобие System.FreeSpace даст вам реальный размер свободного пространства на диске, где расположена папка, которой соответствует `StorageFolder` .

2) Если вам интересно, почему асинхронные методы, такие, как readText и writeText не имеют слова Async в своих названиях, это потому, что дизайнеры WinJS сознательно решили следовать существующим соглашениям об именам, принятых в JavaScript, где такой суффикс обычно не используется. API WinRT, с другой стороны, с другой стороны, не зависит от языков программирования, и имеет собственное соглашение об именовании, допускающее использование суффикса Async.

3) Как насчет краткой комбинации из четырех имен событий? Стоит

так же отметить, что будет срабатывать событие `document.body.DOMNodeInserted`, когда всплывающий элемент будет появляться.

4) Я должен отметить, что пример использует вместо done метод then в последнем асинхронном вызове. Хотя и then работает, там следует использовать done, особенно если вы собираетесь обрабатывать там исключения.

5) Вопреки любым примерам в документации, в запросах всегда следует использовать полные имена свойств Windows, как, например System.ItemDate:, вместо удобного для программистов сокращения date:, так как последний вариант не будет работать в локализованных версиях Windows.

6) Другое свойство, `<code> dateStackOption</code>` (значение из `<code>Windows.Storage.Search.DateStackOption</code>` в данной структуре представлено в виде только для чтения, но оно может быть установлено при создании `<code>QueryOptions</code>`; из `CommonFolderQuery`.

7) Посмотрите материал "Изменения, представляющие интерес для разработчиков приложений, внесенные после выхода Consumer Preview" (ссылка: http://blogs.msdn.com/b/windowsappdev_ru/archive/2012/06/07/consumer-preview.aspx) в блоге для разработчиков приложений для Windows 8 для того, чтобы узнать некоторые подробности об этом.

Ввод данных и сенсоры

Данная лекция содержит описание методик использования сенсорного ввода данных, содержит описание поддержки других способов взаимодействия с системой, а так же - описание особенностей использования различных сенсоров, таких, как компас, акселерометр и другие.

Ввод данных и сенсоры

Файлы к данной лекции Вы можете скачать здесь скачать: http://old.intuit.ru/department/se/uiwin8htmljs/3/3.zip.

Сенсорный экран, безусловно, это одно из самых замечательных средств взаимодействия с компьютером, которое в наши дни, наконец, повзрослело. Конечно, уже много лет у нас есть устройства, которыми можно управлять касаниями. Помню, я работал с экраном, чувствительным к касаниям, в колледже, должен признать, это было до неприличия давно. Тогда сенсорный датчик представлял собой набор прозрачных проводников, встроенных в лист пластика, расположенный над экраном, с общим сенсорным разрешением что-то около 60 точек в ширину и 40 в высоту… и, что действительно выдает давность этих событий, монитор представлял собой полностью текстовый терминал.

К счастью, в последние годы сенсорные экраны чрезвычайно развились. Они достаточно чувствительны для нужд общего назначения (вам не нужно бить по ним, чтобы они зафиксировали событие ввода данных), они встроены в дисплеи высокого разрешения, они сравнительно дешевы, и они могут деть гораздо больше, чем копировать поведение мыши, а именно – они поддерживают множество одновременных касаний и сложные жесты.

Хорошие возможности по сенсорному взаимодействию – это фундаментальная функция отличных приложений, и разработка для сенсорного взаимодействия означает, во многом, переосмысление пользовательского интерфейса. В наших макетах, например, это предусматривает создание целей касания, размер которых подходит для пальцев различного размера. В плане перемещения по содержимому, это подразумевает использование прямых жестов, таких, как протягивание

и жесты сжатия и разведения пальцев, вместо того, чтобы полагаться только на выделение элементов и элементы управления для навигации. Аналогично, разработка для сенсорного взаимодействия подразумевает размышление о том, как жесты могут обогатить опыт взаимодействия пользователя и приложения, и так же, как обеспечить возможности по исследованию элементов и обратную связь пользователям, которые привыкли полагаться лишь на события мыши, наподобие зависания указателя над объектом.

В целом, подходите к дизайну так, будто сенсорное взаимодействие – это единственный способ, который есть у пользователя. В то же время, важно помнить, что новые способы ввода редко полностью замещают устаревшие. Конечно, перфокарты, в конечном счете, исчезли, но изобретение мыши не сделало ненужной клавиатуру. Доступность распознавания речи или рукописного ввода не привели к исчезновению мыши и клавиатуры. Я думаю, то же самое справедливо для сенсорного взаимодействия: это дополнительный способ ввода, который имеет собственные особенные преимущества, но вряд ли он полностью вытеснит другие. Как сказал Билл Бакстон из Microsoft Research: "Каждое отдельное явление, включая сенсорное взаимодействие, хорошо подходит для чего-то одного и плохо – для чего-то другого". Я надеюсь, со временем, мы будем использовать клавиатуру, мышь, и сенсорный экран вместе, так же, как мы научились использовать в повседневной работе мышь в те времена, когда единственным средство ввода информации была клавиатура.

Windows спроектирована для хорошей работы со всеми формами ввода – для отличной работы с сенсорным экраном, мышью, клавиатурой, и все это – на разном аппаратном обеспечении! (И сертификация для Магазина Windows требует, чтобы все приложения обладали этими возможностями.) По этоим причинам, Windows предоставляет унифицированную, основанную на указателе (pointer) модель, где вы можете дифференцировать различные способы ввода, если нужно, но можете обрабатывать их все равнозначно. Так же вы можете сосредоточиться на жестах высокого уровня, которые могут исходить из многих источников ввода, и совсем не беспокоиться об исходных событиях указателя. На самом деле, тот факт, что мы не поднимали до сих пор этот вопрос, показывает, как естественно можно работать со всеми видами указателей ввода и не беспокоиться о них: элементы

управления и другие части пользовательского интерфейса, которые мы использовали, выполняют за нас всю подобную работу. Необходимость в обработке подобных событий возникает преимущественно при создании собственных элементов управления или для выполнения прямых манипуляций с объектами, которые не являются элементами управления.

Клавиатура – это важный инструмент для ввода данных, это относится и к аппаратной клавиатуре, и к экранной "программной" клавиатуре. Последней уделяется много внимания в наше время, в плане устройств, обладающих лишь возможностями сенсорного взаимодействия с пользователем, но, на самом деле, основные вопросы касались доступности. В Windows, так же, программная клавитура включает в себя возможности распознавания рукописного текста, иногда приложения просто включают в себя эту возможность. И когда приложение нуждается в более плотной работе с исходными данными рукописного ввода (ink), оно так же может воспользоваться этими возможностями.

Другая тема, которой мы коснемся в этой лекции, касается датчиков (сенсоров). До тех пор, пока вы не узнаете, что это за датчики, может показаться непоследовательным помещать эту тему вместе с темой о вводе данных. Датчики, вместе с сенсорным экраном – это еще один способ ввода данных. Сенсоры сообщают приложению, что происходит с устройством во внешнем мире: как оно расположено в пространстве (относительно некоторого количества опорных точек), как оно перемещается в пространстве, как оно расположено по отношению к "нормальной" ориентации, и даже то, насколько яркий свет освещает устройство. Видя сенсоры в таком свете (преднамеренный каламбур), мы начинаем видеть возможности прямой интеграции приложений с окружающим миром вместо того, чтобы нуждаться в том, что пользователь сообщит приложению о внешних условиях каким-то абстрактным способом. Хочу предупредить вас, что как только вы увидите, как легко использовать API WinRT для работы с датчиками, вам может захотеться купить новое устройство, которое хорошо ими оснащено!

Ввод данных с помощью мыши, прикосновений и пера

Когда речь идет о вводе данных, основанном на указателе, что включает в себя ввод данных с помощью мыши, касаний, ручки и пера – особое сообщение от Microsoft имело и имеет актуальность: "Дизайн приложений, использующих мышь, касания, перо – без дополнительных усилий". Это – очень важно, как мы увидим дальше, но мы так же обнаружили, что фразы вроде "дизайн, ориентированный на касания" ("first-touch design"), хорошо звучащие для потребителей, может выглядеть пугающей для разработчиков! Когда все внимание обращено на взаимодействие с помощью касаний, нужно учитывать, что пользователи весьма требовательны в своих ожиданиях, и для соответствия продукта этим ожиданиям может потребоваться много работы.

К счастью, Windows 8 предоставляет единую платформу для обработки указателя ввода из всех источников, в итоге, вам не нужно особо задумываться об их различиях до тех пор, пока у вас не будет особых причин сделать это. Таким образом, проектирование приложений с учетом соображений дизайна, ориентированного на касания, это – больше вопрос дизайна, чем вопрос программирования.

В следующем разделе мы подробнее поговорим о проектировании приложений для сенсорного взаимодействия. Для начала мне хотелось бы обсудить, как вам, в роли разработчика, следует подходить к реализации подобного дизайна, до тех пор, пока вам не понадобится делать различия между типами указателя:

- Во-первых – используйте шаблоны и стандартные элементы управления, и вы получите возможности сенсорного ввода без дополнительных усилий, вместе с поддержкой мыши, ручки, пера и клавиатуры. Если вы создаете собственный пользовательский интерфейс на основе стандартных элементов управления, установите соответствующие атрибуты `tabindex` для тех пользователей, которые будут применять клавиатуру, назначьте обработчики для стандартных событий DOM, наподобие `click`, и все готово. Элементы управления наподобие элемента для контекстного масштабирования уже обрабатывают различные виды ввода (как мы видели в лекции 5 курса "Введение в разработку приложений для Windows 8 с использованием HTML, CSS и JavaScript"), CSS-стили, наподобие точек прикрепления и

масштабирования контента обрабатывают различные жесты взаимодействия с приложением.

- Во-вторых, если вам нужно обрабатывать жесты самостоятельно, например, в пользовательском элементе управления, или в других элементах, с которыми пользователь может взаимодействовать напрямую, используйте события жестов, наподобие `MsGestureTap` и `MsGestureHold`, вместе с последовательностями событий для жестов, использующих инерцию (`MSGestureStart`, `MSGestureChange`, и `MSGestureEnd`). Преимущество здесь в том, что эти жесты представляют собой высокоуровневую интерпретацию низкоуровневых событий указателя, что означает, что вам не нужно выполнять подобную интерпретацию самостоятельно. Например, касание (down) с последующим отрывом, поднятием (up) указателя в пределах определенного порога передвижения (для того, чтобы учесть шевеление пальцев) становится единым жестом касания (tap). Касание, за которым следует короткое перетаскивание (drag), после чего указатель поднимается, становится жестом прокрутки (swipe), которое вызывает последовательность событий, возможно, включающих события, указывающие на инерцию (те, которые продолжают вызываться, даже после того, как указатель, наподобие точки касания, физически освобожден). Обратите внимание на то, что если вы хотите захватить и сохранить данные ввода указателя, не обращая внимание на жесты, есть встроенная возможность для рукописного ввода (inking), позже мы ее рассмотрим.
- В-третьих, если вам нужно напрямую обрабатывать события указателя, используйте унифицированные события указателя, наподобие `MsPointerDown`, `MsPointerMove` и так далее.Это – события более низкого уровня, чем события жестов, и они, преимущественно, подходят для приложений, которые не нуждаются в интерпретации жестов. Например, приложению для рисования нужно просто отследить различные указатели и отобразить их на экране, здесь концепции наподобие прокрутки и инерции значения не имеют. События указателя так же предоставляют специфическую информацию от устройств – такую, как давление, поворот, сдвиг, и они отражаются в событиях указателя. Напомню, возможно реализовать обработку жестов на основе прямой обработки событий указателя, как делает большое

количество встроенных элементов управления.

- И, наконец, приложение может работать напрямую со средством распознавания жестов (gesture recognizer) для того, чтобы обеспечить собственный перевод событий указателя в жесты.

А что же о традиционных событиях DOM, которые мы знаем и любим, помимо `click`? Можете ли вы продолжать работать с `mousedown`, `mouseup`, `mouseover`, `mousemove`, `mouseout`, и `mousewheel`? Ответ положителен, так как события указателя от всех источников ввода данных автоматически транслируются в эти традиционные события. Это может быть полезно при портировании кода веб-приложения в код приложения для Магазина Windows, например. Подобная трансляция занимает, однако, дополнительное процессорное время, поэтому новый код, не использующий данные события, отличается лучшей производительностью из-за прямого использования жестов и событий указателя. Традиционные события мыши, кроме того, подразумевают наличие единственного указателя и генерируются только для основной точки касания (с установленным свойством `isPrimary` (ссылка: http://msdn.microsoft.com/library/windows/apps/Hh972358.aspx)). Используйте в коде жесты и события указателя так часто, как сможете.

Примечание. Визуальная обратная связь для сенсорного ввода – это одно из требований к сертификации приложений для Windows 8 (раздел 3.5., ссылка: http://msdn.microsoft.com/library/windows/apps/hh694083.aspx), она применима везде в ваших приложениях, так же, как и к любому веб-содержимому, которые вы можете отображать в элементе iframe. Предоставление обратной связи подразумевает исполнение короткой анимации, которая подтверждает касание. Для этих целей вы можете использовать библиотеку анимаций WinJS или, напрямую, CSS-анимации и переходы, как описано в лекции 5.

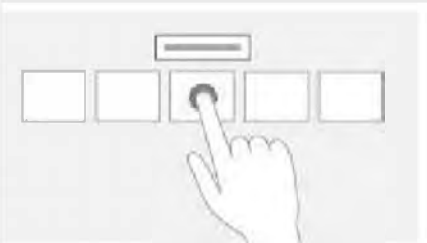
Язык касаний, его перевод и аналоги у мыши и клавиатуры

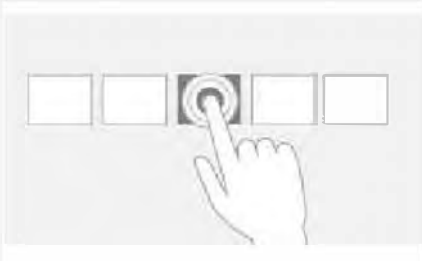
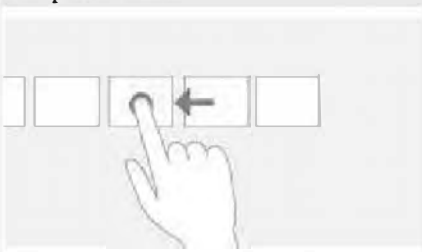
Весьма обширная статья в Центре разработчиков Windows, "Проектирование взаимодействия с сенсорным экраном" (ссылка: http://msdn.microsoft.com/library/windows/apps/hh465415.aspx) полезна и для дизайнеров, и для разработчиков. Она посвящена различным

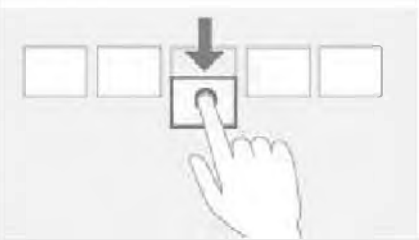
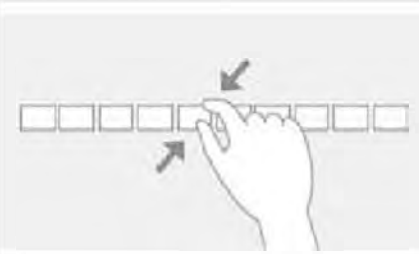
соображениям эргономики, содержит несколько хороших изображений, посвященных различным размерам пальцев, предоставляет четкое руководство по подходящим размерам целей касания, продиктованное реальными условиями, и описывает ключевые принципы дизайна, такие, как обеспечение прямой обратной связи для сенсорного взаимодействия (анимации) и то, как содержимое должно следовать за пальцем.

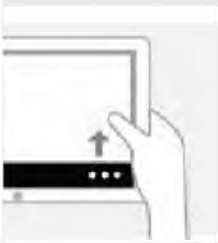
Особенно важно то, что руководство по дизайну так же описывает Язык касаний Windows 8 (Windows 8 Touch Language), который включает в себя восемь основных жестов, которые поддерживаются и системой, и элементами управления. Таблица ниже показывает и описывает эти жесты, и показывает, какие события, воспринимаемые приложением, им соответствуют.

Таблица 3.1.

Жест	Значение и событие жеста
Палец касается экрана и остается в месте касания 	Нажатие с удержанием для вывода информации. Выглядит как события элемента `contextmenu` и `MsGestureHold`.

Палец касается экрана, после чего поднимается. 	Касание для основного действия (для управления), выглядит как события элемента `click` и `MSGestureTap`.
Один или несколько пальцев касаются экрана и перемещаются в одном направлении. 	Скольжение для сдвига (может быть горизонтальным или вертикальным). Выглядит как события прокрутки и как серия событий (`MSGestureStart`, `MSGestureChange`, `MSGestureEnd`, возможно с событиями инерционных жестов, о которых оповещает `MSInertiaStart` плюс - события `MSPointer*`).

Один или несколько пальцев касаются экрана и перемещаются на короткое расстояние в одном направлении 	Скольжение для выбора, управления и перемещения (может быть вертикальным или горизонтальным), та же называется скольжением по диагонали (cross-slide), выглядит как серия жестов (`MSGestureStart`, `MSGestureChange`, `MSGestureEnd`, так же, как и события `MSPointer*`). Средство распознавания жестов, однако, не отличает этот жест от вертикального сдвига, поэтому приложение или элемент управления нуждаются в прямой реализации подобной интерпретации (хорошая причина для использования элементов управления наподобие ListView!)
Два или большее количество пальцев касаются экрана и сдвигаются или раздвигаются. 	Сжатие и растяжение для изменения масштаба. Выглядят как серия жестов (`MSGestureStart`, `MSGestureChange`, `MSGestureEnd`), но приложения могут использовать CSS-стили - `ms-content-zooming: zoom` и `-ms-touch-action: pinch-zoom` для автоматической активации изменения масштаба с помощью жестов.
Два или большее количество пальцев касаются экрана и перемещаются по дуге по часовой стрелке или против часовой стрелки.	Поворот для вращения. Выглядят как серия жестов (`MSGestureStart`,

	`MSGestureChange`, `MSGestureEnd`).
	Скольжение пальцем от верхнего или нижнего края экрана для отображения команд приложения. Автоматически обрабатывается элементом управления AppBar, хотя приложение так же может определять это событие напрямую, посредством `Windows.UI.Input.EdgeGesture`
	Скольжение от края для вывода системных команд. Обрабатывается системой автоматически, приложение принимает события, связанные с активированной чудо-кнопкой, когда применимо, так же события `focus` и `blur`, если меняется приложение переднего плана при скольжении от левого края.

Дополнительные подробности и руководства по дизайну, ориентированные на этот язык касаний можно найти в материале "Жесты, операции и взаимодействия" (ссылка: http://msdn.microsoft.com/library/windows/apps/hh761498.aspx).

Вы можете заметить, что многие жесты языка касаний в вышеприведенной таблице, на самом деле, не имеют отдельного события, связанного с ними (вроде жестов сжатия или вращения), но вместо этого представлены серией жестов или событий указателя. Причина подобного заключается в том, что такие жесты, использующиеся при сенсорном взаимодействии с устройством, обычно включают в себя анимацию содержимого, к которому они относятся, анимация исполняется в ходе выполнения жеста. Жесты скольжения, например, отображают линейное перемещение объекта, который сдвигают или выделяют. Жесты сжатия и растяжения часто активно меняют масштаб содержимого (Семантическое масштабирование – исключение, но в этом случае вы просто позволяете элементу управления обработать все самостоятельно). Жесту вращения, совершенно определенно, следует давать визуальную обратную связь. Коротко говоря, обработка этих жестов при сенсорном взаимодействии, в особенности, подразумевает работу с сериями событий, а не работу с каким-то одним событием.

Это – одна из причин, по которой так полезно (и в плане экономии времени!) использовать встроенные элементы управления так часто, как только возможно, так как они уже обрабатывают все, что связано с жестами. Элемент управления ListView, например, содержит всю логику, относящуюся к жестам и указателям для обработки сдвигов и скольжений, вместе с обычными касаниями. Элемент управления SemanticZoom, как я говорил, реализует жесты сжатия и растяжения, отслеживая события MSPointer*. Если вы посмотрите исходный код этого элемента управления в WinJS, вы оцените, как много он делает

этого элемента управления в WinJS, вы оцените, как много он делает для вас (и увидите, на что похожа реализация функционального пользовательского элемента управления своими силами с использованием средства распознавания жестов!).

Так же вы можете обезопасить себя от множества проблем, используя CSS-свойства `-ms-touch-action`, описанное в разделе "CSS-стили, влияющие на ввод информации". Использование этого свойства добавляет преимущество в виде обработки операций сенсорного ввода в потоке, отличном от потока пользовательского интерфейса, таким образом, предоставляя возможность более плавной манипуляции объектами, чем при использовании обработки событий указателя или жестов.

Продолжая тему: "Дизайн приложений, использующих мышь, касания, перо – без дополнительных усилий", все эти жесты имеют эквиваленты для мыши и клавиатуры, которые так же реализованы во встроенных элементах управления. Эти эквиваленты так же полезно знать, они приведены ниже, в таблице. Раздел "Стандартные сочетания клавиш" ниже в этой лекции так же содержит много сочетаний клавиш, соответствующих командам.

Таблица 3.2.

Жест	Клавиатура	Мышь	Ручка / перо
Нажатие и удерживание (или касание выделенного текста)	Клавиша вызова контекстного меню	Щелчок правой кнопкой	Нажатие и удержание
Касание	Клавиша Enter	Щелчок левой кнопкой	Касание
Скользящее движение (на короткое расстояние)	Клавиши-стрелки	Щелчок левой кнопкой и протягивание, щелчок по свободной области полосы прокрутки, перетаскивание ползунка полосы прокрутки, использование колесика	Касание кнопок полосы прокрутки, перетаскивание ползунка полосы прокрутки, касание и

Скользящее движение с инерцией	Клавиши Page Up / Page Down	Щелчок левой кнопкой и протягивание, щелчок по свободной области полосы прокрутки, перетаскивание ползунка полосы прокрутки, использование колесика мыши.	Касание свободной области полосы прокрутки, перетаскивание ползунка полосы прокрутки, касание и перетаскивание
Скольжение для выделения	Клавиша пробела	Щелчок правой кнопкой	Прикосновение и протягивание
Сжатие / растяжение	Сочетание клавиш Ctrl + "+" и Ctrl + "-"	Клавиша Ctrl + колесико мыши или команда пользовательского интерфейса	Команда пользовательского интерфейса или аппаратная возможность устройства ввода
Скольжение от края	Win+Z, Win+Tab, Win+C или Win+Shift+C	Щелчки по краям экрана. Щелчок правой кнопки мыши открывает панель приложения	Протягивание внутрь от края экрана
Вращение	Ctrl+ "," и Ctrl+ "."	Ctrl + Shift + колесико мыши	Команда пользовательского интерфейса или аппаратная возможность устройства ввода

Вы могли заметить бросающееся в глаза отсутствие двойного щелчка или жеста двойного касания в этом списке. Это вас удивило? В ранних релизах Windows 8 присутствовал жест двойного касания, но он оказался не особо полезным, конфликтуя с жестом масштабирования, и иногда пользователям было очень трудно выполнить его. Я могу сказать, после многих лет наблюдений за друзьями, что двойной щелчок мыши не всегда так уж полезен. Люди с не совсем устойчивыми руками

часто немного перемещают мышь между щелчками, так же они могут слегка переместить пальцы в промежутках между касаниями. В результате, надежность двойного касания невысока, и так как в нем не было особой необходимости, этот жест был полностью исключен из языка касаний.

Врезка: Создание совершенно новых жестов?

В то время, как язык касаний Windows 8 предоставляет простой, хотя и охватывающий подавляющее большинство нужд, набор жестов, не слишком сложно представить другие возможности. Вопрос заключается в том, когда целесообразно вводить новый вид жестов или манипуляций?

Во-первых, имеет смысл то, что приложения обычно не предоставляют новых способов выполнения тех же задач, например, дополнительных жестов для реализации скольжения, масштабирования. Лучше просто подойти творчески к тому, как приложение интерпретирует существующие жесты. Например, жест скольжения может сдвигать прокручиваемую область, но так же может перемещать объекты по экрану – в таком случае не нужно изобретать новый жест.

Во-вторых, если у вас имеется элемент управления, помещенный на экране там, где вам нужно организовать взаимодействие с пользователем, о жестах не нужно думать вовсе, достаточно соответствующим образом обработать данные ввода, полученные от элемента управления.

В-третьих, даже если вы полагаете, что пользовательские жесты нужны, итоговая рекомендация заключается в том, чтобы сделать взаимодействие естественным, вместо реализации того, что вы придумали только ради того, чтобы это придумать. Мы так же рекомендуем, чтобы жесты вели себя единообразно с различными указателями, использовали одинаковые скоростные и временные параметры, и так далее. Например, разделение элемента на три части тремя пальцами путем растягивания и разделение на две части двумя пальцами работает нормально, а вот если растягивание тремя пальцами увеличивает элемент, а растягивание двумя – меняет масштаб

отображения – это уже плохая идея, так как она не понятна интуитивно. Похожим образом, скорость вертикального или горизонтального жеста перелистывания (flick) может воздействовать на скорость перемещения элемента, но использование быстрого жеста перелистывания для перехода на другую страницу, а медленного – для выделения текста – плохая идея. В данном случае различные функции, привязанные к скорости, усложняют пользовательский интерфейс, так как у каждого пользователя свое понимание понятий "быстро" и "медленно" и скорость жеста может быть ограничена их физическими возможностями.

И, наконец, используя любой пользовательский жест, приводит к потенциальной неоднородности между приложениями. Когда пользователь начинает работать с каким-либо приложением новым способом, он может ожидать подобного поведения от других приложений и может смутиться (или расстроиться), если эти приложения не ведут себя таким образом, особенно, если эти приложения используют похожие жесты для совершенно разных целей. Комплексные жесты, так же, могут быть сложными для некоторых, если не для многих, пользователей. Сложность жестов может быть ограничена аппаратным обеспечением устройства (количеством поддерживаемых точек касания, скоростью реакции и так далее), и обычно такие жесты сложнее обнаружить, они не понятны интуитивно. В большинстве случаев, для достижения тех же целей, возможно, проще добавить команду панели приложения, или кнопку на полотне приложения.

Жесты у края экрана

Как мы видели в лекции 1, вам не нужно делать ничего особенного, чтобы появлялись верхняя и нижняя панели приложения: Windows автоматически обрабатывает скольжение от верхнего и нижнего краев, как и щелчок правой кнопкой мыши, сочетание клавиш Win+Z, нажатие клавиши контекстного меню на клавиатуре. Тем не менее, вы можете обнаружить, когда именно происходят эти события, прослушивая события `starting`, `completed` и `canceled` объекта `Windows.UI.EdgeGestures`[1].:

```
var edgeGesture = Windows.UI.Input.EdgeGesture.getForCurrentView();
edgeGesture.addEventListener("starting", onStarting);
edgeGesture.addEventListener("completed", onCompleted);
edgeGesture.addEventListener("canceled", onCanceled);
```

Событие completed вызывается для всех типов ввода. События `starting` и `canceled` происходят только для сенсорных жестов. В этих событиях, свойство `eventArgs.kind` содержит значение из перечисления `EdgeGestureKind`, которое указывает на вид ввода данных, который вызвал событие. События `starting` и `canceled` всегда имеют вид `touch`, очевидно, в то время, как comleted может иметь вид `touch`, `keyboard` или `mouse`:

```
function onCompleted(e) {
// Определяет, было ли это событие инициировано мышью, клавиатуро
if (e.kind === Windows.UI.Input.EdgeGestureKind.touch) {
id("ScenarioOutput").innerText = "Invoked with touch.";
}
else if (e.kind === Windows.UI.Input.EdgeGestureKind.mouse) {
id("ScenarioOutput").innerText = "Invoked with right-click.";
}
else if (e.kind === Windows.UI.Input.EdgeGestureKind.keyboard) {
id("ScenarioOutput").innerText = "Invoked with keyboard.";
}
}
```

Этот код был взят из Сценария 1 примера "Инициация жестов у края экрана" (ссылка: http://code.msdn.microsoft.com/windowsapps/Edge-gesture-invocation-76a474dd). В Сценарии 2 пример показывает, как вы можете предотвратить выполнение событий жестов у края для конкретного элемента, если вы обработаете событие `contextmenu` для этого элемента и вызовите `eventArgs.preventDefault` в вашем обработчике. Подобное делает это для одного элемента на экране, в итоге щелчок правой кнопкой мыши по элементу или нажатие клавиши контекстного меню, когда элемент имеет фокус ввода, предотвратит события жестов у края:

```
document.getElementById("handleContextMenuDiv").
```

```
addEventListener("contextmenu", onContextMenu);

function onContextMenu(e) {
e.preventDefault();
id("ScenarioOutput").innerText =
"The ContextMenu event was handled. The EdgeGesture event will not fire.";
 }
```

Обратите внимание на то, что этот метод не подействует на жесты у края, выполняемые путем касания и на сочетание клавиш Win+Z, которое обычно активирует панель приложения. Это, в основном, для того, чтобы показать, что если вам нужно обработать именно событие contextmenu, вам обычно нужно предотвратить обработку жестов у края.

CSS-стили, влияющие на ввод информации

Во время разговора о вводе данных самое время упомянуть о множестве CSS-стилей, которые могут влиять на ввод данных в приложение.

- `none` – деактивирует прямое выделение, хотя элемент, в целом, может быть выделен, если его элемент-родитель может быть выделен.
- `inherit` – устанавливает поведение выделения элемента таким же, как у элемента-родителя.
- `text` – активирует возможность выделения текста, даже если элемент-родитель установлен в none.
- `element` – активирует возможность выделения для произвольного элеменнта.
- `auto` – значение по умолчанию, может либо активировать возможность выделения, либо нет, в зависимости от типа элемента управления и стилизации элемента-родителя. Для элементов управления, не являющихся текстовыми элементами и не имеющими установки `contenteditable="true"`, возможность выделения не активируется, несмотря на то, что они содержатся в элементе-родителе, который может быть выделен.

Если вы хотите поэкспериментировать с различными вариантами,

обратитесь к примеру "Невыделяемые области содержимого с CSS-атрибутом –ms-user-select" (ссылка: http://code.msdn.microsoft.com/windowsapps/Unselectable-content-areas-963eccd9), которому я присуждаю приз за самое длинное название JavaScript-примера во всем Windows SDK!

Связанный стиль, но не показанный в примере – это `-ms-touch-select`, который может принимать находиться в состоянии `none` или `grippers`, последнее – это стиль, который включает выделение с помощью маркеров в виде окружностей для сенсорного взаимодействия:

Выделяемые текстовые элементы автоматически получают этот стиль, как и другие текстовые элементы с `contenteditable = "true"`. Вы можете использовать `-ms-touch-select` для того, чтобы выключить эту возможность. Для того, чтобы увидеть его действие, попробуйте это с некоторыми элементами в Сценарии 1 вышеупомянутого примера с по-настоящему длинным именем!

В лекции 6 курса "Введение в разработку приложений для Windows 8 с использованием HTML, CSS и JavaScript" мы говорили об использовании точек прикрепления для сдвига, с использованием стилей `-ms-scroll-snap*`. В том же духе, стили, перечисленные в материале "Сенсорное взаимодействие: изменение масштаба и сдвиг" (ссылка: http://msdn.microsoft.com/ru-ru/library/windows/apps/hh453816.aspx), используются для масштабирования содержимого, такие, как `-ms-content-zooming` и `-ms-content-zoom*`, которые предоставляют точки прикрепления и для операций масштабирования. Важно здесь то, что `-ms-content-zooming: zoom` (в противоположность значению по умолчанию – `none`) активирует автоматическое зуммирование с помощью жестов или колеса мыши, обеспечивая элементам того, о чем идет речь, возможность перехода за границы (переполнения) по измерениям X и Y. Существует множество вариантов подобного для сдвига и изменения масштаба, и того, как жесты взаимодействуют с элементами управления WinJS. Оставляю объяснения подробностей этого для примера "HTML-прокрутка, сдвиг и

масштабирование" (ссылка: http://code.msdn.microsoft.com/windowsapps/Scrolling-panning-and-47d70d4c).

И, наконец, стиль `-ms-touch-action` предоставляет элементу множествовозможностей[2)]

- `none` – отключает возможность сенсорного взаимодействия с элементом.
- `auto` – активирует обычное сенсорное поведение.
- `pan-x` / `pan-y` – элементу разрешен горизонтальный / вертикальный сдвиг, что выполняется на ближайшем элементе-предке, который поддерживает возможности горизонтальной / вертикальной прокрутки, на таком, как родительский `div`.
- `pinch-zoom` – активирует изменение масштаба с помощью сведения/разведения пальцев, что выполняется на ближайшем предке, имеющем `-ms-content-zooming: zoom` и возможность переполнения. Например, сам по себе элемент `img` не отреагирует на жест с подобным стилем, однако, если вы поместите его в родительский `div` с установленным свойством `overflow`, то отреагирует.
- `manipulation` – короткое имя для `pan-x pan-y pinch-zoom`.

Для того, чтобы привести пример сдвига и изменения масштаба, попробуем создать простое приложение с разметкой, похожей на эту (используйте здесь любое изображение):

```
<div id="imageContainer">
<img id="image1" src="/images/flowers.jpg" />
</div>
```

И стилизуем контейнер следующим образом:

```
#imageContainer {
overflow: auto;
-ms-content-zooming:zoom;
-ms-touch-action: manipulation;
```

```
}
```

Что могут предоставить возможности ввода данных?

API WinRT в пространстве имен `Windows.Devices.Input` (ссылка: http://msdn.microsoft.com/library/windows/apps/br225648.aspx) предоставляют всю необходимую информацию о возможностях по вводу данных, присутствующих на текущем устройстве, в частности, посредством этих трех объектов: `MouseCapabilities` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.devices.input.mousecap имеет свойства `mousePresent` (0 или 1), `horizontalWheelPresent` (0 или 1), `verticalWheelPresent` (0 или 1), `numberOfButtons` (число), и `swapButtons` (0 или 1). `KeyboardCapabilities` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.devices.input.keyboardc содержит лишь одно свойство: `keyboardPresent` (0 или 1). Обратите внимание, что это свойство не отражает присутствия экранной клавиатуры, которая всегда доступна. Свойство этого объекта указывает на физическое устройство – клавиатуру. `TouchCapabilities` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.devices.input.touchcapa имеет свойства `touchPresent` (0 или 1) и `contacts` (число). Обратите внимание на то, что когда речь идет о сенсорном вводе, может быть интересным свойство Windows.UI.ViewManagement.UISettings.handPreference (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.ui.viewmanagement.uise которое позволяет узнать о направлении пользовательского интерфейса, о том, правша пользователь или левша.

Для проверки доступности сенсорных возможностей, таким образом, вы можете использовать такой код:

```
var tc = new Windows.Devices.Input.TouchCapabilities();
var touchPoints = 0;

if (tc.touchPresent) {
touchPoints = tc.contacts;
```

```
}
```

Примечание. В веб-контексте, где WinRT недоступна, некоторые данные о возможностях по вводу данных могут быть получены с помощью свойств элементов DOM `msPointerEnabled` (ссылка: http://msdn.microsoft.com/library/windows/apps/Hh972607.aspx), `msManipulationViewsEnabled` (ссылка: http://msdn.microsoft.com/library/windows/apps/Hh972607.aspx), и `msMaxTouchPoints` (ссылка: http://msdn.microsoft.com/library/windows/apps/hh779855.aspx). Это так же работает в локальном контексте.

Вы могли заметить, что возможности, перечисленные выше, ничего не сообщают о пере или ручке. Для них, а так же для более подробной информации обо всех устройствах для работы с указателями, включая сенсорный экран и мышь, у нас есть метод `Windows.Devices.Input.PointerDevice.getPointerDevi` Он возвращает массив объектов типа `PointerDevice` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.devices.input.pointerdev каждый из которых имеет следующие свойства:

- `pointerDeviceType` – значение из `Windows.Devices.Input.PointerDeviceType` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.devices.input.poi которое может содержать `touch`, `pen`, или `mouse`.
- `maxContacts` – максимальное количество контактных точек, которое может поддерживать устройство. Обычно 1 для мыши и пера, и любое другое количество для сенсорного экрана.
- `isIntegrated` – значение `true` говорит о том, что устройство ввода встроено в компьютер, то есть на его присутствие можно рассчитывать. Значение `false` указывает на периферийное устройство, которое может отключить пользователь.
- `physicalDeviceRect` – этот объект типа `Windows.Foundation.Rect` предоставляет ограничивающий прямоугольник, в виде которого устройство воспринимает свою рабочую поверхность. Часто разрешение ввода сенсорного экрана не совпадает с экранными пикселями,

что означает, что устройство ввода не способно выделить в точности один пиксель. Один из моих ноутбуков с сенсорным экраном, например, показывает это разрешение как 968x548 для экрана разрешением 1366x768 пикселей (как сообщает `screenRect` ниже). Рабочая поверхность мыши, с другой стороны, обычно один в один совпадает с экранным разрешением. Это может быть важно для приложений для рисования, которые работают с пером, если разрешение ввода меньше, чем разрешение экрана, это может означать некоторую неточность при переводе координат ввода данных в экранные пиксели.

- `screenRect` – этот объект типа `Windows.Foundation.Rect` предоставляет ограничивающий прямоугольник для экрана устройства, то есть, минимальные и максимальные координаты, на которые вам следует рассчитывать, обрабатывая события устройства. Этот прямоугольник принимает в расчет системы с несколькими мониторами, и он подстраивается под масштабирование разрешения.
- `supportedUsages` – массив структур `Windows.Devices.Input.PointerDeviceUsage` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.devices.input.poi которые идентифицируют то, что называется использованием устройств HID (human interface device). Эта тема выходит за пределы данного курса, поэтому я предлагаю вам воспользоваться документом "Использование HID" (ссылка: http://msdn.microsoft.com/library/windows/hardware/ff539946.aspx) в MSDN для того, чтобы получить первоначальные сведения.

Пример "Ввод данных: возможности устройств" (ссылка: http://code.msdn.microsoft.com/windowsapps/Input-device-capabilities-31b67745) в Windows SDK получает эту информацию и выводит ее на экран посредством кода в js/pointer.js. Я не привожу здесь этот код, так как все, что он делает – это обходит массив, создает длинную HTML-строку и помещает ее в DOM. В имитаторе выходные данные выглядят следующим образом. Обратите внимание на то, что имитатор, в данном случае, сообщает о наличии мыши и сенсорного экрана.

```
Output

(1) Pointer Device Type  Touch
(1) Is Integrated        true
(1) Max Contacts         5
(1) Physical Device Rect 0,0,491.3385925292969,907.0866068866787
(1) Screen Rect          0,0,1366,768
(2) Pointer Device Type  Mouse
(2) Is Integrated        false
(2) Max Contacts         1
(2) Physical Device Rect 0,0,1366,768
(2) Screen Rect          0,0,1366,768
```

Любопытная фальсификация? Интересно то, что я запустил этот пример в отладчике Visual Studio на локальном компьютере, на ноутбуке, который совершенно определенно не имеет сенсорного экрана, но о наличии сенсорного устройства ввода, все равно, сообщалось, как на вышеприведенном изображении. Почему? Потому, что все еще был запущен имитатор Visual Studio, который добавляет виртуальное сенсорное устройство к аппаратному профилю. После того, как симулятор был полностью закрыт (а не только свернут в значок), я получил точные сведения о возможностях моего ноутбука. Так что помните об этом, если вы пишете программный код для проверки конкретных возможностей.

Пробовали удаленную отладку? Говоря об отладке, как упомянуто во врезке в лекции 6 курса "Введение в разработку приложений для Windows 8 с использованием HTML, CSS и JavaScript", тестирование приложений на устройствах с разными возможностями – это отличная возможность для применения удаленной отладки в Visual Studio. Если вы еще не сделали этого, сделайте, все настойки займут лишь несколько минут. Для того, чтобы узнать подробности, обратитесь к материалу "Выполнение приложений для Магазина Windows на удаленном компьютере" (ссылка: http://msdn.microsoft.com/library/windows/apps/hh441469%28v=vs.110%29.asp

Унифицированные события указателя

В любой ситуации, когда вы хотите напрямую работать с сенсорным экраном, мышью или пером, возможно, для реализации части языка касаний подобным образом, используйте события `MSPointer*`. Большинство приложений для рисования или художественной обработки изображений, например, используют эти события для

отслеживания и отображения на экране результатов взаимодействия с пользователем. Снова напомню, что указатель – это способ для рассмотрения ввода данных, имеющий более низкий уровень, чем жесты, что мы увидим в следующем разделе. Какие модели ввода данных вы испоьзуете, зависит от типа событий, с которыми вы хотите работать.

Совет. События указателей не вызываются, если система пытается выполнить манипуляцию наподобие сдвига или масштабирования. Для того чтобы отключить у элемента манипуляции, задайте `-ms-content-zooming: none` или `-ms-touch-action: none`, и избегайте использования стилей `-ms-touch-action` в `pan-x`, `pan-y`, `pinch-zoom`, и `manipulation`.

Как и в слудчае с другими событиями, вы можете прослушивать события `MSPointer*` для любых подходящих элементов, помня, опять же, что эти события транслируются в традиционные события мыши, поэтому не следует прослушивать и те и другие. Специальные события, которые описаны ниже, даны в порядке их типичной последовательности:

- `MSPointerOver` (ссылка: http://msdn.microsoft.com/library/windows/apps/hh465907.aspx) указатель перемещен в пределы границ элемента из внешнего пространства.
- `MSPointerHover` (ссылка: http://msdn.microsoft.com/library/windows/apps/hh465895.aspx) указатель завис над элементом (обычно – только для пера или мыши).
- `MSPointerDown` (ссылка: http://msdn.microsoft.com/library/windows/apps/hh465891.aspx) указатель опустился на элемент.
- `MSPointerMove` (ссылка: http://msdn.microsoft.com/library/windows/apps/hh465899.aspx) указатель переместился по элементу.
- `MSPointerUp` (ссылка: http://msdn.microsoft.com/library/windows/apps/hh465912.aspx) указатель был освобожден над элементом. (Если элемент ранее

захватил касание, автоматически вызывается msReleasePointerCapture.) Обратите внимание на то, что если указатель был перемещен за пределы элемента и освобожден, элемент получит событие MSPointerOut, но не MSPointerUp.

- MSPointerCancel (ссылка: http://msdn.microsoft.com/library/windows/apps/hh868516.aspx) событие указателя отменено системой.
- MSPointerOut (ссылка: http://msdn.microsoft.com/library/windows/apps/hh465904.aspx) указатель вышел за пределы элемента, что так же происходит с событием MSPointerUp. Это последнее событие указателя, которое примет элемент.
- MSGotPointerCapture (ссылка: http://msdn.microsoft.com/library/windows/apps/hh465875.aspx) указатель захвачен элементом.
- MSLostPointerCapture (ссылка: http://msdn.microsoft.com/library/windows/apps/hh465883.aspx) захват указателя потерян элементом.
-

Это – те имена, которые используют с addEventListener. Эквивалентные имена свойств имеют, как обычно, форму onmspointerdown. Должно быть очевидным то, что некоторые из этих событий могут не вызываться указателями всех типов. Сенсорные экраны, например, обычно не вызывают события зависания указателя над элементом, хотя некоторые из них, те, что могут определять близость пальца, способны на это.

Совет. Если по каким-то причинам вы хотите предотвратить трансляцию события MSPointer* в традиционное событие мыши, вызовите метод eventArgs.preventDefault внутри соответствующего обработчика события.

Упражнение PointerEvents, которое можно найти среди дополнительной информации к курсу, как показано на рис. 3.1., позволяет вам увидеть, что происходит с событиями мыши, указателей, жестов, выборочно показывая группы событий на экране.

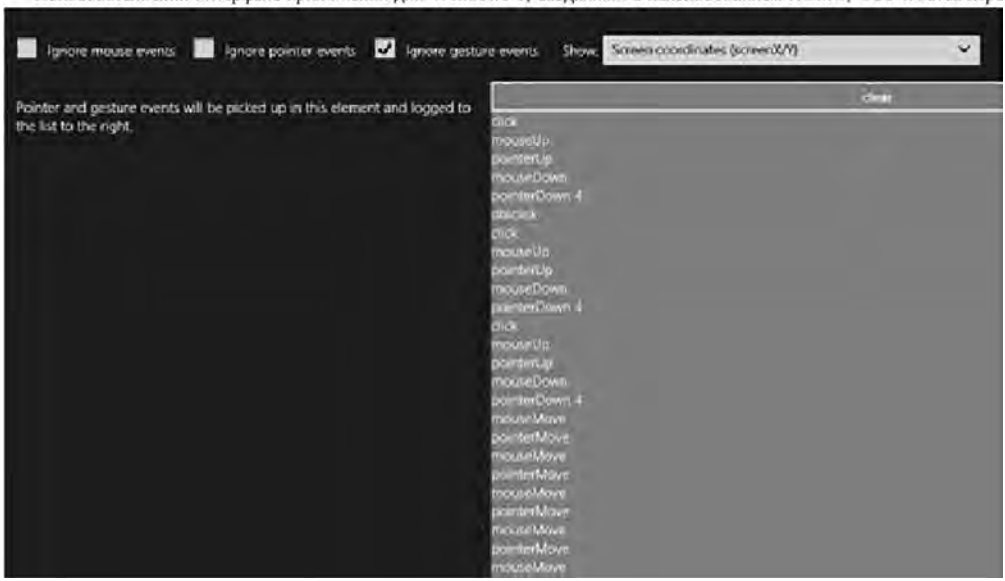


Рис. 3.1. Экран упражнения PointerEvents (экран слегка обрезан для показа подробностей)

Внутри обработчиков для всех событий `MSPointer*`, объект `eventArgs` содержит полный набор свойств. Одно из них, `pointerType`, идентифицирует тип устройства ввода: серсорная панель (2), ручка (3), и мышь (4). Эти свойства позволяют вам, если нужно, реализовывать различное поведение для различных способов ввода. Каждый объект события так же содержит укникальное значение `pointerId`, которое содержит информацию о штрихе или линии для конкретной точки контакта, что позволяет согласовывать первоначальное событие `MSPointerDown` с последующими событиями. Когда мы взглянем, в следующем разделе, на жесты, мы так же увидим, как ипользовать `pointerId` события `MSPointerDown` для сопоставления жеста с указателем.

Полный набор свойств, который поступает вместе с событием, на самом деле, слишком велик, чтобы его здесь приводить, так как он содержит множество обычных свойств DOM (ссылка: http://msdn.microsoft.com/library/windows/apps/hh831236.aspx) вместе со свойствами, имеющими отношение к указателю, из объекта типа `MSPointerEvent` (ссылка: http://msdn.microsoft.com/library/windows/apps/hh441233.aspx). Лучший

способ посмотреть, что там находится – это запустить какой-нибудь код, наподобие примера "Ввод данных: обработка события указателя DOM" (ссылка: http://code.msdn.microsoft.com/windowsapps/Input-DOM-pointer-and-2e5697ed) (это – приложение для рисования на элементе управления `canvas`), установить точку останова внутри обработчика одного из событий, и посмотреть на объект события. Нижеприведенная таблица описывает некоторые из свойсвт (и несколько методов), имеющих отношение к нашей беседе.

Та

Свойства	
`currentPoint`	Объект типа W http://msdn.micr множество дру Windows.In свойство, имен Windows.UI
`pointerType`	Источником со MSPOINTER_ (4). Вы можете устройства вво
`pointerId`	Уникальный и существования Windows.De http://msdn.micr с этим идинти http://msdn.micr описывает воз
`type`	Имя события,
`x, screenX, y, screenY`	Координаты X
`clientX, clientY`	Координаты X пользовательск
`offsetX, offsetY`	Координаты X элемента.
	Определяет кн кнопками). Ле

button	могут комбини нескольких кно
ctrlKey, altKey, shiftKey	Показывает, бы указателя.
hwTimestamp	Отметка време устройства.
relatedTarget	Предоставляет current event, на осуществлено
isPrimary	Показывает, яв указатель мыш
Свойства, присутствующие в зависимости от поддержки устройством (е	
width, height	Ширина конта
pressure	Нажим ручки,
rotation	Движение кур
tiltX	Наклон влево- поверхности),
tiltY	Наклон вперед поверхности) и
Методы	
currentPoint, getCurrentPoint	Предоставляет http://msdn.micro указателя ввод элементу (get
intermediatePoints,getIntermediatPoints	Предоставляет элементу (int

Весьма поучительно запустить вышеупомянутый пример на устройстве, поддерживающем несколько одновременных касаний, так как раздельная обработка каждого `pointerId` позволяет вам рисовать несколькими пальцами одновременно.

Захват указателя

Обычно, при событиях касания и поднятия указателя, устанавливают или снимают захват указателя элементом. Доступны следующие методы для поддержки этих действий каждым элементом DOM. Они применимы к каждому конкретному `pointerId`.

Таблица 3.4.

Метод	Описание
`msSetPointerCapture`	Захватывает `pointerId` для элемента, таким образом, события указателя поступают к этому элементу и не вызываются для других элементов (даже если указатель мыши находится вне этого элемента, внутри другого). `MSGotPointerCapture` так же будет вызвано для элемента.
`msReleasePointerCapture`	Прекращает захват, вызывает событие `MSLostPointerCapture`.
`msGetPointerCapture`	Возвращает элемент, захвативший указатель, если таковой имеется (в противном случае возвращает null).

Мы можем увидеть в примере "Ввод данных: обработка события указателя DOM", где устанавливается захват указателя в событии `MSPointerDown` и указатель освобождается в событии `MSPointerUp`:

```
this.MSPointerDown = function (evt) {
canvas.msSetPointerCapture(evt.pointerId);
// ...
};

this.MSPointerUp = function (evt) {
canvas.msReleasePointerCapture(evt.pointerId);
// ...
};
```

События жестов

Первое, что стоит узнать о событиях `MSGesture*`, это то, что они не вызываются автоматически, как события `click` и `MSPointer*`, и недостаточно просто добавить прослушиватель и завершить тем дело (как для `click`!). Вместо этого вам нужно выполнить некоторые настройки, которые сообщат системе, как именно должны происходить жесты, и вам нужно исползовать `MSPointerDown` для того, чтобы связать настройки жеста с конкретным `PointerId`. Это усложняет схему работы, делая возможным работу приложения с несколькими различными жестами, независимую их обработку, что похоже на то, как работают с событиями указателей. Представьте, например, приложение-головоломку (в некоторой мере оно представлено в одном из примеров "Примеры жестов" ниже), которое позволяет нескольким пользователям сидеть вокруг сенсорного экрана размером со стол и каждому из них работать с отдельной частью головоломки. Используя жесты, каждый из игроков может управлять отдельной частью (или двумя!), перемещать ее, вращать, возможно масштабировать для того, чтобы увидеть ее покрупнее, и, конечно, пробовать поместить кусок головоломки в нужное место. Для приложений Магазина Windows, написанных на JavaScript, полезно то, что приращения манипуляции для сконфигурированных элементов – для переноса, вращения и масштабирования – даются в координатном пространстве родительского элемента, что означает, что довольно просто перенести манипуляции в CSS-трансформации и таким образом сделать манипуляции видимыми. Коротко говоря, это означает большую гибкость там, где она нужна вам. Если вам это не нужно, вы можете использовать жесты простейшим способом. Посмотрим, как все это работает.

Первый шаг к приему событий жестов заключается в создании объекта `MSGesture` (ссылка: http://msdn.microsoft.com/en-us/library/windows/apps/hh968035.aspx) и в связывании его с элементом, для которого вы хотите принимать события. В упражнении PointerEvents этому элементу дано имя divElement, вам нужно сохранить этот элемент в свойстве жеста `target` и сохранить объект жеста в свойстве

элемента `gestureObject` для использования его событием `MSPointerDown`:

```
var gestureObject = new MSGesture();
gestureObject.target = divElement;
divElement.gestureObject = gestureObject;
```

Установив эти связи, вы затем можете добавить прослушиватель события, как обычно. Пример кпоказывает полный набор из шести событий жестов:

```
divElement.addEventListener("MSGestureTap", gestureTap);
divElement.addEventListener("MSGestureHold", gestureHold);

divElement.addEventListener("MSGestureStart", gestureStart);
divElement.addEventListener("MSGestureChange", gestureChange);
divElement.addEventListener("MSGestureEnd", gestureEnd);
divElement.addEventListener("MSInertiaStart", inertiaStart);
```

Однако сейчас работа еще не завершена. Если это все, что вы сделаете в своем коде, вы еще не получите этих событий, так как каждый жест нужно связать с указателем. Это делается в обработчике события `MSPointerDown`:

```
function pointerDown(e) {
//Связываем данный указатель с целевым жестом
e.target.gestureObject.addPointer(e.pointerId);
}
```

Для того чтобы получить возможность выполнять жесты вращения и изменения масштабирования с помощью колеса мыши (а это следует сделать), добавьте обработчик для события `wheel`, установите `pointerId` для этого события в 1 (фиксированное значение для колесика мыши) и отправьте то, что получилось в ваш обработчик `MSPointerDown`:

```
divElement.addEventListener("wheel", function (e) {
e.pointerId = 1;  // Фиксированный pointerId для MouseWheel
pointerDown(e);
```

```
});
```

Теперь события жестов готовы к поступлению в данный элемент. (Помните, что события колесика мыши, сами по себе, транслируются, Ctrl+колесо мыши – это масштабирование, Shift+Ctrl+колесо мыши – вращение). Более того, если возникнет дополнительное событие `MSPointerDown` для того же самого элемента с различными значениями `pointerId`, метод `addPointer` включит этот новый указатель в жест. Это автоматически активирует жесты изменения масштаба и вращения, которые основаны на нескольких точках касания.

Если вы запустите упражнение PointerEvents (установив флаги Ignore Mouse Events (Игнорировать события мыши) и Ignore Pointer Events (Игнорировать события указателя)), и начнете касаться экрана, касаться, удерживая палец на экране, выполнять касание и короткое перетаскивание (с помощью сенсорного экрана или мыши), вы увидите выходные данные, похожие на те, что приведены на рис.3.2

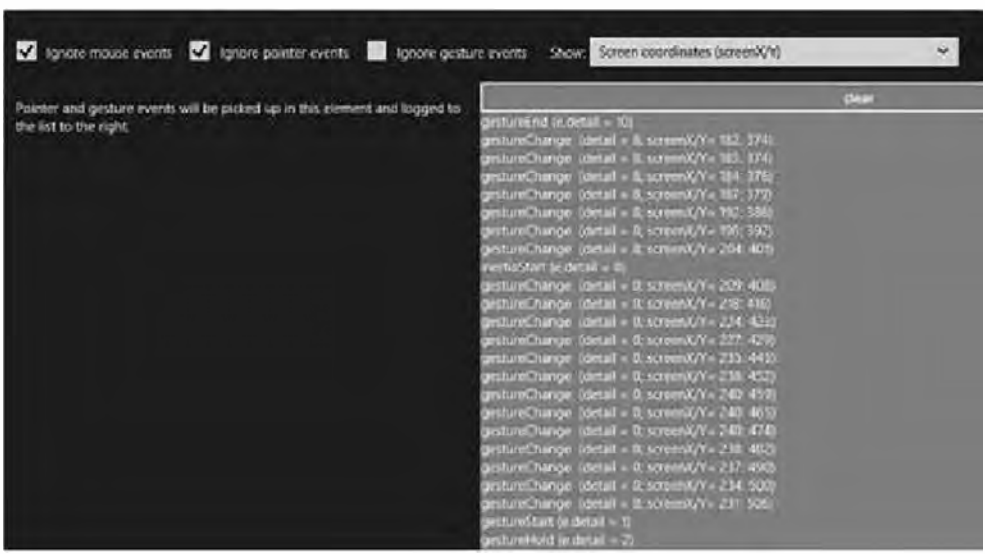
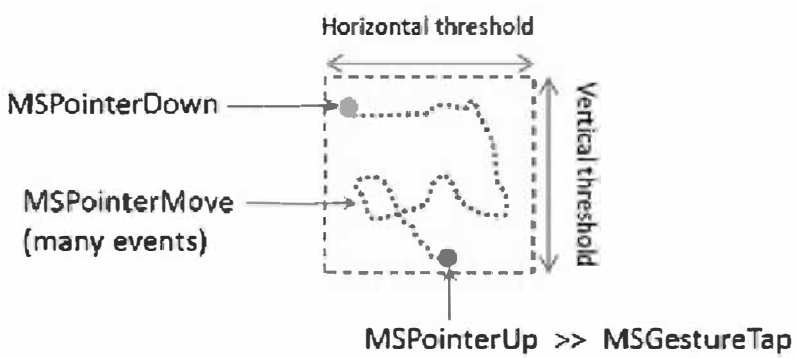


Рис. 3.2. Упражнение PointerEvents, выходные данные для событий жестов (скриншот слегка обрезан для выделения деталей)

Снова хочу отметить, что события жестов вызываются в ответ на последовательность событий указателя, предлагая высокоуровневую интерпретацию низкоуровневых событий указателя. В процессе

интерпретации происходит дифференцирование событий касания/удержания от событий начала/изменения/окончания, выясняется, как и когда началось событие `MSInertiaStart`, и что предпринимает распознаватель жестов, когда объекту `MSGesture` передаются несколько точек.

Начиная с жеста одного указателя, первый аспект дифференцирования – это порог перемещения указателя (pointer movement threshold). Когда распознаватель жестов обнаруживает событие `MSPointerDown`, он начинает наблюдать за событиями `MSPointerMove` для того, чтобы видеть, остаются ли они внутри порогового расстояния, которое является эффективной границей для событий касания и удерживания. Это предназначено для того, чтобы игнорировать небольшое дрожание мыши или точки касания, как показано на рисунке ниже (я бы сказал, что здесь это несколько преувеличено!), когда происходит касание, небольшое перемещение, после чего указатель поднимается, генерируется событие `MSGestureTap`:



Horizontal threshold (Горизонтальный порог перемещения)

Vertical threschold (Вертикальный порог перемещения)

Many events (Множество событий)

Различить события `MSGestureTap` и `MSGestureHold` позволяет временной порог (time threshold): `MSGestureTap` происходит, когда за событием `MSPointerDown` следует событие `MSPointerUp` в пределах временного порога.

MSGestureHold происходит, когда за событием MSPointerDown следует событие MSPointerUp с превышением временного порога. В подобном случае, один раз, при пересечении временного порога, вызывается событие MSGestureHold с eventArgs.detail установленным в 1 (MSGESTURE_FLAG_BEGIN). При условии, что указатель все еще находится в пределах порога перемещения, событие MSGestureHold вызывается еще раз, когда происходит MSPointerUp, с eventArgs.detail установленным в 2 (MSGESTURE_FLAG_END). Вы можете увидеть эти детали в двух первых событиях на рис. 3.2. выше.

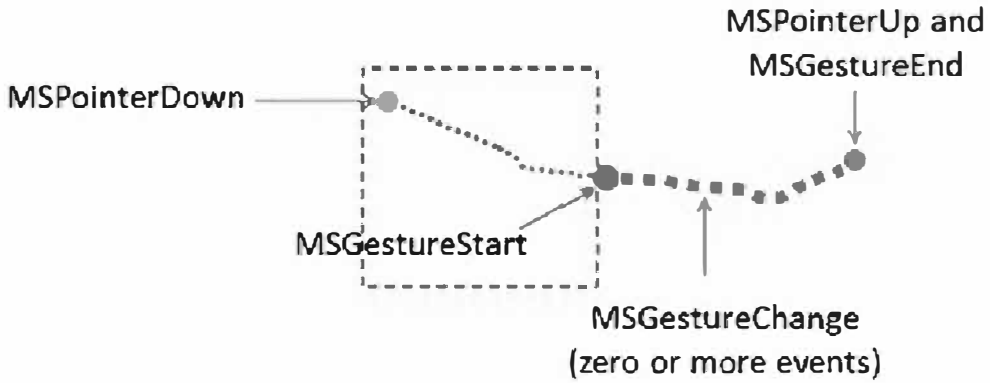
Флаги жестов в значении eventArgs.detail сопровождаются множеством других свойств, касающихся расположения и перемещения в объекте eventArgs, как показано в следующей таблице:

Таблица 3.5.

Свойства	Описание
screenX, screenY	Координаты X и Y центральной точки жеста, спозиционированные
clientX, clientY	Координаты X и Y центральной точки жеста, спозиционированные относительно пользовательской области приожения.
offsetX, offsetY	Координаты X и Y центральной точки жеста, спозиционированные
translationX, translationY	Перенос по осям X и Y.
velocityX, velocityY	Скорость перемещения по осям X и Y.
scale	Коэффициент масштабирования (процентное изменение масштаба).
expansion	Скорость расширения области манипуляции.
rotation	Угол поворота в радианах.
velocityAngular	Угловая скорость в радианах.
	Содержит флаги жеста, которые описывают состояние события жеста. Эти флаги определены как значения в eventArgs:

detail	• eventArgs.MSGESTURE_FLAG_NONE (0): Указывает на текущий жест, такой, как MSGestureChange где представлены изменения координат. • eventArgs.MSGESTURE_FLAG_BEGIN (1): Начало последовательности жестов. Есл взаимодействие включает в себя единственное событие, такое, как MSGestureTap, и флаг MSGESTURE_FLAG_BEGIN, и флаг MSGESTURE_FLAG_END будут установлены (значение 3). • eventArgs.MSGESTURE_FLAG_END (2) Конец последовательности жестов. Опять же, если взаимодействие включает в себя единственный жест, такой, как MSGestureTap, и флаг MSGESTURE_FLAG_BEGIN, и флаг MSGESTURE_FLAG_END будут установлены (значение 3). • eventArgs.MSGESTURE_FLAG_CANCE (4): Жест был отменен.
hwTimestamp	Отметка времени, когда указатель был назначен системой при поступлении входных данных от устройства.

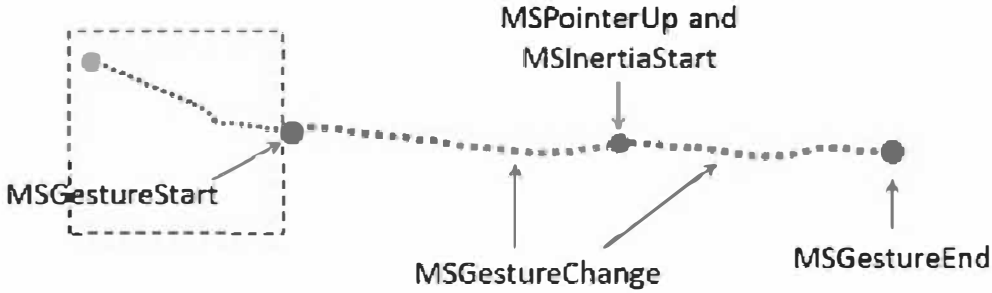
Многие из этих свойств становятся гораздо интереснее, когда указатель выходит за пределы порога перемещения, после чего уже не произойдет события касания (tap) или удерживания (hold). Вместо этого, как только указатель пересечет порог, будет выполнено событие MSGestureStart, после чего возможен вызов событий MSGestureChange, их может не быть вовсе, но обычно их довольно много. В конце вызывается одно событие MSGestureEnd:



Обратите внимание на то, что если указатель не пересечет порог перемещения достаточно долго для вызова первого `MSGestureHold` с флагом `MSGESTURE_FLAG_BEGIN`, но затем указатель переместится за пределы порога, `MSGestureHold` будет вызван второй раз, с флагами `MSGESTURE_FLAG_CANCEL | MSGESTURE_FLAG_END` в `eventArgs.detail` (значение 6), за ним последует `MSGestureStart` с флагом `MSGESTURE_FLAG_BEGIN`. Подобная последовательность позволяет дифференцировать событие удержания (hold) от жестов прокрутки или перетаскивания, даже если пользователь какое-то время удерживал элемент на месте.

События `MSGestureStart`, `MSGestureChange`, и `MSGestureEnd` все вместе задают манипуляцию (manipulation) для элемента, с которым связан жест, когда указатель контактирует с элементом в течение манипуляции. Технически это означает, что указатель больше не перемещается, когда он освобождается.

Если указатель продолжает перемещаться после того, как он освобожден, тогда мы переключаемся с манипуляции на движение по инерции (inertial motion). В этом случае вызывается событие `MSInertiaStart`, оно показывает, что указатель продолжает несмотря на то, что контакт с элементом потерян, или указатель поднят. В результате, вы продолжаете получать события `MSGestureChange` до тех пор, пока движение не будет завершено:



С концептуальной точки зрения разница между манипуляцией и движением по инерции проиллюстрирована на рис. 3.3. Кривые, показанные здесь, не обязательно отражают реальные изменения между сообщениями. Если указатель переместился вдоль зеленой линии и болше не двигается после отпускания указателя, мы видим последовательность жестов, которая описывает манипуляцию. Если указатель отпущен при движении, мы видем событие MSInertiaStart между сериями событий MSGestureChange, и последовательность событий продолжается в виде оранжевой линии.

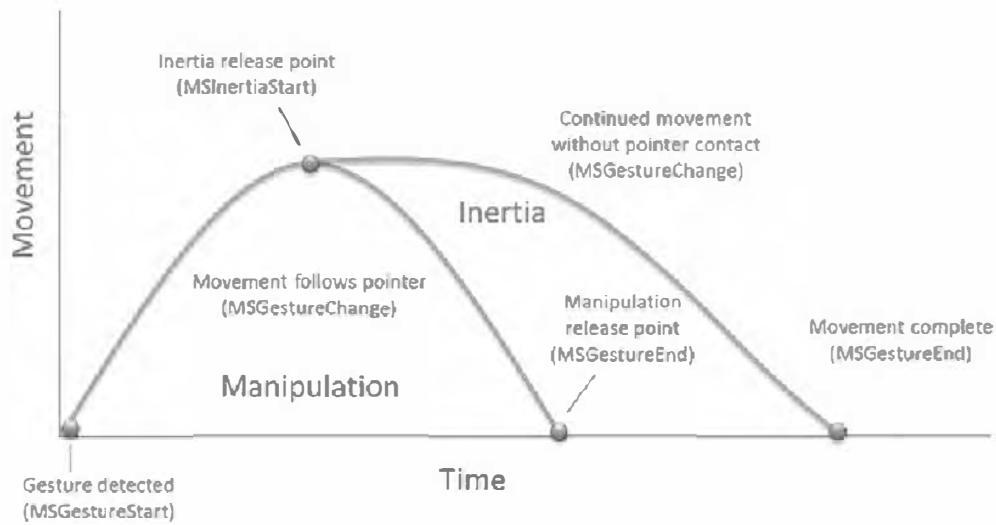


Рис. 3.3. Концептуальное представление манипуляции (зеленая линия) и движения по инерции (оранжевая линия).

- Перемещение (Movement)
- Время (Time)
- Обнаружен жест (Gesture detected)

- Манипуляция (Manipulation)
- Инерция (Inertia)
- Точка освобождения инерции (Inertia release point)
- Перемещение следует за указателем (Movement follows pointer)
- Точка освобождения манипуляции (Manipulation release point)
- Перемещение продолжается, без контакта с указателем (Continued movement without pointer contact)
- Перемещение завершено (Movement complete)

Возвращаясь к рис. 3.2., где выпадающий список Show (Показ) был установлен в значение Velocity (Скорость), выходные данные для событий `MSGestureChange` включали значения `eventArgs.velocity*`. В течение манипуляции, скорость может меняться с любой частотой, в зависимости от того, как движется указатель. Однако когда начинается движение по инерции, скорость постепенно падает до нуля, и тогда происходит событие MSGestureEnded. Количество событий изменения зависит от того, сколько времени нужно для того, чтобы продолжать замедляющееся, вплоть до остановки, движение, но если вы просто перемещаете элемент по экрану с помощью этих событий, несущих информацию об изменениях, пользователь увидит приятную плавную анимацию. Вы можете поэкспериментировать с этим в упражнении PointerEvents, используя выпадающий список Show для того, чтобы посмотреть, как на другие свойства позиционирования влияют различные жесты при манипуляциях и движении по инерции.

1) Как для событий объектов WinRT, для них справедливы соображения, изложенные в лекции 3, в разделе "События WinRT и removeEventListener"

2) Вариант `double-tap-zoom` не поддерживается приложениями для Магазина Windows.

Многоточечные жесты

Все, о чем мы говорили до сих пор, касалось жестов с одной точкой касания, но то же самое справедливо и для многоточечных жестов. Также в лекции рассмотрены ввод данных с клавиатуры и рукописный ввод.

Когда объект `MSGesture` получает несколько указателей через свое событиет `addPointer`, он так же запускает события `MSGestureStart`, `MSGestureChange`, `MSGestureEnd` для жестов поворота и масштабирования, вместе с событием `MSInertiaStart`. В подобных случаях свойства `scale`, `rotation`, `velocityAngular`, `expansion`, и `velocityExpansion` в объекте `eventArgs` приобретают смысл.

Вы можете выборочно просматривать эти свойства для событий `MSGestureChange` посредством выпадающего списка, который расположен в правом верхнем углу, в упражнении PointerEvents. Вы можете заметить, что, когда вы выполняете многоточечные жесты в имитаторе Visual Studio, вы никогда не видите событие `MSGestureTap` для отдельных точек касания. Это происходит потому, что распознаватель жестов может видеть подобные множественные события `MSPointerDown`, которые происходят почти одновременно (здесь играет роль свойство `hwTimestamp`) и тут же объединяет их в одно событие `MSGestureStart` (например, начинает жест масштабирования или вращения).

Теперь я уверен, что вы готовы задать несколько важных вопросов. До сих пор я говорил о сенсорных жестах изменения масштаба и вращения, о жестах переноса, как о рахных вещах, но как, на самом деле, различать их, если все они поступют в приложение посредством одного и того же события `MSGestureChange`? Разве это не запутывает все? Какова стратегия для работы с жестами переноса, вращения и масштабирования?

Хорошо, ответ заключается в том, что вам не нужно их разделять! Что, если вы подумаете немного о том, как вы обрабатываете события `MSGestureChange` и данные, которые каждое из них содержит в зависимости от вида манипуляции, который вы поддерживаете в

пользовательском интерфейсе:

- Если вы поддерживаете лишь перенос элемента (translation), вы просто никогда не обращаете внимание на свойства наподобие `scale` и `rotation` и применяете только необходимые, такие, как `translationX` и `translationY`. Это было бы ожидаемым поведением при выборе элемента в элементе управления коллекцией, например (или в элементе управления, который позволяет манипулировать элементами по технологии "перетащить и опустить", меняя порядок их расположения).
- Если вы поддерживаете лишь изменение масштаба (scale), вы игнорируете все свойства, которые имеют отношение к позиционированию и работаете со свойствами `scale`, `expansion`, и/или со свойством `velocityExpansion`. Это будет тем поведением, которое вы хотите реализовать для элемента управления, который поддерживает оптическое или семантическое масштабирование.
- Если вас интересует лишь вращение (rotation), то ваши друзья – это свойства `rotation` и `velocityAngular`.

Конечно, если вы хотите поддерживать несколько видов манипуляций, вы можете просто реализовать обработку всех этих свойств, передавая их, например в CSS-трансформацию. Подобное может быть применимо к приложениям, которые реализуют различные манипуляции для экранных объектов, и именно этому посвящен один из примеров Windows SDK, рассматривающий жесты.

Ввод данных и жесты, допускающие создание экземпляров

В то время, как упражнение PointerEvents, включенное в эту лекцию, давало нам низкоуровневый вид событий указателей и жестов, для приложений по-настоящему важно то, как применять эти события для реальной манипуляции экранными объектами, что является реализацией частей языка касаний, такими, как жесты сжатия/растяжения и вращения. Для того, чтобы увидеть это, мы можем обратиться к примеру "Ввод: жесты, допускающие создание экземпляров" (ссылка: http://code.msdn.microsoft.com/windowsapps/Input-

Instantiable-deda69ca).

Этот пример, в основном, показывает, как использовать события жестов одновременно с несколькими элементами. В Сценарии 1 и 2, приложение реализует простой пример головоломки, как упоминалось выше. Каждым закрашенным прямоугольником можно управлять по отдельности, используя перетаскивание для перемещения (с инерцией и без), жест сжатия-растяжения для изменения масштаба, и жест вращения для вращения, как показано на рис. 4.1.

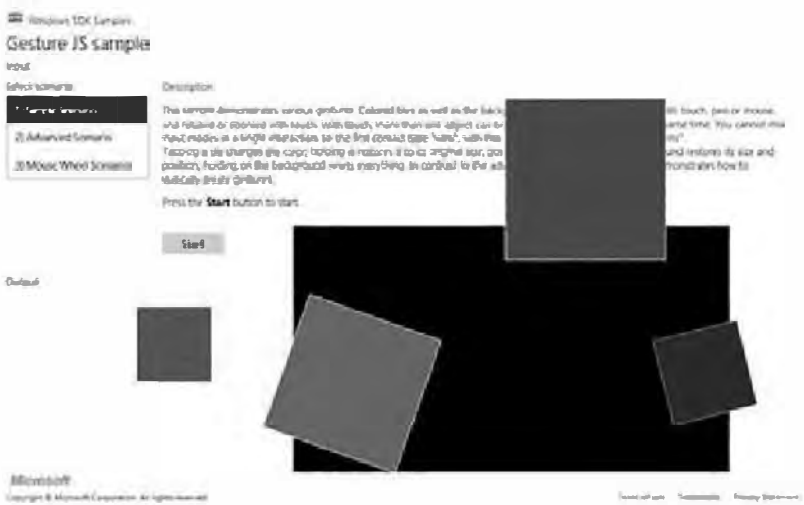


Рис. 4.1. Пример "Ввод: жесты, допускающие создание экземпляров" после того, как с ним немного поработали. Словосочетание "допускающие создание экземпляров" (instantiable) использовано потому, что нам нужно создать экземпляр (instance) объекта MSGesture для приема событий жестов

В Сценарии 1 (js/instantiableGesture.js) объект `MSGesture` создается для кажого экранного элемента вместе с еще одним для черного фона, "стола" при инициализации (в функции `initialize`). Мы уже видели подобное. Похожим бразом, обработчик MSPointerDown (onPointerDown) добавляет указатели к объектам жестов для каждого элемента, уделяя немного больше внимания обработке их положения по оси Z. Это предовращает одновременные касания, указатели мыши и пера работают с тем же самым элементом (что было бы странно!):

```
function onPointerDown(e) {
if (e.target.gesture.pointerType === null) {
 // Первый контакт e.target.gesture.addPointer(e.pointerId);
 // Присоединение указателя к элементу
e.target.gesture.pointerType = e.pointerType;
}
else if (e.target.gesture.pointerType === e.pointerType)
{ // Контакты сходного типа e.target.gesture.addPointer(e.pointerId);
   // Присоединение указателя к элементу
}

// ZIndex изменяется при опускании указателя.
// Элемент, на который опустился указатель, становится самым верхним
var zOrderCurr = e.target.style.zIndex;
var elts = document.getElementsByClassName("GestureElement");
for (var i = 0; i < elts.length; i++) {
if (elts[i].style.zIndex === 3) {
elts[i].style.zIndex = zOrderCurr;
}
e.target.style.zIndex = 3;
}
}
```

Обработчик `MSGestureChange` выполняется для каждого отдельного фрагмента (`onGestureChange`), затем собирает все данные по переносу, вращению, масштабированию в объекте eventArgs и применяет их с помощью CSS. Это показывает, как удобно то, что все эти свойства уже описаны в нужной нам системе координат:

```
function onGestureChange(e) {
var elt = e.target;
var m = new MSCSSMatrix(elt.style.msTransform);

 elt.style.msTransform = m.
translate(e.offsetX, e.offsetY).
translate(e.translationX, e.translationY).
rotate(e.rotation * 180 / Math.PI).
scale(e.scale).
```

```
translate(-e.offsetX, -e.offsetY);
}
```

В коде примера выполняется немного больше всего, но то, что мы показали здесь – это самые важные части. Очевидно, если вам не нужна поддержка манипуляций определенного вида, вы можете просто игнорировать определенные свойства в объекте `eventArgs`.

Сценарий 2 этого примера формирует такое же изображение, но он реализован несколько иначе. Как вы можете видеть, в его функции `initialize` (js/gesture.js), изначально регистрируется лишь одно событие, применяющееся к "столу", который имеет черный фон и окружающую его границу.

Объекты жестов для отдельных фрагментов головоломки создаются и прикрепляются к указателям в событии `MSPointerDown` (`onTableTopPointerDown`). Этот подход более эффективен и масштабируем для приложения-головоломки, которое может иметь сотни или даже тысячи фрагментов, так как объекты жестов хранятся лишь до тех пор, пока осуществляется манипуляция конкретным фрагментом. Эти манипуляции так же похожи на те, что были в Сценарии 1, где свойства `MSGestureChange` применялись посредством CSS-трансформаций. Для того чтобы узнать подробности, обратитесь к комментариям к коду в js/gesture.js, так как они весьма обширны.

Сценарий 3 этого примера предоставляет еще одну демонстрацию выполнения жестов переноса, сжатия-расширения и вращения с использованием колеса мыши. Как показано в упражнении PointerEvents, единственное, что нужно для того, чтобы сделать это – это обработать событие `wheel`, установить `eventArgs.pointerId` в 1 и передать это в обработчик `MSPointerDown`, который затем добавит указатель к объекту жеста:

```
function onMouseWheel(e) {
e.pointerId = 1; // Фиксированный pointerId для MouseWheel
onPointerDown(e);
}
```

Опять же, это все, что нужно. (Мне нравится, когда все так просто!). В качестве упражнения, вы можете попытаться добавить этот код в Сценарии 1 и 2.

Средство распознавания жестов

В случае с инерционными жестами, которые продолжают посылать некоторое количество событий `MSGestureChange`, когда указатель освобожден, вы можете задаться вопросами: "А что, на самом деле, управляет этими событиями? Очевидно, существует особая модель уменьшения скорости, встроенная в эти события, а именно та, на основе которой построены внешний вид и впечатления от взаимодействия с Windows. Но что, если мне нужно другое поведение? И что, если я хочу совсем иначе интерпретировать события указателя?"

Вспомогательный механизм, который интерпретирует события указателя, превращая их в жесты, называется средством распознавания жестов (gesture recognizer). Его вы можете получить напрямую посредством объекта `Windows.UI.Input.GestureRecognizer` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.ui.input.gesturerecognize После создания экземпляра этого объекта с помощью команды `new`, вы можете устанавливать его свойства `gestureSettings` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.ui.input.gesturerecognize для тех видов манипуляций и жестов, которые вам интересны. Документация по Windows.UI.Input.GestureSettings предоставляет полный набор возможностей, в том числе, такие жесты, как `tap`, `doubleTap`, `hold`, `holdWithMouse`, `rightTap`, `drag`, переносы, вращения, масштабирование, перемещение с инерцией, и `crossSlide` (скольжение по диагонали). Например, в примере "Ввод: манипуляции и жесты" (ссылка: http://code.msdn.microsoft.com/windowsapps/Manipulations-and-gestures-26918bb3) (ballineight.js) мы можем видеть, как настраивают средство распознавания жестов для жестов касания (tap), вращения (rotate), переноса (translate) и масштабирования (scale) (с инерцией):

```
gr = new Windows.UI.Input.GestureRecognizer();

// Настройка GestureRecognizer для определения манипуляций вращения
// с инерцией для данных трех компонентов манипуляции и обработки
gr.gestureSettings =
Windows.UI.Input.GestureSettings.manipulationRotate |
Windows.UI.Input.GestureSettings.manipulationTranslateX | Windows.UI.Input.(
Windows.UI.Input.GestureSettings.manipulationScale |
Windows.UI.Input.GestureSettings.manipulationRotateInertia | Windows.UI.Inpu
Windows.UI.Input.GestureSettings.manipulationTranslateInertia |
Windows.UI.Input.GestureSettings.tap;

// Выключение обратной связи пользовательского интерфейса для жесто
// (обратная связь все еще используется для PointerPoints)
gr.showGestureFeedback = false;
```

Объект `GestureRecognizer` так же имеет множество свойств для настройки его специфических событий. Например, для жестов скольжения по диагонали (cross-slides) вы можете задать свойства `crossSlideThresholds`, `crossSlideExact`, и `crossSlideHorizontally`. Вы можете задать быстроту снижения скорости (в пикселях/мс^2) посредством свойств `inertiaExpansionDeceleration`, `inertiaRotationDeceleration`, и `inertiaTranslationDeceleration`.

После настройки, вы начинаете передавать событий MSPointer* к объекту средсва распознавания жестов, в частности, его методам, которые называются `processDownEvent`, `processMoveEvents`, и `processUpEvent` (так же, если нужно, `processMouseWheelEvent`, и `processInertia`). В ответ, в зависимости от конфигурации, средство распознавания жестов запустит некоторое количество собственных событий. Во-первых, это дискретные события, наподобие `crossSliding`, `dragging`, `holding`, `rightTapped`, и `tapped`. Для остальных он запустит серию событий `manipuationStarted`, `manipulationUpdated`, `manipulationInertiaStarting`, и `manipulationCompleted`. Обратите внимание на то, что все эти

события исходят из WinRT, поэтому убедитесь, что вызвали, когда нужно, `removeEventListener`.

Когда вы используете средство распознавания жестов напрямую, другими совами, прослушиваете события `MSPointer*`, передаете их средству распознавания жестов и затем прослушиваете его специфические события и действуете, опираясь на них, как в вышеприведенном примере, вместо работы с событиями `MSGesture*`, которые исходят от средства распознавания жестов по умолчанию, сконфигурированного с помощью объекта `MSGesture`.

Обратитесь к документации `Windows.UI.Input.GestureRecognizer` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.ui.input.gesturerecognize для того, чтобы узнать подробности и к примерам, чтобы посмотреть на образцы кода. В качестве дополнительного примера, вот фрагмент кода, который перехватывает небольшие горизонтальные перемещения с использованием установки `manipulationTranslateX`:

```
var recognizer = new Windows.UI.Input.GestureRecognizer();
recognizer.gestureSettings = Windows.UI.Input.GestureSettings.manipulationTrar
var DELTA = 10;

myElement.addEventListener('MSPointerDown', function (e) {
recognizer.processDownEvent(e.getCurrentPoint(e.pointerId));
});
myElement.addEventListener('MSPointerUp', function (e) {
recognizer.processUpEvent(e.getCurrentPoint(e.pointerId));
});
myElement.addEventListener('MSPointerMove', function (e) {
recognizer.processMoveEvents(e.getIntermediatePoints(e.pointerId));
});

// Помните, что при работе с этими событиями нужен removeEventListen
recognizer.addEventListener('manipulationcompleted', function (args) {
var pt = args.cumulative.translation;
if (pt.x < -DELTA) {
// переместить вправо
}
```

```
else if (pt.x > DELTA) {
// переместить влево
}
});
```

Помимо использования средства распознавания жестов, помните, что вы всегда можете перейти на более низкий уровень и выполнить собственную обработку событий `MSPointer*` , если вам это нужно, и, таким образом, полностью обойти средство распознавания жестов. Это может быть нужно, если конфигурации, допустимые в средстве распознавания жестов, не соответствуют вашим особенным нуждам. В то же время, сейчас неплохая возможность для того, чтобы перечитать врезку "Создание совершенно новых жестов?" в конце раздела о языке касаний. Она посвящена некоторым вопросам о том, когда действительно нужны пользовательские жесты.

Ввод данных с клавиатуры и программная клавиатура

После всего, что мы узнали о сенсорном вводе данных и о других способах ввода, может показаться, что рассмотрение скромной клавиатуры нас разочарует. Но, все же, клавиатура остается крайне важной для ввода текста, будь это физическая клавиатура или экранная "программная" клавиатура. Она особенно важна для обеспечения специальных возможностей, так как некоторые пользователи физически не могут использовать мышь или другие устройства. На самом деле, раздел 3.5. "Сертификационных требований к приложениям для Windows 8" (ссылка: http://msdn.microsoft.com/library/windows/apps/hh694083.aspx) говорит об обязательной поддержке приложениями ввода с клавиатуры.

К счастью, в приложениях для Магазина Windows нет ничего особенного в обработке ввода с клавиатуры, но длинный путь можно пройти короткими шагами. Вот общие положения, взятые из материала "Реализация специальных возможностей клавиатуры" (ссылка: http://msdn.microsoft.com/library/windows/apps/hh700327.aspx):

- Обрабатывайте события `keydown`, `keyup` и `keypress` так,

как вы уже знаете, особенно для реализации специальных сочетаний клавиш. Смотрите раздел "Стандартные сочетания клавиш" далее в этом разделе для того, чтобы взглянуть на типичные сопоставления команд.

- Задавайте атрибут `tabindex` интерактивным элементам, которые должны поддерживать возможность переключения на них с помощью клавиши табуляции. Старайтесь не добавлять `tabindex` к неинтерактивным элементам, так как это может помешать программам чтения с экрана.
- Задавайте атрибут `accesskey` для тех элементов, которым следует иметь клавиатурные сокращения.
- Вызывайте DOM API для задания фокуса для элементов, которые должны быть элементами по умолчанию
- Используйте преимущества клавиатурных сокращений, которые уже присутствуют во встроенных элементах управления, в таких, как панель приложения.

В приложении "Here My Am!", над которым мы работаем в этом курсе, обновленный вариант которого содержится в дополнительных материалах к этой лекции, добавлена полная поддержка клавиатуры. Это, в основном, коснулось добавления `tabindex` нескольким элементам, установки фокуса на область изображения и перехват событий `keydown` элемента `img` для клавиш Enter и Пробел, где мы уже обрабатываем событие `click`. В случае с этими событиями `keydown`, обратите внимания на полезность использования перечисления `WinJS.Utilities.Key` (ссылка: http://msdn.microsoft.com/library/windows/apps/br211775.aspx) для сравнения кодов клавиш:

```
var Key = WinJS.Utilities.Key;
var image = document.getElementById("photo");

image.addEventListener("keydown", function (e) {
if (e.keyCode == Key.enter || e.keyCode==Key.space) {
image.click();
}
});
```

Все это работает и для физической клавиатуры, и для программной. Дело закрыто? Не вполне. Есть две особенности, касающиеся экранной клавиатуры: первая – это как показать ее, вторая – как ее появление подействует на макет приложения. В конце этого раздела мы так же взглянем на стандартные клавиатурные сокращения для команд приложения.

Показ программной клавиатуры происходит по одной и только одной причине: пользователь касается элемента управления для ввода текста с атрибутом `contenteditable="true"` (такого, как `div` или `canvas`). Нет API, которое отображает клавиатуру, она не появляется и тогда, когда вы щелкаете по подобному элементу управления мышью или пером, или перемещаетесь к нему, нажимая клавишу табуляции на физической клавиатуры.

Конфигурация клавиатуры зависит от типа элемента управления для ввода текста. Мы можем увидеть это в Сценарии 2 примера "Ввод: сенсорная клавиатура для ввода текста" (ссылка: http://code.msdn.microsoft.com/windowsapps/Input-Touch-keyboard-text-f86e9bd9), где в файле html/ScopedView.html содержится несколько элементов управления `input` (окружающая разметка опущена), которые выглядят так, как показано на рис. 4.2.

```
<input type="url" name="url" id="url" size="50" />
<input type="email" name="email" id="email" size="50" />
<input type="password" name="password" id="password" size="50" />
<input type="text" name="text" id="text" size="50" />
<input type="number" name="number" id="number" />
<input type="search" name="search" id="search" size="50" />
<input type="tel" name="tel" id="tel" size="50" />
```

Рис. 4.2. Программная клавиатура появляется, когда вы касаетесь поля ввода, как показано в примере

То, что показано на рис. 4.2. – это клавиатура по умолчанию. Если вы прикоснетесь к полю Search (Поиск), вы получите практически то же самое, за исключением того, что клавиша Enter превратится в клавишу Search (Поиск). Для поля Email клавиатура тоже похожа на стандартную, за исключением того, что появятся кнопки @ и .com у клавиши пробела:

Клавиатура для ввода URL выглядит так же, но клавиши @ нет, а Enter превращается в Go (Перейти):

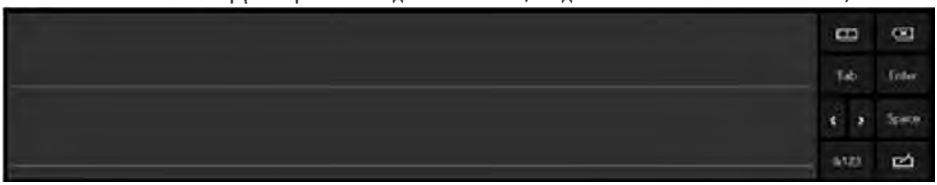
Для поля ввода пароля на клавиатуре появляется клавиша Hide keypresses (Скрыть нажатия клавиш), которая отключает видимую

анимацию нажатий клавиш – очень полезная функция, если вы записываете видео!

И, наконец, поля Number (Число) и Telephone (Телефон) вызывают клавиатуру, ориентированную на ввод чисел:

Во всех этих случаях кнопка в нижнем правом углу (значок которой очень похож на клавиатуру) позволяет вам переключаться в другие режимы отображения клавиатуры:

Доступные возможности здесь – это нормальная (широкая) клавиатура, разделенная клавиатура, панель распознавания рукописного ввода, и кнопка для того, чтобы полностью скрыть клавиатуру. Вот как выглядят разделеная клавиатура и панель для ввода рукописного текста:



Панель для рукописного ввода, это просто еще один режим программной клавиатуры: вы можете переключаться между режимами, и ваш выбор сохраняется между вызовами. (По этой причине, Windows не активирует автоматически панель рукописного ввода для указателя ручки, так как пользователь может предпочитать использование программной клавиатуры даже с пером).

Так же клавиатура настроаивает свой внешний вид при работе с элементами управления для ввода текста для предоставления подсказок и выделения появляющейся кнопки Insert (Вставить). Это показано в Сценарии 1 примера:

Настройка макета для работы с программной клавиатурой

Вторая задача, касающаяся программной клавиатуры (да, я не забыл!) заключается в поддержке макета, когда клавиатура может перекрыть поле для ввода, имеющее фокус.

Когда появляется программная клавиатура или панель для ввода рукописного текста, система пытается убедиться в том, что поле ввода видимо, прокручивая содержимое, если это возможно. Это значает просто установку отрицательного вертикального смещения для всей

вашей страницы в соответствии с высотой программной клавиатуры. Например, если я добавлю (в качестве радикального решения!) кучу элементов </br> в верхнюю часть страницы html/ScopedView.html в примере, так, чтобы элементы управления для ввода текста оказались внизу страницы, и затем коснусь одного из них, вся страница сдвинется вверх, как показано на рис. 4.3.:

Рис. 4.3. Когда появляется программная клавиатура, Windows автоматически смещает страницу приложения вверх для того, чтобы поле ввода не оказалось перекрытым

Хотя такой подход может быть самым простым решением этой задачи, он не всегда идеален. К счастью, вы можете сделать кое-что более интеллектуальное, если хотите, прослушивая события `hiding` и `showing` объекта `Windows.UI.ViewManagement.InputPane` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.ui.viewmanagement.inpu и непосредственно подстраивая макет. Код для того, чтобы это сделать, можно найти (вы готовы к этому?) в примере "Реагирование на отображение экранной клавиатуры" (ссылка: http://code.msdn.microsoft.com/windowsapps/Keyboard-Events-Sample-866ba41c) [1]. К этим событиям очень просто добавить прослушиватели (смотрите в нижней части в js/keyboardPage.js, который, так же, соответствующим образом удаляет прослушиватели):

```
var inputPane = Windows.UI.ViewManagement.InputPane.getForCurrentView()
 inputPane.addEventListener("showing", showingHandler, false);
inputPane.addEventListener("hiding", hidingHandler, false);
```

В обработчике события `showing` объект

`eventArgs.occludedRect` (имеющий тип `Windows.Foundation.Rect`) дает вам координаты и размеры обрасти, которую перекроет экранная клавиатура. В ответ на это мы можете настроить любые необходимые параметры макета и установить свойство `eventArgs.ensuredFocusedElementInView` в `true`. Это сообщит Windows, что автоматический сдвиг выполнять не нужно:

```
function showingHandler(e) {
if (document.activeElement.id === "customHandling") {
keyboardShowing(e.occludedRect);

// Будьте осторожны с этим свойством. Как только оно установлено, си
// не будет делать ничего для того, чтобы элемент, имеющий фокус, был
e.ensuredFocusedElementInView = true;
}
}
```

В примере показаны оба случая. Если вы коснетесь расположенного в нижней левой части приложения элемента defaultHandling, который имеет цвет морской волны, как показано на рис. 4.4, его `showingHandler` ничего не будет делать, и мы увидим поведение по умолчанию

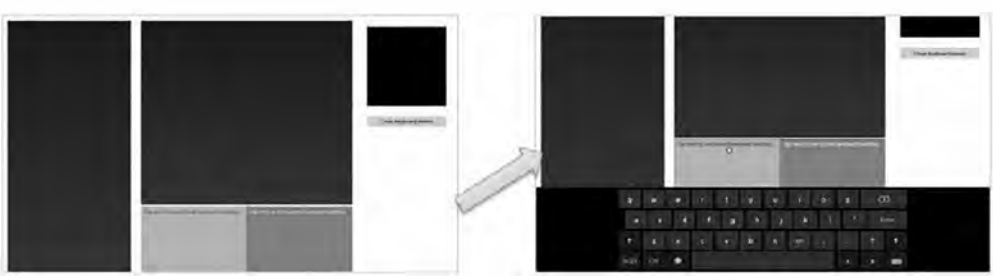


Рис. 4.4. Прикосновение к левому элементу defaultHandling в нижней части приведет к стандартному поведению при появлении клавиатуры, при котором содержимое страницы сдвигается вертикально

Если вы коснетесь элемента customHandling (справа), он вызовет функцию `keyboardShowing` для настройки макета:

```
function keyboardShowing(keyboardRect) {
// Часть кода опущена...
var elementToAnimate = document.getElementById("middleContainer");
var elementToResize = document.getElementById("appView");
var elementToScroll = document.getElementById("middleList");

// Записываем расстояние перемещения. Это упрощает расчеты
displacement = keyboardRect.height;
var displacementString = -displacement + "px";

// Находим последний видимый элемент в списке
var bottomOfList = elementToScroll.scrollTop + elementToScroll.clientHeight;

// Анимируем
showingAnimation = KeyboardEventsSample.Animations.inputPaneShowing(eler
{ top: displacementString, left: "0px" }).then(function () {

// После анимации макет занимает меньшее окно просмотра над клавиа
elementToResize.style.height = keyboardRect.y + "px";

// Прокручиваем список в нужное место, так чтобы прокрутка не была з
elementToScroll.scrollTop = bottomOfList - elementToScroll.clientHeight;
showingAnimation = null;
});
}
```

Этот код немного сложный, так как он анимирует перемещение различных элементов страницы. В результате макет задействованных элементов, а именно, того, к которому вы прикоснулись – подстраивается для того, чтобы дать место для клавиатуры. Другие элементы на странице, в противном случае, не затрагиваются. Результат показан на рис. 4.5.

Рис. 4.5. Прикосновение к серому элементу customHandling справа демонстрирует пользовательскую обработку отображения клавиатуры

Стандартные сочетания клавиш

Последнее, что мне хотелось бы включить в раздел, посвященный клавиатуре – это таблица командных сочетаний клавиш, которые вы можете поддерживать в своем приложении. Это дополняет набор эквивалентов языка касаний, и вы, возможно уже привыкли пользоваться многими из них. Это полезно рассмотреть, так как приложения должны быть доступны для использования исключительно с помощью клавиатуры, и реализация сочетаний клавиш, наподобие этих – это один из шагов к удовлетворении данного требования и позволяет пользователям клавиатуры более эффективно использовать ваше приложение.

Таблица 4.6.

Действие или команда	Сочетание клавиш
Перемещение фокуса	Tab
Назад (Навигация)	Кнопка "Назад" на специальных клавиатурах; клавиш текстовых полей; Alt+стрелка влево

Вперед (Навигация)	Alt+стрелка вправо
Вверх	Alt+стрелка вверх
Отмена/Выход из режима	ESC
Перемещение по элементам в списке	Клавиши-стрелки (и Tab)
Быстрое по элементам списка к соедующей группе, если выделение не следует автоматически за фокусом	Ctrl+клавиши-стрелки
Масштабирование (семантическое и оптическое)	Ctrl+"+" b Ctrl+ "-"
Быстрое перемещение к именованному элементу коллекции	Начало ввода текста
Перемещение на большое расстояние	Page up/down (должно работать в пользовательском и поддерживающем сдвиг в горизонтальном и вертика
Следующая позиция табуляции или группа	Ctrl+Tab
Предыдущая позиция табуляции или группа	Ctrl+Shift+Tab

N-ная позиция табуляции или группа	Ctrl+N (1-9)
Открыть панель приложения (Windows обрабатывает это автоматически)	Win+Z
Контектстное меню	Клавиша контекстного меню
Открыть дополнительную панель/выбрать элемент меню	Enter
Перейти куда-либо/ активировать	Enter (в выделении)
Выбрать	Пробел
Выбрать смежные элементы	Shift+клавиши-стрелки
Закрепить элемент	Ctrl+Shift+!
Сохранить	Ctrl+S
Поиск	Ctrl+F
Печать	Ctrl+P (вызывает `Windows.Graphics.Printing.PrintManag`
Копировать	Ctrl+C
Вырезать	Ctrl+X
Вставить	Ctrl+V
Новый элемент	Ctrl+N
Открыть адрес	Ctrl+L или Alt+D
Вращение	Ctrl+"," и Ctrl+"."
Проигрывание/	Ctrl+P (только медиа-приложения)

Пауза	Ctrl+P (только медиа-приложения)
Следующий элемент	Ctrl+F (конфликтует с Поиском)
Предыдущий элемент	Ctrl+B
Перемотка назад	Ctrl+Shift+B
Перемотка вперед	Ctrl+Shift+F

Рукописный ввод

Помимо встроенной программной клавиатуры и панели для рукописного ввода, приложениям может понадобиться предоставить пользователю поверхность, которая может непосредственно принимать данные указателя в виде рукописного ввода (ink). Под этим я подразумеваю больше, чем использование элемента canvas и обработку событий MSPointer для рисования или создания растровых изображений. Рукописные штрихи – это структура данных, которая поддерживает реально вводимую информацию (в том числе – сведения о нажиме, угле наклона сенсора, скорости, если аппаратное обеспечение это поддерживает), что позволяет выполнять распознавание рукописного ввода и другие виды высокоуровневой обработки информации, которые невозможны при работе с растровыми данными. Работа с рукописным вводом, другими словами, напоминае работу с процессом рисования изображения, а не работу с готовым изображением, и это возможно для всех типов указателей ввода.

Поддержка рукописного ввода в WinRT располагается в пространстве имен `Windows.UI.Input.Inking` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.ui.input.inking.aspx). Это API не зависит от конкретного презентационного фреймворка, не предоставляет она и средств визуализации: она имеет дело лишь с управлением структурами данных, которые приложение может вывести самостятельно, на поверхности для рисования наподобие `canvas`. Вот базовые функции этого API:

- Создание экземпляра управляющего объекта с помощью команды

- Назначение атрибутов рисования путем создания объекта `Windows.UI.Input.Inking.InkDrawingAttributes` и установки атрибутов штриха, таких, как `color`, `fitToCurve` (в противоположность прямым линиям, проводимым по умолчанию), `ignorePressure`, `penTip` (`Windows.UI.Input.Inking.PenTipShape.circle` или `rectangle`), и `size` (объект `Windows.Foundation.Size` со свойствами `height` и `width`).
- Для элементов ввода, прослушивание событий `MSPointerDown`, `MSPointerMove`, и `MSPointerUp`, которые обычно нужны для целей визуализации. `eventArgs.currentPoint` это объект `Windows.UI.Input.PointerPoint`, который содержит идентификатор указателя, координаты точки и свойства наподобие `pressure`, `tilt`, и `twist`.
- Передача объекта `PointerPoint` методам управляющего объекта `processPointerDown`, `processPointerUpdate`, и `processPointerUp`.
- После события `processPointerUp`, управляющий объект рукописного ввода создает объект `Windows.UI.Input.Inking.InkStroke` для проведенного штриха. Штрихи можно получить посредством метода getStrokes управляющего объекта и вывести так, как нужно.
- Жесты высокого уровня могут быть непосредственно конвертированы в объекты `InkStroke` и переданы управляющему объекту посредством метода `addStroke`. Объекты штрихов можно удалять с использованием метода `deleteStroke`.

Управляющий объект, кроме того, предоставляет методы для распознавания рукописного текста на основе содержащихся в нем штрихов, сохранения и загрузки данных и обработки различных режимов, таких, как рисование и стирание штрихов. Полную демонстрацию этого вы можете найти в примере "Ввод: рукописный ввод" (ссылка: http://code.msdn.microsoft.com/windowsapps/Ink-App-sample-61abaec3), который показан на рис. 4.6. Этот пример позволит вам увидеть полный набор возможностей платформы рукописного ввода, в том числе – распознавание рукописного текста.

том числе – распознавание рукописного текста.

Рис. 4.6. Пример работы с платформой рукописного ввода содержит множество команд на панели приложения. Зеленый текст "Hello" в верхнем левом углу был создан после выделения рукописного "Hello" и использования команды Recognition (Распознавание)

SDK так же включает в себя пример "Ввод: упрощенный рукописный ввод" (ссылка: http://code.msdn.microsoft.com/windowsapps/Input-simplified-ink-sample-11614bbf) для демонстрации сценария, ориентированного на распознавание рукописного текста, как показано на рис. 4.7. Вам следует знать, что это пример, который вовсе не поддерживает касания – он ограничен мышью и пером и использует для активации команд клавиатурные сокращения вместо панели приложения. Взгляните на функцию keydown в файле simpleink.js для того, чтобы увидеть список команд вида "Ctrl + другая клавиша". Клавиша пробел активирует распознавание введенного текста, клавиша backspace очищает рабочую область. Как вы можете видеть на рисунке, я могу думать, что распознавание рукописного текста работает очень хорошо! (Это говорит мне о том, что образцы рукописного текста, которые я передал команде инженеров Microsoft где-то в середине 1990-х, стали значительным вкладом в их дело).

recognised text: Windows

Рис. 4.7. Пример отлично распознает мой небрежный текст, написанный мышью

Географическое расположение

Прежде чем мы рассмотрим датчики в целом, я хочу отдельно рассмотреть возможности определения географического расположения в приложениях для Магазина Windows, так как это API устроено не так, как у других датчиков. Мы уже использовали эту возможность, начиная с Главы 2 курса "Введение в разработку приложений для Windows 8 с использованием HTML, CSS и JavaScript", в приложении "Here My Am!", но нам нужен более полной обзор этой крайне полезной возможности.

В отличие от других датчиков, только определение местоположения имеет выделенную возможность, которую вы должны объявить в манифесте. То, где вы находитесь на Земле, это абсолютные данные, которые являются частью личной информации. Таким образом, пользователи должны дать согласие, прежде чем приложение сможет получить эти сведения, и ваше приложение так же должно предоставлять "Соглашение о конфиденциальности" в Магазине Windows. Другие датчики, в отличие от этого, предоставляют относительную информацию – вы не можете, например, узнать что-либо о человеке из данных о наклоне устройства, о его перемещении или из сведений о том, насколько яркий свет его освещает. Соответственно, вы можете использовать другие датчики без

объявления каких-либо особых возможностей.

Как вы можете знать, данные о географическом местоположении могут быть получены двумя различными путями. Первый, и наиболее точный, конечно, это получение данных с GPS-приемника, который получает информацию от геостационарных спутников, расположенных на орбитах в несколько сотен миль. Другим, достаточно полезным, хотя и не всегда точным средством является попытка узнать местоположение на основе IP-адреса проводного сетевого соединения или триангуляция на основе расположения доступных точек доступа WiFi. В любом случае, WinRT делает все возможное для того, чтобы дать вам достойные показатели.

Для доступа к данным о географическом местоположении, вы должны сначала создать экземпляр средства определения местоположения WinRT, `Windows.Devices.Geolocation.Geolocator` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.devices.geolocation.geo Когда он у вас есть, вы затем можете вызвать его метод `getGeopositionAsync`, результаты работы которого (помещенные в обработчик завершения), представляют собой объект типа `Geoposition` (в том же пространстве имен `Windows.Devices.Geolocation`, если не указано иное). Вот код из "Here My Am!":

```
var gl = new Windows.Devices.Geolocation.Geolocator();

gl.getGeopositionAsync().done(function (position) {
//Сохранить для целей общего доступа
lastPosition = { latitude: position.coordinate.latitude,
longitude: position.coordinate.longitude };
}
```

У метода `getGeopositionAsync` имеется вариант использования (ссылка: http://msdn.microsoft.com/library/windows/apps/hh973537.aspx), применяя который вы можете указать два параметра: максимальную давность для кэшированных данных (то есть то, насколько свежими должны быть данные) и значение тайм-аута, в течение которого вы хотите ждать ответ. И то и другое значение измеряются в

миллисекундах.

Объект `Geoposition` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.devices.geolocation.geo содержит два свойства. Первое, это свойство `coordinate`, объект `Geocoodinate` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.devices.geolocation.geo который предоставляет свойства `accuracy` (точность измерения, измеряется в метрах), `altitude` (высота, измеряется в метрах), `altitudeAccuracy` (точность измерения высоты, измеряется в метрах), `heading` (курс, в градусах по отношению к истинному северу), `latitude` (широта, измеряется в градусах), `longitude` (долгота, измеряется в градусах), `speed` (скорость, измеряется в метрах в секунду), и `timestamp` (отметка времени, значения типа Date). Вторая часть `Geoposition` это объект `CivicAddress` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.devices.geolocation.civi который содержит свойство `civicAddress`, которое может содержать свойства `city` (город, строка), `country` (страна, стока, содержащая двухбуквенный код страны в формате ISO-3166), `postalCode` (почтовый индекс, строка), `state` (штат, строка), и `timestamp` (отметка времени, тип Date), если поставщик географических данных поддерживает подобные данные[2)]

Вы можете задать желаемую точность измерений посредством свойства `desiredAccuracy` средства определения местоположения, которое может принимать значение `PositionAccuracy.default` или `PositionAccuracy.high`. Последнее, учтите, создаст большую нагрузку на аппаратное обеспечение. Это может привести к повышению стоимости использования лимитированного сетевого соединения и ускорить разряд батарей, поэтому устанавливайте этой свойство в значение `high` только тогда, когда это имеет ключевую важность для работы с приложением.

Средство определения местоположения так же предоставляет свойство `locationStatus`, которое является перечислением `PositionStatus` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.devices.geolocation.pos содержащим такие члены, как `ready`, `initializing`, `noData`,

`disabled`, `notInitialized`, и `notAvailable`. Должно быть очевидным, что вы не сможете получить данные от средства определения местоположения, которое находится в состоянии, отличном от `ready` (готово). Для того, чтобы отследить это, вы можете прослушивать событие средства определения местоположения `statusChanged`, свойство `eventArgs.status` которого в вашем обработчике будет содержать `PositionStatus`. Это полезно, когда вы обнаруживаете, что GPS-приемнику может понадобиться пара секунд на то, чтобы предоставить необходимые данные. Например, вот как используется данное событие в примере "Определение местопол ожения" (ссылка: http://code.msdn.microsoft.com/windowsapps/Geolocation-2483de66) из Windows SDK:

```
geolocator = new Windows.Devices.Geolocation.Geolocator();
geolocator.addEventListener("statuschanged", onStatusChanged); //Позже не за

function onStatusChanged(e) {
switch (e.status) {
// ...
}
}
```

Обратите внимание на то, что `PositionStatus` и `statusChanged` отражают чтение данных с GPS-устройства, и на чтение не влияет разрешение на получение данных о местоположении в панели параметров. Как показано в "Here My Am!", приложению не нужно проверять разрешение, пытаясь получить значения параметров, что является особенностью получения данных от устройства.

Два других интересных свойства средства определения местоположения – это `movementThreshold`, расстояние в метрах, на которое может переместиться устройство, прежде чем будет вызвана следующая операция чтения данных (это может быть использовано в сценариях с установкой географических границ), и `reportInterval`, который задается в миллисекундах между попытками чтения данных. Используйте последний параметр умеренно, устанавливайте его в то значение, которое вам действительно нужно, так как следует стремиться к минимизации сетевой активности и активности GPS-датчика. В

любом случае, когда средство определения местоположения получает новые данные и обнаруживает, что устройство переместилось за пределы `movementThreshold`, оно вызывает событие `positionChanged`, свойство которого `eventArgs.position` является новым объектом Geoposition. Это так же показано в Сценарии 1 примера об определении местоположения:

```
geolocator.addEventListener("positionchanged", onPositionChanged);

function onPositionChanged(e) {
var coord = e.position.coordinate;

document.getElementById("latitude").innerHTML = coord.latitude;
document.getElementById("longitude").innerHTML = coord.longitude;
document.getElementById("accuracy").innerHTML = coord.accuracy;
}
```

В случае с `movementThreshold` и `reportInterval`, рассматривайте их с позиций нужд приложения, основанных на точности и/или интервале обновления данных о местоположении, которые вы используете. Например, данные о погоде относятся к регионам и могут обновляться лишь раз в час. Поэтому `movementThreshold` может быть установлено в милях или километрах, `reportInterval` – на 15, 30, 60 минут или на более длительное время. Приложения работы с картами или предназначенные для автомобилистов, с другой стороны, работают с данными, весьма чувствительными к местоположению и, таким образом, будут иметь гораздо меньшие пороговые значения и более короткие интервалы обновления.

Когда имеет значение время жизни устройства от батарей, лучше всего просто получать новые данные, когда они нужны пользователю, вместо того, чтобы выполнять это через некоторые регулярные интервалы. Но это, опять же, зависит от сценария приложения, и вы можете так же предоставить параметр, который позволяет пользователю управлять активностью подсистемы определения местоположения.

Так же очень важно отметить, что приложения не получают события

positionChanged или statusChanged, когда они приостановлены, если только вы не зарегистрировали для этой цели фоновую задачу, запускаемую по таймеру и пользователь не добавил приложение на экран блокировки. Мы поговорим об этом подробнее в лекции 2 курса "Программная логика приложений для Windows 8, созданных с использованием HTML, CSS и JavaScript и их взаимодействие с системой", и вы так же сможете увидеть, как это работает в Сценарии 3 примера об определении местоположения. Если, однако, вы не используете фоновые задачи или пользователь не поместил приложение на экран блокировки, и вам все еще нужно отслеживать местоположение пользователя, обрабатывайте соответствующим образом событие resuming и обновляйте там эти данные.

С другой стороны, некоторые сценарии, исопользующие определение местоположения, такие, как навигация, нуждаются в том, чтобы экран устройства был активен (предотвращая автоматическое выключение экрана), даже если пользователь ничего не делает. Для этой цели вы можете использовать класс Windows.System.Display.DisplayRequest (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.system.display.displayre в частности, его методы requestActive и requestRelease, которые вы можете вызывать в начале и в конце сеанса навигации. Конечно, поддержание экрана во включенном состоянии приводит к потреблению большей энергии батарей, используйте эту возможность только когда это необходимо – когда приложение используется для целей навигации – и не прибегайте к выполнению подобного запроса при запуске приложения. В противном случае ваше приложение, возможно, прослывет в Магазине Windows как неумеренный потребитель энергии!

Врезка: Определение местоположения в HTML5

Опытные разработчики HTML/JavaScript могут удивиться, почему WinRT предоставляет API для определения местоположения в то время, как в HTML5 оно уже есть: window.navigator.geolocation и его метод getCurrentPosition , который возвращает объект с координатами. Причина подобного заключается в том, что другие языки,

такие, как C#, VisualBasic и C++ не имеют других API для получения подобных данных, что предоставляет выбор разработчикам на HTML/JavaScript. Если посмотреть глубже, то API HTML5 пользуется теми же данными, что и API WinRT, требует объявления той же возможности Расположение (Location) в манифесте, и нуждается в таком же разрешении пользователя, то есть, в основном, эти два API идентичны. Я дал бы API WinRT небольшое преимущество за счет опции `movementThreshold` , которая помогает приложению экономить электроэнергию и делает возможной простую постановку географических границ. Выполнение тех же действий в HTML5 потребует более частых запросов и приведет к большему потреблению энергии. Во многих сценариях, однако, вы можете использовать либо то, либо другое, с одинаковыми результатами. Однако, как и другие API WinRT, `Windows.Devices.Geolocation` доступно лишь страницам локального контекста приложений для Магазина Windows. На страницах веб-контекста вы можете пользоваться API HTML5.

Датчики

Как я говорил во введении к лекции, мне нравится думать о датчиках как об одной из форм ввода данных. Это имеет смысл, так как каждое устройство, которое интегрированно в наши компьютеры, что мы принимаем как должное, является одним из видов интерфейса для взаимодействия пользователя и машины. Со временем, я полагаю, многие датчики, которые сейчас кажутся нам новыми, станут стандартным оборудование почти везде.

Датчики, опять же, это способ понять взаимоотношения устройства и физического мира, окружающего его, и их использование подразумевает ввод данных, так как вы, будучи человеком, можете влиять на эти взаимоотношения, в основном, перемещая устройство в пространстве или меняя их окружение. Датчики так же могут быть использованы как средства прямого ввода информации, вызывая перемещение объектов на экране, вместо использования какой-то формы абстрактного ввода данных наподобие мыши или клавиатуры. Например, вместо того, чтобы использовать абстрактное сочетание клавиш для того, чтобы наклонить игровое поле, вы можете, используя датчики, просто наклонить устройство. Встряхивание устройства, на самом деле,

является хорошо известным физическим жестом, который может быть связан с командой вроде "Повтори, будь ты проклят! Почему ты не делаешь то, что я хочу?". Разве мы долгие годы не трясли наши компьютеры или не стучали по ним, когда они не делали то, что нам нужно? А компьютер с датчиками вполне может отреагировать на такое воздействие!

Вот, какую информацию могут нам сообщить различные датчики:

- Местоположение (Location). Позиция устройства на Земле (мы обсудили это в предыдущем разделе).
- Компас (Compass) и ориентация (Orientation). Ориентация устройства по отношению к магнитным полюсам Земли, или относительно нормального положения устройства (простая и сложная ориентация).
- Инклинометр, креномер (Inclinometer). Предоставляет данные о поворотах устройства относительно поперечной, продольной и вертикальной осей в трехмерном пространстве.
- Гирометр (Gyrometer). Данные об угловой скорости/вращательном движении устройства в трехмерном пространстве.
- Акселерометр (Accelerometer). Ускорение устройства (в G) в трехмерном пространстве (X, Y, Z).
- Окружающий свет (Ambient light). Яркость света в пространстве, окружающем устройство.

Эти датчики предоставляет API WinRT[3)], некоторые из них созданы программным путем посредством синтеза датчиков (sensor fusion). Это подразумевает получение исходных данных от одного или большего коичества аппаратных датчиков, комбинирование, интерпретацию и представление их в виде, которым может воспользоваться приложение. Как и с указателями, вы можете пользоваться исходными данными, если нужно, но чаще в этом нет необходимости. Например, простой датчик ориентации (Simple Orientation) предоставляет простую интерпретацию расположения устройства по отношению к его расположению по умолчанию, округляя данные до 90-градусных секторов. Полный датчик ориентации, с другой стороны, комбинирует данные гирометра, акселерометра и компаса для того, чтобы предоставить матрицу трехмерной ориентации, которая гораздо более точна, но ориентирована (если можно использовать каламбур!) на более

продвинутые сценарии, нежели обычное определение, расположено ли устройство вниз экраном или на боку.

Так как работа всех этих датчиков очень похожа друг на друга (это сделано специально, за исключением простого датчика ориентации, который не похож на другие), я хочу показать общий шаблон работы с API датчиков вместо детальных примеров для каждого из них. Подобные примеры можно найти среди SDK-примеров "Акселерометр" (ссылка: http://code.msdn.microsoft.com/windowsapps/Accelerometer-Sensor-Sample-22982671), "Компас" (ссылка: http://code.msdn.microsoft.com/windowsapps/Compass-Sensor-Sample-0ed09c55), "Гирометр" (ссылка: http://code.msdn.microsoft.com/windowsapps/Gyrometer-Sensor-Sample-4fe891d9), "Инклинометр" (ссылка: http://code.msdn.microsoft.com/windowsapps/Inclinometer-Sensor-Sample-0cd0bf84), "Датчик освещенности" (ссылка: http://code.msdn.microsoft.com/windowsapps/LightSensor-Sample-4477824c) и "Датчик ориентации" (ссылка: http://code.msdn.microsoft.com/windowsapps/OrientationSensor-sample-0b1732be).

Шаблон их использования выглядит следующим образом, некоторые особенности приведены в таблице ниже:

Получение объекта датчика посредством `Windows.Devices.Sensors.<sensor>.getDefault()`.

Вызов метода getCurrentReading этого объекта для получения данных.

Для последующего получения данных, настроить свойства объекта `minimumReportInterval` и `reportInterval` (и то и другое – в миллисекундах) и прослушивать событие объекта `readingchanged`. В ответ обработчик события получит объект подходящего типа с прочитанными данными. В случае с определением местоположения, разумно подойтите к установке этих параметров, что поможет увеличить время работы устройства от батарей, предотвращая излишний расход электроэнергии датчиками.

Название датчика (Windows.Devices.Sensors.)
Акселерометр (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.devices.sensors.acceler
Компас (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.devices.sensors.compas
Гирометр (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.devices.sensors.gyrome
Инклинометр (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.devices.sensors.inclinom
Датчик освещенности (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.devices.sensors.lightsen
Датчик ориентации (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.devices.sensors.orientat

Вот образец кода из примера "Тирометр" (js/scenario1.js):

```
gyrometer = Windows.Devices.Sensors.Gyrometer.getDefault();

var minimumReportInterval = gyrometer.minimumReportInterval;
var reportInterval = minimumReportInterval > 16 ? minimumReportInterval : 16;
gyrometer.reportInterval = reportInterval;

gyrometer.addEventListener("readingchanged", onDataChanged);
// Не забудьте удалить при необходимости

function onDataChanged(e) {
var reading = e.reading;
document.getElementById("eventOutputX").innerHTML = reading.angularVelocit
document.getElementById("eventOutputY").innerHTML = reading.angularVelocit
document.getElementById("eventOutputZ").innerHTML = reading.angularVelocit
}
```

В случае с датчиком ориентации, понятие кватернион (quaternion) легче всего понять как вращение точки [x,y,z] вокруг одной произвольной оси. В этом – отличие от матрицы вращения, которая представляет вращение по всем трем осям. Математическая основа кватернионов довольно экзотична, она включает в себя геометрические понятия комплексных чисел, математические понятия мнимых чисел, но работать с ними просто и фреймворки наподобие DirectX поддерживают их. Посмотрите пример "Сенсор ориентации" для того, чтобы больше узнать об этом.

Говоря об ориентации, я упоминал, что `SimpleOrientationSensor` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.devices.sensors.simpleo работает немного не так, как остальные. Его цель – в предоставлении информации о квадранте (quadrant) ориентации, а не в предоставлении данных о точной ориентации устройства, и, возможно, это все, что вам нужно. Например, приложению для просмотра звездной карты нужно знать, расположено ли устройство вниз экраном, таким образом оно сможет настроить вывод данных (используя и показания компаса) для того, чтобы то, что оно отображает на экране, соответствовало бы виду неба.

В итоге, использование этого датчика выглядит так:

- Вызовите `Windows.Devices.Sensors.SimpleOrientation.getI` для того, чтобы получить объект.
- Вызовите `getCurrentOrientation` для того, чтобы получить результаты чтения данных.
- Событие `orientationChanged` используется для последующего получения данных, где `eventArgs` содержит свойства `orientation` (результат чтения данных) и `timestamp`.
- Свойство `reading` – это объект `SimpleOrientation` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.devices.sensors.s который содержит следующие свойства:
    - `notRotated` ("portrait up", книжная-сверху), `rotated90DegreesCounterclockwise` ("landscape left" альбомная-слева), `rotated180DegreesCounterclockwise` ("portrait down" книжная-снизу), `rotated270DegreesCounterclockwise` ("landscape right" альбомная-справа). Обратите внимание на то, что все это не имеет никакого отношения к состояниям просмотра наподобие полноэкранного альбомного или полноэкранного портретного режимов просмотра.
    - `faceup`, `facedown` (лицевой стороний вверх или вниз, только для планшентых устройств).

Демонстрацию использования этого датчика вы можете найти в примере "Простой датчик ориентации" (ссылка: http://code.msdn.microsoft.com/windowsapps/SimpleOrientationSensor-d948ac62).

Что мы только что изучили

- Сообщение: "Дизайн приложений, использующих мышь, касания, перо – без дополнительных усилий" соответствует истине, так как

работа с указателями и жестами, поступающими от различных устройств, не требует делать различия между способами ввода.

- Использование встроенных элементов управления – это самый простой способ обработки ввода, но вы так же можете непосредственно обрабатывать события MSPointer* и MSGesture*, если в этом есть необходимость. Так же вы можете передавать события MSPointer в пользовательское средство распознавания жестов (оно инициирует собственные события).
- Язык касаний Windows 8 включает в себя касание, нажатие и удержание, прокрутку/сдвиг, скольжение по диагонали (для выделения), жесты сжатия-растяжения, вращение, жесты у краев экрана (от верхней, нижней и от боковых сторон). Касание обычно обрабатывается в событии `click`, в то время как другие требуют создания объекта `MSGesture`, связи этого объекта с указателем и обработку последовательностей событий `MSGesture*`, которая используется для реализации манипуляций и перемещения с инерцией.
- У языка касаний есть эквиваленты, реализуемые с помощью мыши, пера и клавиатуры. В случае с мышью и пером нужно проделать не так много работы (например, передавать события мыши `wheel` в объект жеста). Поддержка клавиатуры должна быть реализована отдельно, она использует обычные события HTML/JavaScript.
- Поддержка клавиатуры так же включает в себя соответствующую настройку для работы с программной (экранной) клавиатурой, которая появляется автоматически для полей ввода текста и других элементов, поддерживающих редактирование содержимого. Она автоматически настраивает свой внешний вид в зависимости от типа вводимых данных и смещает содержимое приложения вверх, если нужно, для того, чтобы не перекрыть клавиатурой элемент управления, в который нужно вводить текст. Приложение так же может обрабатывать события, связанные с экранной клавиатурой для предоставления большего удобства, чем в случае с автоматическим поведением системы.
- API поддержки рукописного ввода предоставляет приложениям средства для записи, сохранения и вывода последовательностей штрихов, оставленных указателем, штрихи так же можно передать средству распознавания рукописного текста.
- API определения местоположения в WinRT, так же, как похожее

API HTML5, предоставляет приложениям доступ к GPS-данным, а так же – события, показывающие, что устройство пересекло ранее заданное пороговое значение в пространстве.

- API WinRT предоставляеют множество датчиков, которые можно использовать для ввода данных в приложение. В дополнение к датчику определения местоположения, это компас, датчик ориентации, основанный на квадрантах простой датчик ориентации, инклинометр, гиромет, акселерометр и датчик окружающего освещения.
- Большая часть сенсоров используется по одному и тому же шаблону: получить объект сенсора, получить текущий результат чтения данных и, возможно, прослушивать событие `readingChanged`. С ними легко работать, что оставляет большую часть сил на поиск креативных способов их применения!

1) И пока вы думаете, что это второе самое длинное название JavaScript-примера в Windows SDK, оно, на самом деле, может получить лишь бронзовую медаль. Пример "Невыделяемые области содержимого с CSS-атрибутом –ms-user-select" (ссылка: http://code.msdn.microsoft.com/windowsapps/Unselectable-content-areas-963eccd9), который мы уже видели, получает золото, а пример "Использование requestAnimationFrame для анимации, эффективно расходующей электроэнергию" (ссылка: http://code.msdn.microsoft.com/windowsapps/Using-requestAninationFrame-924b039a) получает серебро. Однако, я не против таких длинных названий – я очень доволен тем, что у нас есть столь обширный набор отличных примеров, из которых мы можем взять много полезного!

2) Таким образом, civicAddress может быть недоступен или не содержать данных. В качестве альтернативного способа получения этих данных используйте API Bing Maps (ссылка: http://www.microsoft.com/maps/developers/web.aspx), в частности – класс MapAddress (ссылка: http://msdn.microsoft.com/library/jj663670.aspx) для того, чтобы узнать адрес по имеющимся координатам.

3) Здесь присутствует и датчик для бесконтактного взаимодействий (proximity) близко расположенных устройств (NFC), который сообщает о том, что устройства находятся близко друг от друга или соприкасаются, но это больше средство для установления сетевого соединения, нежели

датчик, похожий на другие. Мы рассмотрим его в лекции 15.

Мультимедиа

Эта лекция содержит описание особенностей работы с мультимедийными данными - изображениями, музыкальными записями и видеоклипами.

Файлы к данной лекции Вы можете скачать здесь скачать: http://old.intuit.ru/department/se/uiwin8htmljs/5/4.zip.

Говорить, что средства аудиовизуальной информации важны для приложений – и для культуры в целом – это грубое преуменьшение. С тех пор, как люди, подобные Эдисону, сделали возможным записывать произведения для того, чтобы позже ими можно было насладиться, влечение человечества к мультимедийным данным – к изображениям, аудиозаписям, видеоклипам – возможно превысило тягу к автомобилям, электричеству и нездоровой пище. В ранние дни Интернета передача изображений легко занимала большую часть сетевого трафика. Сегодня, потоковое видео даже из одного отдельно взятого ресурса вроде Netflix поддерживает усилия по увеличению возможностей инфрастрктуры широкополосных сетей! (У меня дома это так и есть, с любимыми фильмами моего маленького сына вроде "Curious George", "Bob the Builder", "Dinosaur Train" и другими.)

Включение в себя, в той или иной форме, мультимедийной информации, это одна из основных забот большинства приложений для Магазина Windows. Простые, вероятно используют лишь несколько изображений для придания приложению фирменного стиля и отображения привлекательного пользовательского интерфейса, как мы уже видели много раз. Многие другие, особенно игры, конечно, используют и графику, и видео и аудио. В контексте этого курса все это поразумевает использование таких элементов HTML5, как `img`, `svg` (Scalable Vector Graphics – Масштабируемая векторная графика), `canvas`, `audio` и `video`.

Конечно, работа с мультимедийными данными идет дальше, чем просто их отображение, так как приложения могут предоставлять еще и любые из следующих возможностей:

- Организация и правка медиафайлов, в том числе тех, что хранятся

в библиотеках изображений, музыки и видео.
- Конверсия (преобразование) медиафайлов, возможно, с применением различных фильтров и пользовательских кодеков.
- Организация и правка плей-листов.
- Захват аудио и видео с доступных устройств.
- Потоковая передача данных с сервера на устройства, или с устройства на целевые объекты воспроизведений (PlayTo target), возможно, с применением DRM.

Эти возможности, для реализации которых существует множество API WinRT, вместе с мультимедиа-элементами HTML5 и их особыми возможностями в среде Windows 8, будут рассмотрены в этой лекции.

Примечание. Обратите внимание на полный список аудио и видео-форматов, которые поддерживаются WinRT-приложениями: "Поддерживаемые форматы звука и видео" (ссылка: http://msdn.microsoft.com/library/windows/apps/hh986969.aspx).

Врезка: Трюки по повышению производительности для создания более быстрых приложений

Некоторые из рекомендаций, данных в этой лекции, взяты из отличного выступления Джейсона Вебера, который занимается вопросами производительности Internet Explorer. Это выступление называется "50 трюков по повышению производительности, которые позволяют сделать ваше приложение для Windows 8, использующее HTML5, быстрее" (ссылка: http://channel9.msdn.com/events/BUILD/BUILD2011/PLAT-386T). В то время, как некоторые из этих трюков имеют смысл для веб-приложений, которые исполняются в браузере, многие из них полностью применимы к приложениям для Магазина Windows, написанных на JavaScript, так как они исполняются на основе той же инфраструктуры, что имеется в Internet Explorer.

Создание мультимедиа-элементов

Конечно, самый простой способ включить мультимедийные данные в

приложение – это то, что мы делаем уже многие годы: просто использовать подходящий HTML-элемент в разметке, и вуаля! – все сделано. С элементами img , audio, и video, на самом деле, вы без проблем можете исползовать содержимое из почти любого источника. То есть, атрибутам src этих элементов может быть назначены URI, которые указывают на содержимое в пакете приложения (с использованием относительных путей, URI ms-appx:// , или путей, основанных на Windows.ApplicationModel.Package.current.installec которые затем передают в URL.createObjectURL), на файлы в папках данных приложения (с использованием URI ms-appdata:// , или путей, основанных на Windows.Storage.ApplicationData.current с применением URL.createObjectURL), на удаленные файлы с применением http:// и других URI. В случае с элементом img сюда можно еще добавить использование в качестве источников SVG-файлов.

Существуют три способа создания мультимедиа-элемента на странице или на элементе управления страницы:

Во-первых, можно включить элемент напрямую в декларативный HTML-код. Часто полезно использовать атрибут preload="auto" для удаленного аудио или видео для увеличения скорости реакции элементов управления и других элементов пользовательского интерфейса, которые зависят от этих элеменов. (Это важно делать и для локальных медиафайлов, хотя они и так являются локальными!). Часто мультимедийные элементы размещают ближе к верхней части HTML-файла, в порядке приоритета, таким образом, их загрузка может начаться уже в то время, когда остальные части документа еще обрабатываются.

С другой стороны, если пользователь может немного подождать перед запуском проигрывания видео, используйте изображение для предварительного просмотра вместо видеофайла и не начинайте загрузку до того, как это не будет нужно. Код для этого показан ниже в данной лекции в разделе "Проигрывание видео и отложенная загрузка".

Воспроизведение для элементов, заданных декларативно, может быть начато автоматически, благодаря атрибугу autoplay, посредством

встроенного пользовательского интерфейса, если элемент имеет атрибут `controls`, или с помощью вызова из JavaScript команды `<element>.play()`.

Второй метод заключается в создании HTML-элемента в JavaScript посредством `document.createElement` и в добавлении его в DOM с помощью команды `<parent>.appendChild` и похожих методов. Вот пример использования мультимедиа-файлов, который находится в дополнительных материалах к этой лекции, хотя вам нужно поместить этот код в новый собственный проект:

```
//Создаем элемент и добавляем его в DOM, что приведет к изменению
var picture = document.createElement("img");
picture.src = "media/wildflowers.jpg";
picture.width = 300;
picture.height = 450;
document.getElementById("divShow").appendChild(picture);

var movie = document.createElement("video");
movie.src = "media/ModelRocket1.mp4";
movie.autoplay = false;
movie.controls = true;
document.getElementById("divShow").appendChild(movie);

var sound = document.createElement("audio");
sound.src = "media/SpringyBoing.mp3";
sound.autoplay = true; //Запускаем проигрывание как только элемент буде
sound.controls = true; //Если установлено false, аудио проигрывается, но н
document.getElementById("divShow").appendChild(sound);
```

Если элементы для вывода изображений или видео не скрыты с помощью стилей, а так же, если речь идет об элементах с установленным атрибутом `controls`, то их добавление вызывает повторную отрисовку макета документа. Аудио-элемент без данного атрибута не приводит к повторному выводу макета. Как и в декларативном HTML, установка autoplay в значение true приведет к запуску воспроизведения видео или аудио сразу после того, как элемент будет добавлен в DOM.

И, наконец, для аудио, приложения могут создавать объект `Audio` в JavaScript для проигрывания звуков или музыки без какого-либо воздействия на пользовательский интерфейс. Больше об этом будет позже. В JavaScript так же есть класс `Image`, класс `Audio` может быть использован для загрузки видео:

```
//Создаем объекты (с предварительной загрузки), затем нужным образо
var picture = new Image(300, 450);
picture.src = "http://www.kraigbrockschmidt.com/downloads/media/wildflowers.

document.getElementById("image1").src = picture.src;

//Объект Audio можно использовать для предварительной загрузки
(но не воспроизведения) видео
var movie = new Audio("http://www.kraigbrockschmidt.com/downloads/media/M
document.getElementById("video1").src = movie.src;

var sound = new Audio("http://www.kraigbrockschmidt.com/downloads/media/S
document.getElementById("audio1").src = sound.src;
```

Создание объектов Image или Audio из кода не приводит к созданию элементов в DOM, что может быть полезным. Объект Image, например, многие годы используется для предварительной загрузки массивов исходных изображений для использования в задачах наподобие вращения изображений или при реализации всплывающих меню. Предварительная загрузка в данном случае означает лишь то, что изображения были загружены и кэшированы. Таким образом, присваивание того же самого URI атрибуту SRC элемента, который уже находится в DOM, как показано выше, приведет к немедленному появлению изображения. То же самое справедливо для предварительно загруженного видео и аудио, но, опять же, это, в первую очередь, полезно для использования с удаленными источниками, в то время, как файлы из локальной файловой системы загружаются сравнительно быстро и без дополнительных усилий с нашей стороны. Тем не менее, если у вас есть большие локальные изображения, и вы хотите быстро показать их, когда они понадобятся, предварительная загрузка их в па мять – это полезная стратегия.

Конечно, вы можете загружать мультимедиа-файлы только тогда, когда они понадобятся, в таком случае тот же самый код можно использовать с существующими элементами, или вы просто можете создать элемент и добавить его в DOM, как показано ранее.

Графические элементы: Img, Svg и Canvas (и немного CSS)

Знаю, вам, вероятно, не терпится прочесть разделы этой лекции, посвященные аудио и видео, но мы не можем забывать о том, что изображения были основой веб-приложений с самого начала и их важность в реализации опыта взаимодействия пользователя и приложения все еще огромна. Действительно, полезно помнить о том, что видео, концептуально, это лишь последовательный набор статических изображений, быстро сменяющих друг друга! К счастью, HTML5 значительно расширил возможность приложений по включению в себя графических данных путем добавления поддержки формата SVG и элемента `canvas` к проверенному временем элементу управления `img`. Кроме того, применение CSS-анимаций и переходов (этому посвящена лекции 7-8) к элементам изображений, которые иначе неподвижны, позволяет сделать их очень динамичными.

Говоря о CSS, важно отметить, что многие графические эффекты, которые раньше требовали применения соответствующим образом подготовленного статического изображения, могут быть реализованы только с использованием CSS, в частности, с применением CSS3:

- Границы, фоновые цвета, фоновые изображения.
- Изображения закладок папок, меню, панели инструментов.
- Скругленные углы границ, использование нескольких границ и фонов, границы изображений
- Прозрачность
- Встраиваемые шрифты
- Тени объектов
- Тени текстов
- Градиенты

Коротко говоря, если вы когда-либо использовали элементы `img` для

создания маленьких визуальных эффектов, создания градиентных фонов, использования нестандартного шрифта или создания графических элементов навигации, теперь есть возможность сделать это исключительно с помощью CSS. Для того, чтобы узнать подробности, обратитесь к отличному материалу по CSS3 (ссылка: http://coding.smashingmagazine.com/2009/01/08/push-your-web-design-into-the-future-with-css3/) от Smashing Magazine и к спецификациям CSS3 на ссылка: http://www.w3.org/. CSS так же предоставляет возможность декларативно обрабатывать некоторые события и визуальные состояния с использованием псевдо-селекторов для `hover`, `visited`, `active`, `focus`, `target`, `enabled`, `disabled`, и `checked`. Для того, чтобы узнать подробности, обратитесь к материалам ресурса ссылка: http://css-tricks.com/ и к еще материалу Smashing Magazine по псевдо-классам (ссылка: http://coding.smashingmagazine.com/2011/03/30/how-to-use-css3-pseudo-classes/).

Рассмотрим теперь три основные HTML5-элемента для вывода изображений:

- Элемент `img` используется для растровых изображений. Формат PNG предпочтителен по сравнению с другими, особенно для текстов и штриховых рисунков, хотя JPEG-файлы фотографий имеют меньший размер. GIF, как правило, считается устаревшим, в тех случаях, когда использование GIF-файлы давало выигрыш за счет маленького размера, теперь можно пользоваться исключительно CSS. Если речь идет о масштабировании, приложения для Магазина Windows должны учитывать плотность пикселей, как мы видели в лекции 6 курса "Введение в разработку приложений для Windows 8 с использованием HTML, CSS и JavaScript", и предоставлять различные файлы изображений ля каждого масштаба, с которым может столкнуться приложение. Это тот случай, когда JPEG-файлы меньшего размера могут уменьшить общий размер пакета приложения в Магазине Windows.
- Формат SVG лучше всего использовать для плавного масштабирования изображений на мониторах разного размера и разной плотности пикселей. SVG-изображения могут быть описаны внутри HTML-кода, созданы динамически в DOM, ими можно управлять как отдельными файлами и использовать в качестве источника для элемента `img` (в этом случае все

характеристики масштабирования поддерживаются). SVG-файл так же может быть использован как источник для `iframe`, что дает преимущества в виде доступности дочерних элементов SVG в DOM. Как мы видели в лекции 6 курса "Введение в разработку приложений для Windows 8 с использованием HTML, CSS и JavaScript", сохранение соотношения сторон SVG-изображения часто важно, для чего мы использовали `viewBox` и атрибуты `preserveAspectRatio` тега `svg`.

- Элемент `canvas` (холст) предоставляет поверхность для рисования и API для создания изображений с использованием линий, прямоугольников, дуг, текста и так далее. Холст, в итоге, генерирует растровые данные, что означает, что будучи однажды созданным, такое изображение масштабируется как растровое. (Приложение, конечно, обычно перерисовывает холст с масштабированием координат для того, чтобы избежать пикселизации). Холст весьма полезен для выполнения манипуляций с пикселями, даже с отдельными кадрами видео, когда оно проигрывается.

Приложения часто используют все три элемента, опираясь на их сильные стороны. Я говорю это, так как, когда элемент `canvas` впервые стал доступен, разработчики оказались так им очарованы, что, казалось, забыли, как пользоваться img и они игнорируют тот факт, что часто SVG – это наилучший вариант! (И я ведь уже сказал, что CSS сам по себе способен на многое, а?)

В конце концов, полезно полезно воспринимать все графические элементы HTML5 как производящие, в конечно счете, растровые изображения, которые хост приложения просто выводит на экран. Вы можете, конечно, программно анимировать внутреннее содержимое этих элементов в JavaScript, как мы увидим в лекциях 7-8, но для наших целей полезно просто думать о них как о статических, по сути, изображениях.

Что отличает эти элементы друг от друга, так это то, как в них поступают данные. Элементы `img` загружают данные из файла-источника, `svg`-данные заданы в разметке, элементы `canvas` заполняются посредством программного кода. Но, в конце концов, как показано в Сценарии 1 упражнения "Графика в HTML" для этой лекции,

и как отражено на рис. 5.1 все они производят одинаковые результаты:

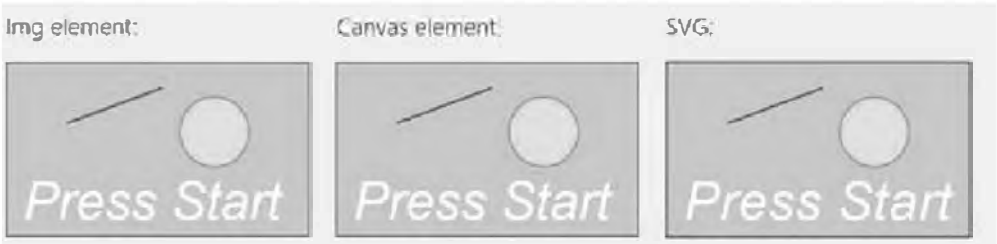


Рис. 5.1. Элементы img, canvas и svg показывают одинаковый результат

Коротко говоря, в том, что может быть выведено с помощью каждого из элемента, нет фундаментальных различий. Тем не менее, у них есть различия, которые становятся очевидными, когда мы начинаем манипулировать этими элементами как и в случае с CSS. Так каждый элемент – это лишь узел в DOM, простой и ясный, и они рассматриваются как и все другие, неграфические элементы: CSS не влияет на внутреннее содержимое элементов, а влияет лишь на то, как они, в итоге, отображаются на странице. Отдельные части SVG, объявленные в разметке, могут, на самом деле, быть отдельно стилизованы до тех пор, пока их можно идентифицировать с помощью CSS-селекторов. В любом случае, подобная стилизация влияет только на внешний вид, таким образом, если применяются новые стили, они применяются к исходному содержимому элемента.

Справедливо и то, что графические элементы могут перекрывать друг друга и неграфические элементы (как и видео), и подсистема визуализации автоматически обрабатывает прозрачность в соответствии с параметром `z-index` этих элементов. Каждый графический элемент может иметь чистые, или прозрачные области, которые встроены в форматы изображений наподобие PNG. В элементе `canvas`, любые области, очищенные с помощью метода `clearRect`, которые иным образом не затрагивают другие вызовы API, будут прозрачными. Похожим образом, любая область прямоугольного элемента SVG, которая не затронута его отдельными частями, будет прозрачной.

Сценарий 2 в упражнении "Графика в HTML" позволяет вам включать несколько стилей (с помощью флагов) для тех же элементов, которые

показаны ранее. В данном случае я сделал фон элемента canvas прозрачным, таким образом, мы можем видеть то, что распложено за ним. Когда стили применяются, элемент `img` оказывается повернутым и трансформированным, к `canvas` применяется масштабирование, и отдельные части `svg`-элемента стилизуются новыми цветами, как показано на рис. 5.2.

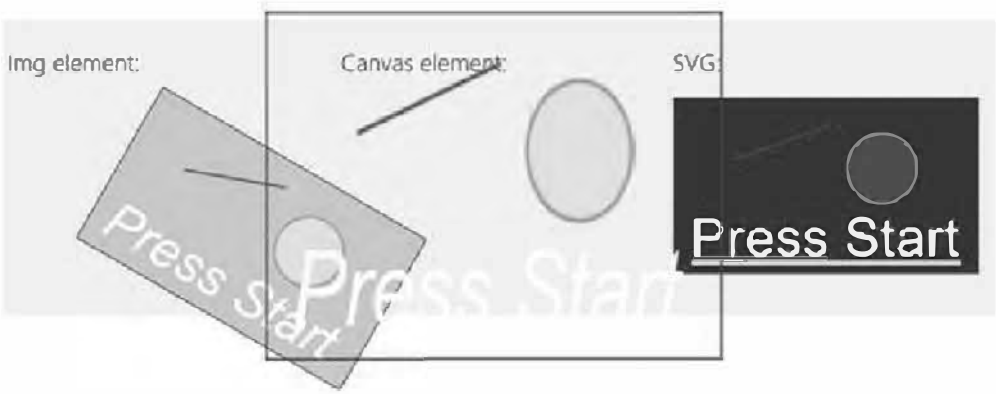


Рис. 5.2. Стили, примененные к графическим элементам; отдельные элементы SVG могут быть стилизованы, если они доступны посредством DOM

Стили, заданные в css/scenario2.css, довольно просты:

```
.transformImage {
transform: rotate(30deg) translateX(120px);
}

.scaleCanvas {
transform: scale(1.5, 2);
}
```

Как и код в js/scenario2.js, который их применяет:

```
function toggleStyles() {
var applyStyles = document.getElementById("check1").checked;

document.getElementById("image1").className = applyStyles ? "transformImag
```

```
document.getElementById("canvas1").className = applyStyles ? "scaleCanvas"

document.getElementById("r").style.fill = applyStyles ? "purple" : "";
document.getElementById("l").style.stroke = applyStyles ? "green" : "";
document.getElementById("c").style.fill = applyStyles ? "red" : "";
document.getElementById("t").style.fontStyle = applyStyles ? "normal" : "";
document.getElementById("t").style.textDecoration = applyStyles ? "underline" : '
 }
```

Другая особенность, которую вы могли заметить при применении стилей, это то, что увеличенный элемент canvas выглядит пикселизованным, как обычно бывает с растровыми изображениями. Это вполне ожидаемое поведение, как показано в следующей таблице о характеристиках масштабирования. Это показано в Сценариях 3 и 4 упражнения "Графика в HTML".

Таблица 5.1.

Элемент	Масштабирование	Обработка изменений макета для лучшего отображения
`img`	растеризованное	Измените атрибут `src` для различных масштабов (или просто используйте SVG-файл в качесте источника).
`canvas`	растеризованное	Перерисуйте холст с использованием масштабированных размеров; часто это можно сделать наилучшим образом, вызвав `<context>.scale` в соответствии с необходимым размером отображения вместо изменения координат, использованных в коде
`svg`	плавное	Нет необходимости. Используйте `viewBox` и `preseveAspectRatio` для пропорционального изменения масштаба.

Дополнительные характеристики графических элементов

Существуют некоторые дополнительные характеристики, о которых следует знать, работая с графическими элементами. Во-первых, это разные виды операций, которые вызывают перерисовку элементов в документе. Во-вторых, это режим работы каждого элемента. В-третьих – это относительные преимущества каждого элемента. Все это приведено в следующей таблице:

Таблица 5.2.

Элемент	Операции, вызывающие перерисовку	Режим	Преимущества
`img`	Изменение атрибута `src` Изменение стиля из JavaScript	Пиксельный	Быстро выводится и трансформируется Отлично подходит для статических элементов и статических/повторяющихся фонов Возможна спрайтовая анимация путем изменения атрибута `src`
	Вызов контекстного API Изменение стиля из JavaScript Обратите внимание: перерисовка происходит	Немедленный: вызовы API	Качественный динамический контент Быстро выводится после перерисовки Возможность

`canvas`	только тогда, когда код возвращает управление хост-процессу и разблокирует поток пользовательского интерфейса. Когда код манипулирует элементом, видимых изменений не наблюдается.	вызовы API приводят к выводу пикселей и не запоминаются	манипуляции изображением на уровне пикселей Отлично подходит для качественного динамического/ интерактивного содержимого с частым использованием вычислений.
`svg`	Изменение структуры элемента Изменение стиля из JavaScript	С запоминанием: все фигуры существуют как элементы DOM (если не используется в атрибуте `img` src)	Плавное масштабирование Точное управление отдельными (запоминаемыми) элементами Возможность манипуляции на уровни фигур Отлично подходит для интерактивной графики, детализированной и масштабируемой стилизации, и динамической настройки атрибут

Так как SVG создает элементы в DOM, эти элементы могут быть стилизованы индивидуально. Вы можете использовать этот факт вместе с медиа-запросами для скрывать разные части SVG в зависимости от их размера. Для того, чтобы сделать ээто, добавьте различные классы к таким SVG-элементам, затем, в CSS, добавьте или удалите стиль `display: none` для этих классов в медиа-запросах наподобие `@media (min-width:300px) and (max-width:499px)` . Вам может понадобиться рассчитать размер SVG по отношению к окну приложения, но это означает, что вы можете эффективно убрать детали из SVG вместо того, чтобы позволять этим частям отрисовываться на слишком малом пространстве, чтобы быть полезными.

В итоге, причина по которой HTML5 имеет все три элемента, заключается в том, что все три по-настоящему нужны. Все они способны получить преимущества от полного аппаратного ускорения, так же, как это происходит в Internet Explorer, так как приложения, написанные на HTML и JavaScript исполняются на той же подсистеме визуализации, что и браузер.

Лучший подход к дизайну приложений – по-настоящему исследовать подходящие варианты использования элементов каждого типа. Каждый элемент может иметь прозрачные области, в итоге, вы можете легко достигнуть весьма забавных эффектов. Например, если у вас есть данные, которые выводят временную информацию о видео или другой текст, в надпись, вы можете просто использовать обработчик интервалов (с интервалом, установленным на необходимую детализации, пол секунды, например), получить свойство видео `currentTime`, получить необходимый текст для данного фрагмента, и вывести текст на пэлементе canvas с прозрачным фоном, который расположен поверх видео. Субтитры и другие сведения о видео можно вывести похожим образом, устраняя необходимость в перекодировании видео.

Некоторые советы и секреты

Работа с графическими элементами HTML5 довольно проста, но знание

Работа с графическими элементами HTML5 довольно проста, но знание некоторых подробностей может помочь в работе с ними в приложениях для Магазина Windows.

Элементы Img

- Используйте атрибут `title` элемента `img` для реализации всплывающей подсказки, а не атрибут `alt`. так же вы можете использовать элемент управления `WInJS.UI.Tooltip`, как показано в лекции 4 курса "Введение в разработку приложений для Windows 8 с использованием HTML, CSS и JavaScript".
- Для создания изображений из потока, расположенного в памяти, воспользуйтесь методом `MSApp.createBlobFromRandomAccessStream` (ссылка: http://msdn.microsoft.com/library/windows/apps/Hh767329.aspx), результат его работы затем можно передать в `URL.createObjectURL` для создания подходящего для атрибута `src` URL. Мы встретимся с этим в данной лекции, и нам это понадобится при работе с контрактом "Общий доступ" в лекции 1 курса "Программная логика приложений для Windows 8, созданных с использованием HTML, CSS и JavaScript и их взаимодействие с системой". Та же техника работает для аудио и видео-потоков.
- Загружая изображение из `http://` или с другого удаленного источника, вы рискуете столкнуться с отображением в элементе изображения-заполнителя с красным значком "X". Для того, чтобы предотвратить это, перехватите событие `img.onerror` и предоставьте собственный элемент-заполнитель:

```
var myImage = document.getElementById('image');
myImage.onerror = function () { onImageError(this);}
function onImageError(source) {
source.src = "placeholder.png";
source.onerror = "";
}
```

- Теги `<script>` не поддерживаются внутри `<svg>`.
- Если у вас есть SVG-файл, вы можете загрузить его в элемент `img`, сославшись на файл в атрибуте `src`, но это не позволит вам обходить SVG в DOM. Если вам нужно подобное поведение, загрузите SVG в `iframe`. Содержимое SVG тогда будет расположено внутри свойства `contentDocument.documentElement`:

```
<!-- В HTML-->
<iframe id="Mysvg" src="myFolder/mySVGFile.svg" />

// В JavaScript
var svg = document.getElementById("Mysvg").contentDocument.docume
```

- Файлы в форматах PNG и JPEG обычно обрабатываются быстрее, чем SVG, поэтому, если технической необходимости в использовании SVG нет, или если вы имеете дело со сценарием, требующем высокой производительности, рассмотрите вариант использования масштабированных растровых изображений. Или вы можете динамически создавать масштабированные статические изображения из SVG для того, чтобы использовать их позже для более быстрого рендеринга:

```
<!-- В HTML-->
<img id="svg" src="somesvg.svg" style="display: none;" />
<canvas id="canvas" style="display: none;" />
// В JavaScript
var c = document.getElementById("canvas").getContext("2d");
c.drawImage(document.getElementById("svg"),0,0);
var imageURLToUse = document.getElementById("canvas").toDataURL(
```

- Вот два полезных материала по SVG: ссылка: http://www.carto.net/papers/svg/samples/ и ссылка: http://srufaculty.sru.edu/david.dailey/svg/.

Элементы Canvas

Все упомянутые здесь имена фукнций – это методы объекта контекста canvas:

- Помните, что элемент `canvas` нуждается в задании атрибутов `width` и `height` (в JavaScript – `canvas.width` и `canvas.height`), а не стилей. Он не принимает таких единиц измерения, как px, em, % и других.
- Несмотря на имя, метод `closePath` не является прямым дополнением метода `beginPath`. Метод `beginPath` используется для создания нового контура, который может быть обведен линией, сбрасывая существующий контур. Метод `closePath`, с другой стороны, просто соединяет две конечных точки текущего контура, как если бы вы исполнили для этих точек команду `lineTo`. Он не сбрасывает контур и не создает новый. Это часто вызывает у программистов путаницу, и поэтому иногда можно видеть окружности, нарисованные с линией, проведенной к их центру!
- Вызов `stroke` необходим для отрисовки контура, до этого вы можете рассматривать его как карандашный набросок чего-либо, который пока не закрашен тушью. Обратите так же внимание на то, что обведение контура означает вызов `beginPath`.
- При анимации содержимого canvas, выполнение `clearRect` для всего элемента и перерисовка каждого кадра обычно легче реализуема, чем очищение множества небольших областей и перерисовывание отдельных частей элемента. Хост-процесс приложения, в итоге, выполняет рендеринг всего элемента, со всем его содержимым, для каждого кадра, в целях управления прозрачностью, поэтому попытка оптимизировать производительность путем очищения небольших областей – это неэффективная стратегия, за исключением случаев, когда вы выполняете небольшое количество вызовов API для каждого кадра.
- API отрисовки элемента canvas реализованы путем конверсии его в соответствующие вызовы Direct2D GPU. При таком подходе фигуры выводятся с автоматическим антиалиасингом. В результате, рисование фигуры, наподобие незаполненной

того, чтобы эффективно очистить пиксели, используйте команду `clearRect` для области, которая немного больше, чем фигура. Это одна из причин, по которой очистка всего элемента и перерисовка каждого кадра часто оказывается самым простым решением.

- Для того чтобы задать в элементе canvas фоновое изображение (то есть, вам не придется каждый раз его перерисовывать), вы можете использовать свойство `canvas.style.backgroundImage` с подходящим URI для изображения
- Используйте метод `msToBlob` (ссылка: http://msdn.microsoft.com/library/windows/apps/hh465735.aspx) для объекта `canvas` чтобы получить объект типа `Blob` (ссылка: http://msdn.microsoft.com/library/windows/apps/hh453178.aspx) для содержимого элемента.
- При использовании `drawImage`, вам может понадобиться подождать какое-то время, пока необходимое изображение загрузится с использованием подобного кода:

```
var img = new Image();
img.onload = function () { myContext.drawImage(myImg, 0, 0); }
myImg.src = "myImageFile.png";
```

- Хотя другие графические API видят окружность как особый случай эллипса (с одними и теми же параметрами радиусов x и y), функция arc элемента canvas работает только с окружностями. К счастью, небольшое использование масштабирования упрощает рисование эллипсов, как показано ниже. Обратите внимание на то, что мы используем команды save и restore, таким образом, команда `scale` применяется только к `arc`. Она не воздействует на stroke, который используется из `main`. Это важно, так как если показатель масштабирования все еще действует, когда вы вызываете stroke, ширина линии будет меняться вместо того, чтобы оставаться постоянной.

```
var img = new Image();
img.onload = function () { myContext.drawImage(myImg, 0, 0); }
myImg.src = "myImageFile.png";
function arcEllipse(ctx, x, y, radiusX, radiusY, startAngle, endAngle, anticl
//Используем меньший радиус как базу и растягиваем другой
```

```
function arcEllipse(ctx, x, y, radiusX, radiusY, startAngle, endAngle, anticl
//Используем меньший радиус как базу и растягиваем другой

var radius = Math.min(radiusX, radiusY);
var scaleX = radiusX / radius;
var scaleY = radiusY / radius;
ctx.save();
ctx.scale(scaleX, scaleY);

//Обратите внимание на то, что центральная точка должна учиты
ctx.arc(x / scaleX, y / scaleY, radius, startAngle, endAngle, anticlockwise)
ctx.restore();
}
```

- Копируя пиксельные данные из видео, возможно с использованием canvas динамически манипулировать видеоданными (не влияя на источник, конечно). Это полезная техника, даже учитывая то, что она интенсивно использует процессор. По этой причине, однако, такой подход может не работать нормально на маломощных устройствах.

Вот пример покадровой манипуляции с видео, техника, которая отлично освещена в блоге команды разработки Windows, в сообщение "Прямая манипуляция пикселями в canvas"[1]. Кроме того, обратитесь к материалу ссылка: http://beej.us/blog/data/html5s-canvas-2-pixel/ (ссылка: http://blogs.windows.com/windows/archive/b/developers/archive/2011/02/15/canv direct-pixel-manipulation.aspx). В упражнении к этой лекции VideoEdit, файл default.html содержит в теле элементы `video` и `canvas`:

```
<video id="video1" src="ModelRocket1.mp4" muted style="display: none"></vi
<canvas id="canvas1" width="640" height="480"></canvas>
```

В коде (js/default.js), мы вызываем функцию `startVideo` из обработчика активации. Эта функция запускает воспроизведение видео и использует requestAnimationFrame для выполнения пиксельной манипуляции для каждого видеокадра:

```
function startVideo() {
```

```
ctx = canvas1.getContext("2d");

video1.play();
requestAnimationFrame(renderVideo);
}

function renderVideo() {
//Копируем кадр из видео в canvas
ctx.drawImage(video1, 0, 0, canvas1.width, canvas1.height);

//Получаем кадр в виде пиксельных данных
var imgData = ctx.getImageData(0, 0, canvas1.width, canvas1.height);
var pixels = imgData.data;

//Обходим пиксели, обрабатывая их так, как нам нужно
var r, g, b;

for (var i = 0; i < pixels.length; i += 4) {
r = pixels[i + colorOffset.red];
g = pixels[i + colorOffset.green];
b = pixels[i + colorOffset.blue];

// Здесь создается негативное изображение
pixels[i + colorOffset.red] = 255 - r;
pixels[i + colorOffset.green] = 255 - g;
pixels[i + colorOffset.blue] = 255 - b;
}

//Копируем измененные пиксели в canvas
ctx.putImageData(imgData, 0, 0);
//Запрашиваем следующий кадр
requestAnimationFrame(renderVideo);
}
```

Страница в этом примере содержит скрытый видеоэлемент (`style="display: none"`), в котором запускается воспроизведение при загрузке страницы (`video1.play()`). В цикле `requestAnimationFrame`, текущий кадр из видео копируется в

воспроизведение при загрузке страницы (`video1.play()`). В цикле `requestAnimationFrame`, текущий кадр из видео копируется в canvas (`drawImage`) и пиксели кадра копируются (`getImageData`) в буфер `imgData`. Затем мы обходим этот буфер и инвертируем цветовую информацию, создавая, таким образом, фотографический негатив изображения (альтернативная формула для перевода изображения в цветовой режим оттенков серого показана в комментариях и опущена выше). Затем мы копируем полученные пиксели обратно в canvas (`putImageData`), таким образом, когда мы возвращаемся, негатив изображения выводится на экран.

Повторюсь, подобные операции интенсивно используют процессор и обычно не подлежат ускорению с помощью GPU, то есть, могут выполняться не очень хорошо на маломощных устройствах (однако, не забудьте запустить Release build (построении Выпуска) программы за пределами отладчика для оценки производительности). Гораздо лучше будет написать DLL для реализации видеоэффектов, там, где это возможно, как обсуждается в разделе "Применение видеоэффектов" ниже. Тем не менее, об этой технике полезно знать. Что на самом деле происходит, так это то, что вместо рисования каждого кадра с помощью вызовов API, мы просто используем видео в качестве источника данных. Мы можем, если захотим, украсить canvas любым другим способом, прежде чем возвращаться из функции `renderVideo`. Пример подобного, который мне очень нравится, приведен в материале "Манипуляция видео с использованием canvas" (ссылка: https://developer.mozilla.org/En/Manipulating_video_using_canvas) на сайте разработчиков Mozilla, которыйди намически делает фоновые пиксели зеленого цвета в видеокадре прозрачными, в итоге, элемент `img`, расположенный под видео, становится видимым сквозь видео, выступая в роли фона. Тот же подход можно использовать для наложения двух видео, в итоге, то видео, которое будет расположено ниже, станет фоном вместо статического изображения. Опять же, помните о производительности маломощных устройств; вы можете решить предоставить параметр, посредством которого пользователь сможет отключить подобные дополнительные эффекты.

Проигрывание видео и отложенная загрузка

время выполнения программы дает вам возможность воспроизводить видео. В нижеприведенном коде источник видео является локальным файлом, видео начинает проигрываться самостоятельно, непрерывно повторяется и обеспечивает элементы управления воспроизведением:

```
<video src="media/ModelRocket1.mp4" controls loop autoplay></video>
```

Как обычно, мы не будем повторять то, что можно найти среди спецификаций W3C по тегам `video` и `audio`. Материалы об этом можно найти по адресу ссылка: http://www.w3.org/html/wg/drafts/html/master/embedded-content-0.html#the-video-element. Спецификации позволяют узнать о свойствах, методах и событиях этих элементов. Обратите особое внимание на сводные данные по событиям в разделе 4.8.10.15 (ссылка: http://www.w3.org/html/wg/drafts/html/master/embedded-content-0.html#event-definitions), и то, что большинство свойств и методов и для того и для другого можно найти в разделе о медиа-элементах, 4.8.10 (ссылка: http://www.w3.org/html/wg/drafts/html/master/embedded-content-0.html#media-elements). Обратите внимание на элемент `track`, который поддерживается и элементом `video`, и элементом `audio`, вы можете найти образец его использования в Сценарии 4 (демонстрацию субтитров) примера "Воспроизведение мультимедиа в HTML" (ссылка: http://code.msdn.microsoft.com/windowsapps/Media-Playback-Sample-3bdbe7c9), больше мы не будем на нем здесь останавливаться.

Так же полезно понимать, что `video` и `audio` тесно связаны, так как они являются частью оной и той же спецификации. На самом деле, если вы просто хотите проиграть звуковую дорожку видеоклипа, вы можете использовать объект `Audio` в JavaScript:

```
//Проигрывание лишь звуковой дорожки видеоклипа
var movieAudio = new Audio("http://www.kraigbrockschmidt.com/downloads/m

movieAudio.load();
movieAudio.play();
```

Для любого видео-элемента вы можете задать ширину и высоту для управления размером области воспроизведений (вплоть до 100% для полноэкранного воспроизведения). Это важно, когда ваше приложение

управления размером области воспроизведений (вплоть до 100% для полноэкранного воспроизведения). Это важно, когда ваше приложение переключается между режимами просмотра и вы, вероятно, имеете CSS-стили видеоэлемента в различных медиа-запросах. Кроме того, если у вас есть элемент управления для полноэкранного воспроизведения, просто сделайте размеры видео такими же, как размер окна просмота (и после вызова `Windows.UI.ViewManagement.ApplicationView.tryUnsna` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.ui.viewmanagement.app если приложение находится в прикрепленном режиме). В дополнение, когда вы создаете видеоэлемент с атрибугом controls, у него автоматически появится элемент управления для перевода в полноэкранный режим, в правом углу, который делает именно то, что нужно для приложений Магазина Windows:

Коротко говоря, вам не нужно делать чего-то особенного для того, чтобы это работало. Когда видео отображается в полноэкранном режиме, похожая кнопка (или клавиша Esc) позволяет вернуться к обычному режиму просмотра приложения.

Примечание. Если вам интересно, элементы audio и video не предоставляют каких-либо псевдо-селекторов CSS для стилизации панели инструментов. Как сказала дошкольная учительница моего сына (в отношении раздачи фруктового мороженного, но такой подход работает и здесь), "Вы получаете то, что получаете, вы не столкнетесь с истерикой и вы счастливы". Если вы хотите сделать что-то особенное с этими элементами управления, вам нужно выключить элементы по умолчанию и предоставить собственные элементы управления, которые будут соответствующим образом вызывать методы элемента.

Реализуя собственные элементы управления, убедитесь в том, что установили тайм-аут для их скрытия (либо скрывайте их, либо изменяйте их z-index), когда они не используются. Это особенно важно, если у вас есть кнопка для перевода видео в полноэкранный режим, как та, что имеется во встроенном наборе элементов, с помощью которой вы изменяете размер элемента в соответствии с разрешением экрана.

оптимизацию производительности, но это не делается, если какие-то еще элементы находятся поверх видео. Хорошая идея – отключать любую выполняемую анимацию и отключать ненужные фоновые процессы, наподобие рабочих веб-процессов.

Вы можете использовать различные события элемента video для того, чтобы узнать, когда видео воспроизводится или приостановлено посредством элементов управления, помимо других вещей (хотя события перехода в полноэкранный режим здесь нет), но вам так же следует соответствующим образом реагировать, когда используются аппаратные кнопки для управления мультимедиа-данными. Для этих целей прослушивайте событий, исходящие из объекта `Windows.Media.MediaControl` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.media.mediacontrol.asp: такие, как `playpressed`, `pausepressed` и так далее. (Это события WinRT, поэтому вызовите removeEventListener, по мере необходимости). Обратитесь к примеру "Конфигурация кнопок для мультимедиа" (ссылка: http://code.msdn.microsoft.com/windowsapps/Media-Buttons-ea57d8e2) для того, чтобы увидеть демонстрацию, но обычно добавление слушателей выглядит примерно так:

```
mediaControl = Windows.Media.MediaControl;
mediaControl.addEventListener("soundlevelchanged", soundLevelChanged, false
mediaControl.addEventListener("playpausetogglepressed", playpause, false);
mediaControl.addEventListener("playpressed", play, false);
mediaControl.addEventListener("stoppressed", stop, false);
mediaControl.addEventListener("pausepressed", pause, false);
```

Я так же упоминал, что вы можете отложить загрузку видео до того момента, как оно понадобится, и вместо него показать изображение для предварительного просмотра. Это можно реализовать с помощью атрибута poster, значением которого является изображение, которое нужно использовать:

```
<video id="video1" poster="media/rocket.png" width="640" height="480"></vid

var video1 = document.getElementById("video1");
var clickListener = video1.addEventListener("click", function () {
```

```
var video1 = document.getElementById("video1");
var clickListener = video1.addEventListener("click", function () {
video1.src = "http://www.kraigbrockschmidt.com/downloads/media/ModelRock

video1.load();

// Удалим прослушиватель для предотвращения пересечения с элемента
 video1.removeEventListener("click", clickListener);

video1.addEventListener("click", function () {
video1.controls = true;
video1.play();
});
});
```

В данном случае я не использую `preload="true"` и не задаю src, в итоге ничего не загружается до прикосновения к видео. Когда происходит прикосновение, прослушиватель удаляется, включаются собственные элементы управления видео и начинается воспроизведение. Это, конечно, обходной метод, чаще вы будете исопользовать `preload="true" controls src="..."` непосредственно в видеоэлементе, в то время, как атрибут poster будет обрабатывать изображения предварительного просмотра.

Отключение скринсейверов и экрана блокировки при воспроизведении

При воспроизведении видео, особенно – полноэкранном, важно отключить любые автоматические тайм-ауты, которые могут очистить экран или заблокировать устройство. Это можно выполнить с помощью объекта `Windows.System.Display.DisplayRequest` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.system.display.displayre Прежде чем начинать воспроизведение, создайте экземпляр этого объекта и вызовите его метод `requestActive`.

```
var displayRequest = new Windows.System.Display.DisplayRequest();
```

Если будет произведен такой вызов, вы можете быть уверены в том, что экран будет оставаться в активном состоянии, несмотря на бездействие пользователя. Когда проигрывание видео завершится, не забудьте вызвать `requestRelease`. Обратите внимание на то, что Windows автоматически деактивирует подобные запросы, когда приложение перемещается в фоновый режим, и повторно активирует их, когда пользователь переключает его в режим переднего плана.

Расширения API видеоэлемента

Помимо стандартов HTML5 для элементов video, в Windows 8 к нему добавлены некоторые дополнительные свойства и методы, как показано в следующей табилце и описано на странице "Видеоэлемент" (ссылка: http://msdn.microsoft.com/library/windows/apps/hh465962.aspx). Кроме того, взгляните на пример "Воспроизведение мультимедиа в HTML" (ссылка: http://code.msdn.microsoft.com/windowsapps/Media-Playback-Sample-3bdbe7c9), где вы можете найти примеры их использования.

Та

Свойства	
`msHorizontalMirror`	Логическое значение, кото когда источником видеоэл изображение в верной ори разделе "Выбор устройств
`msZoom`	Логическое значение, указ дисплея обрезая верхние/н приложению дать пользов области экрана – то есть, д найти в Сценарии 3 прим http://code.msdn.microsoft.co
`msIsLayoutOptimalForPlayback` (`onMSVideoOptimalLayoutChanged`)	Логическое значение, кото воспроизведения на основ `onMSVideoOptimalLa` "Оптимизация представле "Производительность ауди

(onMSVideoOptimalLayoutChanged)	onMSVideoOptimalLa "Оптимизация представле "Производительность ауди
msIsStereo3D	Логическое значение, кото метаданных видеофайла). Windows.Graphics.D http://msdn.microsoft.com/libr Приложения так же могут Windows.Graphics.Display.D http://msdn.microsoft.com/libr для того, чтобы узнать об других особенносте стерес видео в стереорежиме" (сс примера "Воспроизведени
msStereo3DRenderMode	Может принимать значени управлять воспроизведени
msStereo3DPackingMode	Может иметь значение no доступна приложениям, к (смотрите выше).
msRealtime	Активирует уменьшение п Это важно для приложени этим следует осмотритель воспроизведения с низкой к примеру "Связь в реальн eac73290).Активирует уме возможно. Это важно для пользоваться этим следует "Активация воспроизведе http://msdn.microsoft.com/libr (ссылка: http://code.msdn.mi
msPlayToDisabled, msPlayToPrimary, msPlayToSource	Свойства, связанные с фу Смотрите раздел "Воспро что это доступно так же д
	Массив описаний аудиоза комментариев). Установит проигрываемого аудиофай

msAudioCategory	воспроизведением и фоно что установка этого свойс значение "Communicat
msAudioDeviceType	Задает выходное устройст аудиоэлемента"."
Методы	
msFrameStep (onMSVideoFrameStepCompleted)	Перемещает видео на один вызывается, когда перемещ
msInsertVideoEffect msInsertAudioEffect msClearEffects	Добавляет или удаляет ви msInsertVideoEffect не дост
msSetMediaProtectionManager	Используется для DRM и с DRM" ближе к концу этой
msSetVideoRectangle	Задает размер внутренней
onMSVideoFrameStepCompleted (event)	Происходит при изменени

Атрибут source и пользовательские кодеки

Элементы видео (и аудио) могут использовать атрибут HTML5 `source` (источник). В веб-приложениях элементы с множеством источников используются для предоставления альтернативных видеоформатов в том случае, если клиентская система не имеет кодека, необходимая для воспроизведения из основного источника. Учитывая, что список форматов, поддерживаемых Windows 8, хорошо известен (обратитесь снова к материалу "Поддерживаемые форматы звука и видео" (ссылка: http://msdn.microsoft.com/library/windows/apps/hh986969.aspx), это не большая проблема для приложений Магазина Windows. Однакое, использование source все еще полезно, так как этот атрибут позволяет идентифицировать конкретный кодек для источника медиаданных:

```
<video controls loop autoplay>
<source src="video1.vp8" type="video/webm" />
</video>
```

```
<video controls loop autoplay>
<source src="video1.vp8" type="video/webm" />
</video>
```

Это важно, когда вы хотите предоставить пользовательский кодек для приложения посредством `Windows.Media.MediaExtensionManager`, что описано в разделе "Пользовательские кодеры/декодеры и обработчики схем" ниже в этой лекции, так как кодек идентифицирует расширение для загрузки и применения при декодировании. Я показал здесь в качестве примера WebM, так как он не доступен напрямую приложениям для Магазина Windows (хотя присутствует в Internen Explorer). Когда исполняется хост-процесс, приложение для Магазина Windows, содержащее вышеописанный элемемент `video`, будет выполнять поиск подходящего декодера для заданного параметра `type`.

Применение видеоэффектов

Вышеприведенная таблица показывает, что у видеоэлемента есть методы `msInsertVideoEffect` и `msInsertAudioEffect`. WinRT предоставляет встроенный эффект стабилизации видео, который легко применить к элементу. Это показано в Сценарии 3 примера "Мультимедийные расширения" (), где одно и то же видео проигрывается с использованием эффекта и без него, у стабилизированного видео отключен звук:

```
vidStab.msClearEffects();
vidStab.muted = true;
vidStab.msInsertVideoEffect(Windows.Media.VideoEffects.videoStabilization, tr
```

Пользовательские эффекты, как показано в Сценарии 4 примера, применяются в виде отдельных динамически связываемых библиотек (DLL), обычно написанных на C++ для лучшей производительности, и включенных в пакет приложения, так как приложения для Магазина Windows могут устанавливать DLL только для собственного использования, а не для общесистемного доступа. В примере вы

`msInsertVideoEffect`, который является строкой и идентифицирует эффект, экспортируемый DLL (смотрите, например, файл InvertTransform.idl в проекте InvertTransform):

```
vid.msInsertVideoEffect("GrayscaleTransform.GrayscaleEffect", true, null);

vid.msInsertVideoEffect("InvertTransform.InvertEffect", true, null);
```

Второй параметр `msInsertVideoEffect`, кстати, показывает, нужно ли применять эффект, обычно он имеет значение `true`. Третий параметр называется конфигурация (config), он содержит дополнительные сведения, которые передаются в эффект. В случае с геометрическими эффектами в примере, этот параметр задает конкретный вариант эффекта:

```
var effect = new Windows.Foundation.Collections.PropertySet();
effect["effect"] = effectName;
vid.msClearEffects();
vid.msInsertVideoEffect("PolarTransform.PolarEffect", true, effect);
```

Здесь `effectName` может принимать значения "Fisheye", "Pinch", или "Warp".

Аудиоэффекты, не показанные в примере, применяются так же, с помощью `msInsertAudioEffect` (с теми же параметрами). Обратите внимание на то, что каждый элемент может иметь не более двух эффектов в потоке. Элемент `video` может иметь два видеоэффекта и два аудиоэффекта, элемент `audio` может иметь два аудиоэффекта. Если вы попытаетесь добавить больше, метод выдаст исключение. Поэтому полезно вызывать `msClearEffects` перед добавлением эффектов.

Подробности об эффектах и других расширениях мультимедиа вы можете найти в материале "Применение расширений мультимедиа" (ссылка: http://msdn.microsoft.com/library/windows/apps/hh700365.aspx).

Обзор мультимедиа-серверов

Подробности об эффектах и других расширениях мультимедиа вы можете найти в материале "Применение расширений мультимедиа" (ссылка: http://msdn.microsoft.com/library/windows/apps/hh700365.aspx).

Обзор мультимедиа-серверов

Во многих домашних хозяйствах, и в моем тоже, имеется один или большее количество мультимедиа-серверов, доступных в локальной сети, с которых приложения могут воспроизводить мультимедийные данные. Добраться до этих серверов можно с помощью еще одного свойства `Windows.Storage.KnownFolders`, о котором мы пока не упоминали: `mediaServerDevices` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.storage.knownfolders.m Как и в слудчае с другими известными папками, это просто объект `StorageFolder`, посредством которого можно перечислять дополнительные папки и файлы или применять запросы к ним. В данном случае, если вы вызываете его метод `getFoldersAsync`, вы получаете список доступных серверов, каждый из которых представлен собственным объектом `StorageFolder`. Здесь вы можете использовать файловые запросы, как было описанов в лекции 2, для поиска мультимедийных да нных интересующих вас типов, или использовать критерии поиска, заданные пользователем. Вот пример, посвященный этому: "Клиент мультимедиа-сервера" (ссылка: http://code.msdn.microsoft.com/windowsapps/Media-Server-sample-fffbe490).

Воспроизведение и микширование звука

Как и в случае с видеоданными, элемент `audio` предоставляет собственные возможности по воспроизведению, в том числе – элементы управления, циклическое воспроизведение и автовоспроизведение:

```
<audio src="media/SpringyBoing.mp3" controls loop autoplay></audio>
```

Опять же, как описано выше, одни и те же спецификации W3C применимы и к элементу `video`, и к элементу audio. Уже известный код для проигрывания лишь звуковой дорожки видеофайла, мы можем

Как уже упомянуто ранее, создание объекта `Audio` без элементов управления и запуск воспроизведения не воздействует на макет, это обычно используется для воспроизведения звуковых эффектов в играх и в других приложениях.

Как и в случае с видео, для приложения, выполняющего воспроизведение звука, важно соответствующим образом реагировать на события, исходящие из объекта `Windows.Media.MediaControl` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.media.mediacontrol.asp в особенности на события `playpressed`, `pausepressed`, `stoppressed`, и `playpausetogglepressed`. Это позволяет пользователю управлять воспроизведением с помощью аппаратных кнопок, которые можно использовать при воспроизведении музыки, например. Однако, вы можете не применять обработку этих событий, например, при воспроизведении игровых звуков.

Говоря о воспроизведении звука, интересно рассмотреть вопрос о смешивании нескольких звуков, что обычно нужно в играх. Важно понимать, что каждый элемент `audio` может воспроизводить лишь одну запись, у него есть лишь один и только один файл-источник. Однако, несколько элементов audio могут проигрывать звуки одновременно с автоматическим микшированиев в зависимости от их категорий (смотрите раздел "Средство управления воспроизведением и фоновое аудио" ниже). В этом упражнении, фоновая музыка проигрывается непрерывно (`loop` установлено в `true`, громкость установлена на половину), в то время, как другой звук воспроизводится в ответ на касания (смотрите упражнение AudioPlayback в дополнительных материалах к лекции):[2)]

```
var sound1 = new Audio("media/SpringyBoing.mp3");
sound1.load(); //Для предварительной загрузки данных

//Фоновая музыка
var sound2 = new Audio();
sound2.msAudioCategory = "ForegroundOnlyMedia"; //Установим перед уста
sound2.src = "http://www.kraigbrockschmidt.com/mp3/WhoIsSylvia_PortlandO

sound2.loop = true;
```

```
sound1.load(); //Для предварительной загрузки данных

//Фоновая музыка
var sound2 = new Audio();
sound2.msAudioCategory = "ForegroundOnlyMedia"; //Установим перед уста
sound2.src = "http://www.kraigbrockschmidt.com/mp3/WhoIsSylvia_PortlandO

sound2.loop = true;
sound2.volume = 0.5; //50%;
sound2.play();

document.getElementById("btnSound").addEventListener("click", function () {
//Сбрасываем позицию, если уже воспроизводим
sound1.currentTime = 0;
sound1.play();
});
```

Загрузив звук, воспроизводимый при касании, мы знаем, что можем воспроизвести его в любое время. При запуске воспроизведения, полезно сбросить currentTime в значение 0, таким образом, звук всегда будет воспроизводиться сначала.

Вопросы о микшировании, особенно в играх, касаются управления различными звуками без знания наперед времени, когда они будут смешиваться. Вам может быть нужно, например, наложить друг на друга воспроизведение одного и того же звука, с различным временем начала, но объявлять три аудиоэлемента с одним и тем же источником непрактично. Представленная техника касается использования "ротации каналов" ("rotating channels"), как описано здесь: ссылка: http://ajaxian.com/archives/html5-audio-tutorial-rotating-channels. В итоге получается следующее:

1. Объявляем элементы audio для каждого звука (с установкой `preload="auto"`).
2. Создаем пул (массив) объектов `audio` для необходимого количества каналов
3. Для воспроизведения звука
    - Получаем доступный объект `Audio` из пула.

пары десятков каналов.

Подсказка. Нужны звуки для приложения? Загляните на ссылка: http://www.freesound.org/.

Расширения API аудиоэлемента

Как и в случае с элементом video, и для элемента audio доступны некоторы расширения, в частности те, что можно использовать для управления эффектами (`msInsertAudioEffect`), DRM (`msSetMediaProtectionManager`), для воспроизведения на целевых объектах (`msPlayToSource`, и так далее.), `msRealtime`, и `msAudioTracks`, как перечислено раньше в разделе о расшираниях API для видеоэлемента. На самом деле, каждое расширение API для аудио существует и для видео, но два из них имеют особую важность для элементов `audio`:`msAudioDeviceType` позволяет приложению определить, на какое выходное устройство осуществляется отправка звука: `"Console"` (по умолчанию) и `"Communications"`. Это способ, благодаря которому приложение знает, что оно участвует в сеансе связи (наподобие чата), что не связано с воспроизведением мультимедийных аудиоклипов.

`msAudioCategory` идентифицирует тип проигрываемой аудиозаписи (смотрите таблицу в следующем разделе). Это очень важно для идентификации того, следует ли продолжать воспроизведение музыки в фоновом режиме (тем самым предотвращая перевод приложения в приостановленный режим), как описано далее. Обратите внимание на то, что следует всегда устанавливать это свойство перед установкой `src`, и на то, что установка режима `"Communications"` так же устанавливает тип устройства на `"Communications"` и приводит к установке `msRealtime` в `true`.

Обратите внимание на то, что несмотря на схожесть значений в этих свойствах, `msAudioDeviceType` предназначено для выбора выходного устройства, в то время как `msAudioCategory` идентифицирует тип аудиозаписи, которая воспроизводится через любое устройство. Аудио с коммуникационной категорией, например,

так же устанавливает тип устройства на "Communications" и приводит к установке msRealtime в true.

Обратите внимание на то, что несмотря на схожесть значений в этих свойствах, msAudioDeviceType предназначено для выбора выходного устройства, в то время как msAudioCategory идентифицирует тип аудиозаписи, которая воспроизводится через любое устройство. Аудио с коммуникационной категорией, например, может воспроизводиться на консольном устройстве, а мультимедийное аудио – на коммуникационном устройстве. За этими двумя свойствами стоят разные концепции.

Управление воспроизведением и фоновое аудио

Для того, чтобы рассмотреть различные виды воспроизведения аудио, обратимся к примеру "Средство управления воспроизведением msAudioCategory" (ссылка: http://code.msdn.microsoft.com/windowsapps/Playback-Manager-e6526e67). Я не привожу здесь скриншот, так как пример лишь работает с аудио, и там особенно нечего показывать! ВВместо этого позвольте мне изложить особенности поведения различных его сценариев следующей таблице, а так же – некоторые категории, которые не представлены в примере, но могут быть использованы в ваших приложениях. В каждом сценарии вам сначала нужно выбрать аудиофайл с помощью средства выбора файлов:

Таблица 5.4.

Сценарий	msAudioCategory	Описание
		Воспроизводит выбранный аудиофайл, и когда приложен видимо, и в фоновом режим том числе тогда, когда пользователь находится на Рабочем столе, на Начальном экране или на экране блокировки. Приложение не приостанавливается, находяс

		медиафайлов, музыкальных видео и так далее. Использование этой категор требует объявления в маниф и наличия обработчиков аппаратных кнопок управле воспроизведением.
2	`Communications`	Как и в случае `BackgroundCapableMe` воспроизведение звука продолжается и когда приложение находится в фоновом режиме. Используй это для чатов, VoIP-приложе и так далее.
3	`Other` (по умолчанию для элементов `audio`)	Проигрывает выбранный аудиофайл, когда приложени находится на переднем план микшируется с фоновым ауди когда приложение переходит фоновый режим, воспроизведение приостанавливается.
4	`ForegroundOnlyMedia`	Проигрывает выбранный аудиофайл, когда приложени находится на переднем план Воспроизведение приостанавливается, когда приложение переходит в фоновый режим. Когда воспроизводится аудио из эт категории, фоновое аудио отключается.
		Проигрывает выбранный аудиофайл, когда приложени находится на переднем план

		фоновый режим. Когда воспроизводится аудио из эт категории, фоновое аудио отключается.
5	`Alert`	Проигрывает выбранный аудиофайл, когда приложени находится на переднем план ослабляет фоновое аудио. Используется для оповещени наподобие звонков, и для системных сообщений.
n/a	`GameMedia`	Используется для окружающ музыки в играх.
n/a	`GameEffects`	Используется для игровых звуковых эффектов, предназначенных для смешивания с воспроизводимыми звуками немузыкальные звуки).
n/a	`SoundEffects`	Другие звуковые эффекты (за пределами игр), которые предназначены для смешива с воспроизводимым звуком, такие, как короткие звонки, писки, стуки, бульканья, кото сопутствуют неким действия но не являются оповещения

Когда речь идет об отдельном звуковом потоке, между некоторыми из этих категорий нет особой разницы. Действительно, как показано в таблице, звуки разных категорий по-разному воздействуют на другие звуковые потоки, проигрываемые одновременно с ними. Для этой цели в Windows SDK сделан необычный ход: предоставлен второй пример, идентичный вышеописанному, "Дополнительный пример для средства управления воспроизведением" (ссылка: http://code.msdn.microsoft.com/windowsapps/Playback-Manager2-55c5b86d). Это позволяет вам запустить данные приложения одновременно (одно

Однко, самое важное, это чтобы вы присваивали каждому аудиопотоку наиболее соответствующую ему категорию. Эти категории помогают средству управления воспроизведением правильно микшировать звуковые потоки, основываясь на ожиданиях пользователя. Речь идет и о нескольких потоках в одном приложении, и о потоках аудио, поступающих от разных приложений (с ограничением количества приложений, которые могут одновременно воспроизводить аудио в фоновом режиме). Например, пользователи ожидают, что будильник, важная форма оповещения, временно приглушит другие аудиопотоки. Похожим образом, пользователи ожидают, что аудиопоток от приложения переднего плана имеет преимущество перед потоком той же категории, воспроизводимым в фоновом режиме. Как разработчик, поэтому, избегайте развлечений с категориями. Просто на значайте наиболее подходящие категории вашим аудиопотокам и позвольте средству управления воспроизведением делать свою работу с аудио, которое поступает из всех источников и создает последовательный опыт взаимодействия пользователя и системы в целом.

Установка категории для любого элемента `audio`, заключается в простом задании его атрибута `msAudioCategory`. Каждый сценарий в примере выполняет это одинаково, устанавливая категорию перед установкой атрибута `src` (здесь показан код из js/backgroundcapablemedia.js):

```
audtag = document.createElement('audio');
audtag.setAttribute("msAudioCategory", "BackgroundCapableMedia");
 audtag.setAttribute("src", fileLocation);
```

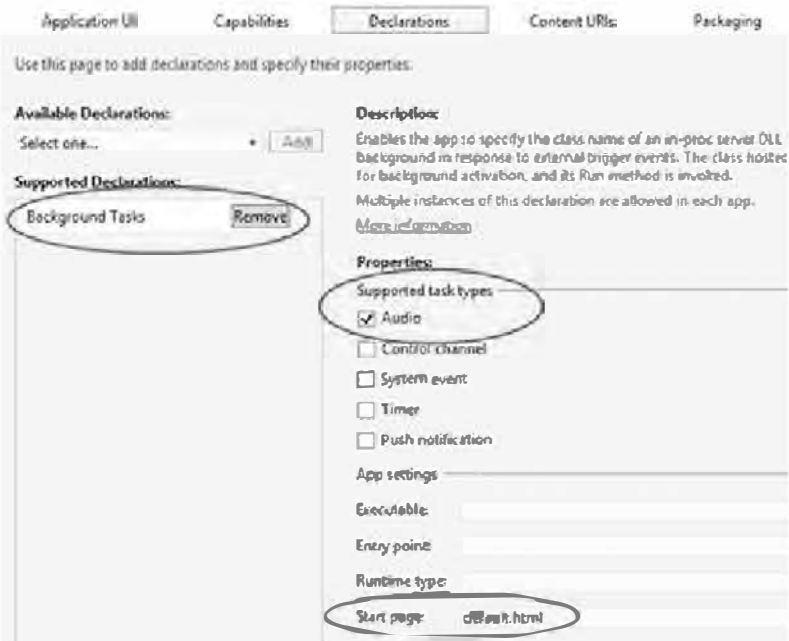
Конечно, вы можете сделать то же самое в разметке. Вот примеры:

```
<audio id="audio1" src="song.mp3" msAudioCategory="BackgroundCapableM
<audio id="audio2" src="voip.mp3" msAudioCategory="Communications"></au
<audio id="audio3" src="lecture.mp3" msAudioCategory="Other"></audio>
```

Однако, в случае с `BackgroundCapableMedia` and `Communcations` недостаточно лишь установить категорию: вам так же нужно задать расширение фоновой задачи в манифесте. Это просто

```
<audio id="audio2" src="voip.mp3" msAudioCategory="Communications"></au
<audio id="audio3" src="lecture.mp3" msAudioCategory="Other"></audio>
```

Однако, в случае с `BackgroundCapableMedia` and `Communcations` недостаточно лишь установить категорию: вам так же нужно задать расширение фоновой задачи в манифесте. Это просто сделать на закладке Объявления (Declarations) редактора манифеста:



Во-первых, выберите Фоновые задачи (Background tasks) из выпадающего списка Доступные объявления (Available declarations). Затем установите флаг Аудио (Audio) в группе Поддерживаемые типы задач (Supported task types) и укажите начальную страницу (Start page) в разделе параметров приложения (App settings). Начальная страница не особенно важна для фонового аудио (так приложение никогда не будет запущено извне для этой цели), но это нужно для соответствия правилам заполнения манифеста.

В XML-файле манифеста эти объявления выглядят следующим образом, взгляните на них:

```
<Application Id="App" StartPage="default.html">
```

```
</Extensions>
</Application>
```

Кроме того, приложения, воспроизводящие фоновое аудио, должны доблять прослушиватели для событий `Windows.Media.MediaControl` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.media.mediacontrol.asp: как мы уже упоминали, так пользователь сможет управлять воспроизведением фонового аудио с помощью элементов управления воспроизведением мультимедиа (смотрите следующий раздел). Это так же необходимо, потому что это позволяет средству управления воспроизведением управлять аудиопотоками, когда пользователь переключается между приложениями. Если вы не предоставите эти прослушиватели, ваше аудио всегда будет приостановлено, звук будет выключен, когда приложение переходит в фоновый режим.

Как это сделать, показано в примере о средстве управления воспроизведением. Вот код (часть опущена) из файла js/communication.js:

```
mediaControl = Windows.Media.MediaControl;

mediaControl.addEventListener("soundlevelchanged", soundLevelChanged, false
mediaControl.addEventListener("playpausetogglepressed", playpause, false);
mediaControl.addEventListener("playpressed", play, false);
mediaControl.addEventListener("stoppressed", stop, false);
mediaControl.addEventListener("pausepressed", pause, false);


// переменная audtag это глобальный аудиоэлемент для страницы
function playpause() {
if (!audtag.paused) {
audtag.pause();
} else {
audtag.play();
}
}
function play() {
audtag.play();
```

```
audtag.pause();
} else {
audtag.play();
}
}
function play() {
audtag.play();
}

function stop() {
// Здесь ничего не делается
}

function pause() {
audtag.pause();
}
function soundLevelChanged() {
//Перехватываем уведомление SoundLevel и определяем его состояние.
//Если звук выключен, приостанавливаем воспроизведение.
var soundLevel = Windows.Media.MediaControl.soundLevel;
//Здесь ничего не делается, параметры приведены для показа содержим
switch (soundLevel) {
case Windows.Media.SoundLevel.muted:
break;
case Windows.Media.SoundLevel.low:
break;
case Windows.Media.SoundLevel.full:
break;
}
appMuted();
}
function appMuted() {
if (audtag) {
if (!audtag.paused) {
audtag.pause();
}
}
}
```

следующем разделе).

Вот некоторые дополнительные замечания, касающиеся фонового аудио:

- Причина, по которой мы различаем события `paypressed`, `pausepressed` и `playpaysepressed` заключается в поддержке разнообразного аппаратного обеспечения, при этом некоторые устройства имеют раздельные кнопки запуска воспроизведения и постановки на паузу, а у некоторых предусмотрена лишь одна кнопка и для того, и для другого.
- Если воспроизведение аудио приостановлено, приложение, воспроизводящее фоновое аудио, будет приостановлено, как и прочие, но если пользователь нажмет кнопку воспроизведения, приложение возобновит работу и продолжит проигрывание аудио.
- Использование функции фонового воспроизведения аудио тщательно оценивается при отправке приложения в Магазин Windows. Если вы попытаетесь проигрывать беззвучную запись для того, чтобы предотвратить приостановку приложения, приложение не пройдет сертификацию.
- Приложения фонового воспроизведения аудио должны осторожно использовать сетевое соединение для приема потоковых данных для поддержки состояния низкого потребления энергии, которое называется режим ожидания с подключением (connected standby). Подробности вы можете узнать в материале "Создание приложений фонового воспроизведения мультимедиа, экономящих энергию" (ссылка: http://msdn.microsoft.com/library/windows/desktop/jj247568.aspx).

Посмотрим теперь на еще одну важную причину, по которой мы должны реализовывать события элементов управления мультимедиа: пользовательский интерфейс, который Windows отображает в ответ на нажатия аппаратных кнопок.

Пользовательский интерфейс управления воспроизведением

Пользовательский интерфейс управления воспроизведением

Как было упомянуто в предыдущем разделе, приложениям, которые воспроизводят фоновое аудио, нужно предоставлять обработчики событий объекта MediaControl. Благодаря этому пользовали могут управлять воспроизведением с помощью аппаратных кнопок (они встроены во многие устройства, в том числе – клавиатуры, пульты дистанционного управления) без необходимости переключаться на приложение. Это особенно важно, так как фоновое аудио продолжает воспроиизводиться не только тогда, когда пользователь переключается к другому приложению, но и тогда, когда пенеходит на Начальный экран, и на рабочий стол, или блокирует устройство.

Стандартный пользовательский интерфейс для управления воспроизведением, как показано на рис. 5.3., расположен в левом верхнем углу экрана, независимо от того, что в данный момент отображается на экране. Прикосновение к имени приложения приводит к переключению на приложение.

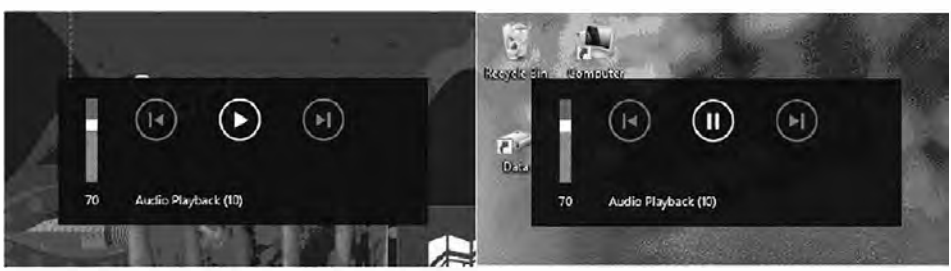
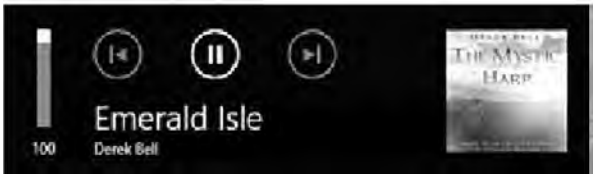


Рис. 5.3. Интерфейс управления воспроизведением аудио отображается поверх Начального экрана (слева) и рабочего стола (справа)

На изображениях вы можете видеть имя приложения, по умолчанию отображаемое в интерфейсе, взятое из свойства Отображаемое имя (Display Name), из манифеста. Хотя это и применимо, в идеале аудиоприложению следует передавать подробные метаданные записи в

Buttons-ea57d8e2), который показывает, как получать сведения об альбоме для аудиозаписи, мы вернемся к этому позже.

С использованием этих метаданных, интерфейс управления воспроизведения выглядит так, как показано ниже. Прикосновение к изображению альбома, названию записи или имени исполнителя приводит к переходу в приложение.



Вы можете заметить в вышеприведенных изображенийх, что кнопки перехода на следующую и предыдущую запись заблокированы. Это потому, что приложение не имеет прослушивателей для событий `nexttrackpressed` и `previoustrackpressed` объекта `MediaControl`. Мы увидим, как пользоваться ими, в следующем разделе. Есть, так же, и другие события, такие, как `channeldownpressed`, `channeluppressed`, `fastforwardpressed`, `rewindpressed`, `recordpressed`, и `stoppressed`, хотя они и не представлены в интерфейсе.

Последовательное воспроизведение аудиозаписей

Приложения, которые воспроизводят аудиозаписи (такие, как музыку, аудиокниги или записи лекций) часто имеют некоторый список записей для последовательного воспроизведения, в особенности, когда приложение исполняется в фоновом режиме. В данном случае важно быстро начинать воспроизведение следующей записи, так как в противном случае Windows приостановит приложение через 10 секунд после завершения воспроизведения текущей записи. Для этой цели прослушивают событие `ended` элемента `audio` и устанавливают `audio.src` на следующую запись. В данном случае хорошая оптимизация заключается в использовании второго объекта Audio и установка его атрибута src после того, как первая запись начнет воспроизводиться. Таким образом, вторая запись будет предварительно

`audio.src` на следующую запись. В данном случае хорошая оптимизация заключается в использовании второго объекта Audio и установка его атрибута src после того, как первая запись начнет воспроизводиться. Таким образом, вторая запись будет предварительно загружена и готова к немедленому воспроизведению, таким образом, устраняя потенциальные задержки между записями при воспроизведении. Это реализовано в упражнении к этой лекции AudioPlayback, где я разделил одну песню на четыре сегмента для непрерывного воспроизведения. Я так же покажу здесь, как обрабатывать событии кнопок "Предыдущая запись" и "Следующая запись", вместе с установкой номера сегмента в качестве названия записи:

```
var mediaControl = Windows.Media.MediaControl;
var playlist = ["media/segment1.mp3", "media/segment2.mp3", "media/segment3.
"media/segment4.mp3"];
var curSong = 0;
var audio1 = null;
var preload = null;

document.getElementById("btnSegments").addEventListener("click", playSegmer
audio1 = document.getElementById("audioSegments");
preload = document.createElement("audio");

function playSegments() {
//Всегда сбрасывайте прослушиватели событий объектов
WinRT для предотвращения дублирования и утечек памяти
mediaControl.removeEventListener("nexttrackpressed", nextHandler);
mediaControl.removeEventListener("previoustrackpressed", prevHandler);

curSong = 0;

//Приостановим другую музыку
document.getElementById("musicPlayback").pause();

//Set up media control listeners setMediaControl(audio1);

//Покажем элемент (изначально скрытый) и начнем воспроизведение
audio1.style.display = "";
```

```
var preload = document.createElement("audio"); preload.setAttribute("preload", "
preload.src = playlist[1];

//Переключаемся на следующую запись, как только очередная окончится
будет нажат кнопка "Следующая запись"
audio1.addEventListener("ended", nextHandler);
mediaControl.addEventListener("nexttrackpressed", nextHandler);
}
}
function nextHandler () {
curSong++;

//Активируем кнопку "Предыдущая запись", если у нас есть
 хотя бы одна предыдущая запись
if (curSong > 0) {
mediaControl.addEventListener("previoustrackpressed", prevHandler);
}

if (curSong < playlist.length) {
//playlist[curSong] уже должен быть загружен
playCurrent();

//Установка следующей заранее загруженной композиции
var nextTrack = curSong + 1;

if (nextTrack < playlist.length) {
preload.src = playlist[nextTrack];
} else {
preload.src = null;
mediaControl.removeEventListener("nexttrackpressed", nextHandler);
}
}
}

function prevHandler() {
//Если уже проигрываем последнюю запись,
снова добавим обработчик кнопки "Следующая запись"
if (curSong == playlist.length - 1) {
mediaControl.addEventListener("nexttrackpressed", nextHandler);
```

```
//Если уже проигрываем последнюю запись,
снова добавим обработчик кнопки "Следующая запись"
if (curSong == playlist.length - 1) {
mediaControl.addEventListener("nexttrackpressed", nextHandler);
}

curSong--;

if (curSong == 0) {
mediaControl.removeEventListener("previoustrackpressed", prevHandler);
}

playCurrent();
preload.src = playlist[curSong + 1]; //Это должно работать всегда
}

function playCurrent() {
audio1.src = playlist[curSong];
audio1.play();
mediaControl.trackName = "Segment " + (curSong + 1);
}
```

При последовательном проигрывании записей, как здесь, из приложения, написанного на JavaScript и HTML, вы можете отметить небольшие паузы между записями, особенно, если первая запись напрямую переходит во вторую. Это существующие ограничения платформы, где существуют уровни между HTML-элементом audio и низкоуровневым API XAudio2, которое, в конечном счете, выполняет реальную работу. В какой-то мере вы можете смягчить этот эффект, например, вы можете организовать плавный переход одной записи в другую, или наложить третью запись на конец первой и начало второй. Так же вы можете использовать отрицательное значение смещения по времени для начала воспроизведения второй записи немного раньше, чем будет окончена первая. Но если вам нужен по-настоящему бесшовный переход между записями, вам нужно обойти элемент audio и использовать API XAudio2 из WinRT-компонента для непосредственного воспроизведения музыки. О том, как это сделать, идет речь в сообщении блога для разработчиков приложений для Windows 8:

1)

2) И, конечно, я играю на гитаре и пою главную партию в этой живой записи, вместе с моим другом Тедом Катлером. Песня "Who is Sylvia?" написана другим моим другом, Дж. Дональдом Уолтерсом, на стихи Шекспира.

Списки воспроизведения

Пользователь может хранить медиафайлы везде, но изображения, музыка и видео обычно хранятся в соответствующих библиотеках пользователя. в лекции рассмотрена занрузка мультимедийных данных и управление ими. Потоковое мультимедиа и целевые объекты воспроизведения.

Пример многосегментного воспроизведения в предыдущем разделе, это лишь учебное упражнение, так как приложения обычно не работают со списками воспроизведения, расположенными в памяти. Веротянее всего, приложение загрузит существующий список воспроизведения или создаст плейлист из файлов, которые выбрал пользователь.

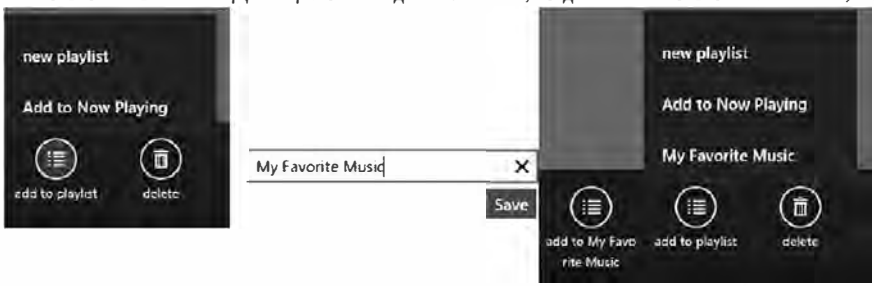
WinRT поддерживает эти действия посредством простого API в пространстве имен `Windows.Media.Playlists` (ссылка: http://msdn.microsoft.com/en-us/library/windows/apps/br206938.aspx), используя форматы WPL (Windows Media Player), ZPL (Zune) и M3U. Пример "Список воспроизведения" (ссылка: http://code.msdn.microsoft.com/windowsapps/Playlist-sample-3d80daee) в Windows SDK (который почти победил в конкурсе на самое короткое название примера!) показывает, как выполнять различные задачи с данным API. В Сценарии 1 он позволяет вам выбрать несколько файлов с использованием средства выбора файлов, создает новый объект Windows. `Media.Playlists.Playlist` (ссылка: http://msdn.microsoft.com/en-us/library/windows/apps/windows.media.playlists.playlist.aspx), добавляет выбранные файлы в его список `files` (вектор объектов `StorageFile`) и сохраняет список воспроизведения с помощью метода `saveAsAsync` (этот код из create.js упрощен и слегка переформатирован):

```
function pickAudio() {
var picker = new Windows.Storage.Pickers.FileOpenPicker();
picker.suggestedStartLocation = Windows.Storage.Pickers.PickerLocationId.mu
picker.fileTypeFilter.replaceAll(SdkSample.audioExtensions);

picker.pickMultipleFilesAsync().done(function (files) {
```

```
if (files.size > 0) {
SdkSample.playlist = new Windows.Media.Playlists.Playlist();

files.forEach(function (file) { SdkSample.playlist.files.append(file);
});

SdkSample.playlist.saveAsAsync(Windows.Storage.KnownFolders.musicLibrar
"Sample", Windows.Storage.NameCollisionOption.replaceExisting,
Windows.Media.Playlists.PlaylistFormat.windowsMedia)
.done();
}
}
```

Обратите внимание на то, что `saveAsAsync` принимает объект типа `StorageFolder` и имя для файла (вместе с необязательным параметром формата). Это соответствует обычно используемому шаблону для списков воспроизведения, где у музыкального приложения есть отдельная папка, где оно хранит списки воспроизведения и предоставляет пользователю простые средства для их именования и/или выбора. При таком подходе списками воспроизведения обычно управляют не так как пользовательскими файлами, когда действия выполняются с помощью средства выбора файлов для выполнения команды Сохранить как (Save as) в произвольной папке. Вы можете использовать `FileSavePicker`, получить `StorageFile`, и использовать его свойство `path` для получения подходящего объекта `StorageFolder`, но вероятнее всего, вы будете сохранять списки воспроизведения в одном месте и представлять их в виде сущностей, которые появляются лишь внутри самого приложения.

Например, приложение Музыка (Music), поставляемое вместе с Windows 8, позволяет вам создавать новый список воспроизведения, когда вы просматриваете записи некоторого альбома. Следующие команды появляются на панели приложения (слева), и когда вы выбираете команду Новый список воспроизведения (New playlist), появляется всплывающий элемент (в середине), запрашивающий имя, после чего всплывающий элемент появляется на панели приложения (справа):



Созданный список воспроизведения затем появляется в приложении в виде другого альбома. Другими словами, хотя плейлисты можно сохранить как самостоятельные файлы, обычно нет нужды представлять их в подобном виде пользователю, и API отражает подобный шаблон их использования.

Для загрузки плейлиста используется метод `Playlist.loadAsync`, которому передается `StorageFile` для плейлиста. Это может быть объект StorageFile, полученный от средства выбора файлов, или из перечисления выделенной для списков воспроизведения папки приложения. Сценарий 2 примера "Список воспроизведения" (display.js) показывает последний подход, после чего осуществляется обход файлов и запрос их музыкальных свойств:

```
function displayPlaylist() {
var picker = new Windows.Storage.Pickers.FileOpenPicker();
picker.suggestedStartLocation = Windows.Storage.Pickers.PickerLocationId.mu
picker.fileTypeFilter.replaceAll(SdkSample.playlistExtensions);

var promiseCount = 0;

picker.pickSingleFileAsync()
.then(function (item) {
if (item) {
return Windows.Media.Playlists.Playlist.loadAsync(item);
}
return WinJS.Promise.wrapError("No file picked.");
```

```
})
.then(function (playlist) { SdkSample.playlist = playlist;
var promises = {};

// Запрос музыкальных свойств для каждого файла в списке.
playlist.files.forEach(function (file) {
promises[promiseCount++] = file.properties.getMusicPropertiesAsync();
});

// Вывод музыкальных свойств каждого файла. По причине асинхронно
// природы вызова по получению музыкальных свойств, данные могут п
// в ином порядке, нежели они заданы в исходном списке воспроизведе
// Для обеспечения верного порядка, мы используем Promise.join с ассоц
// переданным в качестве параметра, содержащим индекс каждого отдел
return WinJS.Promise.join(promises);
})
.done(function (results) {
var output = "Playlist content:\n\n";

var musicProperties;
for (var resultIndex = 0; resultIndex < promiseCount; resultIndex++) {
musicProperties = results[resultIndex];
output += "Title: " + musicProperties.title + "\n";
output += "Album: " + musicProperties.album + "\n";
output += "Artist: " + musicProperties.artist + "\n\n";
}

if (resultIndex === 0) {
output += "(playlist is empty)";
}

}, function (error) {
// ...
});
}
```

Мы вернемся к работе с этими специальными свойствами в следующем разделе, так как их обработка применима и к другим типам

мультимедийных данных.

Другой метод управления плейлистами – это `PlayList.saveAsync`, который принимает единственный `StorageFile`. Такой подход используется, когда вы загружаете и модифицируете список воспроизведения и просто хотите сохранить эти изменения (что обычно выполняется автоматически, когда пользователь добавляет или удаляет элементы плейлиста). Это показано в Сценариях 3, 4 и 5 примера (add.js, js/remove.js и js/clear.js), где просто использованы методы вектора `Playlist.files`, наподобие `append`, `removeAtEnd` и `clear`, соответственно.

Проигрывание списка воспроизведения, конечно, зависит от типов включенных в него элементов, но обычно загружают плейлист и последовательно получают объекты `StorageFile` из его вектора `files`, передают их в `URL.createObjectURL`, и затем присваивают полученный URI атрибугу `src` элемента `audio` или `video`. Вы можете, кроме того, использовать плейлисты для управления списками изображений для слайд-шоу

Загрузка мультимедийных данных и управление ими

Пользователь может хранить медиафайлы везде, но изображения, музыка и видео обычно хранятся в соответствующих библиотеках пользователя. Проще говоря, это папки, которые медиа-приложениям следует использовать по умолчанию до тех пор, пока пользователь не укажет иную папку с помощью средства выбора папок. Как мы видели в лекции 2, приложения могут объявлять о программном доступе к библиотекам изображений, музыки и видео в манифестах, и получать объекты `StorageFolder` для них посредством `Windows.Storage.KnownFolders`:

```
var picsLib = Windows.Storage.KnownFolders.picturesLibrary;
var musicLib = Windows.Storage.KnownFolders.musicLibrary;
var vidsLib = Windows.Storage.KnownFolders.videosLibrary;
```

Приложения для работы с фотографиями обычно объявляют

возможность Библиотека изображений (Pictures Library), и отображают ее содержимое в ListView. Приложения для работы с музыкой и видео так же используют соответствующие библиотеки, как вы можете видеть во встроенных приложениях Windows 8 Фотографии (Photos), Музыка (Music) и Видео (Video). Помните так же, что если вы забудете объявить соответствующие возможности, строки кода выше приведут к выдаче исключения "Доступ запрещен". Вы тут же поймете, если вы забудете эти важные особенности.

Я должен предупредить вас заранее, что работа с мультимедиа может быть очень сложной и запутанной. Поэтому вы, возможно, сочтете полезным просмотреть несколько материалов в документации, таких, как "Обработка файлов изображений" (ссылка: http://msdn.microsoft.com/library/windows/apps/hh465103.aspx), "Перекодировка" (ссылка: http://msdn.microsoft.com/ru-ru/library/windows/apps/hh452806.aspx), "Применение расширений мультимедиа" (ссылка: http://msdn.microsoft.com/library/windows/apps/Hh700365.aspx).

Метаданные мультимедийных файлов

Когда у нас есть `StorageFolder` для некоторой медиабиблиотеки или некоторый набор таких объектов, мы можем использовать, как мы уже видели в лекции 2, их метод `getItemAsync` для получения их содержимого. Так же вы можете использовать файловые запросы для перечисления файлов, которые подпадают под некоторый критерий. В любом случае, в итоге вы получаете коллекцию объектов `storageFile`, с которой вы можете работать так, как вам нужно.

Теперь начинается кое-что интересное. Как я упоминал в лекции 2, вы можете получать дополнительные метаданные для этих файлов. Имеется несколько уровней данных, которые вы будете исследовать, начиная открывать тайные двери класса StorageFile, как показано на рис. 6.1. Следующий раздел, в свою очередь, посвящем этим областям.

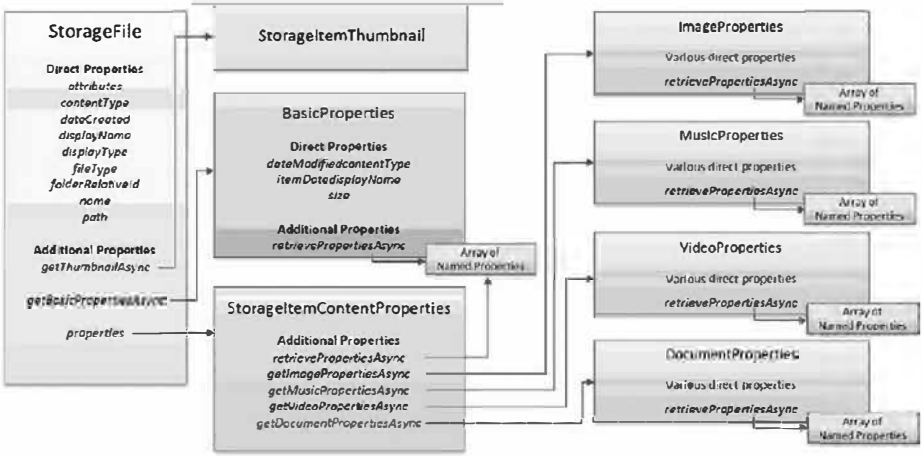


Рис. 6.1. Взаимосвязь между объектом StorageFile и другими объектами, получаемыми с его помощью

Эскизы

Во-первых, вызов StorageFile.getThumbnailAsync предоставляет изображение эскиза, соответствующее конкретному "режиму" из перечисления `Windows.Storage.FileProperties.ThumbnailMode` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.storage.fileproperties.thu Здесь есть такие варианты, как `picturesView`, `videosView`, `musicView`, `documentsView`, `listView`, и `singleItem`. То, что вы получите в обработчике завершения, это объект StorageItemThumbnail (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.storage.fileproperties.stc который предоставляет данные эскиза в виде потока. Вы можете удобно передать его нашему старому другу `URL.createObjectURL` для отображения в элементе `img` и так далее.

Примеры этого найти в примере "Эскизы файлов и папок" (ссылка: http://code.msdn.microsoft.com/windowsapps/File-thumbnails-sample-17575959). Сценарий 1, например, (js/scenario1.js), предоставляет эскиз и отображает его в элементе `img`:

```
file.getThumbnailAsync(thumbnailMode, requestedSize, thumbnailOptions).done(
if (thumbnail) {
outputResult(file, thumbnail, modeNames[modeSelected], requestedSize);
}
// ...
});

function outputResult(item, thumbnailImage, thumbnailMode, requestedSize) {
document.getElementById("picture-thumb-imageHolder").src = URL.createObje
{ oneTimeOnly: true });
// ...
}
```

Общие свойства файлов

Общие свойства файлов – это те свойства, которые имеются у всех файлов – находятся во множестве различных мест. Наиболее общие свойства можно обнаружить непосредственно в объекте `StorageFile`, наподобие `attributes`, `contentType`, `dateCreated`, `displayName`, `displayType`, `fileType`, `name`, и `path`.

Свойства следующей группы можно получить из `StorageFile.getBasicPropertiesAsync`. Этот вызов предоставляет вам объект `Windows.Storage.FileProperties.BasicProperties` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.storage.fileproperties.ba который содержит свойства `dateModified`, `itemDate`, и `size`. "Вот скукота!", скажете вы себе. Ну хорошо, у этого объекта еще есть дополнительный метод, который называется `retrievePropertiesAsync`, который предоставляет вам массив пар имя – значение для разных штуковин.

Весь фокус в том, чтобы понять, что вы можете взять массив имен свойств, которые вам нужны, где каждое имя – это строка, происходящая из очень длинного списка свойств Windows (ссылка:

http://msdn.microsoft.com/library/windows/desktop/dd561977.aspx), таких, как System.FileOwner и System.FileAttributes, и передать его `retrievePropertiesAsync` . Пример этого можно найти в Сценарии 5 примера "Перечисление папок" (ссылка: http://code.msdn.microsoft.com/windowsapps/Folder-enumeration-sample-33ebd000), который мы видели в лекции 2:

```
 var dateAccessedProperty = "System.DateAccessed";
var fileOwnerProperty  = "System.FileOwner";

SdkSample.sampleFile.getBasicPropertiesAsync().then(function (basicProperties
outputDiv.innerHTML += "Size: " + basicProperties.size + " bytes<br />";
outputDiv.innerHTML += "Date modified: " + basicProperties.dateModified + "<

// Получить дополнительные свойства
return SdkSample.sampleFile.properties.retrievePropertiesAsync([fileOwnerProp
}).done(function (extraProperties) {
var propValue = extraProperties[dateAccessedProperty];
if (propValue !== null) {
outputDiv.innerHTML += "Date accessed: " + propValue + "<br />";
}
propValue = extraProperties[fileOwnerProperty];
if (propValue !== null) {
outputDiv.innerHTML += "File owner: " + propValue;
}
});
```

Что здесь особенно удобно, так это то, что здесь вы можете получить почти любое свойство, которое вам нужно (список свойств содержит тысячи позиций), затем модифицировать массив и вызвать `BasicProperties.savePropertiesAsync`. Вуаля! Только что вы обновили эти свойства файлов. Вариация savePropertiesAsync так же позволяет вам передавать специальный массив из пар имя – значение, если вы лишь хотите изменить лишь некоторые из свойств.

Третий набор свойств можно найти, если пройти через тайную дверь `StorageFile.properties`. Здесь содержится объект `StorageItemContentProperties` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.storage.fileproperties.stc

методы которого retrievePropertiesAsync и savePropertiesAsync похожи на те, что мы видели у BasicProperties. Гораздо интереснее то, что у него есть еще четыре метода: getDocumentPropertiesAsync, getImagePropertiesAsync, getMusicPropertiesAsync, и getVideoPropertiesAsync. С их помощью вы можете получить по-настоящему специальные сведения для конкретных типов файлов, как мы увидим дальше.

Специальные свойства мультимедиа

Вместе с классом BasicProperties в пространстве имен Windows.Storage.FileProperties мы так же можем найти классы, возвращаемые методами StorageFile.properties.get*PropertiesAsync: ImageProperties, VideoProperties, MusicProperties, и DocumentProperties. Хотя мы и так довольно далеко забрались, чтобы их найти, каждый из них, по-настоящему глубоко, содержит несметные информационные сокровища. Таблицы ниже, в свою очередь, приводят сводные данные по ним. Обратите внимание на то, что объект каждого типа содержит метод retrievePropertiesAsync, похожий на соответствующий метод BasicProperties, который позволяет вам запрашивать дополнительные свойства по имени, которые не включены в свойства основного объекта. Воспользуйтесь ссылками в верхней части таблицы для того, чтобы воспользоваться справочными разделами, которые позвлят обнаружить наиболее подходящие свойства Windows.

Таблица 6.1.

ImageProperties	из **StorageFile.properties.get**
Дополнительные свойства	**System.Image, System.Phc**
Свойство	Тип данных
title	String
dateTaken	Date

latitude	Double (смотрите ниже)
longitude	Double (смотрите ниже)
cameraManufacturer	String
cameraModel	String
width	Количество в пикселях
height	Количество в пикселях
orientation	Windows.Storage.FileProperties.PhotoOrientation содержит unspecified, normal, flipHorizontal, flipVertical, transpose, transverse, rotate90 rotate180, rotate270
peopleNames	String vector
keywords	String vector
rating	Число (1-99 с 0 означающим "нет оценки")

Таблица 6.2.

VideoProperties	из **StorageFile.properties.getVid**	
Дополнительные свойства	**System.Video, System.Media, System**	
Свойство	Тип данных	
title	String	Sy
subtitle	String	Sy
year	Number	Sy
publisher	String	Sy

rating	Number	Sy
width	Number in pixels	Sy
height	Number in pixels	Sy
orientation	Windows.Storage.FileProperties.-VideoOrientation содержит normal, rotate90, rotate180, rotate270	Sy
duration	Число (в 100-нс единицах, т.е. 1/10th миллисекунды)	Sy
bitrate	Число (в битах в секунду)	Sy Sy
directors	String vector	Sy
producers	String vector	Sy
writers	String vector	Sy
keywords	String vector	Sy
latitude	Double (смотрите ниже)	Sy ко Sy Sy an
longitude	Double (смотрите ниже)	Sy ил Sy Sy Sy an

Таблица 6.3.

MusicProperties	из **StorageFile.properties.getMusicPro**	
Дополнительные свойства	**System.Music, System.Med**	
Свойство	Тип данных	Соответствующее с
title	String	System.Title, System.M
subtitle	String	System.Media.SubTitl
trackNumber	Number	System.Music.TrackN

year	Number	System.Media.Year
publisher	String	System.Media.Publish
artist	String	System.Music.Artist, System.Music.Display
albumArtist	String	System.Music.Display System.Music.AlbumA
genre	String vector	System.Music.Genre
composers	String vector	System.Music.Compo
conductors	String vector	System.Music.Conduc
rating	Число (1-99 с 0 означающим "нет оценки")	System.Rating
duration	Число (в 100-нс единицах, т.е. 1/10th миллисекунды	System.Media.Duratio
bitrate	Число (в битах в секунду)	System.Video.TotalBit System.Video.Encodin
producers	String vector	System.Media.Produc
writers	String vector	System.Media.Writer

Таблица 6.4.

DocumentProperties	from	
Дополнительные свойства	**System**	
Свойство	Тип данных	Соответствующее свойство Windows
title	String	System.Title
Author	String vector	System.Author
keywords	String vector	System.Keywords
Comments	String	System.Comment

Пара замечаний обо всем этом. Во-первых, переменные String vector (строковые векторы) – это, как мы видели ранее, экземпляры IVector (ссылка: http://msdn.microsoft.com/library/windows/apps/br206631.aspx),

который предоставляет методы для манипуляции, наподобие `append`, `insertAt`, `removeA`, и так далее. в JavaScript вы можете получить доступ к членам вектора как к массиву, через квадратные скобки [ ]; следует лишь помнить, что доступные методы более специфичны.

Во-вторых, свойства `latitude` и `longitude` для изображений имеют тип данных `double`, но содержат градусы, минуты, секунды и указание на направление. Пример "Простая работа с изображениями" (ссылка: http://code.msdn.microsoft.com/windowsapps/Simple-Imaging-Sample-a2dec2b0) (в js/default.js) содержит вспомогательную функцию для извлечения компонентов этих значений и конвертирования их в строку:

```
"convertLatLongToString": function (latLong, isLatitude) {
var reference;

if (isLatitude) {
reference = (latLong >= 0) ? "N" : "S";
} else {
reference = (latLong >= 0) ? "E" : "W";
}

latLong = Math.abs(latLong);
var degrees = Math.floor(latLong);
var minutes = Math.floor((latLong - degrees) * 60);
var seconds = ((latLong - degrees - minutes / 60) * 3600).toFixed(2);
return degrees + "°" + minutes + "\'" + seconds + "\'" + reference;
}
```

Подводя итоги, знак величины указывает на направление. Положительное значение для широты (latitude) указывает на Север, отрицательное означает Юг, для долготы (longitude) положительное значение означает Восток, отрицательное – Запад. Целая часть числа представляет собой градусы, дробная содержит значение минут, за базу взято 60. Умножение этого значения на 60 дает полное количество минут, то, что остается, содержит секунды. Это может показаться странным, но подобные исходные данные получают от GPS-приемника и API определения местоположения обычно конвертирует их.

Мультимедийные свойства в примерах

Несколько примеров из Windows SDK показывают, как работать с некоторыми из вышеописанных свойств и как работать со свойствами в общем. Пример "Простая работа с изображениями" (ссылка: http://code.msdn.microsoft.com/windowsapps/Simple-Imaging-Sample-a2dec2b0), в Сценарии 1, предоставляет наиболее полную демонстрацию, так как вы можете выбрать файл изображения и он загрузит и отобразит различные свойства, как показано на рис. 6.2 (я прокрутил список вниз, чтобы показать полный набор свойств). Я могу подтвердить, что дата, производитель и модель камеры, сведения об экспозиции, верны.

Рис. 6.2. Свойства файла изображения в примере "Простая работа с изображениями"

Метод примера `openHandler` получает эти свойства из файла, в частности, выполняя вызов `StorageFile.properties.getImagePropertiesAsync` и используя метод `ImageProperties.retrievePropertiesAsync` для пары дополнительных свойств уже не в ImageProperties. Затем `getImagePropertiesForDisplay` объединяет их в единый

объект, используемый для вывода данных в пользовательский интерфейс примера. Некоторые строки кода, приведенного ниже, опущены.

```
var ImageProperties = {};

function openHandler() {
// Хранит данные между асинхронными вызовами.
var file = {};

Helpers.getFileFromOpenPickerAsync().then(function (_file) {
file = _file;
return file.properties.getImagePropertiesAsync();
}).then(function (imageProps) {
ImageProperties = imageProps;

var requests = [
"System.Photo.ExposureTime",    // В секундах
"System.Photo.FNumber"    // Значение диафрагмы из данных EXIF
];

return ImageProperties.retrievePropertiesAsync(requests);
}).done(function (retrievedProps) {
// Форматирование свойств в виде текста для отображения в интерфейс
displayImageUI(file, getImagePropertiesForDisplay(retrievedProps));
});
 }

function getImagePropertiesForDisplay(retrievedProps) {
// Если заданное свойство не существует, его значением будет null.
var orientationText = Helpers.getOrientationString(ImageProperties.orientation);

var exposureText = retrievedProps.lookup("System.Photo.ExposureTime") ?
retrievedProps.lookup("System.Photo.ExposureTime") * 1000 + " ms" : "";

var fNumberText = retrievedProps.lookup("System.Photo.FNumber") ?
retrievedProps.lookup("System.Photo.FNumber").toFixed(1) : "";

// Опущено: код для конвертации ImageProperties.latitude и ImagePropertie
```

```
// градусы, минуты, секунды, и направление

return {
"title": ImageProperties.title,
"keywords": ImageProperties.keywords, // строковой массив
"rating": ImageProperties.rating, // число
"dateTaken": ImageProperties.dateTaken,
"make": ImageProperties.cameraManufacturer,
"model": ImageProperties.cameraModel,
"orientation": orientationText,
// Опущено: свойства широты/долготы
"exposure": exposureText,
"fNumber": fNumberText
};
}
```

Функция `displayImageUI`, которой передаются эти свойства, в основном просто копирует данные в различные элементы управления. Полезно будет снова отметить, однако, что отображение фотографии реализовано с помощью нашего хорошего друга `URL.createObjectURL`:

```
function displayImageUI(file, propertyText) {
id("outputImage").src = window.URL.createObjectURL(file, { oneTimeOnly: tru
```

Небольшой пример работы с `MusicProperties` можно найти в примере "Список воспроизведения" (ссылка: http://code.msdn.microsoft.com/windowsapps/Playlist-sample-3d80daee), как мы уже видели в разделе "Списки воспроизведения". Вы можете сейчас вернуться назад и взглянуть на код, приведенный в этом разделе, и вы вполне сможете понять, что происходит. В SDK нет примеров использования `VideoProperties` и `DocumentProperties`, работа с ними выполняется по тем же шаблонам, как показано выше для ImageProperties, поэтому написание необходимого кода не вызовет затруднений.

Кроме того, взглянем еще раз на пример "Конфигурация кнопок для

мультимедиа" (ссылка: http://code.msdn.microsoft.com/windowsapps/Media-Buttons-ea57d8e2), который мы видели ранее, в разделе "Пользовательский интерфейс управления воспроизведением". Он показывает, как использовать музыкальные свойства для того, чтобы получить изображение альбома.

Что касается сохранения свойств, пример "Простая работа с изображениями" демонстрирует это в Сценарии 1. Так как поля, показаные ранее, на рис. 6.2, можно редактировать, в примере есть кнопка Apply (Применить), которая вызывает функцию applyHandler, показанную ниже, для записи измененных свойств в файл:

```
function applyHandler() {
ImageProperties.title = id("propertiesTitle").value;

// Ключевые слова хранятся в строковом массиве.
 Разделение строк знаками перевода строки.
ImageProperties.keywords.clear();
if (id("propertiesKeywords").value !== "") {
var keywordsArray = id("propertiesKeywords").value.split("\n");

keywordsArray.forEach(function (keyword) {
ImageProperties.keywords.append(keyword);
});
}

var properties = new Windows.Foundation.Collections.PropertySet();

// При записи оценок, используйте ключ свойства "System.Rating".
// ImageProperties.rating не обрабатывает значение оценки 0 (нет звезд/оц
properties.insert("System.Rating", Helpers.convertStarsToSystemRating(
id("propertiesRatingControl").winControl.userRating
));

// Код опущен: конвертирование дискретных значений широты и
 долготы из пользовательского интерфейса в
// форму, подходящую для свойства, и проверка данных;
конечный результат этих манипуляций
// сохраняется в списке свойств
```

```
properties.insert("System.GPS.LatitudeRef", latitudeRef); properties.insert("Syste
properties.insert("System.GPS.LatitudeNumerator", latNum); properties.insert("S
properties.insert("System.GPS.LatitudeDenominator", latDen);
properties.insert("System.GPS.LongitudeDenominator", longDen);

// Запись массива свойств в файл

ImageProperties.savePropertiesAsync(properties).done(function () {
// ...
}, function (error) {
// Обработка некоторых ошибок и некоторые свойства могут
 не поддерживаться всеми форматами изображений.
});
}
```

Несколько примечательных особенностей этого кода перечислены ниже:

- Он разделяет ключевые слова в элементе управления пользовательского интерфейса и отдельно добавляет каждое к вектору в свойстве `keywords`.
- Он создает новую коллекцию свойств типа `Windows.Foundation.Collections.PropertySet` и использует ее метод `insert` для добавления свойств в список. Этот набор свойств – именно то, чего ожидает метод `savePropertiesAsync`.
- Метод `Helpers.convertStartsToSystemRating` (смотрите js/default.js) производит конверсию оценки, выраженной в 1 – 5 звездах, как используется в элементе управления `WinJS.UI.Rating`, в значение System.Rating, которое использует диапазон значений 1 – 99. Документация по `System.Rating` (ссылка: http://msdn.microsoft.com/library/bb787554.aspx) особо описывает подобное преобразование.

В итоге, вся детализированная информация, которая нужна вам для любого конкретного свойства Windows, может быть найдена на справочной странице этого свойства. Начать углубленное изучение

свойств можно со страницы Свойства Windows (ссылка: http://msdn.microsoft.com/library/windows/desktop/dd561977.aspx).

Манипуляция изображениями и их кодирование

Для того, чтобы сделать с изображениями нечто большее, чем загрузка и отображение (где вы можете применить различные CSS-трансформации для реализации эффектов), вам нужно получить доступ к пикселям с помощью декодера (decoder). Это уже происходит внутри механизмов вывода изображения, когда вы присваиваете URI img.src, но для того, чтобы получить непосредственный доступ к пикселям, нужно выполнить ручное декодирование изображения. С другой стороны, сохранение пиксельной информации в виде файла требует использования кодера.

WinRT предоставляет API и для того, и для другого в пространстве имен `Windows.Graphics.Imaging` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.graphics.imaging.aspx), а именно, в классах `BitmapDecoder`, `BitmapTransform`, и `BitmapEncoder`. Загрузка, выполнение манипуляций с изображением и сохранение его в файл часто включает в себя использование всех трех классов, хотя объект `BitmapTransform` предназначен для поворота и масштабирования, таким образом, вы не будете пользоваться им, если выполняете другие манипуляции.

Демонстрацию этого API можно найти в Сценарии 2 примера "Простая работа с изображениями" (ссылка: http://code.msdn.microsoft.com/windowsapps/Simple-Imaging-Sample-a2dec2b0). Я оставляю вам этот код для того, чтобы вы просмотрели его в редакторе, так как он довольно сложен – до 11 promise-вызовов, объединенных в цепочку для сохранения файла! Так же он выполняет декодирование, манипуляцию и кодирование внутри одной функции, такой, как `saveHandler` (js/scenario2.js). Вот какую последовательность действий реализует этот код:

- Открыть файл с помощью `StorageFile.openAsync`, получив поток.

- Передать поток статическому методу `BitmapDecoder.createAsync`, что позволит получить конкретный экземпляр `BitmapDecoder` для потока.
- Переать декодер статическому методу `BitmapEncoder.createForTranscodingAsync`, который предоставляет экземпляр `BitmapEncoder`. Этот кодировщик создается с помощью `InMemoryRandomAccessStream`.
- Установить свойства в свойстве кодировщика bitmapTransform property (объект BitmapTransform) для настройки масштабирования и поворота. Это позволит создать трансформированное изображение в потоке в памяти.
- Создать набор свойств (`Windows.Graphics.Imaging.BitmapPropertySet`), который включает в себя System.Photo.Orientation и использует команду кодера `bitmapProperties.setPropertiesAsync` для того, чтобы это сохранить.
- Копирование потока из памяти в выходной файловых поток с использованием `Windows.Storage.Stream.RandomAccessStream.copy`
- Закрытие обоих потоков с помощью соответствующих методов (тем самым закрывается файл).

В сложных сценариях, подобных этому, полезно рассмотреть отдельные стадии этого процесса, для этой цели в материалах к этой лекции есть упражнение ImageManipulation. Он позволит вам выбрать и загрузить изоражение, конвертировать его в оттенки серого и сохранить конвертированное изображение в новый файл. Результат работы показан на рис. 6.3. Кроме того, это дает вам возможность, как мы можем отправлять декодированное изображение в элемент HTML `canvas` и сохранять содержимое элемента в виде файла.

Рис. 6.3. Результаты работы упражнения ImageManipulation из дополнительных материалов к лекции

Обработчик кнопки Load Image (Загрузить изображение) (loadImage в js/default.js) предоставляет первоначальное отображение. Он позволяет вам выбирать изображение с помощью средства вабра файлов, отображает полноразмерное изображение в элементе img с помощью `URL.createObjectURL`, вызывает `StorageFile.properties.getImagePropertiesAsync` для получения свойств `title` и `dateTaken`, и применяет `StorageFile.getThumbnailAsync` для получения эскиза, который отображается в верхней части. Мы уже видели все эти API в действии.

При щелчке на кнопке Grayscale (Оттенки серого), мы входим в обработчик `setGrayscale`, где происходят интересные действия. Мы вызываем `StorageFile.openReadAsync` для того, чтобы получить поток, вызываем `BitmapDecoder.createAsync` с этим потоком для того, чтобы получить декодер, кэшируем некоторые подробности из декодера в локальном объекте (`encoding`), вызываем `BitmapDecoder.getPixelDataAsync` и копируем эти пиксели в элемент `canvas` (и здесь лишь три асинхронных операции, объединенных в цепочку!):

```
var Imaging = Windows.Graphics.Imaging; //Короткое имя
var imageFile;          //Полученный из средства выбора файла
var decoder;          //Полученный из BitmapDecoder.createAsync
var encoding = {};       //Для кэширования некоторых деталей из декодера

function setGrayscale() {
//Декодируем файл изображения в пиксельные данные для canvas

//Получаем входной поток длф файла (объект StorageFile полученный по
imageFile.openReadAsync().then(function (stream) {
//Создаем декодер, используя статический метод createAsync и поток фаі
return Imaging.BitmapDecoder.createAsync(stream);
}).then(function (decoderArg) {
decoder = decoderArg;

//Настраиваем декодер, если хотим.
Параметры по умолчанию, это BitmapPixelFormat.rgba8 и
//BitmapAlphaMode.ignore.
Параметризованная версия getPixelDataAsync может так же
//управлять трансформацией, а так же ExifOrientationMode,
и ColorManagementMode если нужно.

//Кэшируем эти параметры для кодирования
encoding.dpiX = decoder.dpiX;
encoding.dpiY = decoder.dpiY;
encoding.pixelFormat = decoder.bitmapPixelFormat;
encoding.alphaMode = decoder.bitmapAlphaMode;
encoding.width = decoder.pixelWidth;
encoding.height = decoder.pixelHeight;

return decoder.getPixelDataAsync();
}).done(function (pixelProvider) {
//detachPixelData получает реальные пиксели (массив нельзя вернуть из
//асинхронной операции)
copyGrayscaleToCanvas(pixelProvider.detachPixelData(), decoder.pixelWidth, d
});
}
```

Метод декодера `getPixelDataAsync` (ссылка:

http://msdn.microsoft.com/library/windows/apps/windows.graphics.imaging.bitmap существует в двух формах. Более простая форма, показанная здесь, выполняет декодирование с использованием настроек по умолчанию. Версия, которую можно полностью контролировать, позволяет вам задать другие параметры, как пояснено в комментариях к вышеприведенному коду. Обычное использование этого метода ведет к выполнению трансформации с использованием объекта `Windows.Graphics.Imaging.BitmapTransform` (как упомянуто выше), что включает в себя масштабирование (с различными моделями интерполяции), поворот (с приращением в 90 градусов), обрезку и отражение.

В любом случае, то, что вы получаете из `getPixelDataAsync`, это не массив пикселей из-за ограничений механизма языковой проекции WinRT, где асинхронные операции не могут возвращать массивы. Вместо этого операция возвращает объект `PixelDataProvider` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.graphics.imaging.pixelda единственный восхитительный синхронный метод которого `detachPixelData` дает вам необходимый массив. (И этот метод можно вызвать лишь один раз, он не будет работать при последующих вызовах, отсюда и имя "detach" (отсоединить)). В итоге мы получаем данные, которые нам нужны для манипуляции пикселями и отображения результатов в элементе canvas, что и демонстрирует функция `copyGrayscaleToCanvas`. Вы можете, конечно, заменить эту функцию любой подпрограммой для манипуляции пикселями:

```
function copyGrayscaleToCanvas(pixels, width, height) {
//Настраиваем контекст canvas и получаем его пиксельный массив
var canvas = document.getElementById("canvas1");
canvas.width = width;
canvas.height = height;
var ctx = canvas.getContext("2d");

//Обходим данные и копируем значения пикселей
в canvas после конвертации в оттенки серого
var imgData = ctx.createImageData(canvas.width, canvas.height);
var colorOffset = { red: 0, green: 1, blue: 2, alpha: 3 };
var r, g, b, gray;
```

```
var data = imgData.data; //Очень влияет на производительность!

for (var i = 0; i < pixels.length; i += 4) {
r = pixels[i + colorOffset.red];
g = pixels[i + colorOffset.green];
b = pixels[i + colorOffset.blue];

//Присваиваем каждое rgb-значение яркости
gray = Math.floor(.3 * r + .55 * g + .11 * b);

data[i + colorOffset.red] = gray;
data[i + colorOffset.green] = gray;
data[i + colorOffset.blue] = gray;
data[i + colorOffset.alpha] = pixels[i + colorOffset.alpha];
}

//Показываем результат в элементе canvas ctx.putImageData(imgData, 0, 0);

//Активируем кнопку сохранения
document.getElementById("btnSave").disabled = false;
}
```

Это – хороший случай убедиться в том, что JavaScript – это не лучшия язык для работы с большими группами пикселей вроде этой, хотя в данном случае производительность Release build (построения Выпуска), запущенного вне отладчика, весьма неплоха. Подобные подпрограммы лучше реализовывать в виде WinRT-компонентов на языках вроде C# или C++ и делать вызываемыми из JavaScript. У нас будет возможность сделать это в лекции 5 курса "Программная логика приложений для Windows 8, созданных с использованием HTML, CSS и JavaScript и их взаимодействие с системой", где мы так же увидим ограничения элемента canvas, которые требуют несколько иного подхода.

Сохранение данных из canvas в файл происходит в функции `saveGrayscale`, где мы используем средство выбора файла для получения `StorageFile`, открываем поток, берем пиксельные данные из элемента `canvas` и передаем их `BitmapEncoder`:

```
function saveGrayscale() {
var picker = new Windows.Storage.Pickers.FileSavePicker();
picker.suggestedStartLocation =
Windows.Storage.Pickers.PickerLocationId.picturesLibrary;
picker.suggestedFileName = imageFile.name + " - grayscale";
picker.fileTypeChoices.insert("PNG file", [".png"]);

var imgData, fileStream = null;

picker.pickSaveFileAsync().then(function (file) {
if (file) {
return file.openAsync(Windows.Storage.FileAccessMode.readWrite);
} else {
return WinJS.Promise.wrapError("No file selected");
}
}).then(function (stream) {
fileStream = stream;
var canvas = document.getElementById("canvas1");
var ctx = canvas.getContext("2d");
imgData = ctx.getImageData(0, 0, canvas.width, canvas.height);

return Imaging.BitmapEncoder.createAsync( Imaging.BitmapEncoder.pngEncode
}).then(function (encoder) {
//Задаем пиксельные данные, подразумевая, что объект "encoding"
имеет необходимые параметры.
//Конверсия из данных элемента canvas в Uint8Array необходима,
так как тип массива из элемента canvas
//не соответствует тому, что нужно здесь WinRT.
encoder.setPixelData(encoding.pixelFormat, encoding.alphaMode,
encoding.width, encoding.height, encoding.dpiX, encoding.dpiY,
new Uint8Array(imgData.data));

//Начинаем кодирование
return encoder.flushAsync();
}).done(function () {
fileStream.close();
}, function () {
//Пустой обработчик ошибки (если пользователь закрыл средство выбо
ничего не происходит)
```

```
});
}
```

Обратите внимание на то, как `BitmapEncoder` принимает идентификатор кодека в первом параметре. Мы используем `pngEncoderId`, который, как вы можете видеть, определен как статическое свойство класса `Windows.Graphics.Imaging.BitmapEncoder`; другие возможные значения – это `bmpEncoderId`, `gifEncoderId`, `jpegEncoderId`, `jpegXREncoderId`, и `tiffEncoderId`. Все это – форматы, поддерживаемые API. Вы можете установить дополнительные свойства `BitmapEncoder` прежде чем зададите пиксельные данные, такие, как его `BitmapTransform`, эти установки будут применены при кодировании.

Одна особенность, о которой нужно знать, заключается в том, что пиксельный массив, полученный из элемента canvas (тип DOM `CanvasPixelArray` (ссылка: http://msdn.microsoft.com/library/windows/apps/hh465731.aspx) не совместима напрямую с байтовым массивом WinRT, который нужен кодировщику. По этой причине мы используем вызов new `Uint8Array` в последнем параметре.

Перекодировка и пользовательские форматы изображений

В предыдущем разделе мы, в основном, видели использование `BitmapEncoder`, созданного с помощью статического метода этого класса `createAsync` для записи нового файла. Все это хорошо, но вам может захотеться узнать о некоторых других возможностях кодировщиков.

Во-первых, это метод `BitmapEncoder.createForTranscodingAsync` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.graphics.imaging.bitmap который был кратко упомянут в контексте примера 'Простая обработка

изображений". С его помощью создается новый кодировщик, который инициализируется на основе существующего объекта `BitmapDecoder`. Это, в основном, используется для манипуляции некоторыми аспектами исходного файла изображения, оставляя оставшиеся данные нетронутыми. Говоря точнее, вы можете сначала изменить эти аспекты, посредством метода кодировщика `setPixelData`: формат пикселей (rgba8, rgba16, и bgra8, смотрите `BitmapPixelFormat` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.graphics.imaging.bitmap параметры прозрачности (premultiplied, straight, или ignore, смотрите BitmapAlphaMode (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.graphics.imaging.bitmap размер изображения, DPI изображения, и, конечно, сами пиксельные данные. Помимо этого, вы можете изменить другие свойства посредством метода кодировщика `bitmapProperties.setPropertiesAsync`. На самом деле, если все, что вам нужно – это изменить несколько свойств, и вы не собираетесь работать с данными пикселей, вы можете использовать вместо этого `BitmapEncoder.createForInPlacePropertyEncodingAsyr` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.graphics.imaging.bitmap (ничего себе имя метода!). Кодировщик позволяет вызывать лишь `bitmapProperties.setPropertiesAsync`, `bitmapProperties. getPropertiesAsync`, и `flushAsync`, и так как это подразумевает неизменность исходных данных, это выполняется быстрее, чем при использовании его более гибкого эквивалента и требует меньше памяти.

Кодировщик из `createForTranscodingAsync` не приспособлен для изменения формата файла изображения (например, JPEG на PNG); для этого вам нужно использовать `createAsync`, где вы можете задать конкретный тип кодирования. Как мы уже видели, первый аргумент createAsync это идентификатор кодека, куда обычно передают одно из статических свойств из `Windows.Graphics.Imaging.BitmapEncoder`. Я еще не упоминал о том, что вы можете так же задавать пользовательские кодеки в этом первом параметре, и что `createAsync` может так же

поддерживать необязательный третий аргумент, в котором можно передать параметры для конкретного кодека. Однако здесь имеются некоторые сложности и ограничения.

Позвольте мне сначала остановиться на параметрах. Существующая документация по значениям кодеков BitmapEncoder (наподобие pngEncoderId) не содержит многих подробностей о доступных параметрах. Для того, чтобы их узнать, вам нужно обратиться к документации по Windows Imaging Component (WIC), в частности, к материалу Native WIC Codecs (ссылка: http://msdn.microsoft.com/library/windows/desktop/gg430027.aspx), который посвящен тому, что WinRT предоставляет приложениям для Магазина Windows. Если вы пройдете на страницу конкретного кодека, вы увидите раздел "Encoder Options" (Свойства кодера), который содержит описания того, что можно использовать. Например, кодек JPEG (ссылка: http://msdn.microsoft.com/library/windows/desktop/gg430026.aspx) поддерживает свойства наподобие `ImageQuality` (качество изображения, значение в диапазоне от 0.0 до 1.0), а так же встроенную функцию поворота. Кодек PNG (ссылка: http://msdn.microsoft.com/library/windows/desktop/gg430028.aspx) поддерживает свойства наподобие FilterOption для различных оптимизаций сжатия файлов.

Для того, чтобы задать эти свойства, вам нужно создать новый `BitmapPropertySet` и вставить в этот набор запись для каждого необходимого параметра. Если, например, у вас есть переменная, содержащая данные о качестве изображения, которое вы хотите использовать при кодировании JPEG, вы можете создать кодер, таким образом:

```
var options = new Windows.Graphics.Imaging.BitmapPropertySet();
options.insert("ImageQuality", quality);
var encoderPromise = Imaging.BitmapEncoder.createAsync(Imaging.BitmapEnc
stream, options);
```

Вы используете тот же самый `BitmapPropertySet` для любых свойств, которые вы можете передать при вызове метода кодера `bitmapProperties.setPropertiesAsync`. Здесь мы

используем тот же механизм для свойств кодировщика.

Что касается пользовательских кодеков, это просто означает, что первый аргумент `BitmapEncoder.createAsync` (как и `BitmapDecoder.createAsync`) это GUID (идентификатор класса или CLSID) для этого кодека, реализация которого должна быть выполнена в виде DLL. Подробности о том, как писать подобные кодеки, можно найти в материале "Как написать WIC-кодек" (ссылка: http://msdn.microsoft.com/library/windows/desktop/ee719883.aspx). Загвоздка здесь в том, что включение пользовательских кодеков для изображений в пакет приложения не поддерживается. Если кодек уже присутствует в системе (то есть, установлен традиционным образом), это будет работать. Однако, политика Магазина Windows не позволяет одним приложениям зависеть от других, так что маловероятно, что вы сможете хотя бы отправить подобное приложение, если только оно не предустановлено на каком-то конкретном OEM-устройстве и нужная DLL не является частью образа системы (Приложения, написанные на C++ могут большее в да нной области, но это выходит за границы данного курса).Короче говоря, в случае с приложениями, написанными на JavaScript и HTML, вы, в работе с различными форматами изображений, по-настоящему ограничены применением кодеков, которые изначально поддерживаются системой.

Заметьте, что эти ограничения не применяются для пользовательских аудио- и видеокодеков. Пример "Расширения мультимедиа" (ссылка: http://code.msdn.microsoft.com/windowsapps/Media-extensions-sample-7b466096) показывает, как работать с пользовательским видеокодеком, что мы увидим в следующем разделе.

Манипуляция аудио и видео

Как и при работе с изображения, если все, что нам нужно – это загрузить содержимое `StorageFile` в аудио- или видеоэлемент, мы можем просто передать этот `StorageFile` в `URL.createObjectUrl` и пприсвоить результат атрибуту src. Похожим образом, если нам нужны исходные данные, мы можем просто использовать методы `StorageFile.openAsync` или `openReadAsync` для получения файлового потока.

Честно говоря, открытие файла, это, вероятнее всего, самое сложное, что вам придется делать в JavaScript с исходными аудио- или видеоданными, если вообще придется этим пользоваться. В то время, как процесс обработка изображений находится где-то на грани приемлемости в окружении JavaScript, обрабатывать аудио, и особенно видео, лучше всего в высокопроизводительных DLL C++. На самом деле, легко доступны множесто платформенно нейтральных библиотек C++ сторонних разработчиков для подобных целей, которые вы сможете легко включить в состав подобных DLL. В подобном случае вы можете просто позволить DLL открыть нужный файл.

Тем не менее, WinRT предоставляет средства для перекодирования (конвертации) различных мультимедиа-форматов и предоставляет модель расширения для пользовательских кодеков, эффектов и обработчиков схем. На самом деле, мы уже видели, как применять пользовательские видео эффекты в примере "Расширения мультимедиа" (ссылка: http://code.msdn.microsoft.com/windowsapps/Media-extensions-sample-7b466096), и те же DLL можно использовать при кодировании, где все, что выполняет код JavaScript – это связывание вместе нужных компонентов (что он делает очень хорошо). Посмотрим как это делается, сначала на примере перекодирования видео, а затем – на примере пользовательского кодека.

Перекодирование

Перекодирование аудио и видео реализуется посредством класса `Windows.Media.Transcoding.MediaTranscoder` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.media.transcoding.medi который поддерживает выходные форматы mp3 и wma для аудио, и mp4, wmv и m4a для видео. Процесс перекодирования позволяет вам применять эффекты, осуществлять обрезку композиций.

Перекодирование происходит либо из одного объекта `StorageFile` в другой, или из одного `RandomAccessStream` в другой, в каждом из случае это производится в соответствии с `MediaEncodingProfile` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.media.mediaproperties.r Для настройки операции перекодирования, вы вызываете методы

`prepareFileTranscodeAsync` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.media.transcoding.medi или `prepareStreamTranscodeAsync` объекта MediaTranscoder, которые возвращают объект `PrepareTranscodeResult` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.media.transcoding.prep Он представляет собой операцию, которая готова к работе, но обработка не начинается, пока вы не вызовете его метод `transcodeAsync`. В JavaScript, подобный результат является promise-объектом, который позволяет вам предоставлять обработчики завершения и выполнения для отдельной операции, а так же позволяет комбинировать операции с помощью `WinJS.Promise.join`. Это позволяет сначала настроить операции, а потом запустить их выполнение, что полезно для пакетной обработки и выполнения автоматической отправки на сервисы наподобие YouTube, пока вы спите! (В подобное время я вытаскиваю упаковку льда из морозилки и кладу ее под мой ноутбук из-за его плохой системы охлаждения)

Пример "Перекодирование мультимедиа" (ссылка: http://code.msdn.microsoft.com/windowsapps/Media-Transcode-Sample-f7ba5269) показывает нам пару сценариев перекодирования. В Сценарии 1 (js/presets.js) мы можем выбрать видеофайл, задать целевой формат, выбрать профиль перекодирования и предоставить компьютеру возможность сделать свою работу (информация о продвижении операции предоставляется), как показано на рис. 6.4.

Рис. 6.4. Пример перекодирования мультимедиа обрабатывает клип с моим двухлетним сыном, который исследует измерительную рулетку

Код, который выполняется, когда вы нажимаете на кнопку Transcode (Перекодировать) приведен ниже (кое-что опущено; в примере использованы вложенные promise-вызовы, что не рекомендуется, если вам нужна обработка ошибок, этот код поглощает любые исключения, которые происходят до вызова `transcodeAsync`):

```
function onTranscode() {
// Создадим объект для перекодирования.
var transcoder = null;
transcoder = new Windows.Media.Transcoding.MediaTranscoder();

// Получим профиль перекодирования.
getPresetProfile(id("profileSelect"));

// Создадим выходной файл и начнем перекодировку.
var videoLib = Windows.Storage.KnownFolders.videosLibrary;
var createFileOp = videoLib.createFileAsync(g_outputFileName,
Windows.Storage.CreationCollisionOption.generateUniqueName);

createFileOp.done(function (ofile) {
g_outputFile = ofile;
g_transcodeOp = null;
var prepareOp = transcoder.prepareFileTranscodeAsync(g_inputFile, g_outputF
g_profile);

prepareOp.done(function (result) {
if (result.canTranscode) {
g_transcodeOp = result.transcodeAsync();
g_transcodeOp.done(transcodeComplete, transcoderErrorHandler, transcodePr
} else {
transcodeFailure(result.failureReason);
}
}); // prepareOp.done
id("cancel").disabled = false;
}); // createFileOp.done
}
```

Метод `getPresentProfile` получает подходящий объект профиля, соответствующий параметрам, выбранным в приложении. Для параметров, выбранных на рис. 6.4 (WMV и WVGA) мы используем следующие части этой функции:

```
function getPresetProfile(profileSelect) {
g_profile = null;
var mediaProperties = Windows.Media.MediaProperties;
var videoEncodingProfile;

switch (profileSelect.selectedIndex) {
// Остальные варианты опущены
case 2:
videoEncodingProfile = mediaProperties.VideoEncodingQuality.wvga;
break;
}
if (g_useMp4) {
g_profile = mediaProperties.MediaEncodingProfile.createMp4(videoEncodingPr
} else {
g_profile = mediaProperties.MediaEncodingProfile.createWmv(videoEncodingPr
}
}
```

В Сценарии 2 всегда используется WVGA-кодирование, но здесь вы можете установить необходимые значения для размерностей видео, частоты кадров, битрейта аудио и видео, настроить аудиоканалы и аудиосэмплинг. Данные параметры используются в `getCustomProfile` (js/custom.js), просто конфигурируя параметры профиля после его создания:

```
function getCustomProfile() {
if (g_useMp4) {
g_profile = Windows.Media.MediaProperties.MediaEncodingProfile.createMp4(
Windows.Media.MediaProperties.VideoEncodingQuality.wvga);
} else {
g_profile = Windows.Media.MediaProperties.MediaEncodingProfile.createWmv
```

```
Windows.Media.MediaProperties.VideoEncodingQuality.wvga);
}

// Берем значения настроек из элементов пользовательского интерфейса
g_profile.audio.bitsPerSample = id("AudioBPS").value;
g_profile.audio.channelCount = id("AudioCC").value;
g_profile.audio.bitrate = id("AudioBR").value;
g_profile.audio.sampleRate = id("AudioSR").value;
g_profile.video.width = id("VideoW").value;
g_profile.video.height = id("VideoH").value;
g_profile.video.bitrate = id("VideoBR").value;
g_profile.video.frameRate.numerator = id("VideoFR").value;
g_profile.video.frameRate.denominator = 1;
}
```

И, в завершении, Сценарий 3 похож на Сценарий 1, но позволяет вам задать время начала и конца записи для ее обрезки, которые затем сохраняются в свойствах перекодировщика `trimStartTime` и `trimStopTime` (js/trim.js):

```
transcoder = new Windows.Media.Transcoding.MediaTranscoder();
transcoder.trimStartTime = g_start;
transcoder.trimStopTime = g_stop;
```

Хотя это и не показано в примере, вы можете применять эффекты при операции перекодирования, используя методы перекодировника `addAudioEffect` и `addVideoEffect`.

Пользовательские кодеры/декодеры и обработчики схем

Очевидно то, что в мире гораздо больше аудио- и видеоформатов, чем поддерживается в исходной поставке Windows, поэтому WinRT предоставляет механизм расширения, позволяющий работать с пользовательскими объектами битовых потоков, источниками мультимедийных данных, пользовательскими кодеками и эффектами.

Важно снова отметить, что все эти расширения доступны только самому приложению и не доступны другим приложениям. Более того, Windows всегда предпочитает встроенные компоненты пользовательским, а это означает, что вам не следует впустуют тратить время на разработку нового декодера mp3, так как он никогда не будет использован.

Как было сказано ранее, о пользовательских форматах изображений, эти вопросы ведут в весьма обширную область вокруг Windows Media Foundation (WMF) SDK (ссылка: http://msdn.microsoft.com/library/ms694197.aspx). И WinRT лишь предоставляет оболочку для этого, поэтому изучение WMF существенно в этих вопросах, и это занятие не для слабонервных!

Аудио- и видеорасширения объявляются в манифесте приложения, вам понадобится непосредственно править XML. Как можно увидеть в примере "Расширения мультимедиа" (ссылка: http://code.msdn.microsoft.com/windowsapps/Media-extensions-sample-7b466096), для каждой DLL в решении объявление выглядит так:

```
 <Extension Category="windows.activatableClass.inProcessServer">
<InProcessServer>
<Path>MPEG1Decoder.dll</Path>
<ActivatableClass ActivatableClassId="MPEG1Decoder.MPEG1Decoder"
ThreadingModel="both" />
</InProcessServer>
</Extension>
```

`ActivatableClassId` это то, как идентифицируется расширение при вызовах WinRT APIs, что ясно отражается в манифесте для каждой DLL, которая должна быть загружена.

В зависимости от использования расширения, вам может понадобиться зарегистрировать его в WinRT посредством методов `Windows.Media.MediaExtensionManager: registerAudio[Decoder | Encoder]`, `registerByteStreamHandler` (потоки мультимедиа), `registerSchemeHandler` (источники мультимедиа/файлы-контейнеры), и `registerVideo[Decoder | Encoder]` . В сценарии 1 примера о расширениях мультимедиа (js/LocalDecoder.js), мы

можем видеть, как задать пользовательский декодер для воспроизведения видео.

```
 воспроизведения видео.
var page = WinJS.UI.Pages.define("/html/LocalDecoder.html", {
extensions: null,
MFVideoFormat_MPG1: { value: "{3147504d-0000-0010-8000-00aa00389b7
NULL_GUID: { value: "{00000000-0000-0000-0000-000000000000}" },

ready: function (element, options) {
if (!this.extensions) {
// Добавьте здесь любой код инициализации
this.extensions = new Windows.Media.MediaExtensionManager();
// Зарегистрируем пользовательский ByteStreamHandler и пользовательск
this.extensions.registerByteStreamHandler("MPEG1Source.MPEG1ByteStreamH
".mpg", null);
this.extensions.registerVideoDecoder("MPEG1Decoder.MPEG1Decoder", this.M
}

// ...
```

Здесь CLSID MPEG1Source.MPEG1ByteStreamHandler реализован в одной DLL (смотрите проект C++ MPEG1Source в решении), а CLSID MPEG1Decoder.MPEG1.Decoder реализован в другой (C++ проект MPEG1Decoder).

Сценарий 2, в свою очередь, показывает использование пользовательского обработчика схемы, где обработчик (в проекте C++ GeometricSource) генерирует видеокадры в процессе выполнения программы. Притягательная возможность, но находящаяся за пределами тем данного курса.

Эффекты, как мы можем видеть, довольно просто использовать, если у вас есть реализация эффекта, просто передайте ее CLSID методу наподобие `msInsertVideoEffect` и `msInsertAudioEffect` видео или аудиоэлемента. Так же вы можете применить эффект в процессе перекодирования, в методах `addAudioEffect` и `addVideoEffect` класса `MediaTranscoder`. То же самое справедливо и для процесса захвата мультимедийных данных, который

мы сейчас рассмотрим.

Захват мультимедийных данных

Есть моменты, когда мы по-настоящему ценим работу, которая выполнена для защиты неприкосновенности частной жизни. Например, когда я знаю, используется ли камера моего компьютера, так как я часто использую его поздно вечером, сидя на кровати, или утром, до похода в душ, когда, по словам моего тестя, моя голова похожа на ананас.

И бывают ситуации, когда мы хотим включить камеру или микрофон и что-то записать: сделать фотографию, аудиозапись или видеоклип. Конечно, приложения не могут знать заранее, какие именно камеры и микрофоны могут быть установлены в системе. Ключевой шаг в захвате мультимедийных данных заключается в определении устройства для использования – а это то, что красиво предоставляет нам API Windows.Media.Capture, вместе со средствами для выполнения захвата в файл, поток или в какой-нибудь пользовательский объект, в зависимости от того, как приложение хочет обрабатывать захваченные данные или манипулировать ими.

В лекции 2 курса "Введение в разработку приложений для Windows 8 с использованием HTML, CSS и JavaScript", в приложении "Here My Am!", мы видели, как просто WinRT позволяет делать фотографии. Нам лишь нужно объявить возможность Веб-камера (Webcam) в манифесте и добавить несколько строк кода:

```
function capturePhoto() {
var that = this;

var captureUI = new Windows.Media.Capture.CameraCaptureUI();

//Показывает, что мы хотим захватить PNG-файл, размером не больше,
//пользовательский интерфейс автоматически выведет рамку, обрезающ
captureUI.photoSettings.format = Windows.Media.Capture.CameraCaptureUIP
captureUI.photoSettings.croppedSizeInPixels =
{ width: this.clientWidth, height: this.clientHeight };

captureUI.captureFileAsync(Windows.Media.Capture.CameraCaptureUIMode.
```

```
.done(function (capturedFile) {
//Убедитесь, что проверили, подходит ли вам возвращенный элемент: о
//пользователь отменил действие.
if (capturedFile) {
lastCapture = capturedFile; //Сохраним для целей общего доступа
that.src = URL.createObjectURL(capturedFile, {oneTimeOnly: true});
}
}, function (error) {
console.log("Unable to invoke capture UI.");
});
}
```

Пользовательский интерфейс, который Windows предоставляет нам посредством данного API, предоставляет настройки для обрезки и повторной съемки изображения, и для установки параметров камеры. Другой пример получения фотографии можно найти в Сценарии 1 примера "Пользовательский интерфейс камеры" (ссылка: http://code.msdn.microsoft.com/windowsapps/CameraCaptureUI-Sample-845a53ac), вместе с примером захвата видео в Сценарии 2. В последнем случае (js/capturevideo.js) мы настраиваем объект интерфейса захвата данных на видеоформат и указываем видеорежим при вызове `captureFileAsync`:

```
function captureVideo() {
var dialog = new Windows.Media.Capture.CameraCaptureUI();
dialog.videoSettings.format = Windows.Media.Capture.CameraCaptureUIVideo

dialog.captureFileAsync(Windows.Media.Capture.CameraCaptureUIMode.vide
.done(function (file) {
if (file) {
var videoBlobUrl = URL.createObjectURL(file, {oneTimeOnly: true});
} else {
//...
}
}, function (err) {
//...
});
}
```

Следует отметить, что возможность Веб-камера (Webcam) в манифесте применима лишь для захвата фотографий или записи видео без звука. Если вы хотите захватывать аудиосигнал, убедитесь в том, что объявили возможность Микрофон (Microphone) на закладке Возможности (Capabilities) редактора манифеста.

Если вы взглянете на объект `Windows.Media.Capture.CameraCaptureUI`, вы увидите множество параметров, которые вы можете настраивать. Его свойство `photoSettings`, это объект `CameraCaptureUIPhotoCaptureSettings`, который позволяет вам указывать размер захватываемого изображения и соотношение сторон, формат и максимальное разрешение. Его свойство `videoSettings`, это объект `CameraCaptureUIVideoCaptureSettings`, который позволяет вам задавать максимальную длительность и разрешение видео, и задает следует ли пользовательскому интерфейсу позволять обрезку. Все это очень полезно! Вы можете найти дополнительные сведения об этом в материале "Захват или воспроизведение звука, видео и изображений" (ссылка: http://msdn.microsoft.com/library/windows/apps/hh465156.aspx), который включает в себя сведения по управлению вызовами на устройстве связи Bluetooth.

Конечно, интерфейс камеры по умолчанию не подходит для всех случаев ее использования. Например, он всегда направляет вывод данных в файл, но если вы пишете коммуникационное приложение, например, вам скорее нужно отправить захваченное видео в поток или отправить его по сети, вовсе не используя файлы. Вам может понадобиться организовать предварительный просмотр видео перед началом захвата. Более того, вам может понадобиться добавить эффекты при захвате, реализовать поворот изображения, и, возможно, применить пользовательское кодирование.

Все эти возможности доступны посредством класса `Windows.Media.Capture.MediaCapture` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.media.capture.mediacap

Таблица 6.5.

Свойства	Описание (классы в про
audioController	Свойство AudioDeviceC предоставляет возможность воздействующими на аудио
mediaCaptureSettings	Свойство MediaCapture параметры режима, и позво photo).
videoController	Свойство VideoDeviceC изображения (яркостью, отт и так далее), предоставляя н частоты переменного тока, другими свойствами, воздеі
События	
failed	Вызывается, когда в поцесс
recordLimitationExceeded	Вызывается, когда пользова длительность которых прев
Методы	
initializeAsync	Инициализирует объект Me или с использованием объе MediaCaptureInitial самое, что содержится в Me
addEffectAsync	Применяет эффект.
clearEffectsAsync	Удаляет все текущие эффект
capturePhotoToStorageFileAsync capturePhotoToStreamAsync	Захватывает изображение в Оба варианта принимают э управление форматом (JPE произвольными свойствам http://msdn.microsoft.com/libra как описано ранее, в раздел
getEncoderProperty setEncoderProperty	Управляет специфическими
startRecordToStorageFileAsync startRecordToStreamAsync	Начинают и останавливают MediaEncodingProfil

stopRecordAsync	вместе с битрейтом, качеств
getRecordRotation setRecordRotation	Применимы для видео, раб градусным приращением) в аудио.
startRecordToCustomSinkAsync	Начинает запись в пользова реализации Windows.Mec использованием набора пар
startPreviewAsync startPreviewToCustomSinkAsync stopPreviewAsync getPreviewRotation setPreviewRotation	Выполняют те же функции, предварительного просмотр URL.createObjectURL качестве перевого параметр атрибуту src элемента vic отображены в этом элемент
getPreviewMirroring setPreviewMirroring	Управляют зеркальным отра подразумевает горизонтальн различной установкой, кото пользователя (камеры, уста планшетных компьютерах), на мониторах или встраива раздел: "Выбор устройства з

Для того, чтобы увидеть очень простую демонстрацию возможностей предварительного просмотра видео в элементе video, мы можем обратиться к примеру "Параметры пользовательского интерфейса камеры" (ссылка: http://code.msdn.microsoft.com/windowsapps/CameraOptionsUI-Sample-44c06873), к его файлу js/showoptionsui.js. Когда вы нажимаете на кнопку Start Preview (Начать предварительный просмотр), создается и инициализируется объект MediaCapture:

```
function initializeMediaCapture() {
mediaCaptureMgr = new Windows.Media.Capture.MediaCapture();
mediaCaptureMgr.initializeAsync().done(initializeComplete, initializeError);
 }
```

Где обработчик initializeComplete вызывается в

startPreivew:

```
function startPreview() {
document.getElementById("previewTag").src = URL.createObjectURL(mediaC
document.getElementById("previewTag").play();
startPreviewButton.disabled = true;
showSettingsButton.style.visibility = "visible";
previewStarted = true;
}
```

В этом коде так же показан вызов Windows.Media.Capture.CameraOptionsUI , который происходит, когда вы нажимаете на кнопку Show Settings (Показать параметры), результат показан на рис. 6.5. Это всплывающий элемент, предоставленный системой с параметрами, которые соответствуют захватываемому потоку мультимедиа:

```
function showSettings() {
if (mediaCaptureMgr) {
Windows.Media.Capture.CameraOptionsUI.show(mediaCaptureMgr);
}
}
```

Кстати, если у вас возникают проблемы с запуском примера, подобного этому, в имитаторе Visual Studio, в частности, если вы сталкиваетесь с исключением, когда пытаетесь включить камеру – попытайтесь, вместо этого, запустить его на локальном компьютере или на удаленной машине.

Рис. 6.5. Пользовательский интерфейс настроек параметров камеры из примера (пустая нижняя часть обрезана)

Более сложный сценарий, включающий в себя использование класса `MediaCapture` (и некоторых других) можно найти в примере "Захват мультимедиа с использованием устройств для захвата" (ссылка: http://code.msdn.microsoft.com/windowsapps/Media-Capture-Sample-adf87622), в частности, это предварительный просмотр и захват видео, динамическое изменение свойств (Сценарий 1), выбор конкретного мультимедиа-устройства (Сценарий 2) и запись звука, без видео (Сценарий 3).Начнем со Сценария 3 (js/AudioCapture.js, он самый простой), здесь присутствует базовый код для создания и инициализации объекта `MediaCapture` для аудиопотока (смотрите свойство `streamingCaptureMode` в параметрах инициализации), где поток направляется в файл в библиотеке музыки посредством `startRecordToStorageFileAsync` (некоторая часть кода опущена для краткости):

```
var mediaCaptureMgr = null;
var captureInitSettings = null;
var encodingProfile = null;
var storageFile = null;

// Это вызывается при загрузке страницы
function initCaptureSettings() {
captureInitSettings = new Windows.Media.Capture.MediaCaptureInitializationSe
captureInitSettings.audioDeviceId = "";
captureInitSettings.videoDeviceId = "";
captureInitSettings.streamingCaptureMode =
Windows.Media.Capture.StreamingCaptureMode.audio;
 }

function startDevice() {
mediaCaptureMgr = new Windows.Media.Capture.MediaCapture();

mediaCaptureMgr.initializeAsync(captureInitSettings).done(function (result) {
// ...
});
 }
```

```
function startRecord() {
// ...
// Начинаем запись. Windows.Storage.KnownFolders.videosLibrary.createFile
Windows.Storage.CreationCollisionOption.generateUniqueName)
.done(function (newFile) {
storageFile = newFile;
encodingProfile = Windows.Media.MediaProperties
.MediaEncodingProfile.createM4a(Windows.Media.MediaProperties
.AudioEncodingQuality.auto);
mediaCaptureMgr.startRecordToStorageFileAsync(encodingProfile, storageFile)
// ...
});
});
}

function stopRecord() {
mediaCaptureMgr.stopRecordAsync().done(function (result) {
displayStatus("Record Stopped. File " + storageFile.path + " ");

// Воспроизведение записанного аудио
var audio = id("capturePlayback" + scenarioId);
audio.src = URL.createObjectURL(storageFile, { oneTimeOnly: true });
audio.play();
});
 }
```

Сценарий 1, в целом, использует тот же код, но захватывает видеопоток и фотографии, результат его работы показан на рис. 6.6. Эти особенности включены посредством следующих свойств в установках инициализации (смотрите js/BasicCapture.js, внутри `initCaptureSettings`):

```
captureInitSettings.photoCaptureSource =
Windows.Media.Capture.PhotoCaptureSource.videoPreview;
captureInitSettings.streamingCaptureMode =
Windows.Media.Capture.StreamingCaptureMode.audioAndVideo;
```

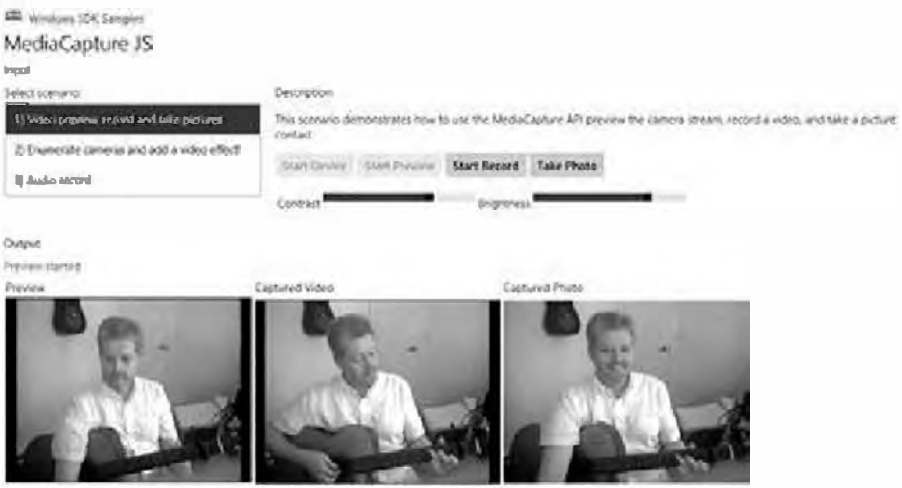


Рис. 6.6. Предварительный просмотр и запись видео с устройства по умолчанию в Сценарии 1 примера о захвате мультимедиа. (Изображение обрезано, так как мне понадобилось запустить этот пример с использованием опции Локальный компьютер в Visual Studio, и я не думаю, что вам надо видеть скриншот размером 1920x1200 с большим количеством пустого пространства!)

Обратите внимание на элементы управления Contrast (Контраст) и Brightness (Яркость) на рис. 4.9. Их изменение меняет видео во время предварительного просмотра, вместе с записываемым видео. В примере это реализовано с помощью свойств `contrast` и `brightness` объекта `MediaCapture.videoDeviceController`, демонстрируя возможность динамической настройки этих (и других) свойств контроллера. Обратитесь к функции `getCameraSettings` в js/BasicCapture.js, которая подключает события `change` ползунка в обычной анонимной функции для обновления необходимого свойства.

Выбор устройства захвата мультимедиа

Взглянем теперь на Сценарий 2 (js/AdvancedCapture.js), он более или менее похож на Сценарий 1, но он позволяет вам выбирать конкретное

устройство захвата. До настоящего момента мы пользовались лишь устройством по умолчанию, но вы не ограничены этим, конечно. Вы можете использовать API `Windows.Devices.Enumeration` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.devices.enumeration.asp для получения списка устройств в классе интерфейсов устройств. В примере использован заранее определенный класс `videoCapture`:

```
function enumerateCameras() {
var cameraSelect = id("cameraSelect");
deviceList = null;
deviceList = new Array();
while (cameraSelect.length > 0) {
cameraSelect.remove(0);
}
//Перечисляем вебкамеры и добавляем их в список
var deviceInfo = Windows.Devices.Enumeration.DeviceInformation;
deviceInfo.findAllAsync(Windows.Devices.Enumeration.DeviceClass.videoCapt
.done(function (devices) {
// Добавляем устройства в deviceList
if (devices.length > 0) {
for (var i = 0; i < devices.length; i++) {
deviceList.push(devices[i]);
cameraSelect.add(new Option(deviceList[i].name), i);
}
//Выбираем первую вебкамеру
cameraSelect.selectedIndex = 0;
initCaptureSettings();
} else {
// блокируем кнопки.
}
}, errorHandler);
}
```

Выбранные ID устройств затем копируются внутри `initCaptureSettings` в свойство `MediaCaptureInitializationSetting.videoDeviceId`:

```
var selectedIndex = id("cameraSelect").selectedIndex;
var deviceInfo = deviceList[selectedIndex];
captureInitSettings.videoDeviceId = deviceInfo.id;
```

Кстати, вы всегда можете получить ID устройства по умолчанию, воспользовавшись методами объекта `Windows.Media.Devices.MediaDevice` (ссылка: http://msdn.microsoft.com/library/windows/apps/br226802.aspx) и прослушивая его события на предмет изменения устройств по умолчанию. Важно отметить, что `DeviceInformation` (в переменной `deviceInfo` выше) включает в себя свойство, которое называется `enclosureLocation`. Оно сообщает о том, направлена ли камера на пользователя, или от него, что вы можете использовать для поворота видео или фото так, чтобы их отображение соответствовало восприятию пользователя:

```
var cameraLocation = null;

if (deviceInfo.enclosureLocation) {
cameraLocation = deviceInfo.enclosureLocation.panel;
}

if (cameraLocation === Windows.Devices.Enumeration.Panel.back) {
rotateVideoOnOrientationChange = true;
reverseVideoRotation = false;
} else if (cameraLocation === Windows.Devices.Enumeration.Panel.front) {
rotateVideoOnOrientationChange = true;
reverseVideoRotation = true;
} else {
rotateVideoOnOrientationChange = false;
}
```

Сценарий 2 демонстрирует, так же, использование `MediaCapture.addEffectAsync` с эффектом перевода в оттенки серого, что показано на рис. 6.7., который реализован в DLL (проект GrayscaleTransform в решении). Это работает в точности так, как работало уже рассмотренное нами перекодирование, и вы можете обратиться к функциям `addRemoveEffect` и

`addEffectToImageStream` в файле примера js/AdvancedCapture.js для того, чтобы увидеть подробности. Вы можете здесь заметить, что эти функции выполняют некоторое количество проверок с использованием значения `MediaCaptureSettings.videoDeviceCharacteristic` для того, чтобы убедиться, что эффект добавлен в верное место.

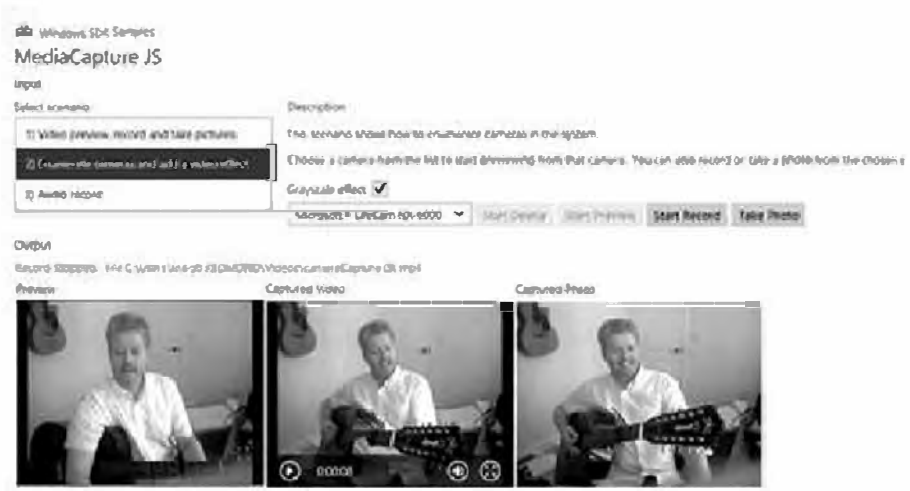


Рис. 6.7. Сценарий 2 примера о захвате мультимедиа, где можно выбирать конкретное устройство захвата и применять эффект (Окно обрезано из скриншота большего размера). Вы достаточно внимательно посмотрели на этот рисунок, чтобы заметить, что я поменял гитары?

Потоковое мультимедиа и целевые объекты воспроизведения

Грубым преуменьшением будет говорить, что потоковое мультимедиа популярно в наши дни. Как я упоминал во введении к лекции, один только Netflix занимает немалую долю полосы пропускания современного Интернета (и в моем доме – тоже). YouTube тоже не стоит в стороне – и ваше приложение так же может внести свой вклад в это дело!

Передача мультимедийных данных с сервера приложению это простой

и наболее распространенный случай. Это происходит автоматически, когда вы устанавливаете атрибут `src` элементов `audio` или `video` на удаленный URI. Для того, чтобы улучшить этот процесс, у Microsoft есть Smooth Streaming SDK для Windows 8-приложений (ссылка: http://visualstudiogallery.msdn.microsoft.com/04423d13-3b3e-4741-a01c-1ae29e84fea6?SRC=Home) (он имеет статус бета-версии, когда я это пишу), который призван помочь вам создавать мультимедиа-приложения с большим набором богатых возможностей, включая живое воспроизведение и защиту содержимого PlayReady. Я не описываю в данном курсе этот SDK, я лишь хочу, чтобы вы о нем знали.

На последних страницах данной лекции мы рассмотрим соображения по управлению цифровыми правами (DRM), и поговорим о потоковой передаче данных, но не из сети, а в сеть. Например, о захвате аудио и видеоинформации коммуникационным приложением, и о передаче приложением потокового мультимедиа на целевые устройства воспроизведения (PlayTo).

Потоковая передача данных с сервера и DRM

Потоковая передача мультимедиа с сервера, это то, что вы уже делали, применяя элементы `audio` и `video` с URI, указывающими на удаленные ресурсы. Все это выполняется автоматически. На самом деле, все, что делает отличный мультимедиа-клиент – это общается с веб-сервисом, получает метаданные и списки элементов, помогает пользователю перемещаться по этой информации и, в конце концов, получает URI, который можно передать в атрибут src элемента `audio` или `video`. Затем, как только приложение получает событие `canplay`, вы можете вызвать метод `play` элемента для того, чтобы начать воспроизведение.

Конечно, мультимедийные записи часто защищены DRM, в противном случае содержимое платных сервисов не приносило бы денег владельцам прав! Таким образом, здесь нужен механизм для получения и проверки прав где-то между установкой атрибута src элемента и получением события canplay. К счастью, есть простые средства для того, чтобы это сделать:

Перед установкой атрибута `src`, создается экземпляр

`Windows.Media.Protection.MediaProtectionManager` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.media.protection.mediap и настраивается его свойство `properties`.

Прослушивается событие этого объекта `serviceRequested`, обработчик которого производит проверку прав и устанавливает флаг завершения, если все верно. (Два других события, о которых я просто хочу упомянуть, это `componentloadfailed` и `rebootneeded`.)

Присвоение средства управления защитой к элементу audio/video с использованием метода расширения `msSetMediaProtectionManager` (ссылка: http://msdn.microsoft.com/library/windows/apps/hh465953.aspx).

Установка атрибута `src`. Это вызовет событие `serviceRequested` для начала обработки DRM, которая предотвращает вызов `canplay` до тех пор, пока проверка прав не завершится успешно.

Вы можете обратиться к материалам "Использование подключаемого DRM" (ссылка: http://msdn.microsoft.com/library/windows/apps/hh452779.aspx), и "Обработка ошибок DRM" (ссылка: http://msdn.microsoft.com/library/windows/apps/hh452767.aspx) для того, чтобы узнать подробности, здеь приведен минимальный гипотетический пример кода, который все это реализует:

```
var video1 = document.getElementById("video1");

video1.addEventListener('error', function () {
var error = video1.error.msExtendedCode;
//...
}, false);

video1.addEventListener('canplay', function () {
video1.play();
}, false);

var cpm = new Windows.Media.Protection.MediaProtectionManager();
```

```
cpm.addEventListener('servicerequested', enableContent, false); //Удалите это п
video1.msSetContentProtectionManager(cpm);
video1.src = "http://some.content.server.url/protected.wmv";


function enableContent(e) {
if (typeof (e.request) != 'undefined') {
var req = e.request;
var system = req.protectionSystem;
var type = req.type;

//Предпринять необходимые действия на основе системы и типа
}
if (typeof (e.completion) != 'undefined') {
//Запрошенное действие завершено
var comp = e.completion;
comp.complete(true);
}
}
```

Как конкретно вы проверяете права, зависит от сервиса, из которого вы получаете данные – и это не то, что вам захочется опубликовать!

Более полная демонстрация работы с DRM находится в примере "Простое использование PlayReady" (ссылка: http://code.msdn.microsoft.com/windowsapps/Simple-PlayReady-sample-5c1aefaf), которому нужно, чтобы вы скачали и установили Microsoft PlayReady Client SDK (ссылка: http://visualstudiogallery.msdn.microsoft.com/e02ccac7-f3eb-4b53-b11a-c657d5631483). PlayReady, если вы еще с ним не знакомы, это сервис лицензирования, который предоставляет Microsoft, таким образом, вам не нужно создавать что-то подобное с нуля. PlayReady Client SDK предоставляет дополнительные инструменты и подержку фреймворка для приложений, которые нуждаются в реализации онлайновых и оффлайновых сценариев работы с мультимедиа, таких, как прогрессивная загрузка, загрузка для владения, выдача напрокат и подписка. В дополнение к этому, с использованием этого SDK вам не нужно отправлять ваше приложение для тестирование на соответствие

требованиям DRM. В любом случае, вот как пример использования PlayReady настр аивает свое средство защиты содержимого, просто для того, чтобы показать, как API WinRT используются с конкретными идентификаторами DRM-сервисов:

```
mediaProtectionManager = new Windows.Media.Protection.MediaProtectionMa
mediaProtectionManager.properties["Windows.Media.Protection.MediaProtectio
'{F4637010-03C3-42CD-B932-B48ADF3A6A54}'

var cpsystems = new Windows.Foundation.Collections.PropertySet();
cpsystems["{F4637010-03C3-42CD-B932-B48ADF3A6A54}"] =
"Microsoft.Media.PlayReadyClient.PlayReadyWinRTTrustedInput";
mediaProtectionManager.properties[
"Windows.Media.Protection.MediaProtectionSystemIdMapping"] = cpsystems;
```

Потоковая передача мультимедиа из приложения в сеть

Следующий случай касается ситуации, когда приложение является источником потоковых мультимедиа-данных, вместо того, чтобы быть их потребителем, что означает, что клиентские приложения в других местах выступают в таком качестве. На самом деле, в этом сценарии – аудио- или видеокоммуникации и реализации конференц-связи – обычно приложение играет обе роли, выполняя потоковую передачу данных другим клиентским приложениям и получая данные от них. Так работают Windows Live Messenger, Skype и другие подобные программы, вместе с приложениями наподобие игр, которые включают в себя возможности чата.

Вот как такие приложения обычно работают:

- Настройка необходимых каналов связи через сеть, это могут быть одноранговые системы, или системы с привлечением некоей централизованной службы.
- Захват аудио и видео в поток с использованием API WinRT, которые мы уже видели (в особенности `MediaCapture.startRecordToStreamAsync`), или захват в пользовательский объект.

- Выполнение любой дополнительной обработки данных в потоке. Обратите внимание, однако, что здесь применяются эффекты, подключенные к механизму захвата (`MediaCapture.addEffectAsync`), вместо тех, которые обрабатывают уже захваченные данные.
- Кодирование потока для передачи любым необходимым способом.
- Передача потока через сетевой канал связи.
- Прием передач от других подключенных приложений.
- Декодирование переданных приложению потоков и конверсия в большой двоичный объект (blob) с использованием `MSApp.createBlobFromRandomAccessStream`.
- Использование `URL.createObjectURL` для подключения элементов `audio` и `video` к потоку.

Для того, чтобы увидеть это в действии, обратитесь к примеру "Коммуникация в реальном времени" (ссылка: http://code.msdn.microsoft.com/Simple-Communication-Sample-eac73290), который реализует видеочат в Сценарии 2 и показывает работу с различными режимами задержки с Сценарии 1. Последние два шага в вышеприведенном списке так же показаны в примере "Приемник целевого объекта воспроизведения" (ссылка: http://code.msdn.microsoft.com/windowsapps/PlayToReceiver-sample-607f00ed), который настроен на прием мультимедиа-потоков из других источников.

Целевые объекты воспроизведения

Последний вариант работы с потоковыми данными построен вокруг возможности передачи данных на целевые объекты воспроизведения (PlayTo), которая была представлена в Windows 7. Проще говоря, эта возможность представляет собой средство, с помощью которого приложения могут подключать локальное воспроизведение или просмотра элементов `audio`, `video` и `img` к удаленным устройствам.

Работа ведется с помощью API `Windows.Media.PlayTo` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.media.playto.aspx) вместе с методами расширений, добавленных к медиа-элементам. Если,

например, вы хотите запустить процесс передачи данных на целевой объект воспроизведения, непосредственно вызывая пользовательский интерфейс, выполните следующую последовательность действий:

- `Windows.Media.PlayTo.PlayToManager` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.media.playto.play
  :
  - `getForCurrentView` возвращает объект.
  - `showPlayToUI` запускает всплывающий элемент интерфейса, где пользователь выбирает приемник.
  - событие `sourceRequested`, когда пользователь выбирает приемник.
- В `sourceRequested`
  - Получить объект `PlayToSource` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.media.play из элемента `audio`, `video`, или `img` (свойство `msPlayToSource`) и передать в `e.setSource`.
  - Установить свойство `PlayToSource.next` на `msPlayToSource` другого элемента для непрерывного воспроизведения.
- Принять событие медиа-элемента `ended` для воспроизведения дополнительных мультимедийных данных

Другой подход, как показано в примере "Проигрывание мультимедиа на целевых устройствах воспроизведения" (ссылка: http://code.msdn.microsoft.com/windowsapps/Media-PlayTo-Sample-fedcb0f9), идет дальше и проигрывает мультимедийные файлы локально, затем позволяя пользователю выбрать целевое устройство воспроизведения из меню чудо-кнопки Устройства. В подобном случае вам не нужно ничего предпринимать, так как Windows возьмет текущий элемент, выполняющий воспроизведение, и соответствующим образом направит вывод. Но приложение может прослушивать событие `statechanged` объекта `msPlayToSource.connection` элемента (тип `PlayConnection` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.media.playto.playtoconr которое будет вызвано, когда пользователь выберет целевое устройство и когда происходят другие изменения.

На самом деле, функция воспроизведения на целевом устройстве предназначена для передачи потоковых данных на медиа-проигрыватели, которые могут быть подключены к телевизору или к другим большим экранам. С помощью этого механизма вы можете выбрать локальное содержимое на устройстве, которое работает под управлением Windows 8 и отправить его прямо на проигрыватель, способный принимать потоковые данные. Но, кроме того, подобный приемник можно реализовать программно, то есть, создать приложение, которое может принимать потоковое содержимое из PlayTo-источника. В примере "Приемник PlayTo" (ссылка: http://code.msdn.microsoft.com/windowsapps/PlayToReceiver-sample-607f00ed) реализован такой подход, и если вы запустите его на другом компьютере в вашей локальной сети, он будет показан в интерфейсе чудо-кнопки Устройства следующим образом:

Вы даже можете запустить это приложение с основного рабочего компьютера, воспользовавшись возможностями удаленной отладки Visual Studio, что позволит вам пошагово исполнить и код приложения-источника, и код приложения-приемника в одно и то же время! (Другая возможность – это запустить Windows Media Player на одном компьютере, установить параметр Разрешить удаленное управление проигрывателем (Allow Remote Control of My Player) в меню Поток (Stream). В результате данный компьютер появится в списке целевых устройств функции Воспроизвести на (PlayTo)).

Для того чтобы выполнять функции приемника, приложению обычно нужно объявить некоторые дополнительные сетевые возможности в манифесте, а именно - Интернет (клиент и сервер) (Internet (Client&Server)), и Частные сети (клиент и сервер) (Private Networks (Client&Server)), иначе приложение не будет работать так, как нужно.

Затем приложение создает экземпляр `Windows.Media.PlayTo.PlayToReceiver`, как показано в функции `startPlayToReceiver` (js/audiovideoptr.js), в примере "Приемник PlayTo":

```
function startPlayToReceiver() {
if (!g_receiver) {
g_receiver = new Windows.Media.PlayTo.PlayToReceiver();
}
```

Далее, нужно подключить обработчики для элементов, которые будут воспроизводить мультимедиа-поток:

```
var dmrVideo = id("dmrVideo");
dmrVideo.addEventListener("volumechange", g_elementHandler.volumechange, f
g_elementHandler.ratechange, false); dmrVideo.addEventListener("loadedmetada
dmrVideo.addEventListener("durationchange", g_elementHandler.durationchange
g_elementHandler.seeking, false); dmrVideo.addEventListener("seeked", g_elem
("playing", g_elementHandler.playing, false); dmrVideo.addEventListener("pause"
dmrVideo.addEventListener("ended", g_elementHandler.ended, false);
dmrVideo.addEventListener("error", g_elementHandler.error, false);
```

А так же – обработчики для событий, которые вызывает объект приемника:

```
g_receiver.addEventListener("playrequested", g_receiverHandler.playrequested,
g_receiver.addEventListener("pauserequested", g_receiverHandler.pauserequeste
g_receiver.addEventListener("sourcechangerequested", g_receiverHandler.source
g_receiver.addEventListener("playbackratechangerequested", g_receiverHandler.
g_receiver.addEventListener("currenttimechangerequested", g_receiverHandler.c
g_receiver.addEventListener("mutechangerequested",
g_receiverHandler.mutedchangerequested, false); g_receiver.addEventListener("
g_receiverHandler.volumechangerequested, false); g_receiver.addEventListener("
g_receiverHandler.timeupdaterequested, false); g_receiver.addEventListeer("stop
g_receiverHandler.stoprequested, false); g_receiver.supportsVideo = true;
g_receiver.supportsAudio = true;
g_receiver.supportsImage = false;
```

```
g_receiver.friendlyName = 'SDK JS Sample PlayToReceiver';
```

Последняя строка выше, как вы можете понять по вышеприведенному изображению, это строка, которая отобразится в интерфейсе чудо-кнопки Устройства для данного приложения-приемника, как только оно станет доступным в сети. Это выполняется путем вызова `startAsync`:

```
// Анонсирует приемник в локальной сети и начинает прием команд
g_receiver.startAsync().then(function () {
g_receiverStarted = true;

// Предотвращает блокировку экрана
if (!g_displayRequest) {
g_displayRequest = new Windows.System.Display.DisplayRequest();
}
g_displayRequest.requestActive();
});
```

Среди всех событий объекта-приемника, наиболее важное – `sourcechanderequested`, где `eventArgs.stream` содержит мультимедиа-данные, которые мы хотим проиграть в выбранном элементе. Это просто реализуется путем создания большого двоичного объекта из потока, и затем URI из этого объекта, которое мы можем присвоить атрибуту src элемента:

```
sourcechangerequested: function (eventIn) {
if (!eventIn.stream) {
id("dmrVideo").src = "";
} else {
var blob = MSApp.createBlobFromRandomAccessStream(eventIn.stream.conte
id("dmrVideo").src = URL.createObjectURL(blob, {oneTimeOnly: true});
}
}
```

Все остальные события, как вы можете представить, в основном

касаются связывания элементов управления источника и приемника, таким образом, нажатие на кнопку постановки на паузу, переключения дорожки или какое-то другое воздействие на проигрываемые данные в источнике отразится на приемнике. Здесь много событий, но обрабатывать их просто, как вы можете увидеть в примере.

Что мы только что изучили

- Мультимедиа-элементы сожно создать в разметке или в коде, используя стандартные элементы `img`, `svg`, `canvas`, `audio` и `video`.
- Три графических элемента – `img`, `svg` и `canvas` – могут выводить изображения, выглядящие одинаково, но обладающие разными характеристиками, касающиеся их создания и того, как они масштабируются. Все они, однако, могут быть стилизованы с помощью CSS.
- Объект `Windows.System.Display.DisplayRequest` предназначен для отключения скринсейверов и экрана блокировки в течение проигрывания видео (или в других подходящих случаях).
- И аудио- и видеоэлементы предоставляют множество расширений API (свойств, методов и событий) для работы с различными специфическими возможностями платформы Windows 8, таких, как горизонтальное отражение, масштабирование, оптимизация воспроизведения, 3D-видео, воспроизведение с низкой задержкой, воспроизведение на целевых устройствах, управление воспроизведением различных типов или категорий аудиозаписей, эффекты (обычно реализуемые в виде DLL в пакете приложения), и управление цифровыми правами.
- Фоновое аудио поддерживается для различных категорий, для реализации этой возможности нужны объявления возможностей в манифесте, и обработчики для событий элементов управления воспроизведения (таким образом, звук может быть соответствующим образом воспроизводиться и приостанавливаться). События элементов управления мультимедиа так же важны для поддержки пользовательского интерфейса управления воспроизведением.
- Посредством API WinRT приложения могут управлять богатым

набором метаданных и свойств мультимедиа-файлов, в том числе – эскизами, изображениями альбомов и свойствами, специфичными для типа содержимого, в том числе, они имеют доступ к очень большому списку свойств Windows (ссылка: http://msdn.microsoft.com/library/windows/desktop/dd561977%28v=vs.85

- API WinRT предоставляет средства для декодирования и кодирования мультимедиа-файлов и потоков, с помощью которых содержимое может быть конвертированы, могут быть изменены свойства содержимого. Это включает в себя поддержку пользовательских кодеков.
- WinRT предоставляет богатое возможностями API для захвата мультимедиа (фотографий, видеоклипов, аудиозаписей) в том числе, встроенный пользовательский интерфейс для этих целей, вместе с возможностью реализовать собственный, а так же – возможности по перечислению имеющихся устройств и по доступу к ним.
- Поддерживается воспроизведение потокового мультимедиа с сервера (с применением управления цифровыми правами, или без них, и с возможностью использовать PlayReady), между приложениями (как работа с входящими данными, так и с исходящими), и с приложений на целевые устройства воспроизведения (PlayTo). Приложение, кроме того, может быть настроено в качестве PlayTo-приемника.

Целенаправленная анимация

Здесь читателю предлагается описание особенностей применения анимации в приложениях для Магазина Windows. В частности, рассмотрены примеры из библиотеки анимации WinJS, реализация анимации с помощью средств CSS и создание покадровой анимации.

Файлы к данной лекции Вы можете скачать здесь скачать: http://old.intuit.ru/department/se/uiwin8htmljs/7/5.zip.

В начале 1990-х удивительный мир мультимедиа впервые пришел на персональные компьютеры с Windows. До этого таким машинам тяжело было воспроизводить аудио и видео и работать с компакт-дисками (помните о них?), и обеспечивать тот богатый опыт взаимодействия, который сегодня мы считаем привычным. Опыт работы с мультимедиа был восхитительно новым, и многие люди от всего сердца отдались ему, в том числе и группа инженеров поддержки в Microsoft, которая специализировалась в этой области. Хотя моя команда (мы занимались пользовательским интерфейсом) была расположена более чем в 100 футах от них, мы могли отлично слышать, - в течение всего дня!, - всякие писки и звонки, исходившие из динамиков их машин на фоне мягких звуков дождя в бассейне Амазонки.

Тогда же многие пользователи Windows развлекались, назначая всяческие безумные звуки каждому щелчку мыши, открытию окна, поступлению почты и каждому системному событию, о котором можно подумать. Тем не менее, через месяц-другой этой какофонии, делающей работу похожей на карнавал, большинство людей начало отключать многие, если не все эти звуки.Я, например, в конце концов, выключил их все. Проще говоря, я устал от чрезмерного шума.

В том же духе, вы можете вспомнить, что когда появились, во всей красе, первые DVD, почти каждое меню диска было полно визуальных украшений. Теперь это не так: многие зрители, я думаю, устают ждать, пока все это будет показано, и просто хотят воспользоваться диском по его прямому назначению, приступив к просмотру фильма так быстро, как только возможно.

Сегодня мы переживаем то же самое с высококачественной анимацией.

Сейчас, когда большое количество систем имеет высокочувствительные сенсорные экраны и графические процессоры, позволяющие воспроизводить качественные графические эффекты, заманчиво использовать анимацию везде, где это возможно. Однако, несмотря на то, что анимация, на самом дел, добавляет функциональности и смысла происходящему в приложении, ползователи могут устать от нее, как от DVD-меню, особенно, если она начинает вмешиваться в рабочий процесс, постоянно заставляя пользователей ждать, когда закончится анимация. Держу пари, что каждый читатель этого курса, хотя бы однажды, без всякого толка много раз нажимал на кнопку Меню пульта DVD-проигрывателя.

Именно поэтому дизайн приложений для Магазина Windows особое внимание уделяет целенаправленной анимации: если у анимации в вашем приложении нет реальной цели, спросите себя: "Зачем я собираюсь это применять?". Найдите время для того, чтобы посмотреть на то, как Windows 8 и некоторые встроенные приложения используют и не используют анимацию. Обратите внимание на то, как много анимаций предназначены для отслеживания действий, или, иначе, для предоставления немедленной обратной связи при сенсорном взаимодействии. Такой подход целенаправленно помогает пользователям знать, что система восприняла их действия (и, на самом деле, это есть в требованиях сертификации для Магазина Windows). Другие анимации, такие, которые происходят, когда элемент был добавлен в спискок или удален их него, предназначены для привлечения внимания к изменениям, для смягчения их визуального воздействия и создание ощущения плавности. В других случаях вы можете обнаружить приложения, которые, возможно, чрезмерно используют анимацию, просто применяя ее потому что у них есть такая возможность, или пытаясь точно передать поведение физических объектов там, где в этом нет необходимости.

Таким образом, чрезмерная анимация представляет собой, своего рода "визуальное оформление", обладающее тем же эффектом, что и прочие элементы оформления, которые отвлекают пользователя от содержимого, которое ему по-настоящему нужно. (Если вы не можете удержаться от соблазна добавить немного подобных эффектов, предусмотрите, как минимум, возможность для их отключения).Позвольте мне сказать это иначе. Размышляя об

анимациях, спросите себя: "О чем они сообщают?". Анимации – это разновидность коммуникации, что-то вроде визуального языка. Я даже рискнул бы сказать (как я решился сделать сейчас), что анимации, на самом деле, лишь сообщают об одном или нескольких событиях следующих трех видов:

- "Спасибо, я услышал", - когда что-то на экране перемещается естественным образом в ответ на жест пользователя. Без подобного "ответа", пользователь может подумать, что система не восприняла его жест, и почти наверняка ткнет в экран пальцем еще раз.
- "Привет" и "Пока", - когда элементы появляются или исчезают, или одни сменяются другими. Без подобной коммуникации, изменения, которые происходят с экранными элементами, могут быть такими же резкими, как мгновенное исчезновение Бильбо Бэггинса из "Властелина колец", который надел Кольцо всевластия. Я не хочу сказать, что все пользователи – это недоверчивые хоббиты, конечно, но идею я вам дал.
- "Эй, посмотри на меня!", - когда что-то перемещается только для того, чтобы привлечь внимание или привлекательно выглядеть.

Если бы я назначил этим трем категориям процентные значения, представляющие то, как часто они используются или могли бы использоваться, я, возможно, дал бы каждой из них, соответственно, 80%, 15% и 5% (хотя некоторые из анимаций служат нескольким целям). Иными словами, первая часть коммуникации заключается в том, чтобы слушать и отвечать, и именно это приложению следует делать большую часть времени. Вторая часть касается вежливости, которая хороша в разумных пределах, так как рукопожатия, объятия и приветствия, если их слишком много, могут вызвать раздражение. Третья часть, которую можно сравнить с взмахом руки, в свою очередь, может быть полезна лишь тогда, когда для ее использования есть реальные причины, в противном случае она становится лишь средством внешнего оформления.

Есть и еще одна веская причина разумному и экономному использованию анимации: потребление электоэнергии. Неважно, как она реализована, основана ли на ресурсах центрального процессора или графического ядра, анимация – это энергозатратный процесс. Каждый

ватт, утекающий из батарей устройства пользователя, должен быть направлен на удовлетворение целей, ради которых он использует устройство, а не выброшен на ветер. Именно поэтому данная лекция называется "Целенаправленная анимация", а не просто "Анимация"!

В любом случае, вы и ваши дизайнеры принимаете окончательное решение о том, как и где использовать анимации. Позвольте мне подчеркнуть, что анимация должна быть частью дизайна приложения, а не одной из деталей его реализации. Анимация – это важная часть общего опыта взаимодействия пользователя и приложения. Часто работа над дизайном приложения сосредоточена на статических каркасах и статических моделях, но ни то, ни другое не отражает динамических элементов наподобие анимаций и переходов. Анимация так же тесно связана с макетом приложения и ее следует проектировать вместе с макетом, на ранних стадиях работы над дизайном приложения.

В этой необычно короткой лекции, таким образом, мы сначала посмотрим, что предоставляет нам библиотека анимаций WinJS (WinJS Animations Library) – коллекция анимаций, построенных на основе CSS, которые уже воплощают стиль Windows 8 во многих обычно используемых операциях. После этго мы рассмотрим возможности CSS, лежащие в основе этих анимаций, которые вы, конечно, можете использовать напрямую. На самом деле, если не говорить об играх и других приложениях, основное содержимое которых состоит из анимированных объектов, вы, вероятно, можете использовать CSS для удовлетворения большинства потребностей в анимации. Это хорошая идея, так как подсистема CSS очень хорошо оптимизирована для того, чтобы извлекать преимущества из аппаратного ускорения, а при выполнении покадровой анимации средствами самого JavaScript это не так. Тем не менее, мы окончим эту лекцию рассмотрением этого последнего подхода, так как есть некоторые советы и секреты, касающиеся того, как сделать это хорошо в приложениях для Магазина Windows.

Включение и выключение анимаций во всей системе

Прежде чем мы продолжим, я хочу рассказать об одном параметре, который спрятан глубоко в настройках Панели управления, о котором вам надо знать, так как она влияет на то, как ведет себя библиотека

анимаций WinJS, и ей следует влиять на ту анимацию, которую вы реализуете своими силами. Находясь на Рабочем столе, откройте панель чудо-кнопок, нажмите на чудо-кнопку Параметры и выберите пункт Панель управления. Затем перейдите в раздел Специальные возможности > Оптимизация изображения на экране, прокрутите содержимое окна к его нижней части и в разделе Упрощение зрительного восприятия изображений на экране, установите флаг "Отключить всю излишнюю анимацию (по возможности)" (Turn off all unnecessary animations (when possible), рис. 7.1.

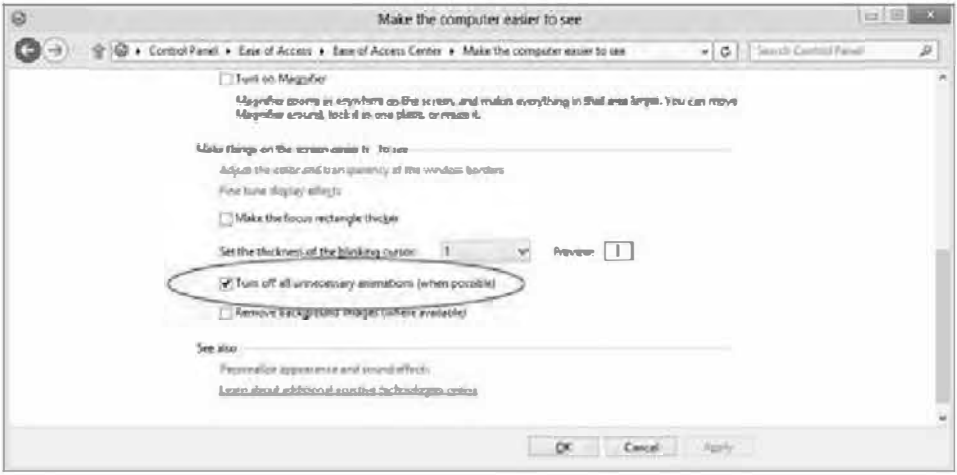


Рис. 7.1. Очень важный параметр, касающийся анимации в Панели управления, вызванной с Рабочего стола

Смысл этого флага заключается в том, что некоторым пользователям анимации по-настоящему мешают, что усложняет их работу с системой в целом. Кроме того, по медицинским причинам, некоторые пользователи могут предпочесть минимизировать движения на экране для того, чтобы сделать работу спокойнее. Таким образом, когда данный параметр включен, анимации WinJS не выполняются. Рекомендуется, чтобы приложения так же отключали большинство собственных анимаций, если не все из них.

Данный параметр Панели управления может быть получен с помощью свойства `animationsEnabled` класса `Windows.UI.ViewManagement.UISettings` (ссылка: http://msdn.microsoft.com/library/windows/apps/windows.ui.viewmanagement.uise

```
var settings = new Windows.UI.ViewManagement.UISettings();
var enabled = settings.animationsEnabled;
```

Так же вы можете просто вызвать метод WinJS.UI.`isAnimationEnabled`, который возвратит `true` или `false` в зависимости от значения этого свойства. WinJS использует этот параметр для управления поведением анимации.

WinJS так же имеет счетчик, который вы можете использовать для временного включения или выключения анимации в соответствии со значением `animationsEnabled`. Вы можете изменять его, вызывая `WinJS.UI.enableAnimations` и `WinJS.UI.disableAnimations`, их эффект последователен, и свойство animationsEnabled считается равным 0, если параметр Панели управления включен, и 1 – если выключен.

Реализуя собственную анимацию, либо с помощью CSS, либо – с помощью механизмов наподобие `setInterval` или `requestAnimationFrame`, правильно будет ориентироваться на установку `animationsEnabled` там, где это имеет смысл. Я добавил это условие, так как если анимация важна для содержимого приложения, наподобие игры, эта установка к нему не применима. То же самое относится к анимации, наподобие часов в приложении-часах. Это касается анимации, которая добавляет эффект быстроты и подвижности к содержимому, но может быть отключена без потери функциональности приложения.

Библиотека анимации WinJS

Рассуждая об использовании анимации в приложении, в первую очередь вы должны принять во внимание библиотеку анимации в WinJA, которую можно найти в пространстве имен `WinJS.UI.Animation` (ссылка: http://msdn.microsoft.com/library/windows/apps/br229780.aspx). Каждая анимация представлена в виде функции в данном пространстве имен, которую вы вызываете, если хотите, чтобы включилась определенная анимация или переход. Преимущество использования библиотеки заключаются в том, что она уже воплощает в себе стиль Windows, и, на

самом деле, применяется WinJS для анимации встроенных элементов управления, всплывающих элементов и так далее, обеспечивая соответствие пользовательского интерфейса руководствам по дизайну. Более того, так как библиотека построена на основе анимаций и переходов CSS, она независима от WinRT и полностью функциональна на страницах веб-контекста, которые может загружать WinJS (но, опять же, и при работе с такими страницами нужно обращать внимание на настройки воспроиз ведени анимации, описанные в предыдущем разделе).

Все анимации, перечисленные в таблице ниже, имеют руководства, касающиеся того, когда и как их следует использовать. Это, опять же, больше вопрос дизайна, чем реализации, как упомянуто выше. Зная о библиотеке анимации, дизайнеры будут более готовы к тому, чтобы видеть, где могут быть применены анимации и включать их в проект на ранней стадии дизайна приложения, что сделает ваши действия, как разработчика, более предсказуемыми.

Вы можете найти полное руководство в материалах "Анимация пользовательского интерфейса" (ссылка: http://msdn.microsoft.com/library/windows/apps/hh465165.aspx) и "Анимация пользовательского интерфейса со всплывающими элементами" (ссылка: http://msdn.microsoft.com/library/windows/apps/hh465259.aspx) в документации, где так же содержатся конкретные руководства по отдельным анимациям, представленным ниже. Здесь я лишь привожу их краткий обзор.

Важный вопрос. Встроенные элементы управления и другие элементы пользовательского интерфейса, наподобие тех, с которыми мы работали ранее, уже используют подходящие анимации. Например, вам не нужно анимировать нажатие на кнопку в элементе button, не нужно анимировать появление или исчезновение элементов наподобие `WinJS.UI.AppBar`. Вы, в основном, будете использовать анимации, реализуя пользовательский интерфейс непосредственно в HTML-макете, или создавая пользовательские элементы управления.

Таблица 7.1.

Название анимации	Методы WinJS.UI.Animation

Переход страницы (Page Transition)	`enterPage, exitPage`	А о п о с с с н с п п з п м с п И и е к у н и т п м с э
		А о п о о с и ч и

Переход содержимого (Content Transition)	`enterContent, exitContent`	ко н п ст за ко ст и вн ко со дч п от и ко сл ра м п эл п ан ра сн (е и (r
		И п сн п эл ум п и п

Появление и исчезновение (Fade In/Out)	`fadeIn, fadeOut`	п п к з з з э ж в а у с к а п
Плавный переход между элементами (Crossfade)	`crossFade`	С м р с э И с о с п ч о с з
Касание и отрыв		П н о у э п ш

указателя (Pointer Up/Down)	`pointerUp, pointerDown`	О в ч э у т и в э
Разворачивание и сворачивание (Expand/Collapse)	`createExpandAnimation, createCollapseAnimation`	Д с д п в н п м с о с п к н
Перемещение (Reposition)	`createRepositionAnimation`	И п э н
		И о с в э и т в о

Показ и скрытие всплывающих окон (Show/Hide Popup)	`showPopup, hidePopup`	дл ко за эл ко п п ов п (д од и П In н д ко ча п п вн и ан П со (С и и F ж н и п вс эл уп ка

		п н а
		И п э п и к п п п к А э и к п д к в н р А и п п д к в б р
Показ и скрытие элементов пользовательского интерфейса по		Э с и э

интерфейса по краям (Show/Hide Edge UI) Показ и скрытие панелей (Show/Hide Panel UI)	showEdgeUI, hideEdgeUI, showPanel, hidePanel	эл п и ко вы кр за кр В м и ан П (Н А П м эл (С об п п од сп В эл ум от кр п п п п п ан
		А

Считывание (Для плиток) (Peek (for tiles))	`createPeekAnimation`	о с п п о и т Г "Т л п W с и Н Ja в с м и ц о д п а и ж Н W
Обновление индикатора событий (Badge Update)	`updateBadge`	И о ч и с п
		И

Подсказка прокрутки (Swipe Hint)	`swipeReveal`	ка уд то п эл бы п
Выбор и отмена выбора прокруткой (Swipe Select/Deselect)	`swipeSelect, swipeDeselect`	А эл п вы от
Добавление и удаление элементов списка	`createAddToListAnimation, createDeleteFromListAnimation`	А до эл сп уд сп ка и эл уп L А до сп п су эл чт м эл до эл А уд

List)		сп эл сп п о эл И и эт д и ко д и вн со ко В и П со (С
Добавление и удаление элементов списка поиска (Add/Delete from Search List)	`createAddToSearchListAnimation`, `createDeleteFromSearchListAnimation`	Э п ко и ан со от бо вн и за сп ро п го

		п го и д
Начало и окончание перетаскивания или перетаскивание между началом и окончанием (Start/End Drag-Drop)	`dragSourceStart, dragSourceEnd, dragBetweenEnter, dragBetweenLeave`	Д о в п ка э п н А н п и п эл сл и вн А н о и п в п чт п п вн со п эл до эт

		э о р п ц (г та м о п у о

Если вы хотите увидеть, как работают эти анимации, вы можете найти их все в исходном коде WinJS, в файле iu.js. Просто найдите там методы и вы увидите, как они устроены. Анимация Плавный переход между элементами (Crossfade), например, анимирует прозрачность появляющегося элемента от 0 до 1 в течение 167 мс, с применением линейной функции, у исчезающего элемента прозрачность анимируется таким же образом, но от 1 до 0.

Анимация Касание указателя (Pointer Down) изменяет масштаб элемента со 100% до 97,5% в течение 167 мс, в соответствии с кубической кривой безье, в то время как анимация Отрыв указателя (Pointer Up) выполняет противоположные действия.

Знание этих характеристик, или метрик анимации (animation metrics) важно при создании веб-содержимого для интеграции с вашим приложением. Как отмечено выше, требования к сертификации приложений для Windows 8, в частности, раздел 4.5, показывают, что цели касания должны давать визуальную обратную связь. Так как вы не можете использовать WinJS в удаленных веб-страницах, знание того, как работают эти анимации поможет вам имитировать подобное поведение на таких страницах.

Внутри приложения, однако, вы должны получать метрики анимации программным образом, посредством API в `Windows.Core.UI.AnimationMetrics` (ссылка: http://msdn.microsoft.com/library/windows/apps/br241916.aspx), вместо того,

демонстрацию использования этого API в примере "Метрики анимации" (ссылка: http://code.msdn.microsoft.com/windowsapps/animation-metrics-sample-acb0220c).

Как ни интересны были бы подобные подробности, конечно, их всегда можно изменить. В конце концов, самое важное то, чтоб вы выбираете анимации не из-за визуальных эффектов, а основываясь на их смысловом значении, используя нужные анимации в нужное время и в нужном месте. Поэтому давайте посмотрим, как это сделать.

Совет №1. Все анимации WinJS реализованы с использованием функций `WinJS.UI.executeAnimation` (ссылка: http://msdn.microsoft.com/library/windows/apps/hh779762.aspx) и `WinJS.UI.executeTransition` (ссылка: http://msdn.microsoft.com/library/windows/apps/hh779763.aspx), которые вы так же можете использовать для пользовательских анимаций.

Совет №2. При исполнении анимаций всегда избегайте изменения содержимого элемента или его CSS-стилей, которые воздействуют на те же его свойства, что и анимации. Результаты непредсказуемы и ненадежны, и могут приводить к проблемам с производительностью.

Анимации в действии

Для того, чтобы увидеть все WinJS-анимации в действии, запустите пример "Библиотека анимации HTML" (ссылка: http://code.msdn.microsoft.com/windowsapps/Using-the-Animation-787f3720). В нем показано очень много разных анимаций, и этот пример, без сомнения, получает награду за самое большое количество сценариев: двадцать два! На самом деле, первое, что вам следует сделать, это перейти в Сценарий 22 и посмотреть, включены ли анимации, так как это, при прочих равных условиях, сильно подействует на то, что вы можете увидеть. В этом сценарии вы увидите, установлен ли флаг UISettings.animationsEnabled, и здесь вы можете увеличить или уменьшить счетчик активности анимации в WinJS. Поэтому взгляните на него прямо сейчас, так как если вы похожи на меня (мне не нравится ждать, пока моя панель задач выдвигается и задвигается), вы могли выключить системные анимации для того, чтобы вам удобнее было работать с

рабочим столом. Поначалу я не знал, что эта установка воздействует и на WinJS!

Очевидно, я не буду приводить здесь код каждого из 22 сценариев, на самом деле, делать это необязательно, так как многие из них работают сходным образом. Единственное реальное различие наблюдается между теми, методы которых начинаются с `create`, и теми, у которых это не так, что мы скоро увидим.

Все методы анимаций возвращают promise-объекгы, которые вы можете использовать для того, чтобы предпринять дополнительные действия, когда анимация завершена (здесь будет вызван обработчик завершения). Если вы уже знаете немного о CSS-переходах и анимациях, вы можете верно догадываться, что эти promise-объекты включают в себя события наподобие `transitionend` и `animationend`, поэтому вам не нжужно непосредственно прослушивать эти события, если вы хотите объединить анимации в цепочку или синхронизировать. Для объединения в цепочку, вы можете просто объединить в цепочку promise-вызовы, для целей синхронизации вы можете получить promise-объекты для нескольких анимаций и подождать их завершения, используя методы наподобие `WinJS.Promise.join` или `WinJS.Promise.any`.

Promise-объекты анимаций, кроме того, поддерживают метод cancel, который убирает анимацию элемента. Этот метод немедленно устанавливает значение свойства, на которое воздействует, в его конечное состояние, что приводит к немедленному видимому переходу элемента в это состояние. И отменили ли вы анимацию, или она сама окончилась, promise-объект решает, что операция успешно завершена. Отмена анимации, другими словами, вызывает обработчик завершения promise-объекта, а не его обработчик ошибки.

Помните, что так как все WinJS-анимации реализованы с помощью CSS, они не начинаются, пока вы не передадите управление обратно в поток пользовательского интерфейса. Это означает, что вы можете настроить несколько анимаций, зная, что они начнутся примерно в одно и то же время, как только вы вернетесь из функции. Поэтому, хотя методы анимации возвращают promise-объекты, они не похожи на другие асинхронные операции WinRT, которые начинают исполняться в другом

В любом случае, давайте взглянем на код! В самом простом случае все, что вам нужно сделать – это вызвать один из методов анимации и анимация будет исполнена, когда вам нужно. Сценарий 6 примера, например, просто добавляет следующие обработчики к событиям `MSPointerDown` и `MSPointerUp` трех разных элементов (js/pointerFeedback.js):

```
function onPointerDown(evt) { WinJS.UI.Animation.pointerDown(evt.srcElemen
}

function onPointerUp(evt) { WinJS.UI.Animation.pointerUp(evt.srcElement);
}
```

Нам обычно не нужно что-либо делать, когда анимация завершится, поэтому нам не нужно вызывать `done` или предоставлять завершающую функцию. На самом деле, использование многих из анимаций так же просто.

Анимация `crossFade`, в свою очередь (Сценарий 10), работает с двумя элементами: появляющимся и исчезающим (все они должны быть видимыми и являться частями DOM в процессе анимации, учтите!). Вызов анимации выглядит так: (js/crossfade.js):

```
WinJS.UI.Animation.crossFade(incoming, outgoing);
```

Но это еще не все. Обычная возможность анимаций заключается в том, что вы можете предоставить массив элементов, на которых должна быть исполнена одна и та же анимация, или, в случае с `crossfade`, два массива элементов. В то время, как это не особенно удобно для анимаций наподобие `pointerDown` и `pointerUp` (каждое событие указателя следует обрабатывать независимо), это, без сомнения, удобно для большинства остальных.

Рассмотрим анимацию `enterPage`. В своей единственной форме она принимает элемент для анимации и необязательное начальное смещение, где должен находиться элемент по отношению к своей конечной позиции. (Вообще говоря, вы вам следует опустить это смещение, если оно вам не нужно, так как это улучшает

конечной позиции. (Вообще говоря, вы вам следует опустить это смещение, если оно вам не нужно, так как это улучшает производительность – в примере туда передается `null`, что я опустил в нижеприведенном коде). Анимация `enterPage` так же может принимать коллекцию элементов, такую, как результат работы `querySelectorAll`. Сценарий 1 (html/enterPage.html и js/enterPage.js) предоставляют возможность выбора того, сколько элементов нужно анимировать раздельно:

```
switch (pageSections) {
case "1":
// Анимировать всю страницу
enterPage = WinJS.UI.Animation.enterPage(rootGrid);
break;
case "2":
// Анимировать заголовок и тело страницы
enterPage = WinJS.UI.Animation.enterPage([[header, featureLabel], [contentHo
break;
case "3":
// Анимировать заголовок, области ввода и вывода
enterPage = WinJS.UI.Animation.enterPage([[header, featureLabel], [inputLabel
break;
}
```

В то время, как аргумент элемента – это массив, аргумент смещения, если он задан, может быть либо отдельным смещением, которое применяется ко всем элементам, либо массивом, задающим отдельные смещения для каждого элемента. Каждое смещение – это объект, свойства которого определяют смещение. Смотрите js/dragBetween.js для Сценария 13, где этот подход используется с анимацией `dragBetweenEnter`:

```
WinJS.UI.Animation.dragBetweenEnter([box1, box2],
[{ top: "-40px", left: "0px" }, { top: "40px", left: "0px" }]);
```

Вот модификация, где одно и то же смещение применяется к обоим элементам:

```
WinJS.UI.Animation.dragBetweenEnter([box1, box2], { top: "0px", left: "40px" }
```

Сценарий 4 (js/transitioncontent.js) показывает, как вы можете объединить в цепочку два promise-объекта для выполнения перехода между двумя различными блоками содержимого:[1)]

```
WinJS.UI.Animation.exitContent(outgoing, null).done( function () {
outgoing.style.display = "none";
incoming.style.display = "block";
return WinJS.UI.Animation.enterContent(incoming, null);
});
```

Происходящее становится немного интереснее, когда мы посмотрим в методы анимации `create*`, которые касаются так называемой анимации макета (layout animation), для выполнения добавления и удаления элементов в списке, разворачивания и сворачивания содержимого и так далее. Каждый из них предусматривает трехступенчатую обработку, где вы создаете анимацию, выполняете манипуляции с DOM, и затем исполняете анимацию, как показано в Сценарии 7 (js/addAndDeleteFromList.js):

```
// Создать анимацию addToList.
var addToList = WinJS.UI.Animation.createAddToListAnimation(newItem, affec

// Вставить новый элемент в дерево DOM. Элементы, на которые это п
list.insertBefore(newItem, list.firstChild);

// Исполнить анимацию.
addToList.execute();
```

Причина для трехступенчатой обработки заключается в применении анимацию к только что добавленному элементу, или элементу, который убирают из списка, всем им нужно присутствовать в DOM во время исполнения анимации. Такой порядок работы позволяет вам создавать анимацию на основе начального состояния содержимого, манипулировать DOM (или просто устанавливать стили, и так далее) для создания конечного состояния и затем исполнять анимацию, позволив всему встать на свои места. Вы затем можете использовать метод done promise-объекта, возвращенного из execute для выполнения любой окончательной обработки. Сценарий 5 (js/expandAndCollapse.js) разьясняет это:

```
// Создание анимации сворачивания.
var collapseAnimation = WinJS.UI.Animation.createCollapseAnimation(element,

// Удаляем сворачивающийся элемент из содержимого документа, таким
// положение. Не удаляйте сворачиваемый элемент из DOM и не скрыв
// анимация этого элемента не будет выполнена.
element.style.position = "absolute";
element.style.opacity = "0";

// Исполнение анимации сворачивания.
collapseAnimation.execute().done(
// После завершения анимации (или если произошла ошибка), скрываем
function () { element.style.display = "none"; },
function () { element.style.display = "none"; }
);
```

В качестве последнего примера, так как я знаю, что вы достаточно сообразительны для того, чтобы самостоятельно посмотреть другие варианты, хочу показасть Сценарий 21 (js/customAnimation.js). Здесь показано, как использовать методы `WinJS.UI.executeAnimation` и `WinJS.UI.executeTransition`:

```
function runCustomShowAnimation() {
var showAnimation = WinJS.UI.executeAnimation(
target,
{
// Обратите внимание: этот ключевой кадр относится к ключевому кадр
// Если он не определен в CSS, анимация не будет работать.
keyframe: "custom-opacity-in",
property: "opacity",
delay: 0, duration: 500,
timing: "linear", from: 0,
to: 1
}
);
}
function runCustomShowTransition() {
var showTransition = WinJS.UI.executeTransition(
target,
```

```
{
property: "opacity", delay: 0,
duration: 500,
timing: "linear", to: 1
}
);
}
```

Если вы хотите объединить несколько анимаций (как делается во многих анимациях WinJS), отметим, что обе эти функции возвращают promise-объекты, в итоге, вы можете комбинировать несколько результатов с помощью `WinJS.Promise.join` и использовать один обработчик завершения, который выполняет пост-обработку. Это именно то, что происходит внутри WinJS.

И если вы уже знаете что-либо о CSS-анимациях и переходах, вы, возможно, можете сказать, что объекты, которые вы передаете `executeAnimation` и `executeTransition` – это просто краткие наименования стилей CSS, которые вы используете. Коротко говоря, эти методы дают вам простой способ создавать собственные пользовательские анимации и переходы, пользуясь возможностями CSS. Посмотрим сейчас непосредственно на эти возможности.

Врезка: параллактическая и панорамная анимации

Разработчики часто спрашивают, как создать параллактическую или панорамную анимацию фона, как на Начальном экране Windows. Если вы не знакомы с этой концепцией, перейдите на Начальный экран и прокрутите его, обращая внимание на то, что фон так же прокручивается, но медленнее, чем плитки. Это создает ощущение, будто плитки плавают над фоном.

Хотя в JavaScript можно реализовать подобный эффект (смотрите пример "KidBook" (ссылка: http://ie.microsoft.com/testdrive/HTML5/KidsBook/Default.html) на сайте Internet Explorer TestDrive), мы не рекомендуем этого делать, или, по крайней мере, рекомендуем предусмотреть параметр, позволяющий отключить его. Речь идет о том, что особенности работы с потоками и

перемещениям, за исключением работы подобной анимации на мощных устройствах. Этот эффект будет ярко выражен на маломощных компьютерах, особенно на ARM-устройствах. Кроме того, подобная анимация может потреблять слишком много ресурсов процессора и нерационально использовать батареи.

Это – один из случаев, где использование C++ и DirectX (или даже C#/VB и XAML) имеет явное преимущества перед JavaScript, и это нужно принять во внимание, если подобный эффект в вашем приложении абсолютно необходим.

1) Обратите внимание на то, что в примере переменная `output` передается в качестве первого параметра в `exitContent` и `enterContent`. код должен выглядеть как показано здесь, с outgoing, который передается в exitContent и `incoming`, который передается в `enterContent`.

CSS-анимации и переходы

Создание сбственных анимаций, пример независимой анимации.

Как упомянуто выше, многие нужды анимации могут быть реализованы посресдтсвом CSS, а не JavaScript-кода, исполняющего покадровую анимацию. Библиотека анимации WinJS, как мы уже видели, полностью построена на основе CSS. Использование CSS освобождает нас от необходимости писать большие объемы кода, который имеет дело с тем, как надо переместить каждый элемент в каждом кадре, основываясь на прошедшем времени и синхронизированно со скоростью обновления экрана. Вместо этого мы можем просто объявить то, что нам нужно (возможно, используя вспомогательные функции WinJS.UI.executeAnimation (ссылка: http://msdn.microsoft.com/library/windows/apps/hh779762.aspx) и WinJS.UI.executeTransition (ссылка: http://msdn.microsoft.com/library/windows/apps/hh779763.aspx) и предоставить хост-процессу приложения заботу о подробностях. Делегирование обязанностей в наилучшем виде! В этом разделе давайте внимательнее посмотрим на возможности CSS для приложений Магазина Windows.

Другое огромное преимущество выпонения анимаций и переходом посредством CSS – это то, что оно воздействует только на трансформацию и прозрачность элементов – это позволяет создавать то, что мы называем независимой анимацией, которая выполняется в потоке графического ядра вместо того, чтобы выполняться с использованием ресурсов центрального процессора. Зависимые анимации происходят, когда вы создаете их в JavaScript, используя временные интервалы, используем CSS-анимации и переходы со свойствами, отличными от трансформаций и изменения прозрачности, или выполняем анимацию на элементах, которые частично или полностью перекрыты другими элементами.

Мы вернемся к этой теме, когда посмотрим код примеров. Как я предполагаю, вы уже хотя бы немного знакомы с этим, поэтому сначала давайте посмотрим, как работают анимации и переходы в CSS. Я говорю об анимациях и переходах, так как это, на самом деле, две разные спецификации: CSS-анимации (ссылка:

http://www.w3.org/TR/css3-animations/) и CSS-переходы (ссылка: http://www.w3.org/TR/css3-transitions/). Итак, в чем же разница?

Обычно, когда меняются CSS-свойства, их значения резко изменяются со старого значения на новое, что приводит к мгновенному визуальному изменения. Переходы сообщают хост-процессу приложения, как изменять одно или несколько значений свойств постепенно, в соответствии с заданной задержкой и временными параметрами. Все это объявляется в специальном правиле стиля для элемента (как для псевдо-элементов `:before` и `:after`) в применении к индивидуальным стилям:

`transition-property` (`transitionProperty` в JavaScript). Идентифицирует CSS-свойства, на которые воздействует переход (свойства, подвергаемые действию переходов (transitionable), которые перечислены в Разделе 7 спецификаций переходов).

`transition-duration` (`transitionDuration` в JavaScript). Задает длительность перехода в секундах (дробные значения секунд поддерживаются, например, `.125s`, отрицательные значения приводятся к `0s`).

`transition-delay` (`transitionDelay` в JavaScript). Задает отложенное начало перехода, по отношению момента изменения свойства, в секундах. Если дано отрицательное значение, переход начнется раньше, но эффект не будет в это время виден.

`transition-timing-function` (`transitionTimingFunction` в JavaScript). Задает, как значение свойства будет меняться во времени, эти функции - `ease`, `linear`, `ease-in`, `ease-out`, `ease-in-out`, `cubic-bezier`, `step-start`, и `step-end`. В спецификациях W3C есть полезные диаграммы, которые поясняют это, но лучший способ увидеть разницу – попробовать все это на практике.

Например, переход для отдельного свойства выглядит так:

```
#div1 {
transition-property: left;
```

```
transition-duration: 2s;
transition-delay: .5s;
transition-timing-function: linear;

}
```

При определении перехода для нескольких свойств, каждое значение в каждом стиле отделяется запятой:

```
.class2 {
transition-property: opacity, left;
transition-duration: 1s, 0.25s;
 }
```

Опять же, переходы не задают начальные и конечные значения свойств – они лишь определяют, как происходят изменения, независимо от того, установлены ли новые свойства посредством другого правила CSS или в JavaScript Так, в первом случае выше, если `left` изначально имеет значение 100px и установлено в 200px посредством правила `:hover`, оно будет меняться по прошествии 0.5 секунд со 100px до 300px в течение 2 секунд. Можно вычислить, что визуальное перемещение с временной функцией `linear` происходит со скоростью 100 пикселей в секунду. Другие временные функции покажут различную скорость перемещения в разные моменты в течение 2-х секунд.

Если JavaScript-код потом установит значение в -200 – в идеале – после завершения первого перехода и вызовет его событие `thransitionend` – значение снова изменится за то же время, но теперь – с 300px до -200px (всего 500px). В результате элемент будет перемещаться быстрее (250 пикселей в секунду, снова с помощью временной функции `linear`), так как ему нужно пройти большее пространство при той же продолжительности перехода.

Для переходов так же справедливо то, что если вы присваиваете элементу стиль (например, `class2` выше), ничего не произойдет, до тех пор, пока свойство, на которое осуществляется воздействие, не изменит значение. Изменение стиля, наподобие этого, так же не возымеет действия, если исполняется анимация. Исключение – это если вы изменяете значение `transition-property`, в таком случае

переход прекратится. В связи с этим, важно отметить, что значение этого свойства по умолчанию `all`, и его очистка (установка в `""`) не останавливает переход, а включает его! Вместо этого вам нужно установить данное свойство в `none`.

Примечание. Элементы с `display: none` вовсе не исполняют CSS-анимаций и переходов, по очевидным причинам. То же самое нельзя сказать об элементах с `display: hidden`, `visibility: hidden`, `visibility: collapsed`, или `opacity: 0`, что значит, что элемены с настройками, отличающимися от `display: none` могут исполнять анимацию невидимых элементов, что является пустой тратой ресурсов. Коротко говоря, используйте `display: none`.

Анимации работают не так, как переходы. Анимации определяются раздельно от любого правила стиля CSS, а затем присоединяются к правилам. Присвоение стиля элементу приводит к немедленному запуску анимации. Более того, группы зависимых свойств объявляются вместе, в ключевых кадрах,(keyframes), и, таким образом, анимируются вместе.

CSS-анимация, другими словами, это инструкция по постепенному обновлению одного или нескольких значений свойств CSS за некоторое время. Значения меняются от начального элемента к конечному, проходя через различные промежуточные состояния, заданные в наборе ключевых кадров. Вот пример (из Сценария 1 примера "Независимые анимации HTML" (ссылка: http://code.msdn.microsoft.com/windowsapps/Independent-animations-app-c00b2962), на который мы будем ссылаться):

```
@keyframes move {

from { transform: translateX(0px); }
50% { transform: translateX(400px); }
to { transform: translateX(800px); }
}
```

В более общем виде:

- Описание начинается с `@keyframes <identifier>`, где `<identifier>` - это любое имя, которое вы хотите назначить ключевому кадру (наподобие имени move выше). Вы будете ссылаться на этот идентификатор в правилах стиля.
- Внутри описания ключевого кадра, вы создаете любое количество наборов правил (rule sets), каждое из которых представляет разное состояние анимируемого элемента в различные этапы общего процесса анимации, отмеченные в процентах. Ключевые слова `from` и `to`, как показано выше, просто соответствуют, соответственно, 0% и 100%.
- В каждом наборе правил вы затем задаете желаемое значение любого количества свойств стиля (в примере выше это лишь `transform`), разделяя их точкой с запятой, как в CSS-стилях. Если значение для свойства не отличается от значения, установленного предыдущим правилом, анимация для данного свойства выполняться не будет. Если значения различаются, подсистема визуализации анимирует изменение между двумя значениями этого свойства в отрезок времени, эквивалентный `<overall animation time> * (<toPercentage> <fromPercentage>)/100`. Временная функция так же может быть задана для каждого набора правил с использованием стиля `animation-timing-function`, например:

  ```
  50% { transform: translateX(400px); animation-timing-function: ease-in;}
  ```

  Вы можете здесь заметить что хотя ключевой кадр может предусматривать использование временной функции, он ничего не сообщает о реальных таймингах. Это оставлено для установки в конкретном правиле стиля, которое ссылается на ключевой кадр. В Сценарии 1 примера, например:

  ```
  .ball {
  animation-name: move;
  animation-duration: 2s;
  animation-timing-function: linear;
  animation-delay: 0s;
  animation-iteration-count: infinite;
  animation-play-state: running;
  }
  ```

- Здесь стиль `animation-name` (`animationName` в JavaScript) задает ключевой кадр для применения. Другие стили, , затем описывают то, как именно должен исполняться ключевой кадр:
- `animation-duration` (`animationDuration` в JavaScript). Длительность анимации в секундах (дробные значения секунд, конечно, разрешены). Отрицательные значения приравниваются `0s`.
- `animation-timing-function` (`animationTimingFunction` в JavaScript). Задает, как и в случае с переходами, как значение свойства будет интерполироваться во времени: — `ease` (по умолчанию), `linear`, `ease-in`, `ease-out`, `ease-in-out`, `cubic-bezier`, `step-start`, и `step-end`.
- `animation-delay` (`animationDelay` в JavaScript). Определяет количество секунд, после которого анимация начнется после применения стиля. Значение может быть отрицательным, как в случае с переходами, что означает начало анимации на полпути ее обычного цикла.
- `animation-iteration-count` (`animationIterationCount` в JavaScript). Показывает, сколько раз повториться анимация (значение по умолчанию – 1). Это может быть либо числом, либо значением `infinite`, как показано выше.
- `animation-direction` (`animationDirection` в JavaScript). Показывает, следует ли анимации воспроизводиться в режиме `normal` (в прямом направлении), `reverse`, `alternate` (туда и обратно), или `alternate-reverse` (туда и обратно, начиная с обратного направления). Значение по умолчанию – `normal`.
- `animation-play-state` (`animationPlayState` в JavaScript). Позволяет проигрывать или приостанавливать анимацию. Состояние по умолчанию –`running` – проигрывает анимацию. Установка в `paused` приостанавливает анимацию до тех пор, пока вы не установите стиль обратно в значение `running`.
- `animation-fill-mode` (`animationFillMode` in JavaScript). Задает, какое значение свойства упомянутого ключевого кадра будет применено, когда анимация не

исполняется, например, в течение первоначальной задержки или после завершения анимации. Значение по умолчанию `none` применяет значения для `0%` или `from` из набора правил, если направление анимации установлено в `forward` и для направления `alternate`; применяются значения для 100% или to из набора правил, если направление установлено в `reverse` или `alternate-reverse`. Режим заполнения `backwards` выполняет обратные действия. Режим заполнения `forwards` всегда применяет значения для `100%` или `to` (если только количество повторений не равно нулю, в таком случае это работает как `backwards`). Другие параметры, оба, это то же самое, что указание и `forwards` и `backwards`.

- `animation` (`animation` in JavaScript). Короткое имя стиля для всего вышеперечисленного (за исключением play-state) в порядке name (имя), duration (длительность), timing function (временная функция), delay (задержка), iteration count (число повторений), direction (направление), и fill mode (режим заполнения).

Применение стиля, который содержит `animation-name` запускает анимацию для элемента. Это может произойти автоматически, если анимация упомянута в стиле, который применен по умолчанию. Это так же может произойти по запросу, если стиль назначен элементу в JavaScript, или если вы зададите свойство элемента `animation`.

Ключевые кадры, которые обычно задаются в CSS, так же могут быть определены в JavaScript. Первый шаг заключается в построении строки, которая соответствует тому, что вы записали бы в CSS, затем эту строку вставляют в таблицу стилей. Это показано в Сценарии 7 примера независимых анимаций HTML (js/scenario7.js):

```
var styleSheet = document.styleSheets[1];
var element1 = document.getElementById("ballcontainer");
var animationString = '@keyframes bounce1 {'

// ...
+ '}';

styleSheet.insertRule(animationString, 0);
```

```
window.setImmediate(function () {
element1.style.animationName = 'bounce1';
});
```

Обратите внимание на то, как этот код использует `setImmediate` для выхода из потока пользовательского интерфейса перед установкой свойства `animationName`, которое запускает анимацию. Это гарантирует то, что другой код (здесь не показанный), будет исполнен в первую очередь, так как он выполняет некоторые другие действия, которые должны быть завершены прежде чем начнется анимация.

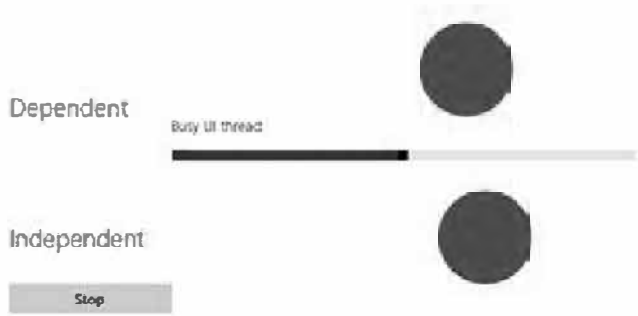
В целом, полезно снова напомнить, что CSS-анимации и переходы начинаются только тогда, когда вы возвращаетесь из функции, которая их задает. Таким образом, ничего не отобразится до тех пор, пока вы не вернетесь обратно в поток пользовательского интерфейса и подсистема визуализации снова примется за свою работу, так же, как происходит, когда вы изменяете свойства, не имеющие отношения к анимации. Это означает, что вы можете настроить столько анимаций и переходов, сколько нужно, и они все будут исполнены одновременно. Использование обратного вызова с setImmediate, как показано выше, это просто способ сообщить системе: "Выполни этот код как только в потоке пользовательского интерфейса не будет другой работы"[1]. Подобный шаблон обычно применим для запуска одной или нескольких анимаций, когда все остальное настр оено.

В качестве последнего замечания для этого раздела, хочу отметить, что вам может быть интересен материал "Руководство по CSS-анимации: принципы и примеры" (ссылка: http://coding.smashingmagazine.com/2011/09/14/the-guide-to-css-animation-principles-and-examples/) из Smashing Magazine. Из материала вы узнаете много всего об анимациях, помимо того, как задавать их в коде.

Пример независимой анимации

Обратимся к примеру "Независимые анимации HTML" (ссылка: http://code.msdn.microsoft.com/windowsapps/Independent-animations-app-c00b2962), в Сценарии 1 показана демонстрация сравнения

независимой и зависимой анимации при создании некоторой нагрузки на поток пользовательского интерфейса (то есть, блокировки этого потока) в соответствии с ползунком. В результате, верхний красный шар (смотрите изображение ниже) перемещается прерывисто, особенно, когда вы увеличиваете нагрузку на поток пользовательского интерфейса, перемещая ползунок. Зеленый шар внизу, с другой стороны, перемещается все время плавно.



Непросто для понимания то, что в этом примере оба шара используют одно и то же правило стиля CSS, которое называется ball, которое мы видели в предыдущем разделе. На самом деле, практически во всем эти элементы совпадают. Почему же красный шар перемещается прерывисто, когда поток пользовательского интерфейса занят дополнительной работой?

Секрет заключается в стиле `z-index: -1;` для красного шара в css/scenario1.css (и, соответственно, отсутствие `position: static`, что делает отрицательным `z-index`). Для независимо выполняемой анимации, элемент не должен перекрываться. Это, на самом деле, касается того, как создается макет в подсистеме визуализации HTML/CSS хост-процесса приложения, как элемент, находящийся где-то среди других, в соответствии с порядком взаимного расположения, может быть зависимым или независимым. Коротко говоря, стиль `z-index` является основным стилем, который может вам позволить сделать это.

Как я упоминал выше, независимые анимации ограничены воздействием на свойства элемента `transform` и `opacity`. Если вы анимируете любое свойство, которое воздействует на макет, например, `width` или `left`, анимация будет исполняться как зависимая (в любом

случае, похожих результатов можно достичь с трансформациями масштабирования и переноса). Другие факторы так же влияют на независимые анимации, как описано в материале "Анимации" (ссылка: http://msdn.microsoft.com/library/windows/apps/hh849087.aspx) в документации. Например, если в системе перегружен графический процессор, если вы нагружаете его слишком большим количеством независимых анимаций, или если элементы слишком велики, некоторые анимации превратятся в зависимые. Это – еще одна хорошая причина использовать анимации целенаправленно – чрезмерное их использование создает ужасный опыт взаимодействия на маломощных устройствах, таким образом, использование анимаций для повышения качества опыта взаимодействия пользователя и системы теряет смысл.

Другой сценарий в примере позволяет вам поэкспериментировать с CSS-переходами и анимациями, задавая значения в различных элементах управления и затем запуская анимацию. Сценарии 2 и 3 работают с CSS-переходами для 2D и 3D-трансформаций, соответственно, эффект, создаваемый последним сценарием, показан на рис. 8.1. Как вы можете видеть, элемент, который анимирован в примере – это контейнер для всех элементов управления для ввода данных! Сценарии 5 и 6 позволяют вам делать похожие вещи с CSS-анимациями. Во всех этих случаях, необходимые стили задаются непосредственно в JavaScript, вместо использования декларативного CSS, поэтому для того, чтобы узнать подробности обратитесь к .js-файлам вместо .css-файлов.

Рис. 8.1. Вывод данных в Сценарии 3 примера независимых анимации HTML

Сценарии 3 и 7, в свою очередь, показывают кое-что такое, о чем мы еще не говорили, несколько простых событий, которые вызываются для переходов и анимаций (независимые и зависимые анимации здесь не сравниваются). В последнем случае, любой элемент, для которого вы исполняете CSS-переход, вызывает события `transitionstart` и `transitionended`. Вы можете использовать эти события для объединения анимаций в цепочку.

В случае с анимациями, имеются три события: `animationstart` (которое происходит после истечения срока задержки), `animationend` (при завершении анимации), и `animationiteration` (в конце каждого цикла анимации, если только тогда же не вызывается `animationend`). Как и в случае с переходами, эти события можно использовать для объединения анимаций в цепочку или синхронизации. Событие animationiteration так же полезно, если вам нужно выполнить некоторый код каждый раз, когда завершается очередной цикл анимации. В подобном обработчике вы можете проверить условия, которые могут привести к остановке анимации, в подобном случае вы можете установить animationPlayState в состояние приостановки, если нужно.

Создание собственных анимаций: советы и секреты

Если вы чем-то похожи на меня, я могу думаю, что первым, что вы сделали, когда начали экспериментировать с анимациями JavaScript, были: настройка начальных условий, создание таймеров с помощью `setInterval`, выполнение каких-то вычислений в обработчиках и обновление элементов (или рисование в элементе canvas), и вы продолжали все это до тех пор, пока не получали то, что вам нужно. В конце концов, подобные вещи лежат в основе наших любимых развлечений! (В качестве введения, если только вы не попробовали еще это сами, посмотрите материал "Анимация графики полотна" (ссылка: http://msdn.microsoft.com/library/windows/apps/hh465053.aspx).

Таким образом, сообщество разработчиков накопило немало ценных

сведений по этим вопросам, если вы решили заняться этим. Я сказал так, потому что сейчас, посмотрев на библиотеку анимаций WinJS и на возможности CSS, вы вполне можете решить, стоит ли вам вообще этим заниматься. Некоторые люди подсчитали, что подавляющиее большинство анимаций, необходимых большинству приложений, могут быть полностью реализованы посредством CSS: просто задайте стиль и позвольте хост-процессу приложения сделать все остальное. Но если вы решили, что вам все еще нужна низкоуровневая анимация, вот первое, о чем вы должны спросить себя:"Каков подходящий интервал анимации?"

Это очень важный вопрос, так как часто разработчики не знают, какой интервал использовать для анимации. Это не столько вопрос о длинных интервалах, наподобие 500 мс или 1 с, но рабзрабочтики часто просто используют 10 мс, так как результат его применения кажется им "достаточно быстрым".

Честно говоря, 10 мс – это излишество по целому ряду причин. 60 кадров в секунду (frames per second, fps) соответствует интервалу анимации в 16,7 мс – это меньше, чем способен различить человек и лучше, чем способно отобразить множество дисплеев. На самом деле, лучшие результаты получаются, когда кадры вашей анимации синхронизированы с частотой обновления дисплея.

Рассмотрим это подробнее. Вы когда-нибудь смотрели на экран, когда ели что-нибудь хрустящее и замечали, что пиксели выглядят "пляшущими"? Это потому что дисплей – это не просто пассивное средство просмотра того, что находится в видеопамяти. Вместо этого дисплеи (даже LCD и LED-дисплеи) обычно просматривают видеопамять с заданной частотой обновления, которая обычно равняется 60 Гц, или 60 fps (но может равняться и 50 Гц, и 100 Гц).

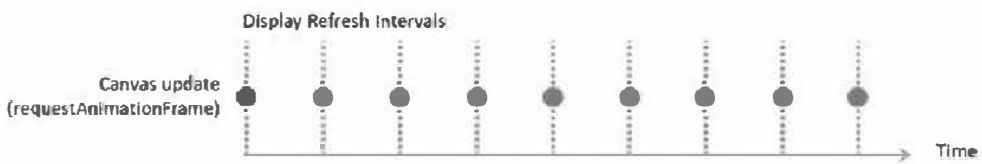
Это означает, что попытка вывода анимации с интервалом, меньшим, чем частота обновления дисплея – это пустая трата времени, пустая трата энергии (было показано, что это способно сократить время работы от батарей на 25%!), это приводит к выпадению кадров. Последнее заявление проиллюстрировано ниже, где красные точки – это кадры, которые были нарисованы на чем-то вроде элемента canvas, но никогда не появились на экране, так как другой кадр был нарисован до обновления экрана:

- Обновление элемента canvas с интервалом менее 10 мс (Canvas update (<10ms intervals))
- Интервалы обновления дисплея(Display Refresh Interfals)
- Время (Time)

Именно поэтому обычно анимацию выводят с интервалами, кратными 16,7 мс, испольуя setInterval. Однако, использование значения в 16,7 подразумевает наличие дисплея, имеющего частоту обновления в 60 Гц, а это не всегда так. Правильное решение, таким образом, и для приложений Магазина Windows на JavaScript, и для веб-приложений – использовать `requestAnimationFrame`. Это API просто принимает функцию, которую нужно вызывать для каждого кадра:

```
requestAnimationFrame(renderLoop);
```

Вы можете заметить, что здесь нет параметра интервала. Функция лишь дает вам способ синхронизировать обновление кадров с частотой обновления дисплея, таким образом, вы выводите данные только тогда, когда система готова что-то отобразить:



Более того, `requestAnimationFrames` принимает во внимание и видимость, что означает, что если элемент невидим (и анимация, таким образом, бесполезна), кадры отрисовываться не будут. Это означает, что вам не нужно самостоятельно обрабатывать события изменения видимости страницы для включения или выключения анимации: вы просто можете положиться на `requestAnimationFrames`.

Совет. Если вы по-настоящему хотите оптимизировать вывод данных,

постарайтесь выполнять все задачи рисования (а не только анимации) с помощью обратного вызова `requestAnimationFrames`. Таким образом, при обработке изменений, например, при реакции на событие ввода, обновляйте данные и вызывайте `requestAnimationFrames` с вашей функцией отрисовки вместо того, чтобы напрямую вызывать эту функцию. И всегда перерисовывайте что-либо только тогда, когда вам нужно сделать это, как мы скоро увидим, для того, чтобы наилучшим способом использовать процессор и батареи.

Полезно знать, что попытка анимации элемента canvas, который частично перекрыт элементом с display: `inline-block`, приведет к значительному падению производительности и большим промежуткам между кадрами из-за чрезмерного аннулирования областей. Использование различных моделей отображения, таких, как `table-cell`, помогает избежать этой проблемы.

Единственный вызов этого метода активирует обратный вызов для одного кадра. Для того, чтобы выполнять длительную анимацию, ваш обработчик должен вызвать `requestAnimationFrames` снова. Это показано в примере "Использование `requestAnimationFrames` для энергоэффективной анимации" (ссылка: http://code.msdn.microsoft.com/windowsapps/Using-requestAnimationFrame-924b039a) (этот пример выигрывает второе место в конкурсе на самое длинное имя!), который выводит и анимирует часы с секундной стрелкой:

Первый вызов `requestAnimationFrames` происходит в методе страницы `ready`, затем обратный вызов выполняет запрошенные изменения (js/scenario1.js):

```
window.requestAnimationFrame(renderLoopRAF);

function renderLoopRAF() {
drawClock();
window.requestAnimationFrame(renderLoopRAF);
}
```

Здесь функция `drawClock` принимает текущее время и вычисляет угол для рисования стрелок часов:

```
function drawClock() {
// ...

// Обратите внимание: пример модифицирован, Date создается лишь од
var date = new Date();
var hour = date.getHours();
var minute = date.getMinutes();
var second = date.getSeconds();

// ...

var sDegree = second / 60 * 360 - 180;
var mDegree = minute / 60 * 360 - 180;
var hDegree = ((hour + (minute / 60)) / 12) * 360 - 180;

// Код для использования методов контекста translate, rotate, и drawImage
// для рисования каждой стрелки
}
```

Вот вам вопрос: "Что не так с этим кодом?". Запустите пример и посмотрите на секундную стрелку. Затем подумайте о том, как `requestAnimationFrames` обеспечивает соответствие с циклами обновления экрана с интервалом 16,7 мс. Что не так с изображением?

Проблема здесь в том, что хотя секундная стрелка совершает видимые перемещения раз в секунду, код метода `drawClock` исполняется примерно 50, 60 или 100 раз в секунду – то есть гораздо чаще, чем нужно!. Таким образом, заголовок "Эффективные и плавные анимации" примера, означает все, что угодно, кроме того, что мы видим! На самом

деле, если вы запустите Диспетчер задач, вы можете видеть, что эти простые и "эффективные" часы, по иронии судьбы, потребляют 15-20% ресурсов процессора. Ну ничего ж себе!

Efficient Animations JS sample	20.2%	39.6 MB	0 MB/s	0 Mbps

Помните, что интервалы ориентированы на частоту обновления экрана в 16,7 мс (для 60 Гц-экрана), что подразумевает рендеринг 60 кадров в секунду. Если вам не нужно так много, вам следует пропускать кадры самостоятельно, ориентируясь на время, и экономить таким образом энергию, а не слепо перерисовывать экран, как сделано в этом примере. На самом деле, все, что нам нужно в подобных часах – это одно перемещение стрелки в секунду, постоянное вызывание `requestAnimationFrames` – это сущее излишество. Мы можем, вместо этого, использовать конструкцию `setInterval(function () { requestAnimationFrame(drawClock) }, 1000)` для того, чтобы скоординировать необходимый нам 1-секундный интервал с частотой обновления экрана. Если вы внесете это изменение, например, в метод `ready`, нагрузка на процессор значительно снизится:

Efficient Animations JS sample	0.5%	39.6 MB	0 MB/s	0 Mbps

Но допустим, мы, на самом деле хотим выполнять анимацию с частотой 60 Гц и потребление при этом 20% процессорного времени нас устраивает. В подобном случае нам следует, как минимум, сделать движения секундной стрелки более плавными, что можно сделать, просто использовав миллисекунды при вычислении угла:

```
var second = date.getSeconds() + date.getMilliseconds() / 1000;
```

Тем не менее, 20% мощности процессора – это слишком много на что-то столь простое, и 60 кадров в секунду, это все еще серьезное излишество. Возможно, что-то около 10 кадров в секунду нам вполне подойдет. В таком случае мы можем вычислять прошедшее время в `renderLoopRAF` и вызывать `drawClock` только тогда, когда прошла 0,1 секунды:

```
var lastTime = 0;
```

```
function renderLoopRAF() {
var fps = 10; // Целевое значение кадров в секунду
var interval = 1000 / fps;
var curTime = Math.floor(Date.now() / interval);

if (lastTime != curTime) {
lastTime = curTime;
drawClock();
}

requestAnimationFrame(renderLoopRAF);
}
```

Движения уже не такие плавные, вывод с частотой 10 кадров в секунду создает впечатление механических движений, но такой подход позволяет значительно снизить нагрузку на процессор:

Efficient Animations JS sample	5.2%	39.8 MB	0 MB/s	0 Mbps

Предлагаю вам поэкспериментировать со всем этим для того, чтобы увидеть, интервалы какой длины вы способны различить. 10 кадров в секунду и 15 дают ощущение механических движений, на 20 кадрах в секунду я не вижу особой разницы с 60-ю, а загрузка процессора состоавляет 7 – 10%. Вы можете так же попробовать что-то вроде 4 кадров в секунду (интервалы в четверть секунды) для того, чтобы увидеть их эффект. В дополнительных материалах к этой лекции я разместил вариант исходного примера, где вы можете выбирать из различных целевых частот рендеринга.

Еще в этом примере можно реализовать рисование часовой и минутной стрелки с использованием дробных углов. В исходном коде, минутрая стрелка перемещается сразу же, когда секундная выполняет один оборот. Многие аналоговые часы работают не так: их сложные механизмы перемещают часовую и минутрую стрелки понемного с каждой секундой. Для имитации такого поведения нам всего лишь нужно использовать вычисление секунд при вычислении минут, и использовать минуты при вычислении часов, например, так:

```
var second = date.getSeconds() + date.getMilliseconds() / 1000;
```

```
var minute = date.getMinutes() + second / 60;
var hour  = date.getHours() + minute / 60;
```

При реальном использовании, например, в игре, обычно избегают выполнения бесконечных циклов анимации, как здесь: если по экрану не перемещается ничего, что должно анимироваться (для чего вы можете использовать `setInterval` в качестве таймера), и нет событий ввода, на которые нужно реагировать, нет причин вызывать `requestAnimationFrames`. Кроме того, удостоверьтесь, что когда приложение "поставлено на паузу", вы прекращаете вызывать `requestAnimationFrames` до тех пор, пока, анимация не будет снова нужна (Так же вы можете испоьзовать `cancelAnimationFrames` для того, чтобы остановить уже запрошенную анимацию). То же самое справедливо и для `setTimeout` и `setInterval`: не делайте ненужные вызовы функции обратного вызова до тех пор, пока вам не нужно выполнять анимацию. Для этой цели используйте событие `visibilitychange` (ссылка: http://msdn.microsoft.com/library/windows/apps/hh441213.aspx), чтобы узнать, отображается ли ваше приложение на экране. Так как `requestAnimationFrames` принимает во внимание видимость (использование примером процессора падает до 0% перед приостановкой), вам нужно это сделать с помощью `setTimeout` и `setInterval`.

В итоге, главная цель здесь – по-настоящему понять, какой интервал анимации вам нужен (то есть, какая частота кадров) для того, чтобы наилучшим образом использовать `requestAnimationFrames`, если это нужно, или использовать `setInterval/setTimeOut`. И у того и у другого есть собственные варианты использования, которые позволяют предоставить хороший опыт взаимодействия пользователя и приложения при соответствующей нагрузке на систему.

Знаете ли вы? Одно из изменений в Windows 8 и Internen Explorer заключается в том, что `setTimeout`, `setInterval`, вместе с `setImmediate`, поддерживают включение параметров, которые вы можете передать в функцию обратного вызова?

Что мы только что изучили

- В панели управления на Рабочем столе, пользователи могут задать параметр, отключающий большинство анимаций, не являющихся существенными для работы системы. Приложениям следует принимать это во внимание, как это делает WinJS, проверяя свойство `Windows.UI.ViewManagement.UISettings.animatior`
- Библиотека анимаций WinJS содержит множество встроенных анимаций, которые воплощают индивидуальность Windows 8. Они рекомендованы для использования в соответствующих им сценариях приложения, в таких, как переходы содержимого и страниц, выделение, манипуляции со списками и в других.
- Все анимации WinJS построены на основе CSS и, таким образом, используют преимущества аппаратного ускорения. Когда соблюдены верные условия, такие анимации исполняются на GPU, и, таким образом, не подвержены влиянию деятельности в потоке пользовательского интерфейса.
- Приложения так же могут непосредственно использовать CSS-анимации и переходы в соответствии со спецификациями W3C.
- Помимо возможностей WinJS и CSS, приложения могут использовать функции наподобие `setInterval` и `requestAnimationFrame` для реализации непосредственной покадровой анимации. Метод `requestAnimationFrame` синхронизирует отрисовку кадров с частотой обновления экрана, что ведет к лучшей общей производительности.

1) Подробности об этом вы можете найти здесь: ссылка: https://dvcs.w3.org/hg/webperf/raw-file/tip/specs/setImmediate/Overview.html

Содержание

Титульная страница 2
Выходные данные 3
Лекция 1. Командный пользовательский интерфейс 4
Лекция 2. Состояния, параметры, файлы и документы 72
Лекция 3. Ввод данных и сенсоры 166
Лекция 4. Многоточечные жесты 204
Лекция 5. Мультимедиа 242
Лекция 6. Списки воспроизведения 290
Лекция 7. Целенаправленная анимация 350
Лекция 8. CSS-анимации и переходы 377