Onlinevarsity
your e-way to learning

Windows Store Apps Development – I

Are you □□□□□□□□□□ with Onlinevarsity.com?	R G E T R E E S I D
Did you □□□□□□□□ this ebook?	D O O W L N A D
Do you have a □□□□□ copy of this ebook?	L G A E L
Are you a victim of □□□□□□?	P C I A R Y

Answers: REGISTERED, DOWNLOAD, LEGAL, PIRACY

Download a **LEGAL** copy of this ebook which you can say is **mine**

because **PIRACY** is a **crime**

Windows Store Apps Development – I

APTECH LIMITED
Contact E-mail: ov-support@onlinevarsity.com

Edition 1 - 2014

Dear Learner,

We congratulate you on your decision to pursue an Aptech Worldwide course.

Aptech Ltd. designs its courses using a sound instructional design model – from conceptualization to execution, incorporating the following key aspects:

- Scanning the user system and needs assessment

Needs assessment is carried out to find the educational and training needs of the learner

Technology trends are regularly scanned and tracked by core teams at Aptech Ltd. TAG* analyzes these on a monthly basis to understand the emerging technology training needs for the Industry.

An annual Industry Recruitment Profile Survey is conducted during August - October to understand the technologies that Industries would be adapting in the next 2 to 3 years. An analysis of these trends & recruitment needs is then carried out to understand the skill requirements for different roles & career opportunities.

The skill requirements are then mapped with the learner profile (user system) to derive the Learning objectives for the different roles.

- Needs analysis and design of curriculum

The Learning objectives are then analyzed and translated into learning tasks. Each learning task or activity is analyzed in terms of knowledge, skills and attitudes that are required to perform that task. Teachers and domain experts do this jointly. These are then grouped in clusters to form the subjects to be covered by the curriculum.

In addition, the society, the teachers, and the industry expect certain knowledge and skills that are related to abilities such as *learning-to-learn, thinking, adaptability, problem solving, positive attitude etc*. These competencies would cover both cognitive and affective domains.

A precedence diagram for the subjects is drawn where the prerequisites for each subject are graphically illustrated. The number of levels in this diagram is determined by the duration of the course in terms of number of semesters etc. Using the precedence diagram and the time duration for each subject, the curriculum is organized.

- Design & development of instructional materials

The content outlines are developed by including additional topics that are required for the completion of the domain and for the logical development of the competencies identified. Evaluation strategy and scheme is developed for the subject. The topics are arranged/organized in a meaningful sequence.

The detailed instructional material – Training aids, Learner material, reference material, project guidelines, etc.- are then developed. Rigorous quality checks are conducted at every stage.

- Strategies for delivery of instruction

Careful consideration is given for the integral development of abilities like thinking, problem solving, learning-to-learn etc. by selecting appropriate instructional strategies (training methodology), instructional activities and instructional materials.

The area of IT is fast changing and nebulous. Hence considerable flexibility is provided in the instructional process by specially including creative activities with group interaction between the students and the trainer. The positive aspects of Web based learning –acquiring information, organizing information and acting on the basis of insufficient information are some of the aspects, which are incorporated, in the instructional process.

- Assessment of learning

The learning is assessed through different modes – tests, assignments & projects. The assessment system is designed to evaluate the level of knowledge & skills as defined by the learning objectives.

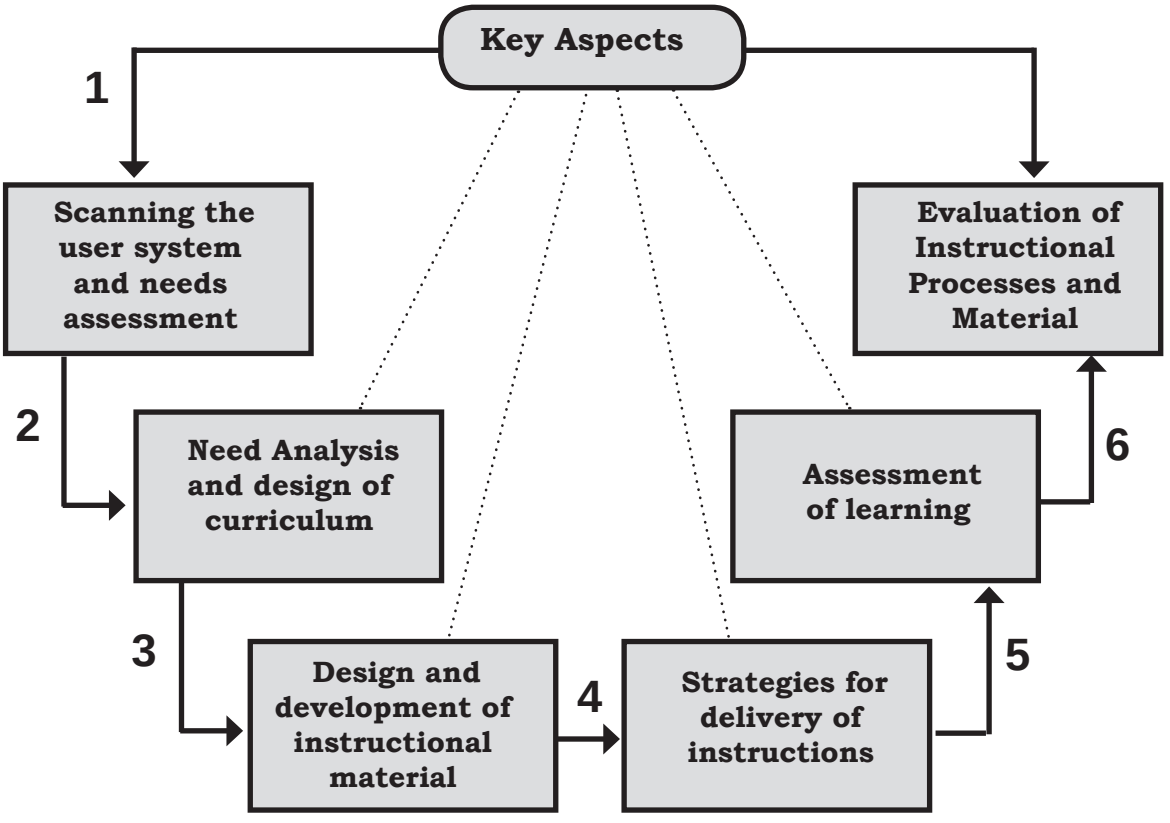
- Evaluation of instructional process and instructional materials

The instructional process is backed by an elaborate monitoring system to evaluate - on-time delivery, understanding of a subject module, ability of the instructor to impart learning. As an integral part of this process, we request you to kindly send us your feedback in the reply pre-paid form appended at the end of each module.

*TAG – Technology & Academics Group comprises of members from Aptech Ltd., professors from reputed Academic Institutions, Senior Managers from Industry, Technical gurus from Software Majors & representatives from regulatory organizations/forums.

Technology heads of Aptech Ltd. meet on a monthly basis to share and evaluate the technology trends. The group interfaces with the representatives of the TAG thrice a year to review and validate the technology and academic directions and endeavors of Aptech Ltd.

Aptech New Products Design Model



Onlinevarsity
your e-way to learning

To enhance your knowledge,

visit the **REFERENCES** *page*

@

Preface

Windows Store Apps also known as Metro-styled Apps is a mobile app that runs on Smartphones, tablet, computers, and other mobile devices running Windows 8 or Windows 8.1. Windows Store App is a new application that runs on Windows 8 devices. Users can create their own apps using programming language such as C#. Apps have one Window that supports multiple views that interact with each other.

The book, Windows Store Apps Development – I begins with the brief description of the Windows 8 Platform and Windows 8 User Interface principles. This book introduces the Windows 8 platform and features and basics of Windows App interface. It also allows to create the user interface layout and structure by using XAML. It also covers the XAML basics, data presentation controls, WinRT controls, and App Bars.

This book is the result of a concentrated effort of the Design Team, which is continuously striving to bring you the best and the latest in Information Technology. The process of design has been a part of the ISO 9001 certification for Aptech-IT Division, Education Support Services. As part of Aptech's quality drive, this team does intensive research and curriculum enrichment to keep it in line with industry trends.

We will be glad to receive your suggestions.

Design Team

Onlinevarsity

your e-way to learning

GROWTH

RESEARCH

OBSERVATION

UPDATES

PARTICIPATION

www.onlinevarsity.com

Sessions

Onlinevarsity

your e-way to learning

Visit the

Frequently Asked Questions

section @

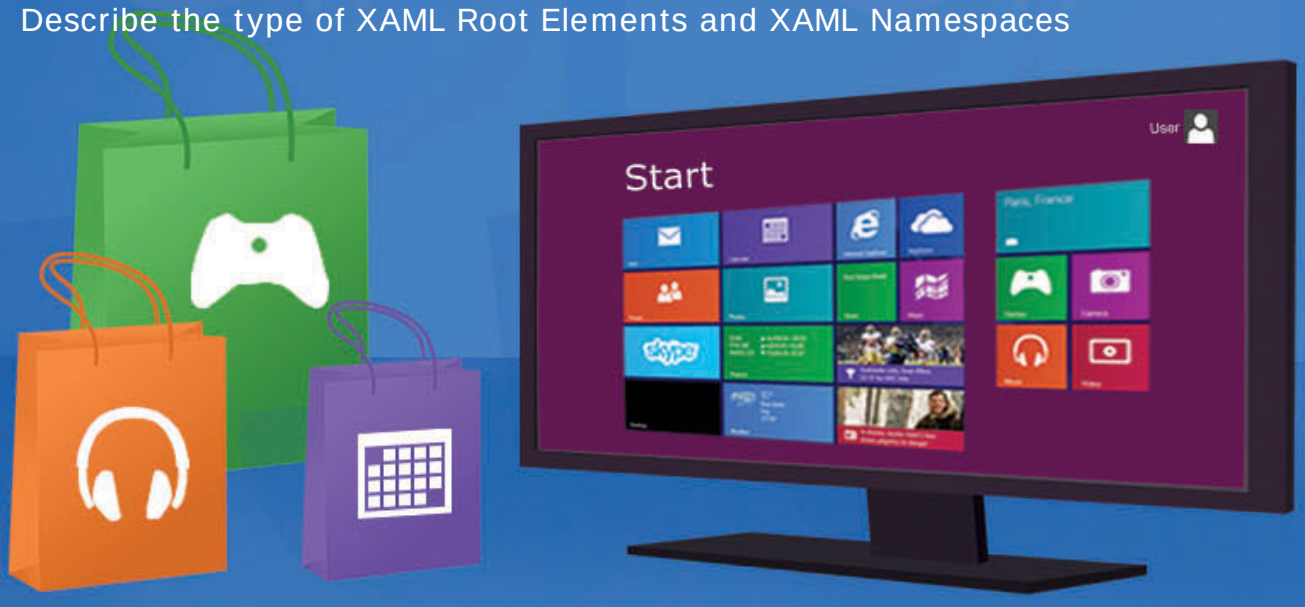
Session - 1
Getting Started with Windows App Architecture

Welcome to the Session, **Getting Started with Windows App Architecture**.

The session will provide a information about the Windows 8 Platform and Windows Store Apps. It will also introduce about basics of XMAL.

In this Session, you will learn to:

- Write the basic information about Windows 8 Platform and Windows Store Apps
- Describe the features of App models
- Write the basics of XML
- List the advantages of XAML over code
- List the properties of XML content
- Describe the Markup extensions
- Convert one type of Object or primitive values to other type
- Describe the type of XAML Root Elements and XAML Namespaces



1.1 Overview

Many options are there in Windows 8 operating system to program Windows Store Apps. It can be written in C++, C#, or Web technologies such as HTML5, JavaScript, and CSS3 using Visual Studio 2012. From the Visual Studio IDE, it is distributed to the Windows Store.

Some of the programming languages which can be used to create Windows Store Apps are as follows:

- C++/CX and Microsoft DirectX
- C# and Extensible Application Markup Language (XAML)
- Visual C++ component extensions (C++/CX) and XAML
- Microsoft Visual Basic and XAML
- JavaScript and HTML5

Depending on developer's suitability, programming languages can be selected for app development.

1.2 App Model Architecture

As data model consists of database design, an app model is meant for application design. It is a set of design patterns and files that is used to implement architecture. The app model consists of corresponding programming language and it is characterized by features such as app's entry point, file layout, and the presentation technology. Figure 1.1 displays the App Model Architecture.

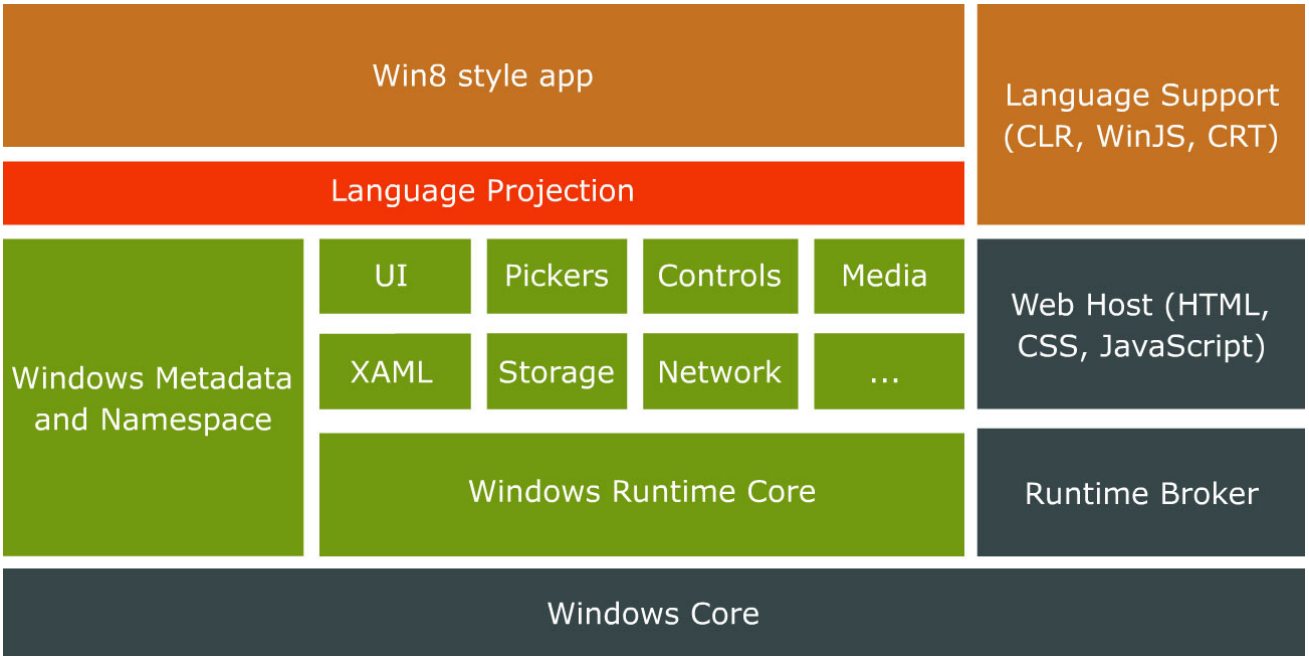


Figure 1.1: App Model Architecture

Concepts

1.3 Entry Points

Based on the programming language, the system calls app's code to start running. This function or method of Windows Store Apps is called **Entry point**.

Figure 1.2 displays the project templates.

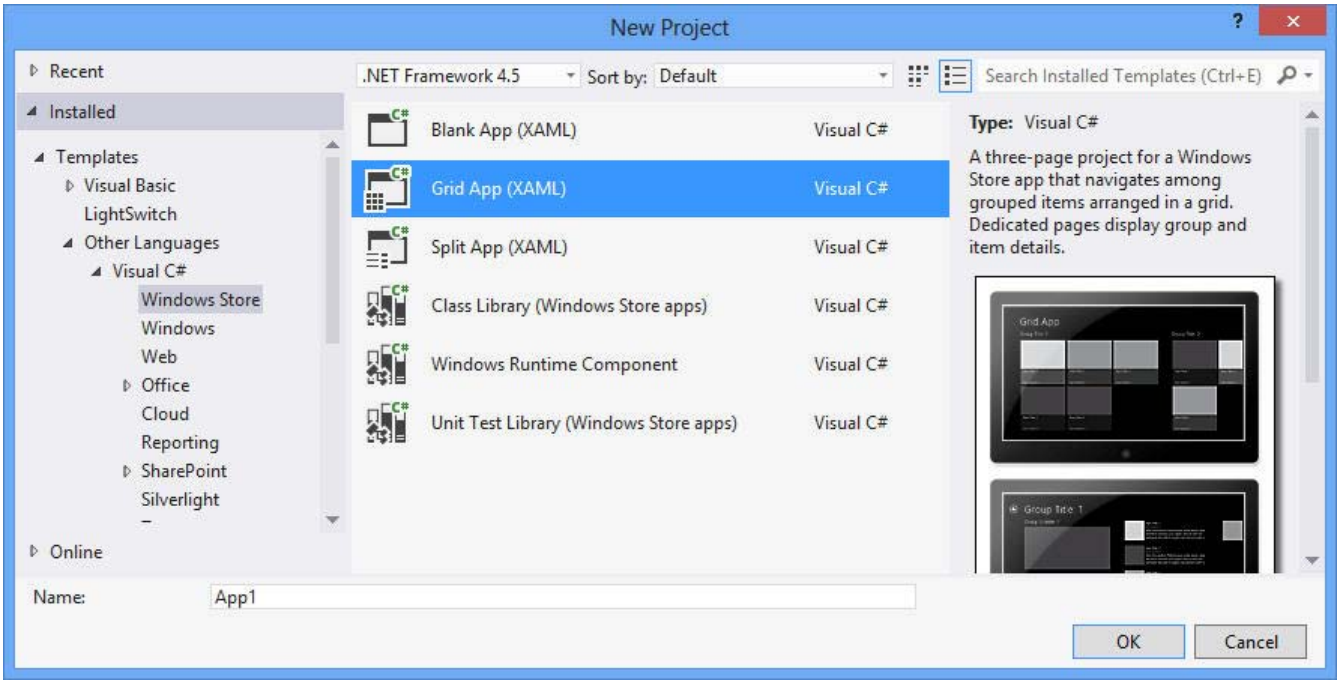


Figure 1.2: Project Templates

Table 1.1 lists the Visual Studio project templates.

App Model	Entry Point	File Location
JavaScript	function `onactivated`	`default.js in js folder`
Visual Basic/ C#	method `OnLaunched`	`App. xaml.vb /App.xaml. cs`
C++/CX with XAML	`OnLaunched` method	`App.xaml.cpp`
C++/CX with Microsoft Direct3D	`main` function	`Direct3DApp1.cpp`

Table 1.1: Project Templates

1.4 Presentation Technologies and Assets

The look and feel of the app can be defined by the app model's presentation technology. XAML, HTML5, and DirectX are the three distinct technologies used to create Windows Store Apps.

Table 1.2 shows the programming languages that can be used with specific presentation technologies.

Programming Language	Presentation Technology
C++/CX	XAML, DirectX, and XAML/DirectX interop
C#/Visual Basic	XAML
JavaScript	HTML5

Table 1.2: Programming Languages

Required assets for Windows Store Apps are four images, named Logo.png, SmallLogo.png, SplashScreen.png, and StoreLogo.png. Table 1.3 lists the App model.

App Model	Location of Assets
C++/CX with Direct3D	Assets folder
C++/CX with XAML	Assets folder
C#/Visual Basic	Assets folder
JavaScript	Images folder

Table 1.3: App Model

1.5 Asynchronous Programming

Asynchronous programming helps apps to stay responsive when work takes time to load an extra amount of time.

Table 1.4 lists the Asynchronous Programming.

App Model	Asynchronous
C#/Visual Basic	async/await keywords
C++/CX	Task class / IAsyncOperation
JavaScript	Common JS Promises/A

Table 1.4: Asynchronous Programming

1.6 Creating Windows Store Components

The code can be factored into reusable components for Windows Store Apps, which are called Windows Runtime Components. The specific interface requirements are there to which the Windows Runtime Components must stick to. It can be created by Windows Runtime Components using Visual Basic, C#, or C++/CX app models.

1.7 Packaging and Deployment

Using the Manifest Designer in Visual Studio Apps Package can be deployed. Using simple interface settings can be changed in the app manifest file. The package build process changes the .appxmanifest file into the AppxManifest.xml file that is included in the package contents.

1.8 Uploading App to the Windows Store

Windows Store components use all languages and components can be created in any programming language, except JavaScript.

1.9 Introduction to XAML

XAML is a declarative markup language and these files have .xaml extension. In .NET Framework programming model, XAML simplifies by making a UI for a .NET Framework application. The UI definition can be separated from the runtime logic using backend coding. XAML provides direct representation of objects in a set of backing types defined in assemblies. Using potentially different tools, it enables a workflow where anybody could work upon the UI and the application logic.

1.10 Advantages of XAML

Following are the advantages of XAML:

- XAML code is clear to read and short.
- There is a separation between designer code and logic.
- Most of the graphical design tools such as Expression Blend and many more require XAML as source.
- Provides different roles for developer and designer.

1.11 XAML vs. Code

Following is an example in which `StackPanel` is constructed with a `button` and a `TextBlock` in XAML and compare it with the C# code:

Example:

```
<StackPanel>
<TextBlock Margin="30">Hello world</TextBlock>
<Button Margin="20" HorizontalAlignment="Center">OK</Button></StackPanel>
```

Concepts

The same when expressed in C# will look similar to this.

The XAML code is as follows:

Example:

```
<Rectangle>
 <Rectangle.Fill>
  <SolidColorBrush Color="Green" />
 </Rectangle.Fill>
</Rectangle>
```

In this example you can observe that XAML code is much shorter and clearer to read.

1.12 XAML Syntax in Brief

Now, you will learn the basic forms of XAML syntax, using a short markup example. These sections will not provide complete information about each syntax form. Next few sections give basic information about XAML. The concept defined by XAML works only within the markup form and XML language.

➔ **Elements of XAML Object**

An instance of a type is typically declared by an `Object` element and this type is defined in the assemblies. The syntax of an `Object` element starts with an opening angle bracket (<) followed by the name of the type where an instance can be created. Later, attributes can be optionally declared on the `Object` element. Use a closing angle bracket (>) to complete the Object element tag. Even a self-closing form can be used to complete an element tag, which is a forward slash and closing angle bracket together (/>). Following is the example used to explain XAML Object elements:

Example:

```
<StackPanel>
<Button Content="Click Me"/>
</StackPanel>
```

The example specifies two `Object` elements:

1. `<StackPanel>` (content and a closing tag)
2. `<Button .../>` (self-closing form, some attributes)

An instruction is created for XAML, when a `Object` element tag is specified, which in turn processed to create a new instance. By calling the default constructor, each instance can be created.

➔ **Attribute Syntax (Properties)**

Properties of an `Object` itself are a attributes of the `Object` element. The attribute syntax property is set by the assignment operator (=) and as a string the value of an attribute is specified quotation marks.

In the following example, markup creates a button:

```
<Button Background="Blue" Foreground="Red" Content="This is a button"/>
```

In this example, the markup creates a button that has a blue background and red text.

➔ **Property Element Syntax**

There is a lack of property value in some properties of an `Object` element, attribute syntax is not possible. For such situations, property element syntax can be used.

The property element start tag syntax is `<typeName.propertyName>`. The content of the tag is an element of an `Object` of the type in which property takes as its value. After content is specified, the property element must be closed with an end tag. The end tag syntax is as follows:

`</typeName.propertyName>`.

Use of attribute syntax is more useful and this enables a more compact markup.

As in the previous attribute syntax example, the following example sets the properties:

Example:

```
<Button>
<Button.Background>
<SolidColorBrush Color="Blue"/>
</Button.Background>
<Button.Foreground><SolidColorBrush Color="Red"/>
</Button.Foreground>
<Button.Content>This is a button
</Button.Content>
</Button>
```

➔ **Collection Syntax**

The XAML language consists of few optimizations which produce more human readable markup. Property's value of the child elements become part of the collection when particular property takes a collection type. The value the collection property is set by the collection of child `Object` elements.

Concepts

The following example shows collection syntax for setting values of the `GradientStops` property:

Example:

```
<LinearGradientBrush>
<LinearGradientBrush.GradientStops>
<!--no explicit new GradientStopCollection, parser knows how to find or create
-->
<GradientStop Offset="0.0" Color="Red" />
<GradientStop Offset="1.0" Color="Blue" />
</LinearGradientBrush.GradientStops>
</LinearGradientBrush>
```

1.13 XAML Content Properties

As XAML specifies a language feature same as that a class indicates XAML content property which is a child element of that `Object` element. That means property element cannot be omitted when setting that property in XAML markup.

In the following example, the `Border` specifies a content property of `Child`. Advantage of the content property syntax is taken by first one and omits the `Border.Child` property element, and `Border.Child` is shown by second one.

Example:

```
<Border>
<TextBox Width="300"/>
</Border>
<!--explicit equivalent-->
<Border>
<Border.Child>
<TextBox Width="300"/>
</Border.Child>
</Border>
```

In the beginning itself on the `Object` element, the value of a XAML content property to be given entirely, after any other property elements.

In the following example markup does not compile:

Example:

```
<Button>I am a
<Button.Background>Blue</Button.Background>
blue button</Button>
```

➔ **Text Content**

To enable XAML elements to process directly text as their content, one must be true among the following cases:

The content property declared by class should be an assignable type to a string or a object. For example, content property is used as `Content` for any `ContentControl` and it is of type `Object`. This `Object` supports the following usage such as a Button:

`<Button>Hello</Button>`.

In the following example to initialize a text the text content is used as type converter:

`Brush>Blue</Brush>`.

Observe the following example:

Example:

```
<StackPanel>
<Button>First Button</Button>
<Button>Second Button</Button>
</StackPanel>
```

In this example, `StackPanel` is streamlined and each Button is a child element, `StackPanel` markup omits two tags for two different reasons which are given here.

1. Panel is derived by `StackPanel`. `Panel.Children` is defined by Panel as its XAML content property.
2. The `IList` implemented by the `Panel.Children` property which takes the type `UIElementCollection`. Based on the XAML rules for processing collections such as `IList`, the collection's element tag can be omitted.

Example:

```
<StackPanel>
<StackPanel.Children>
<!--<UIElementCollection>-->
<Button>First Button</Button>
<StackPanel>
<StackPanel.Children>
<!--<UIElementCollection>-->
<Button>First Button</Button>
<Button>Second Button</Button>
<!--</UIElementCollection>-->
</StackPanel.Children>
</StackPanel>
```

➔ **Attribute Syntax (Events)**

Name of the event is the attribute name and its syntax can be used for members more than properties. In XAML, the WPF implementation of events is the attribute's value which is the name of a handler. In the following example, markup is assigned to a Button by a handler for the `Click` event:

Example:

```
<StackPanel Grid.Row="1" Margin="120,30,0,0">
    <TextBlock Text="Attribute syntax - events" />
    <StackPanel Orientation="Horizontal" Margin="0,20,0,20">
     <TextBox x:Name="myTextBox" Width="300" HorizontalAlignment="Left"
/>
     <Button Content="hello world" Click="Button_Click"/>
    </StackPanel>
   </StackPanel>
```

The attribute values for every XAML syntax also has an event attribute. In the event attribute, you supply the function name for an event handler function which is probably written in C#, VB, or C++.

Concepts

➔ **Case and Whitespace in XAML**

Following are some reasons for the sensitivity of XAML:

1. Since CLR is case sensitive, by the same rules XAML is also case sensitive.
2. Primitives and keywords present in XAML language are also case sensitive.
3. Values present in the XAML are not always case sensitive. It is depend on the type converter behavior.

WPF XAML will normalize any significant whitespace and serializers and processors will ignore all unimportant whitespace. This happens only when strings are specified within XAML content properties. That means XAML tab characters, linefeed, and converts space into spaces, and then, reserves one place at the end of a contiguous string.

1.14 Markup Extensions

XAML language concept are used to provide the value of attribute syntax and it is used to extend markup. The curly braces ({ and }) indicate a markup extension and it directs the XAML processing.

Binding is the common markup extensions used in WPF application programming. It is used for data binding expressions. Attribute syntax can be used to provide values for properties by using markup extensions. Intermediate expression types are used by markup extensions to enable deferring values or referencing other objects.

In the following example, using attribute syntax markup sets the value of the `Style` property. The attribute references a particular markup extension, `StaticResource` and the `Style` property takes an instance of the `Style` class. It returns a reference to a style, when that markup extension is processed.

Example:

```
<Page.Resources>
<SolidColorBrush x:Key="MyBrush" Color="Gold"/>
<Style TargetType="Border" x:Key="PageBackground">
<Setter Property="Background" Value="Blue"/>
</Style>
</Page.Resources>
<StackPanel>
<Border Style="{StaticResource PageBackground}">
...
</Border>
</StackPanel>
```

For more reference listing of markup extensions for XAML implemented in WPF, refer WPF XAML Extensions and refer listing of the markup extensions that are defined by `System.Xaml`. For .NET Framework XAML implementations, refer XAML Namespace (x:) Language Features.

1.15 Type Converters

As stated in the Attribute Syntax section, string sets value of attribute. Based on the `String` type, these are converted into other `Object` types or primitive values. Many WPF types extend the basic string attribute working behavior which specifies strings and attributes.

Thickness structure implies measurements in a rectangle within one another. It is an example of a type that has a type conversion enabled for XAML usages which is used as the value for properties such as Margin. In the following example, type conversion and attribute syntax are used to provide a value for a Margin:

```
<Button Margin="10, 20, 10, 30" Content="Click me"/>
```

The previous example (attribute syntax) is equivalent to the following example which is containing lengthy syntax, where the `Margin` is set through property element syntax. The `Thickness` is set as attributes on the new instance through the four key properties.

Example:

```
<Button Content="Click me">
<Button.Margin>
<Thickness Left="10" Top="20" Right="10" Bottom="30"/>
</Button.Margin>
</Button>
```

1.16 XAML Root Elements and XAML Namespaces

To develop a well-formed XML file, a XAML file must have only one base element. For typical WPF scenarios, a root element can be used in the WPF application model.

(Some of the examples are `ResourceDictionary` for an external dictionary, Application for the application definition, Window or Page for a page).

In the following example, the root element shows a typical XAML file:

Example:

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

The root element containing the attributes, `xmlns` and `xmlns:x` indicates to a XAML processor. For backing types the XAML namespaces contain the type definitions. The `xmlns` attribute specifically indicates the default XAML namespace. Without a prefix `Object` elements markup can be specified. For WPF sections of the SDK and WPF application scenarios, the default XAML namespace is mapped to the WPF namespace `http://schemas.microsoft.com/winfx/2006/xaml/presentation`. The `xmlns:x` attribute indicates an additional XAML namespace and this maps to the XAML language namespace `http://schemas. microsoft.com/winfx/2006/xaml`.

1.17 Check Your Progress

1. Function or a method which starts running based on the programming language is called _____.

(A)	Keyword	(C)	App model
(B)	Entry point	(D)	Windows Store Apps

2. Name the presentation technology that can be used with JavaScript.

(A)	C++	(C)	JavaScript
(B)	C#	(D)	Visual Basic

3. What can be used to deploy apps Package?

(A)	C#	(C)	JavaScript
(B)	C++	(D)	Visual Studio

4. To which type of language XAML belongs to?

(A)	Declarative markup language	(C)	Coding language
(B)	Programming language	(D)	Backend coding language

5. Which operator sets the attribute syntax property?

(A)	Assignment operator (=)	(C)	Slash operator (\)
(B)	Addition operator (+)	(D)	And operator (&)

1.17.1 Answers

1.	B
2.	C
3.	D
4.	A
5.	A

- As data model consists of database design, an app model is meant for application design.
- Function or a method which starts running based on the programming language is called Entry point.
- The look and feel of the app can be defined by the app model's presentation technology.
- The code can be factored into reusable components for Windows Store Apps, which are called Windows Runtime Components.
- Using simple interface, settings can be changed in the app manifest file (Package.appxmanifest).
- XAML is a declarative markup language and these files have .xaml extension.
- An instance of a type is typically declared by an Object element and this type is defined in the assemblies.
- Properties of an Object itself are an attributes of the Object element.

Session - 2

Creating a UI for Windows Store App

Welcome to the Session, **Creating a UI for Windows Store App.**

The session will discuss how to choose the right UI surfaces for your Windows Store app, how to layout your app, and how to add controls and content for you app's UI

In this Session, you will learn to:

- Explain how to choose the right UI surfaces for your Windows Store App
- List the design principles
- Apply commands and touch interaction patterns in Windows Store App
- List the guidelines to develop advertising and branding patterns
- Apply UX guidelines in Windows Store App
- Develop Layout in Windows store app
- Add controls and content for your app's UI

2.1 User Experience (UX) Design Patterns for Windows Store Apps

Microsoft develops, manufactures, licenses, and supports a wide range of products and services related to computing. Three years ago a change has happened through infectious, exciting ideas which transformed from a flat design to a set of design principles.

2.1.1 Modern Design

The design language is based on following three distinct pillars which are based on design principles:

- **Motion design**: Motion helps to explain how a thing is accomplished in creating a stage for innovativeness, manifestation, and generativeness in products.

- **The Bauhaus**: This design is also known as Modern Design Movement. Earlier in 1919, the Bauhaus design school had developed a powerful design philosophy. Its focus is on making the core function simply beautiful. Bauhaus philosophy clears away excessive additions to focus on the essence of the function.

- **Swiss Style**: This style inspires Microsoft design keeping its central themes such as purity, legibility, and neutrality. From this style we have taken a principled and systematic way and the use of asymmetric layouts.

2.2 Modern Design Principles

Microsoft has its own set of principles, which guides it in every design venture that is commenced. These principles will be applied everywhere in one way or the other in developer's designs. Following are the Microsoft design principles:

- **Authentically digital**

  Being authentically digital is about creating new and distinctive things by crossing all the limits of physical world. Users are familiar with technology and digital interactions and are ready for a more powerful and scalable design language.

- **Quick and rapid**

  This principle guides the animations, transitions, and pleasant responses that may not be accepted but are useful to users. This principle focuses more on people rather than technology. A great design can be achieved by focusing on delivering the users requirement with speed and perfection.

- **Win as one**

  This principle is about understanding how designing work fits within the framework of the company to make a customer feel like it came from the same brand. This can happened with a thoughtful application of the core design principles.

➔ **Expertise skills**

Throughout the history, professionals such as designers, technicians, and creators have distinguished themselves by taking pride in their work. It is about designing part that is perfect and polished at every stage.

➔ **Doing more task with less time**

The focus of this principle is about particular thing and reducing functions to their core purpose. It helps to communicate about most important thing which focuses on the task at user hand.

2.3 Command Patterns and Touch Interactions

Commands are the interactive User Interface (UI). Commands controls can be placed on several surfaces in Windows Store app. Using commands a user can take action on the current page, whereas navigation link element takes the user to a different page.

➔ **Command Types**

Table 2.1 lists the command types.

Command types	Description
Filter	Removes or hides content a data set
Pivot	Gives a different view by reorganizing content within a data set
Sort	Changes the content order is to be displayed
View	Changes the style or pattern of content display

Table 2.1: Command Types

2.3.1 Command Types

Following are the command types:

➔ **Command placement**

Based on your requirement you can place commands within the app bar by considering the following guidelines:

- Use less commands to avoid from complicated look of app.
- Place specific commands, to improve the speed and to make it easy to work.
- Across all views of app, use consistent interaction and command placement.
- Keep short text labels and choose icons that are easy to relate and grasp.

➔ **Design principles for touch**

Windows 8 provides a set of touch interactions which makes user feel familiar to what they already know.

➔ **Provide immediate feedback**

Whenever user touches the screen, provide immediate visual feedback to them such as change the color or size or by moving.

➔ **Keep interactions reversible**

Touch interactions should behave in a reversible way i.e. provide visual feedback if users do some touch interaction.

➔ **Allow any number of fingers**

Without their notice users often touch with more than one finger. So provide provision accordingly when number of fingers touching the screen, the touch interactions shouldn't change. Instead provide a provision of allowing any number of fingers for sliding and touching.

2.3.2 Touch Interaction

➔ **Press and hold to get tips**

This option provides detailed information about the app. Press and hold can be used for selecting both horizontal and vertical panning.

➔ **Tap for primary action**

Tapping on a screen invokes its primary action, for e.g. execution of a command.

➔ **Slide to panning**

For basic usage like moving, drawing or writing purpose, target small, densely packed elements can be used.

➔ **Swipe to select and move**

By sliding the finger a short distance or in perpendicular direction the object can be selected. In this condition panning is constrained to one direction which displays the app bar with related commands.

➔ **Pinch and stretch to zoom**

Pinch and stretch to zoom enables jumping to anywhere within the content. The pinch and stretch gestures are used for resizing. Semantic zoom provides a zoomed out view for displaying groups of items and faster ways to return back into them.

➔ **Turn to rotate**

To rotate an object on a screen, rotate with two or more fingers and to rotate the entire screen, rotate the device itself.

2.4 Guidelines for Advertising and Branding the Windows Store Apps

Advertising is the widely accepted fact and it is the most important to each and every company or firm for marketing of products and services. All over the world companies universally has accepted this fact. Advertising is considered to be a key monetization option which is a profitable investment.

Following are the guidelines to design your app for ads:

➔ Plan how much space is required for each component. Integrate advertising into the original design and content layout throughout the app.

➔ In real time many ad formats are available but all of them may not engage users or offer a flawless experience. So choose ad formats that make sense for proper app.

➔ Use well-established metrics for monetozation. Follow industry norms and standards.

➔ Check with all kinds of view state such as landscape, portrait, and snap view.

➔ Design for all possible view states.

➔ **Branding your Windows Store apps**

In a very creative manner app designers and developers can use and incorporate branding into Windows Store apps. While designing the Windows Store apps, it is essential to consider that your apps incorporate the essence and standard of your brand.

➔ **How to incorporate your brand**

Use the visual elements which shows the value of your brand (knobs and dials manipulated through code) to create a specific look in your Windows Store app.

Table 2.2 lists the visual elements.

Visual Element	Description
Colors	Apply the basic color connected with your brand to convey users that this app has come from your company. This app comes from your business.
Graphics	Usage of more graphics may distract user, so be careful while adding graphics.

Concepts

Visual Element	Description
Images	Reuse the same graphics, images, and styles as illustrations and photography also reflect your brand.
Grid	The grid system helps to focus and unite the visual elements of the apps.
Layout	Provides consistency across pages and content types and placement of visual elements for all pages that are relevant to a brand.
Logo	Usage of company logo will help people to recognize your brand quickly.
Typography	The right choice of Typefaces such as color, logo, or layout is impactful to the brand, so be thoughtful about its use.

Table 2.2: Visual Elements

2.5 User Experience Guidelines

Following are the guidelines for apps that support panning:

- Along a single axis when the content area exceeds beyond the viewport, then, use one-dimensional panning.
- Don’t use mandatory snap-points, instead use proximity snap points.
- Along both axes when the content area exceeds beyond the viewport, then, use two-dimensional panning.
- For unstructured content use the freeform panning by bypassing the default behavior.
- Resize each element to fit the view.
- At each section header and logical boundary a snap-point should be placed.
- To provide location and size cues display panning indicators and scroll bars.
- Within a single-axis content area, chaining is used for panning.
- To enable people to rotate an object in clockwise or counter clockwise direction, the Windows Store app has provided touch-optimized technique.
- The selected object in a rotation gripper can be moved using a mouse or active pen/stylus.

➔ Touch or passive pen/stylus is used to rotate the object in any direction.

2.6 Defining Layouts and Views

➔ Laying out an app page

Some of the layout patterns are described here which can be used to lay out app page's UI elements.

Before talking about the Windows 8 silhouette; you will see the Windows 8 grid system.

- As grid system is a design tool, it is used to achieve visual unity across different apps.
- The basic unit of measurement of grid system is unit. It consists of units and sub-units (16 sub-units per square unit).
- Each unit of grid system is equals to 20 × 20 pixels. It is further divided into sub-units of 5 × 5 pixels. Figure 2.1 shows the grid present in the upper-left corner of a screen.

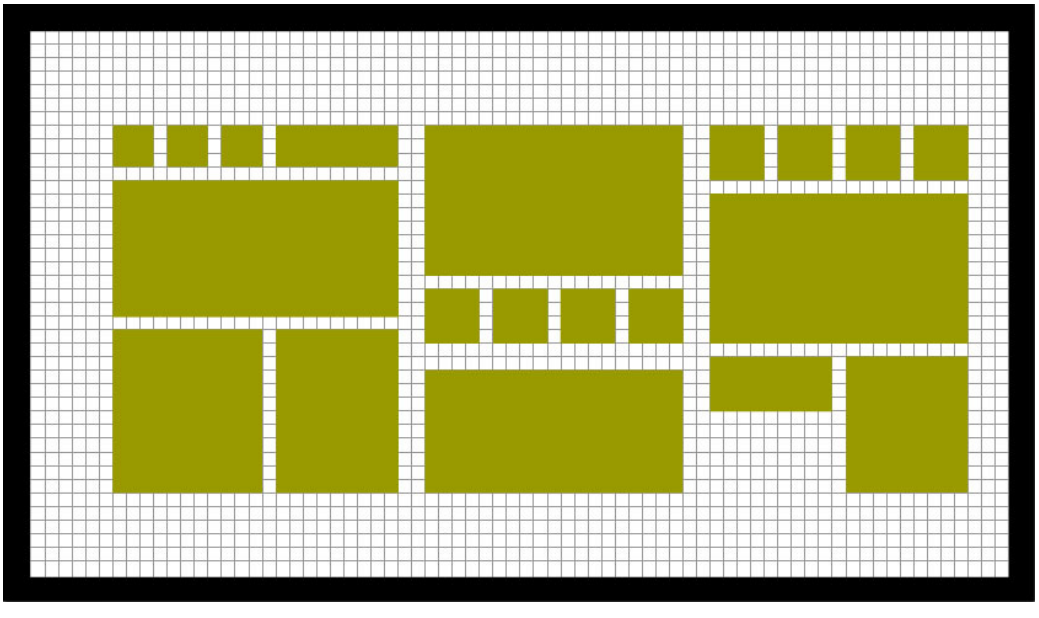


Figure 2.1: Laying out an App Page

➔ Page header

For the page header

- The baseline should be 5 units (100 pixels from the top).
- The left margin should be 6 units (120 pixels).

Note - SegoeUI Stylistic Set 20, light weight should be chosen for the Windows 8 page header.

➔ Content region

For the content region

- The top margin of 7 units (140 pixels).
- The left margin is 6 units (120 pixels).
- The bottom margin can take any value (flexible).
- Horizontal panning content should not be more than 6.5 units (130 pixels) and less than 2.5 units (50 pixels).
- The top and left margins stay unchanged in the vertically panning content.

➔ Horizontal padding values

Table 2.3 shows how the horizontal padding varies depending on the items.

Horizontal Padding Item	Value(Padding between Columns)
Padding lists	2 units, or 40 pixels
Hard-edged items	2 sub-units, or 10 pixels

Table 2.3: Value of Horizontal Padding

Figure 2.2 shows the horizontal padding values.

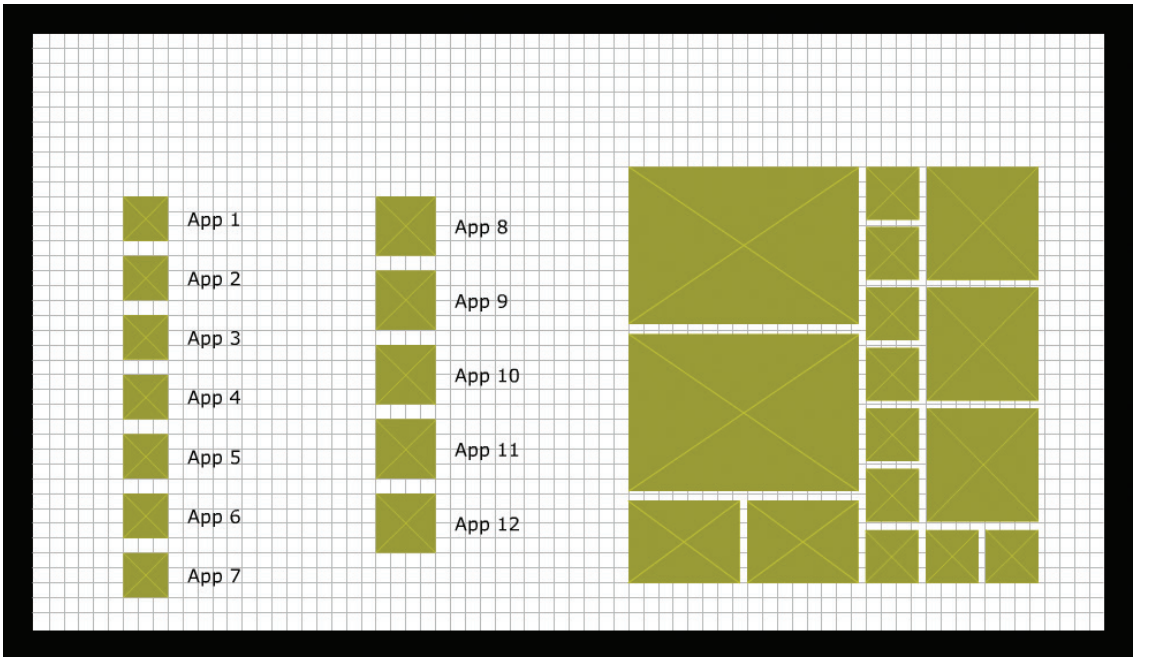


Figure 2.2: Horizontal Padding Values

➔ **Vertical padding values**

Table 2.4 shows the vertical padding between content items. It also varies depending on the types of items.

Vertical Padding Item	Value(Padding between Rows)
Tile and text lists	1 unit, or 20 pixels
Hard-edged objects	2 sub-units, or 10 pixels

Table 2.4: Value of Vertical Padding

➔ **Horizontal padding between groups**

- When the user is panning the padding allows them to distinguish from one group to another.
- The padding between groups is 4 units (80 pixels).

2.7 Adding Controls and Content

1. **Adding buttons (Windows Store apps using C# and XAML)**

The button control in the XAML UI framework is used to submit or reset a form. It provides a click event to respond to user input.

Different kinds of content(text or an image) can be put in a button. A button can be restyled by giving it a new look. For instance, when the user presses a `RepeatButton`, the `Click` event occurs continuously. In a normal state, this control acts as a Button. Code Snippet 1 demonstrates how to add a Button in XAML.

Code Snippet 1:

XAML

```
<Button Content="My First Button" Click="ButtonClickHandler"/>
```

C#

```
private async void ButtonClickHandler(object sender, RoutedEventArgs e)
{
Windows.UI.Popups.MessageDialog messageDialog =
new Windows.UI.Popups.MessageDialog("You have clicked my first button.");
await messageDialog.ShowAsync();
}
```

2. **Customizing the button content**

- Any object can be set as button's content as button is a `ContentControl`.
- It is represented in the form of button if the content is a UI Element or else if content is in the other form its string representation is shown in the button.

In following example a `StackPanel` contains an image and text. It is declared as the content of a Button control.

XAML

```
<Button Click="ButtonClickHandler" Background="#FF0D000" Height="50"
Width="200" >
<StackPanel>
<Image Source="btnBanner.png"/>
<TextBlock Text="My First Button" HorizontalAlignment="Center"/>
</StackPanel>
</Button>
```

3. **Add a HyperlinkButton**

The default feature of `HyperlinkButton` appears as a text hyperlink and on click; it navigates to a particular page (specified in the `NavigateUri` property).

2.7.1 Adding Flyouts and Menus

A `Flyout` control displays lightweight UI also called a `flyout`, which is either requires user interaction or information. It is used to show what user is doing at present. This UI allows the user choose from a relative list of simple commands or options.

➔ **When to use a flyout**

- On a response to a user tap or click a `flyout` should be shown.
- The `flyout` should not be dismissed programmatically until the user presses a command button or until the user selects a menu item present in the `flyout`.
- Without performing any action also the user can dismiss a `flyout`.

  Use a flyout for:

  - Collecting information
  - Displaying more info or show more details

Concepts

- Warnings and confirmations to let the user to confirm the action
- Declare the contents of the `flyout`

➔ **Adding a MenuFlyout**

Contents of the menu can be defined by adding `MenuFlyoutItem`, `ToggleMenuFlyoutItem`, and `MenuFlyoutSeparator` objects to the `MenuFlyout`.

Table 2.5 shows some of the `MenuFlyout` options.

MenuFlyout Options	Description
`MenuFlyoutItem`	Performs an immediate action
`ToggleMenuFlyoutItem`	Switches an option on or off
`MenuFlyoutSeparator`	Separates visually menu items

Table 2.5: MenuFlyout Options

Following example shows how to create the `Button` control with a `MenuFlyout`:

Example:

```
<Button Content="Options">
<Button.Flyout>
<MenuFlyout>
<MenuFlyoutItem Text="Reset" Click="Reset_Click"/>
<MenuFlyoutSeparator/>
<ToggleMenuFlyoutItem Text="Shuffle" IsChecked="{Binding IsShuffleEnabled,
Mode=TwoWay}"/>
<ToggleMenuFlyoutItem Text="Repeat" IsChecked="{Binding IsRepeatEnabled,
Mode=TwoWay}"/>
  </MenuFlyout>
 </Button.Flyout>
</Button>
```

2.7.2 Adding Selection Controls

Selection controls allows users to select between multiple options.

Table 2.6 shows Selection controls.

Controls	Description
Checkbox	Clears or selects an option

Controls	Description
Radiobutton	Selects a single option out of multiple option
Combobox	Selects item from a drop-down list
Listbox	Presents an inline list of items to select
Slider	Selects item between a range of values
Toggleswitch	Toggles between two states.

Table 2.6: Selection Controls

➔ **Add a CheckBox in XAML**

1. Add a CheckBox control to the App.Set the m:Name attribute to a string value to assign a name to the CheckBox.
2. A control must have a name to refer to it in code else, a naming is not required.
3. If it is required to assign the Content property to a string value, then, to the CheckBox allot a label.
4. Add a handler for the checked event when the CheckBox state changes, to perform an action. Add code to perform some action in the checked event handler.

```
<CheckBox m:Name="chkBox1" Content="MyCheck Box"
Checked="chkBoxClickHandler"/>
```

➔ **Using a design tool**

1. **Add a check box**

- Select the CheckBox control or pick the CheckBox in the **Assets** pane in **Blend** for **Visual Studio** or select the CheckBox in the **Toolbox** pane in **Microsoft Visual Studio**.
- The Select Controls are present in the left hand side of the **Assets** pane.
- On the design surface add a CheckBox.

Then, do one of the following:

Double-click the CheckBox. OR

Drag and drop the CheckBox to the design surface. OR

Draw the CheckBox control on the design surface.

- Type a name into the Name property text box if it is required to assign a name to the CheckBox.

Note - The Name property text box is present at the top of the **Properties** pane.

A control must have a name to refer to a control in code; otherwise a name is not required.

Concepts

2. **Add a radio button**

- Select the RadioButton from the **Toolbox** pane in **Microsoft Visual Studio**.
- In Blend for **Visual Studio** choose the RadioButton which is present in the **Assets** pane.
- If it is not selected then, select the controls in the left hand side of the **Assets** pane.

Code Snippet 2 shows how to perform some action when the radio button state changes.

Code Snippet 2:

XAML:

```
<StackPanel Margin="40">

  <RadioButton x:Name="radioButton1" Content="RadioButton 1"
GroupName="Group1" Checked="RadioButton_Checked"/>

  <RadioButton x:Name="radioButton2" Content="RadioButton 2"
GroupName="Group1" Checked="RadioButton_Checked" IsChecked="True"/>

  <RadioButton x:Name="radioButton3" Content="RadioButton 3"
GroupName="Group1" Checked="RadioButton_Checked"/>

 <RadioButton x:Name="radioButtonA" Content="RadioButtonA"
GroupName="Group2" Checked="RadioButton_Checked"/>

  <RadioButton x:Name="radioButtonB" Content="RadioButton B"
GroupName="Group2" Checked="RadioButton_Checked"/>
  </StackPanel>
```

➔ **To add a RadioButton**

Do one of the following:

Double-click the radio button. OR

Drag and drop the radio button to the design surface. OR

Draw the radio button control on the design surface.

- To name the RadioButton enter the name into the text box of Name property as shown in the Code Snippet 3.

Code Snippet 3:

```
<RadioButton x:Name="rdoButton1" Content="Male" GroupName="Gender"
      Checked="RadioButton_Checked"/>

<RadioButton x:Name="rdoButton2" Content="Female" GroupName="Gender"
      Checked="RadioButton_Checked"/>
```

Note - You can find the Name property text box at the top of the **Properties** pane. Update the RadioButton content.

Concepts

After selecting the radio button, click it to make the label content editable. Then, on the design surface do the following:

- Enter new content into the radio button label.

OR

- Type the content string inside the `Content` property text box.

2.7.3 Progress Controls

The progress control graphically represents present activity or the progress of a task. Type of progress bar:

1. **Determinate progress bar**: It shows the progress in the app by filling color from left to right.
2. **Indeterminate progress bar**: On the screen, dots, or slant lines appear from left to the right along a bar until they reach the end of the bar and then, disappear. While this task is taking place user can interact with the UI.
3. **Indeterminate progress ring**: Indicates that user action is stopped until the app completes the task. For example while loading a task, several dots move clockwise in a circle.

Following are the three styles of progress controls:

1. Determinate progress bar style
2. Indeterminate progress bar style
3. Indeterminate ring style

Figure 2.3 shows determinate progress bar style.

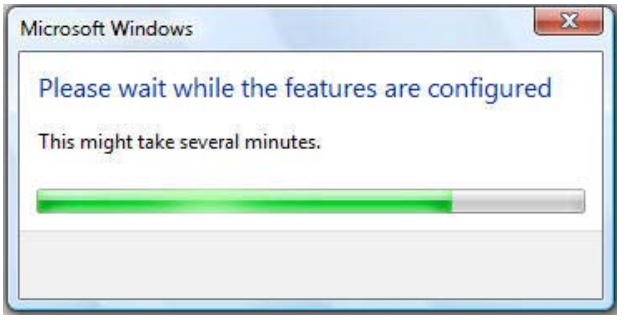


Figure 2.3: Determinate Progress Bar Style

Concepts

Figure 2.4 shows indeterminate progress bar style.

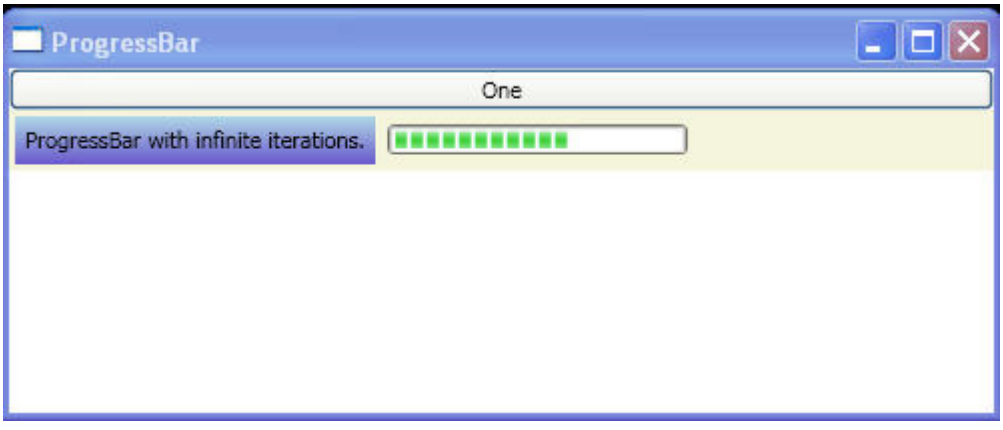


Figure 2.4: Indeterminate Progress Bar Style

Figure 2.5 shows indeterminate ring style.

Figure 2.5: Indeterminate Ring Style

➔ **Creating a determinate progress bar**

- In the Windows store app initially add a `ProgressBar` control.
- Then, set the indeterminate property to false(default value).
- Change the value `Maximum` property if not willing to set it to default value.
- Then, set the `Value` property to show the progress.

Following example shows how to create a determinate progress bar. Here the value is set to 0 to 100.

Example:

```
<ProgressBar IsIndeterminate="False" Maximum="100" Value="0" Height="15"
Width="300"/>
```

Concepts

2.7.4 Tooltip

A tooltip can be added to a UI element in the Extensible Application Markup Language(XAML) editor. Also a tool tip can be added using a design tool.

To add a text tooltip in XAML

➔ Choose an element to associate the `ToolTip` with the `UIElement`.

➔ Set the attached property (`ToolTipService.ToolTip`) to associate the `ToolTip` with the `UIElement`. Code Snippet 4 shows how to add a text tooltip.

Code Snippet 4:

```
<Button Style="{StaticResource BackButtonStyle}"
ToolTipService.ToolTip="This button will take to previous screen" />
```

To add an image tooltip in XAML

➔ Choose an element to associate the `ToolTip` with the `UIElement`.

➔ To associate the `ToolTip` with the `UIElement`, set the `ToolTipService.ToolTip` attached property to the syntax using XAML property element.

➔ Now to the `ToolTip` content add an Image element.

Code Snippet 5 shows how to add an image of the `HelpLogo.png` asset.

Code Snippet 5:

XAML

```
<TextBlock Text="Help">
<ToolTipService.ToolTip>
<Image Source="HelpLogo.png"/>
</ToolTipService.ToolTip>
</TextBlock>
```

2.7.5 Adding ListView, SemanticZoom, and Other Data Controls

Following are the several controls provided by XAML UI framework for Windows Store apps. This makes it easy to display and manipulate data.

Table 2.7 lists the controls.

Control	Description
`ListView`	Presents a collection of items in a list which can be scroll vertically.
`GridView`	Items which can be scrolled horizontally are presented in a rows and columns.
`FlipView`	Presents a collection of items that a user can flip through, one item at a time.
`SemanticZoom`	Allows the user to zoom between two views.

Table 2.7: Choosing a ListView or GridView

The `ListView` and `GridView` have similar functionality, but display data differently. These are used to display collections of data in the app and they are derived from the `ItemsControl` class.

Note - The `ItemsControl` refers to both `ListView` and `GridView` controls.

Figure 2.6 shows `ListView`.

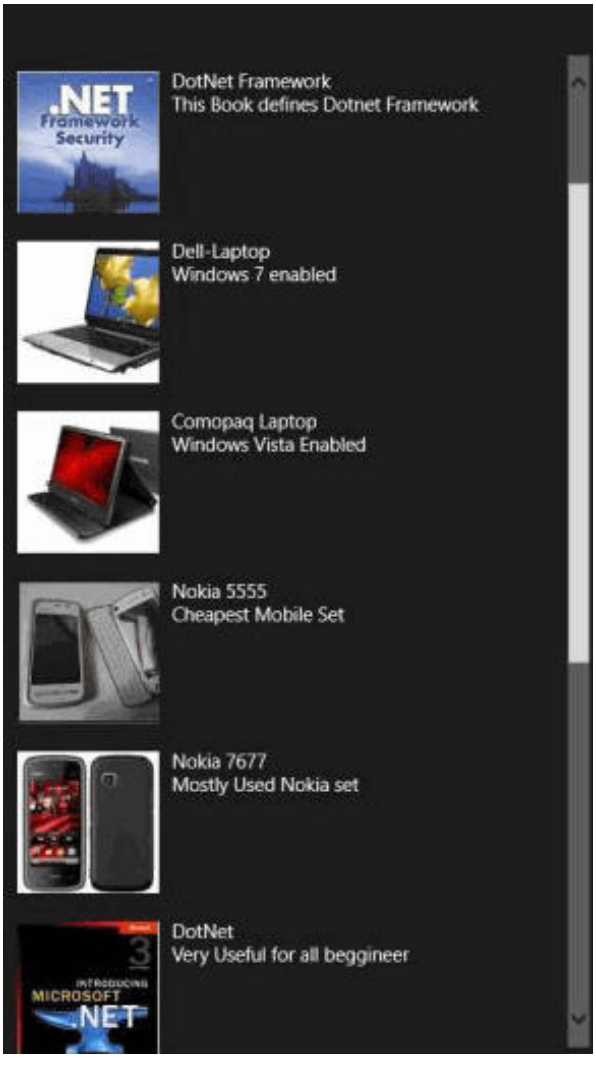


Figure 2.6: ListView

Concepts

Figure 2.7 shows `GridView`.

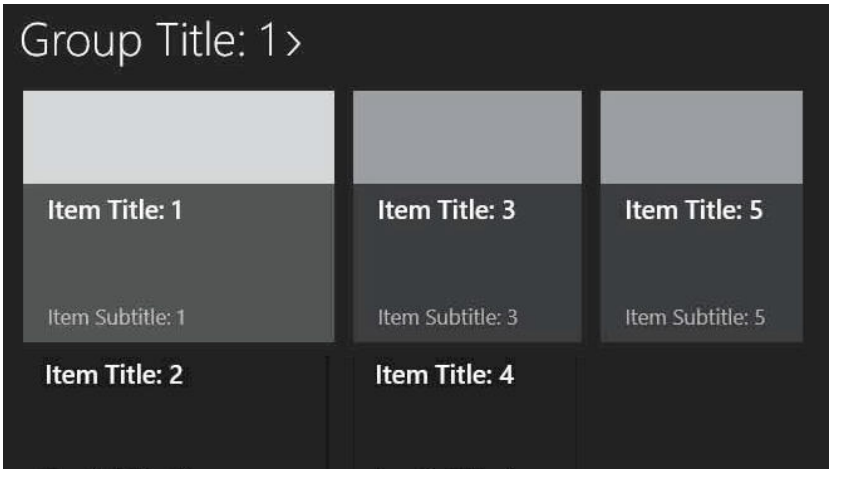


Figure 2.7: GridView

The `ListView` usually used to show a prearranged list of items, such as list of mails or search results. It displays data stacked vertically and also useful in master-detail scenarios.

The `GridView` displays data stacked horizontally and it shows a rich visualization such as a photo gallery.

Adding items to the Items collection

An item can be added to the Items collection using XAML or using code. Code Snippet 6 includes a `ListView` with items defined inline using XAML, and a `GridView` with items and it also demonstrates the use of the `ListView` control.

Code Snippet 6:

XAML

```
<ListView x:Name="myLv1" SelectionChanged="lvSelectionChangeHandler">
<x:String>My Item 01</x:String>
<x:String>my Item 02</x:String>
</ListView>
```

C#

```
GridView gv1 = new GridView();
gv1.Items.Add("My Item 01");
gv1.Items.Add("My Item 02");
gv1.SelectionChanged += gvSelectionChangeHandler;
```

Concepts

➔ **Setting the items source**

The Windows store apps mainly uses a `ListView` or `GridView` to show data from a source such as an Internet or the database. So, set its `ItemsSource` property to an `ItemsControl` from a data source.

➔ **SemanticZoom control**

When there exist two different views, the `SemanticZoom` control allows the user to zoom between these two which have same content.

1. The main view of the content.
2. Allows users to quickly navigate through this view.

The `SemanticZoom` control uses the zoomed-in view and the zoomed-out view, to provide the zooming functionality.

In markup, construct the `SemanticZoom` control and assign the `GridView` controls to the `ZoomedOutView` and `ZoomedInView` properties.

In XAML, add the `SemanticZoom` control. Code Snippet 7 shows the markup before the `GridView` controls are added.

Code Snippet 7:

```
<SemanticZoom>
<SemanticZoom.ZoomedOutView>
<!--Add the content-->
</SemanticZoom.ZoomedOutView>
<SemanticZoom.ZoomedInView>
<!--Add the content-->
</SemanticZoom.ZoomedInView>
<SemanticZoom.ZoomedInView>
<!--Add the content-->
</SemanticZoom.ZoomedInView>
</SemanticZoom>
```

2.8 Displaying and Editing Text

Visual Basic provides several controls to represent a text. `TextBlock` and `RichTextBlock` are the controls for displaying read-only text.

TextBlock

TextBlock is the primary control and using its Text property it can be displayed in a TextBlock control. Code Snippet 8 demonstrates how to define a TextBlock control and how to set its Text property to a string.

Code Snippet 8:

XAML:

```
<TextBlock Text="My First text Display" />
```

In XAML, every string has different formatting and a series of strings can be displayed in a TextBlock .

This one can be achieved by separating each Run element with a LineBreak element and by using a Run element to display each string with its formatting for example, <LineBreak/>.

➔ **RichEditBox**

RichEditBox can be used to control the type and edit rich text documents which contain hyperlinks, formatted text, and images.

- By setting IsReadOnly property to true one can make a RichEditBox read-only and by default, the RichEditBox supports spell checking.

  To get the Document property of the RichEditBox, use its content. It is a Windows.UI.Text.ITextDocument object.

2.9 Displaying Images, Graphics, and Thumbnails

An Image object or the ImageBrush object can be used to represent an image. An image is rendered by the Image Object and the ImageBrush object will paint another object by using an image.

Code Snippet 9 demonstrates creating an image by using the Image object.

Code Snippet 9:

```
<Image Width="500" Source="imageFile.jpg" />
```

An image can be used to paint an area using a brush object with the ImageBrush object.

➔ **Stretching an image**

Stretch property sets the Width and Height and the area is created in the shape of a rectangle in which the image is displayed.

Table 2.8 shows `Stretch` property.

Stretch Property	Usage
`None`	A larger image cannot be clipped because developer does not have control over the viewport.
`Uniform`	This is the default value and the aspect ratio of the content is retained while changing the image size.
`Uniform-ToFill`	The original aspect ratio is scaled completely to fill the output area.
`Fill`	The image is adjusted to fit the output dimensions, but the image is distorted completely to fill the output area.

Table 2.8: Stretch Property

➔ Cropping an image

Cropping of an image can be done by using `Clip` property as shown in the Code Snippet 10.

Code Snippet 10:

```
<Image Source="licorice.jpg" Height="500">
<Image.Clip>
<RectangleGeometry Rect="10,10,490,490" />
</Image.Clip>
</Image>
```

In the example the crop start from point `10,10` and end at `490, 490`.

➔ Applying an Opacity

An image can be turn into semi-translucent applying an Opacity to an image. The range of opacity is between 0.0 to 1.0.In this 1.0 is fully opaque and 0.0 is fully transparent.

➔ File formats of an image

Image can display the following image file formats:

1. Bitmap (BMP)
2. Tagged Image File Format (TIFF)
3. JPEG XR
4. Portable Network Graphics (PNG)
5. Graphics Interchange Format (GIF)
6. Icons (ICO)
7. Joint Photographic Experts Group (JPEG)

2.10 Check Your Progress

1. Which of the following will provide detailed information about the app?

(A)	Tap for Primary action	(C)	Press and hold to learn
(B)	Slide to pan	(D)	Stretch to zoom

2. Name the control which presents an inline list of items.

(A)	Combobox	(C)	Radiobutton
(B)	Listbox	(D)	Checkbox

3. Which control allows user to present a collection of items?

(A)	ListView	(C)	FlipView
(B)	GridView	(D)	SemanticZoom

4. Name the stretch property which is the default value and the image is adjusted to fit the output dimensions.

(A)	Uniform	(C)	Fill
(B)	Uniform ToFill	(D)	All the above

5. Which command type changes the style or method in which content is displayed?

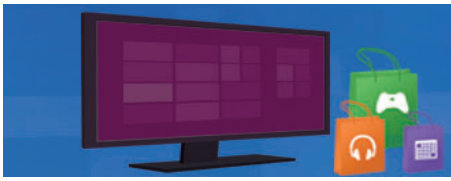
(A)	Filter	(C)	Sort
(B)	Pivot	(D)	View

6. Which MenuFlyout option performs an immediate action?

(A)	ToggleMenuFlyoutItem	(C)	MenuFlyoutItem
(B)	MenuFlyoutSeparator	(D)	None of the above

2.10.1 Answers

1.	C
2.	B
3.	C
4.	A
5.	D
6.	C



Summary

- Microsoft has its own set of principles, which guides it in every design venture that is commenced.
- Advertising is the widely accepted fact and it is the most important tool to company and firm for marketing of products and services.
- A Flyout control displays lightweight UI also called as flyout which either requires user interaction or informational.
- While branding it is essential to consider that your apps incorporate the essence and standard of your brand.
- Button controls are used to submit or reset a form.
- Contents of the menu can be defined by adding MenuFlyoutItem, ToggleMenuFlyoutItem, and MenuFlyoutSeparator objects to the MenuFlyout.
- Selection controls allows users to select between multiple options.
- ListView, SemanticZoom, and other data controls makes easy to display and manipulate data.

Session - 3

Designing and Implementing Navigation in a Windows Store App

Welcome to the Session, **Designing and Implementing Navigation in a Windows Store App**.

The session will discuss the hierarchical navigation pattern for your Windows Store app, different types of hierarchical system of navigation and how to place navigation elements on top App bar.

In this Session, you will learn to:

- Use a hierarchical navigation pattern for your Windows Store App
- Write types of hierarchical system of navigation
- Write about features and uses of flat pattern
- Explain about different types of labels present on canvas navigation
- Place navigation elements on top App bar
- Write about overview and types of Layout
- Navigate between pages
- Display and edit text in XAML framework or Visual Basic Displaying and editing text in Windows Store Apps

3.1 Hierarchical Pattern

While designing your Windows Store App the content can be separated into distinct, using a hierarchical navigation pattern. This is a most frequent and well-known pattern that works well with relating to information architecture, passed over in a preferred manner or arrangement.

Depends on the scenarios of app uphold, you have to select the navigation pattern. A hierarchical pattern is most suitable for your app if it supports different types of feel and component with organization and structure.

Figure 3.1 Shows Hierarchical System of Navigation.

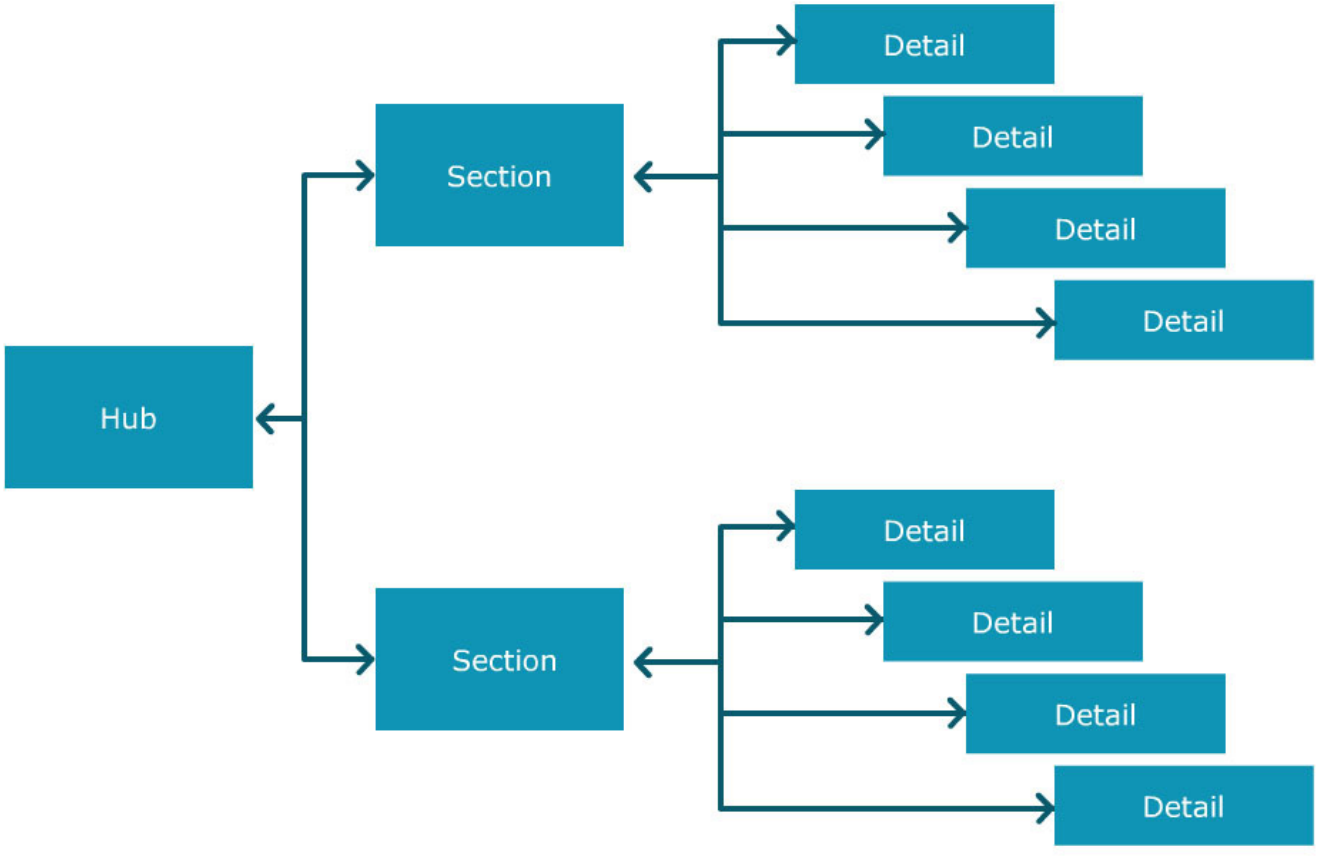


Figure 3.1: Hierarchical System of Navigation

Hierarchical system of navigation pattern makes Windows Store apps quick and rapid. Due to this most of the apps use a hierarchical navigation pattern. It is foremost for a user to explore for apps with huge content collections or many separate sections of content.

➔ **Types**

Hub pages provide an entry point to the app. Content is presented in a rich horizontally panning view which gives users a brief information about new items. The hub contains different types of content in which each one relates to the app's section pages.

Concepts

Figure 3.2 shows a hub page.

Figure 3.2: Hub Page

Image Courtesy: msdn.com

The section contains the scenario which is a second level of an app. The content that can be displayed in any form represents the scenario in which the content consists of individual items. It has its own detail page which is a third level of an app. Detail pages may contain a single object or a lot of information (functionality).

➔ Flat Pattern

Many Windows Store apps practice a flat navigation pattern such as browsers, games, or document creation. These patterns let the user to move between tabs, modes, or pages that all exist within the same hierarchical level. There is no actual back button or navigation stack to move back to previous page. So moving from one page to another can be done through links within the content, as shown in figure 3.3 or through the navigation bar, as shown in figure 3.4.

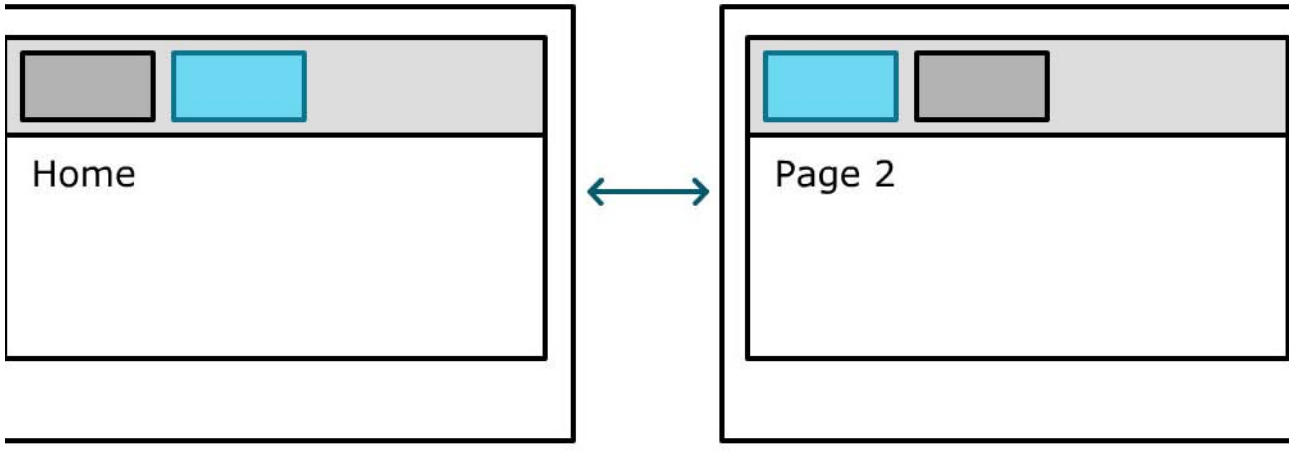


Figure 3.3: Navigation through Direct Links within the Content

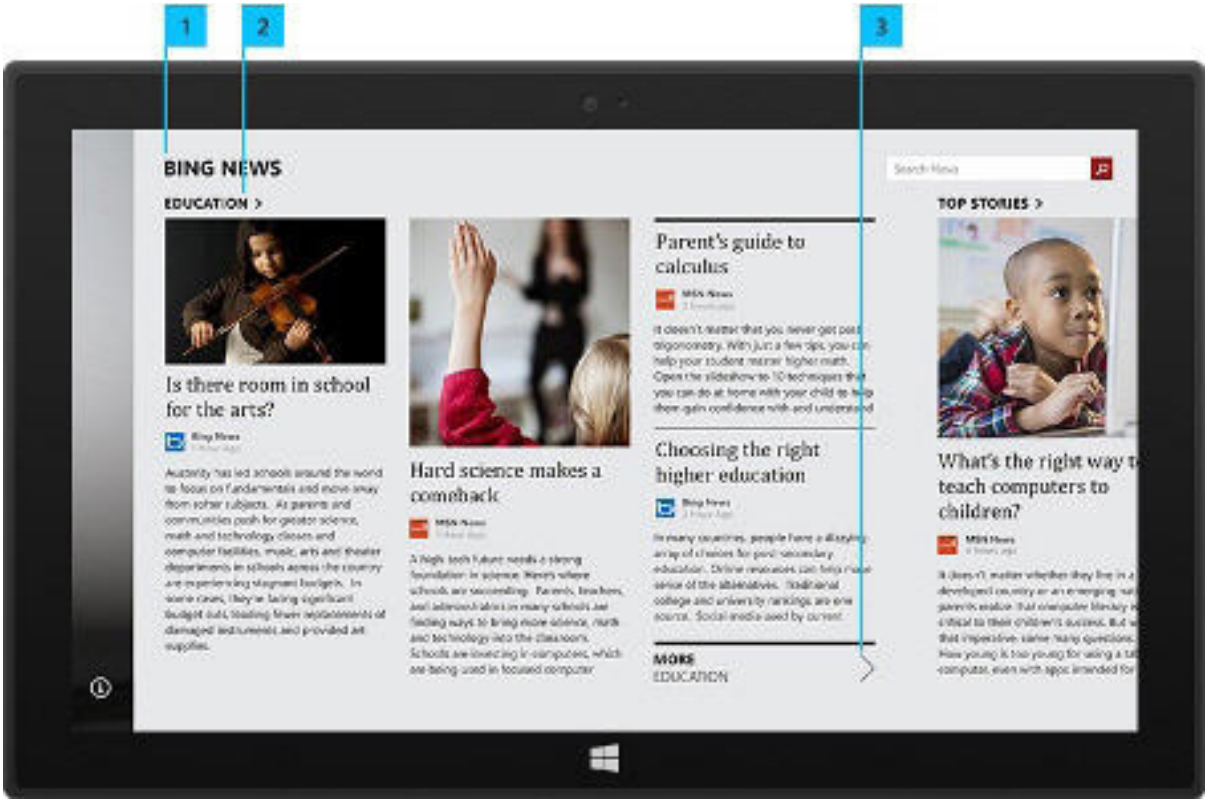


Figure 3.4: Navigating through the Navigation Bar

Image Courtesy: msdn.com

3.2 On-canvas Navigation

➔ **Header and Back Button**

The header provides the present page and is helpful for indicating user is in which page. The back button helps to go back to previous location where user was.

➔ **Section or Category Labels**

Section or category labels help the user to move from content of one section or category to another.

➔ **Other Targets**

Customized targets such as buttons, search results, tiles, arrows, or other navigation elements are part of the On-canvas navigation.

Concepts

3.3 Top App Bar

Figure 3.5 displays the Top App Bar.

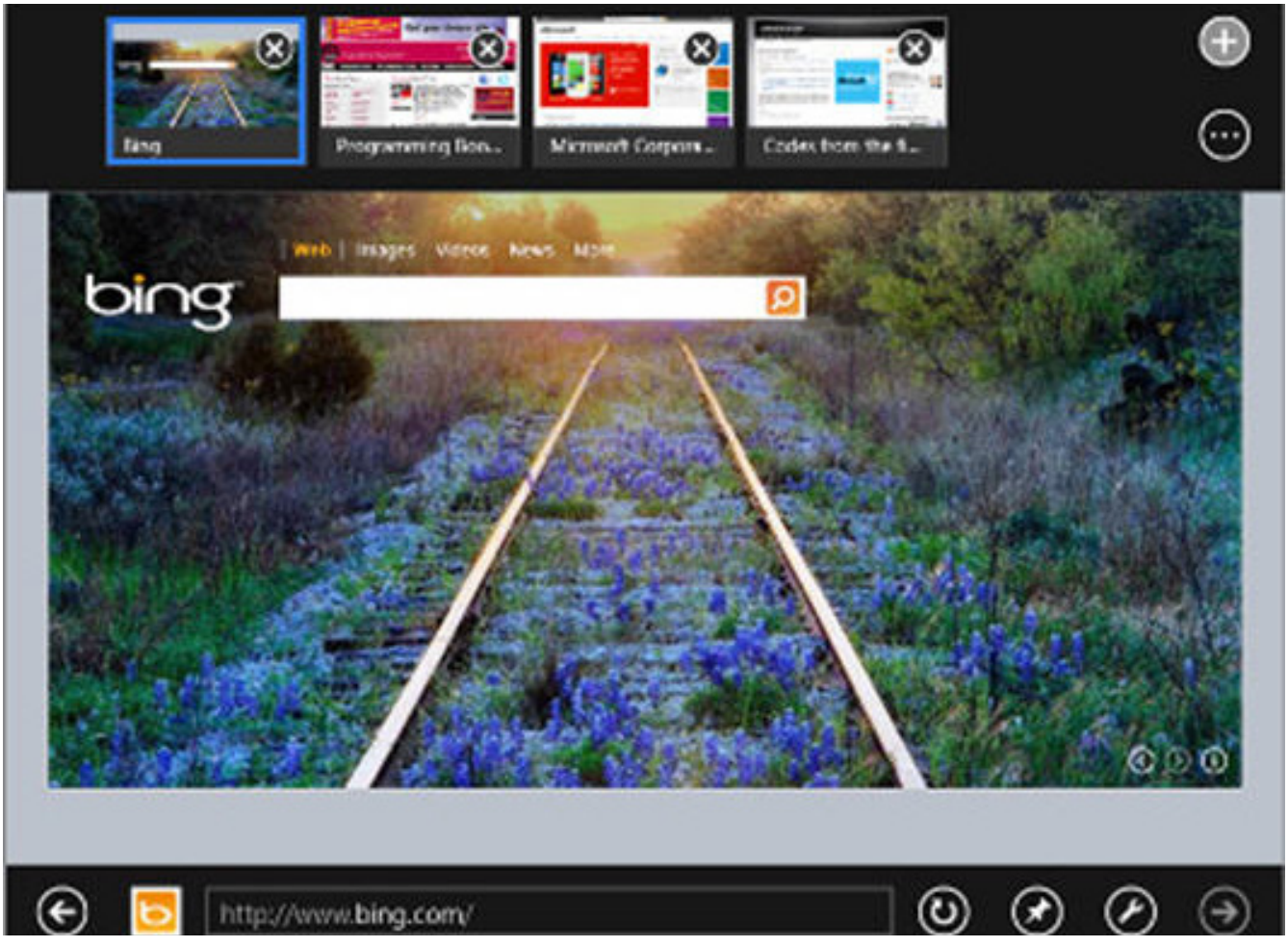


Figure 3.5: Top App Bar

Image Courtesy: Bing.com

You can place the navigation elements on the canvas or App bar appears by swiping the edge (top or bottom) of the screen. If users frequently need them then, navigation bar can be placed on canvas, but that should not impact the app experience. Navigation elements can be placed in the top App bar (also known as navigation bar or `nav bar`) which provides more screen area for the content in app.

3.4 Handling Navigation in Windows Store Apps

Some of the layouts for handling navigation are described in detail.

3.4.1 Defining Layouts

➔ **Layout Overview**

Layout is the process which is used to position visual objects by sizing and positioning objects in UI. To position a visual object it must be put in a Panel control or other container object. Grid, Stack, Panel, and Canvas are the different types of Panel controls provided by the XAML framework and it supports absolute layout and dynamic layout.

➔ **Absolute Layout**

In an absolute layout, controls are located using x and y coordinates and the positioning will not consider the size of the screen. By specifying the exact locations according to their parent element, the child elements in a layout panel can be arranged. In absolute positioning of UI elements, different pages can be designed for different screen resolutions.

The XAML framework set a Canvas control for absolute positioning. The `Canvas.Left` and `Canvas.Top` helps to position a control on a Canvas.

➔ **Dynamic Layout**

To make the interface appear correctly on various screen resolutions, dynamic layout can be used. As screen resolutions differ and based on the sizes child elements can be arranged according to their parent. For example, controls should wrap horizontally and clearly describe once it is placed on a panel.

Some special values need to be assigned to Height and Width properties for automatic sizing using a StackPanel or a Grid. The recommended settings for a dynamic layout are as follows:

- Set the Width and Height of the layout or control to Automatic. Auto sizing is supported for controls which takes the specified values and fills the cell that contains it.
- For text containing controls set the `MinWidth` or `MinHeight` properties. This helps to increase the text readability.

➔ **Auto and Star Sizing**

Using auto sizing controls can be fit to their content, though content changes size. Star sizing is useful to adjust the free space available between the rows and columns of a grid according to a ratio.

➔ **Panel Controls**

In XAML the built-in layout panels can be presented. The XAML code snippet for this is as follows:

```
<Canvas>
<Rectangle Canvas.Left=”50” Canvas.Top=”50” Fill=”blue” Width=”600”
Height=”4000” />
</Canvas>
```

Concepts

➔ StackPanel

The child elements can be arranged in a single line using a `StackPanel`. The child elements can be specified using Orientation property in which orientation can be arranged into horizontally or vertically. When it is required to organize a small subsection of the UI on the page, the default value can be used for the controls i.e. `Orientation.Vertical.StackPanel.`

```
<StackPanel Margin="10">

<Rectangle Fill="Blue" Width="300" Height="400" Margin="5" />

<Rectangle Fill="Green" Width="300" Height="400" Margin="5" />

</StackPanel>
```

➔ Grid

The Grid control can be used to arrange controls in multiple row and column layouts. Using a row and column definitions of Grid controls the `RowDefinitions` and `ColumnDefinitions` properties can be specified. In a similar way using the `Grid.Column` and `Grid.Row` properties the objects can be positioned in specific cells of the Grid. The space within a column or a row can be distributed by using Auto or star sizing. Using the `Grid.RowSpan` and `Grid.ColumnSpan` properties content can span upon multiple rows and columns.

Following example creates three rows and two columns Grid:

Example

XAML:

```
<Grid Margin="12,0,12,0">



<Grid.RowDefinitions>

<RowDefinition Height="auto" />

<RowDefinition />

<RowDefinition Height="auto" />
```

```
</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="*" />

<ColumnDefinition Width="*" />

</Grid.ColumnDefinitions>
<TextBox Text="Text - R1 C1 " FontSize="20" Grid.Column="0" Grid.Row="0" />

<Button Content="Button - R1 C2" FontSize="20" Grid.Column="1" Grid.Row="0"
/>

<TextBox Text="Text R3 C1" FontSize="20" Grid.Column="0" Grid.Row="2" />

<Button Content="Button - R3 C2" FontSize="60" Grid.Column="1 " Grid.Row="2"
/>
</Grid>
```

3.4.2 Navigating Between Pages

Similar to navigating between pages on a single Web site, user can navigate between the pages within the app and can also create multiple pages for Windows Store app. The page templates present in Microsoft Visual Studio provides basic navigation support using C#.

➔ **To Create a New Project**

1. Open **Visual Studio 2012 IDE**.
2. Select **File** → **New** → **Project**. Then, the **New Project** dialog box appears.
3. In the Installed pane, expand **Templates**, then either select **Visual C#**.
4. Select template type as **Windows Store app**. In the center pane, select XAML Blank App.
5. Type a name for the project as **NavigationQuickstart**.
6. Click **OK**. The project is created.

➔ **To Add a Page to an App**

- Click the Project menu, a drop-down list appears. Select Add New Item option, Add New Item dialog box appears.
- In the center pane, select Basic Page change the Name value to myPage. Click Add option.
- After repeating these steps twice, you will find the following files are added to the project.myPage1.xaml.
- myPage.xaml.cs
  myPage2.xaml
  myPage2.xaml.cs
- Make the following changes to myPage1.xaml.

Name the TextBlock element to pageTitle and then, change the property of the text to Page1. The XAML should look as follows:

Code Snippet:

XAML:

```
<TextBlock x:Name="myPageTitle" Text="My First Page"/>
```

In the following example, rootGrid acts as a second child element. The StackPanel element contains the back button and page title which is sibling to the Grid.

```
<StackPanel Grid.Row="1">


<HyperlinkButton Content="Click here to go to my Page 2"
Click="hlbtnClickHandler"/>


</StackPanel>
```

In myPage2.xaml apply following changes.

Identify the TextBlock element which is named pageTitle and then, change the Text property to MySecond Page. The XAML looks as follows:

Code Snippet:

```
<TextBlock x:Name="MyPageTitle" Grid.Column="1" Text="MySecondPage"
Style="{StaticResource PageHeaderText-Style}"/>
```

XAML:

```
    <TextBlock x:Name="myPageTitle" Text="My First Page"/>
```

In the following example root Grid acts as a second child element and the back button and page title are `StackPanel` element which is a sibling to the `Grid`:

Example:

XAML:

```
<StackPanel Grid.Row="1">


<TextBlock HorizontalAlignment="Left" Name="myTxtBox1" Text="Hello
Friends! Welcome"/>


</StackPanel>
```

Add the following code in the class `BasicPage1` present in `myPage1.xaml.cs`:

C#:

```
private void hlbtnClickHandler(object sender, RoutedEventArgs e)
{


this.Frame.Navigate(typeof(myPage2));


}
```

Now, the new pages are prepared and you will make `myPage1` as the first page that appears when the app starts. First open `app.xaml.cs` instead of `BlankPage` call `Frame.Navigate` by changing `OnLaunched` method `using` `myPage1`. The following code appears:

Code Snippet:

C#:

```
if(!rootFrame.Navigate(typeof(myPage1), e.Arguments))


{...}
```

To throw an app exception here the code uses the return value of Navigate and when the Navigate returns true, it indicates that the navigation happens.

To test the app now start the app and click the link `Click to go to page 2`. Then, at the top the second page 'Page 2' appears.

Note - To the left of the page title there is a back button, by clicking this button returns to the first page.

➔ Frame and Page Classes

Now, learn how the added pages provide navigation support for the app. As you know that Frame class is primarily responsible for navigation which implements methods such as `Navigate` (used to display content in the Windows Frame) `GoBack`, and `GoForward`.

In the last example, the method `App.OnLaunched` creates a Frame which passes `BasicPage1` to the Navigate method. Here, the content of the app's current window is set to the `myPage1` Frame.

`myPage1` is a subclass of the Page class which is a Frame property. Content of myPage2 is displayed by the frame, when the Click event handler of the Hyperlink Button calls `Frame.Navigate(typeof(myPage2))`.

3.5 Check Your Progress

1. Name the second level of an app which gives brief information about the scenario in which the content consists of individual items.

(A)	Detail page	(C)	Hub page
(B)	Section page	(D)	App bar

2. Which navigational bar provides the present page and helps for indicating user?

(A)	Header	(C)	Back button
(B)	Footer	(D)	Previous button

3. Which label helps the user to move to from content of one section or category to other?

(A)	Hub	(C)	Section
(B)	Tiles	(D)	Buttons

4. Which category Grid, Stack, Panel, and canvas belongs to?

(A)	Buttons	(C)	Control panels
(B)	Objects	(D)	Framework

5. Which element is used to fit the control to their content?

(A)	Absolute layout	(C)	Dynamic layout
(B)	Auto and star sizing	(D)	Panel controls

6. Which method is used to display content in the Windows Frame?

(A)	Navigate	(C)	GoForward
(B)	GoBack	(D)	None of the above

3.5.1 Answers

1.	B
2.	A
3.	C
4.	C
5.	B
6.	A

Summary

- The content can be separated into distinct, using a hierarchical navigation pattern.
- Hub pages Provides an entry point to the app.
- The section contains the scenario which is a second level of an app.
- The content consists of individual items and has its own detail page which is a third level of an app.
- The back button helps to go back to previous location where users were.
- Navigation elements can be placed in the top App bar (also known as navigation bar, or nav bar) which provides more screen area for the content in app.
- Layout is the process which is used to position visual objects by sizing and positioning objects in UI.
- In an absolute layout, controls are located using x and y coordinates and absolute positioning doesn't consider the size of the screen.
- To make the interface appear correctly on various screen resolutions, dynamic layout can be used.
- Using auto sizing controls can be fit to their content, though content changes size.
- The Grid control can be used to arrange controls in multiple row and column layouts.

Session - 4

App Bars and Layout Controls

Welcome to the Session, **App Bars and Layout Controls**.

The session will discuss how to achieve data binding. It will also discuss the methods and techniques to achieve various customizations of objects and collection of objects.

In this Session, you will learn to:

- Write the overview of an app bar
- Choose right AppBar and CommandBar to Windows Store app
- Add contextual commands and menu
- Share an app bar across pages
- Use App bar styles and templates
- List do's and don'ts to place commands on the app bar
- List the guidelines to follow when designing the controls for your app
- Comparison of the main features of panels

4.1 Overview of an App Bars

App bars are used to show navigation, tools, and commands to hide whenever they are not required and it is hidden by default. Apps can be shown or make it hidden when the user presses Windows+Z, right-clicks or swipes from the edge (top or bottom) of the screen. Apps can be shown programmatically, when the user interacts with the app or makes a selection.

4.1.1 Choose an App Bars or CommandBar

Command Bar and AppBar controls helps to display UI that appears at the edge (top or bottom)of the screen which appears by swiping on an edge. The AppBar enables to add any XAML content to it and by assigning `CommandBar` or `AppBar` control (`TopAppBar` or `BottomAppBar` property of a Page) to an app, app bars can be added.

To show navigation in your app use the navigation top app bar and the bottom app bar is used to show commands and tools.

Following example demonstrates how to add an App Bar control on the top of the page:

Example:

```
<Page.TopAppBar>
<AppBar>
<!-- Your application bar content goes here -->
</AppBar>
</Page.TopAppBar>
```

Following example shows how to add `CommandBar` control as the Bottom App Bar:

Example:

```
<Page.BottomAppBar>
<CommandBar>
<!-- Your Command bar content goes here-->
CommandBar>
</Page.BottomAppBar>
```

4.2 Using an App Bar

Commands can be added to a `CommandBar` which apply when a `CommandBar` between pages are shared or an item is selected. A command can be added to a shared `CommandBar` which makes it easier to capture the command to elements in that page. Add a command when the user moves from one page to another and take it off when the user moves away from the page.

4.2.1 Adding Contextual Commands

Following example shows how the CommandBar is added as the bottom app bar using a single button:

Example:

```
<Page.BottomAppBar>
<CommandBar x:Name="myBottomAppBar">
<AppBarButton Icon="Help" Label="Help" Click="btnHelpClikHandler"/>
</CommandBar>
</Page.BottomAppBar>
```

Figure 4.1 displays the output of this example.

Figure 4.1: Output

When a user navigates to the page do the following settings,

- To add the CommandBar, create an AppBarSeparator and AppBarButton.
- For the AppBarButton set the Icon and Label.
- For the click event handler register it.
- To the CommandBar add the button.

```
void btnHelpClikHandler(object sender, RoutedEventArgs e)
{
    // Your code goes here
}
```

Visibility property can be used to show and hide commands based on user selection.

4.2.2 Adding Menu to an App Bar

Menu helps to present more options in less space and it provides interactive controls. In the following example, the Sort menu displays a simple list that helps to provide options to choose.

When the button is clicked the flyout opens automatically in which the MenuFlyout is added to the Button's Flyout property.

Code Snippet 1 demonstrates the concept.

Code Snippet 1:

```
<AppBarButton Icon="Sort" Label="Sort">
<AppBarButton.Flyout>
<MenuFlyout>
<!-- Your code related to menu -->
</MenuFlyout>
</AppBarButton.Flyout>
</AppBarButton>
```

When user makes each choice in the menu then, add a `MenuFlyoutItem` for it and set its Text property. When the user makes a choice, to take some action assign a command. Code Snippet 2 demonstrates the concept.

Code Snippet 2:

```
<MenuFlyout>
<MenuFlyoutItem Text="More Info" Click="mfiClickHandler" Tag="info"/>
</MenuFlyout>
```

C#:

```
private void mfiClickHandler(object sender, RoutedEventArgs e)
{
    // your code here
}
```

4.3 Sharing an App Bar Across Pages

The app bar use in Windows store apps development can be specific to a particular page or common across for related pages, for e.g., you might want to use an app bar in all pages that allows you to navigate through the desired pages. This app bar needs to be shared across pages instead of creating it each time on a page. The following is an example code that does it.

Example:

```
<Page.TopAppBar>
    <AppBar x:Name="globalAppBar" Padding="10,0,10,0">
    <AppBar.Resources>
    <Style TargetType="Button">
    <Setter Property="Width" Value="150"/>
```

```
<Setter Property="Height" Value="50"/>
<Setter Property="Margin" Value="5"/>
</Style>
</AppBar.Resources>
<Grid>
<StackPanel x:Name="leftCommandPanel"
Orientation="Horizontal" HorizontalAlignment="Left">
     <Button  x:Name="Back" Content="Back" AutomationProperties.Name="Back"
       Click="btnBackClickHandler"/>
     </StackPanel>
<Stack Panel  x:Name="rightCommandPanel"    Orientation="Horizontal"
HorizontalAlignment="Right">
         <Button x:Name="page1Button"  Content="P-1" AutomationProperties.
Name="P-1"  Click="p1ClickHandler"/>
<Button x:Name="page2Button" Content="P-2" AutomationProperties.Name="P-2"
Click="p2ClickHandler"/>
</StackPanel>
</Grid>
</AppBar>
</Page.TopAppBar>
```

Following example displays the commands required to add navigate between pages:

Example:

```
Page mainPage = null;
protected override void OnNavigatedTo(NavigationEventArgs e)
{
mainPage = e.Parameter as Page;
frmContent.Navigate(typeof(Page1), this);
}
private void btnBackClickHandler (object sender, RoutedEventArgs e;
{
if (frmContent.CanGoBack)
```

Concepts

```
{
frmContent.GoBack();
}
else if (mainPage != null && mainPage.Frame.CanGoBack)
{
mainPage.Frame.GoBack();
}
}
private void p1ClickHandler(object sender, RoutedEventArgs e);
{
frmContent.Navigate(typeof(Page1), this);
}
private void p2ClickHandler(object sender, RoutedEventArgs e);
{
frmContent.Navigate(typeof(Page2), this);
}
Add a Frame in the root page to host the app pages.
<Grid Background="{StaticResourceApplicationPageBackgroundThemeBrush}">
<Frame x:Name="frmContent"/>
</Grid>
```

4.4 App Bar Styles and Templates

The styles and templates for the `AppBarButton` and the default `ControlTemplate` control can be modified to give the control a unique appearance.

Table 4.1 lists the `VisualState` names.

VisualState Name	VisualStateGroup Name	Description
`FloatingVisible`	`FloatingStates`	App Bar visible.
`FloatingHidden`	`FloatingStates`	App Bar hidden.
`Top`	`DockPositions`	App Bar is present at the top place of the screen.

Concepts

VisualState Name	VisualStateGroup Name	Description
Bottom	DockPositions	App bar is present at the bottom place of the screen.

Table 4.1: VisualState Elements

Following is the xaml code snippet that adds an appbar named 'globalappbar'. Add a button resource to style the properties. You then, have opened a `Grid` and two StackPanels named '`leftCommandPanel`' and '`rightCommandPanel`'. The '`leftCommandPanel`' have one button resource named 'Back' added to it. The '`rightCommandPanel`' have three button resources named 'Right', 'P-1', and 'P-2' respectively. Code Snippet 3 demonstrates the concept.

Code Snippet 3:

```
<Page.TopAppBar>
     <AppBar x:Name="globalAppBar" Padding="10,0,10,0">
     <AppBar.Resources>
     <Style TargetType="Button">
     <Setter Property="Width" Value="150"/>
<Setter Property="Height" Value="50"/>
<Setter Property="Margin" Value="5"/>
</Style>
</AppBar.Resources>
<Grid>
<StackPanel x:Name="leftCommandPanel"
     Orientation="Horizontal" HorizontalAlignment="Left">
<Button x: Name="Back" Content="Back" AutomationProperties.Name="Back"
       Click="btnBackClickHandler"/>
     </StackPanel>
<StackPanel  x: Name="RightCommandPanel"
     Orientation="Horizontal" HorizontalAlignment="Right">
     <Button  x: Name="page1Button" Content="P-1" AutomationProperties.
Name="P1"   Click="p1ClickHandler"/>
<Button x:Name="page2Button" Content="P-2" AutomationProperties.Name="P2"
     Click="p2ClickHandler"/>
```

```
</StackPanel>
</Grid>
</AppBar>
</Page.TopAppBar>
```

Following is the code where you add commands to navigate between pages when the corresponding page numbers, right or back button is clicked. The Navigate method is used in the click event handler or each button where the address of the page to be navigated is the first argument and the second argument is the address of the current object that holds the button. Code Snippet 4 demonstrates the concept.

Code Snippet 4:

```
        Page mainPage = null;
        protected override void OnNavigatedTo(NavigationEventArgs e)
        {
        mainPage = e.Parameter as Page;
        frmContent.Navigate(typeof(Page1), this);
        }
        private void btnBackClickHandler(object sender, RoutedEventArgs e)
        {
        if (frmContent.CanGoBack)
          {
        frmContent.GoBack();
          }
        else if (mainPage != null &&mainPage.Frame.CanGoBack)

          {
        mainPage.Frame.GoBack();
          }
        }
        private void p1ClickHandler(object sender, RoutedEventArgs e)
        {
        frmContent.Navigate(typeof(Page1), this);
        }
        private void p1ClickHandler(object sender, RoutedEventArgs e)
        {
```

```
        frmContent.Navigate(typeof(Page1), this);
        }
private void p2ClickHandler(object sender, RoutedEventArgs e)
{
frmContent.Navigate(typeof(Page2), this);
}
//You then add a Frame in the root page to host the app pages
<Grid Background="{StaticResourceApplicationPageBackgroundThemeBrush}">
<Frame x:Name="frmContent"/>
</Grid>
```

Following is the similar output of the code. This code has a back, right, and page numbers mentioned. In this output instead of page number as the button name, you have the thumbnail images as button label. Figure 4.2 displays the previous and the next buttons.

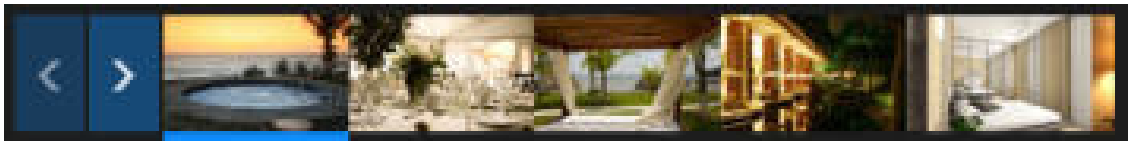


Figure 4.2: Buttons

4.4.1 App Bar Button Style Images

The styles and templates for the `AppBarButton` and the default `ControlTemplate` control can be modified to give the control a unique appearance. Table 4.2 lists the default styles of VisualState elements.

VisualState Name	VisualStateGroup Name	Description
FullSize	ApplicationViewStates	This shows Compact is false.
Compact	ApplicationViewStates	This shows Compact is true.
Normal	CommonStates	Default state.
PointerOver	CommonStates	Pointer is placed above the control.
Pressed	CommonStates	Control is pressed.
Disabled	CommonStates	Control is disabled.
Focused	FocusStates	Control has got focus.
Unfocused	FocusStates	Control doesn't have focus.

VisualState Name	VisualStateGroup Name	Description
PointerFocused	FocusStates	The control has got focus through a pointer action.

Table 4.2: Default Styles of VisualState Elements

The StandardStyles.xaml file present in the common folder contains styles and resources that give a look and feel of a Windows Store app. This includes 192 styles to use with app bar buttons. These Windows Store app provides the visual states and template for the app bar button and its styles are based on the `AppBarButtonStyle` resource. The `AppBarButtonStyle` uses a hieroglyph from the Segoe UI Symbol font. The icon is shown in the button.

By default in StandardStyles.xaml the app bar button styles are commented and these are divided into 10 groups. Before using a style, it must be uncommented and the order in which they are divided is shown here.

Following images are the button styles used in the app bar of the Windows Store apps. The styles of these buttons are set in the StandardStyles.xaml file and they are divided into 10 groups in no particular order as shown here.

Figure 4.3 displays the buttons from AV buttons.

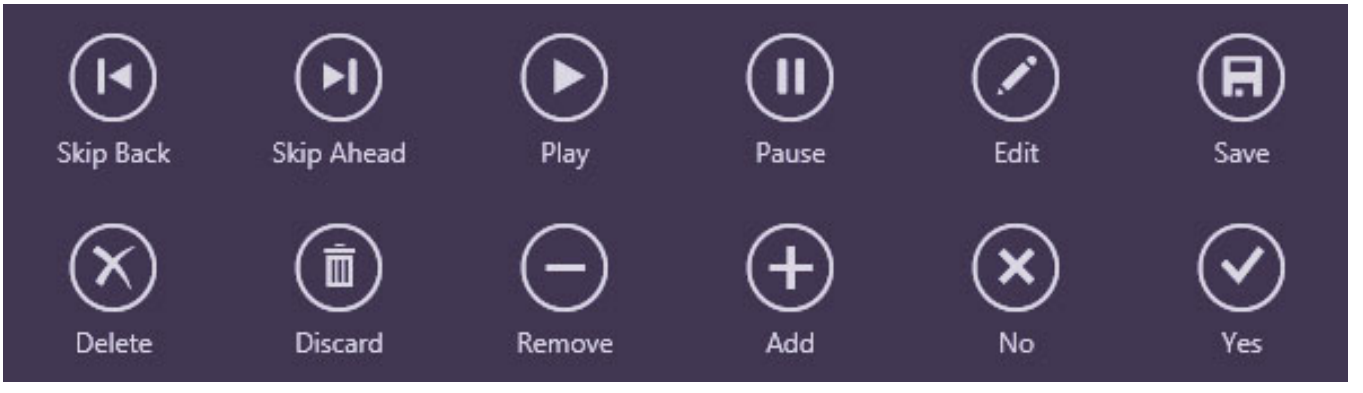


Figure 4.3: AV Buttons

Figure 4.4 displays the Internet based applications.

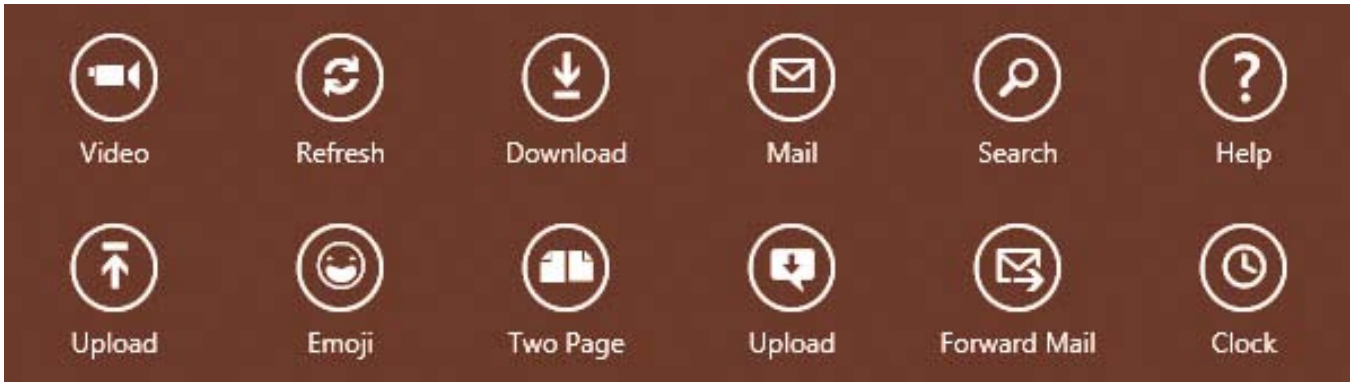


Figure 4.4: Internet-based Applications

Figure 4.5 displays the format menu.

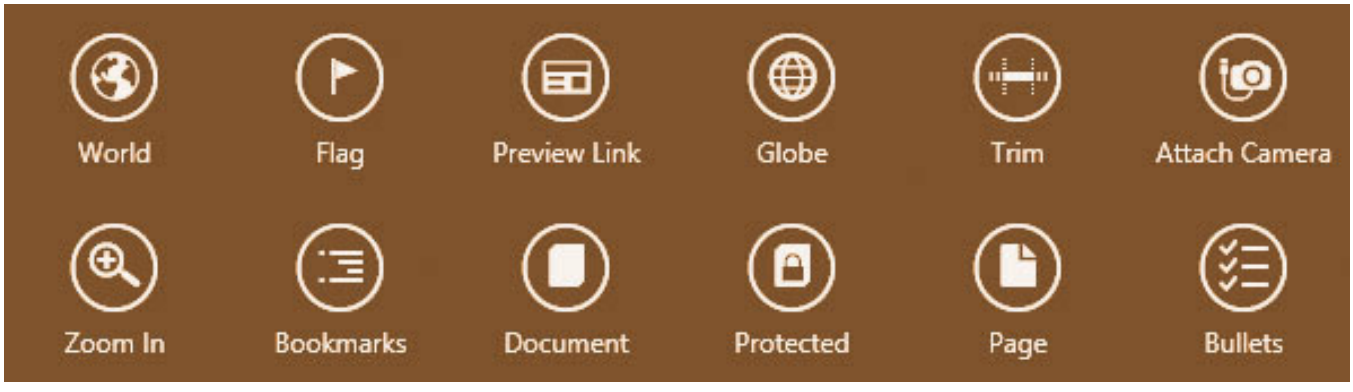


Figure 4.5: Format Menu

Figure 4.6 displays the desktop applications.

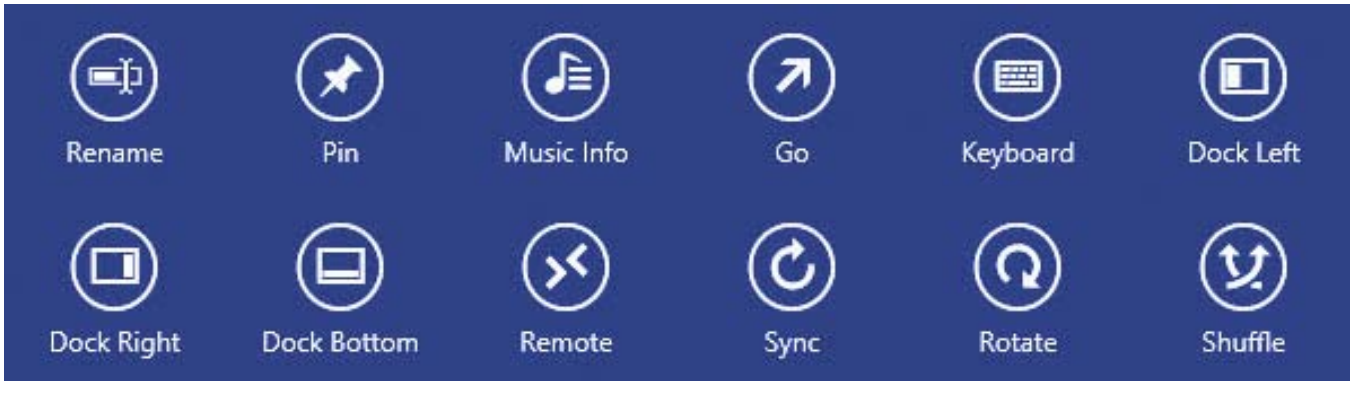


Figure 4.6: Desktop Applications

Figure 4.7 display the buttons from image.

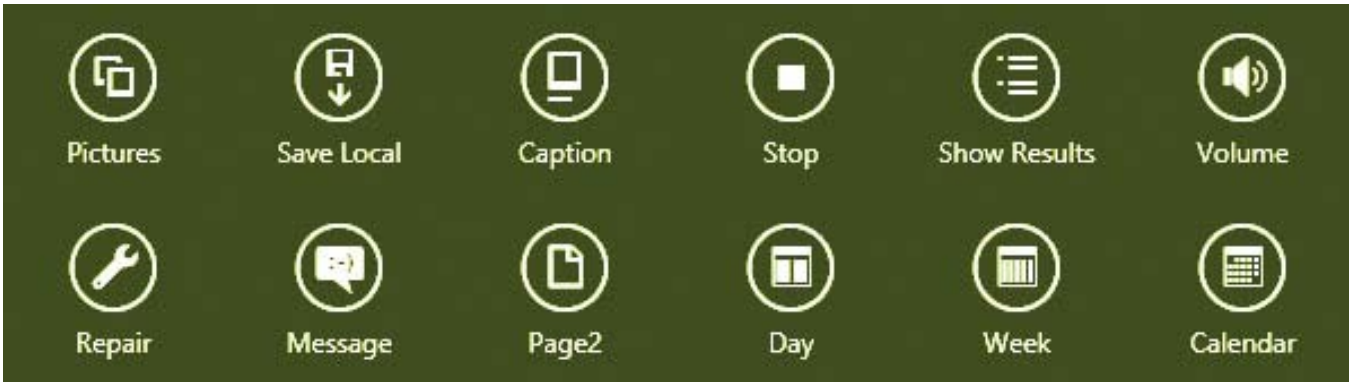


Figure 4.7: Buttons from Image

Figure 4.8 displays the Edit menu-based desktop applications.

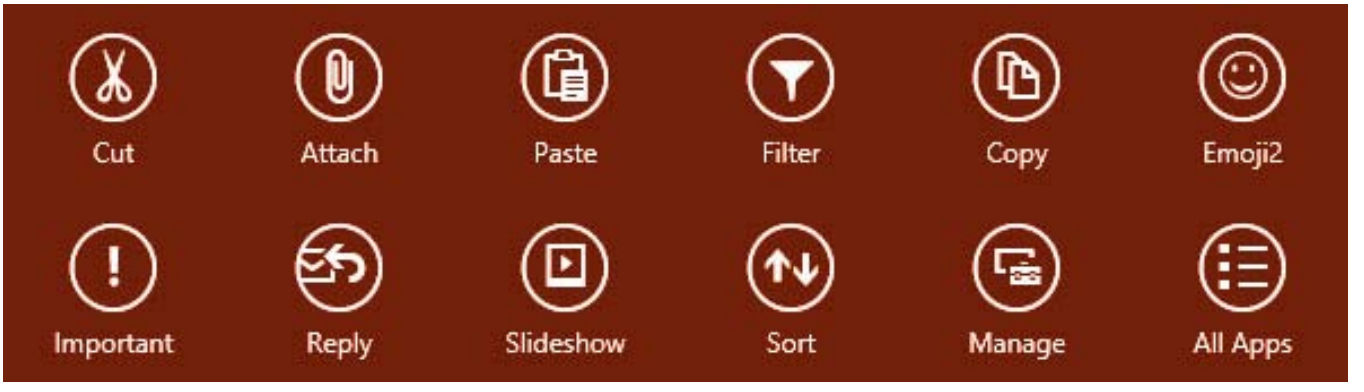


Figure 4.8: Buttons Inclusive of Edit Menu

Figure 4.9 displays the mail-based applications.

Figure 4.9: Buttons inclusive of Edit Menu, AV buttons, and Internet based Applications

Concepts

Figure 4.10 displays the format menu.

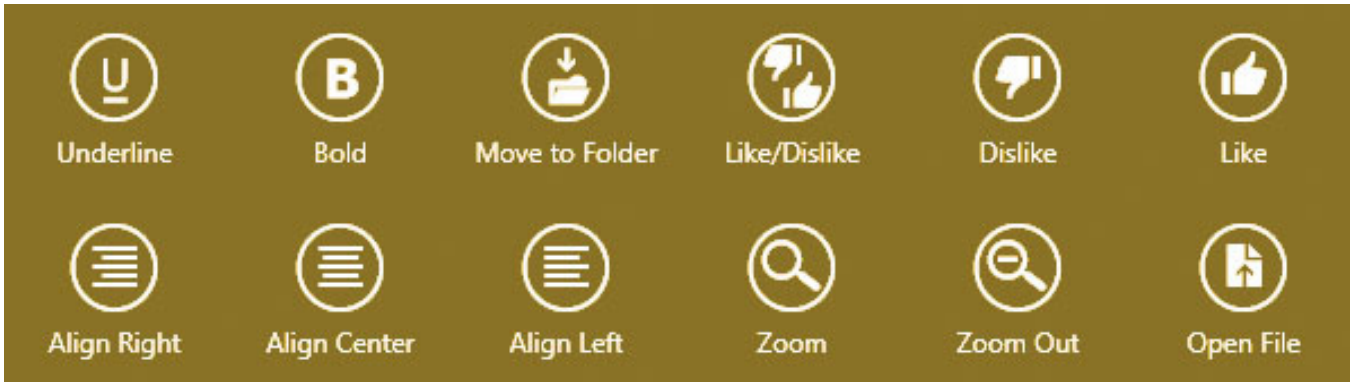


Figure 4.10: Buttons Inclusive of Edit and Format Menu

Figure 4.11 displays the buttons menu.

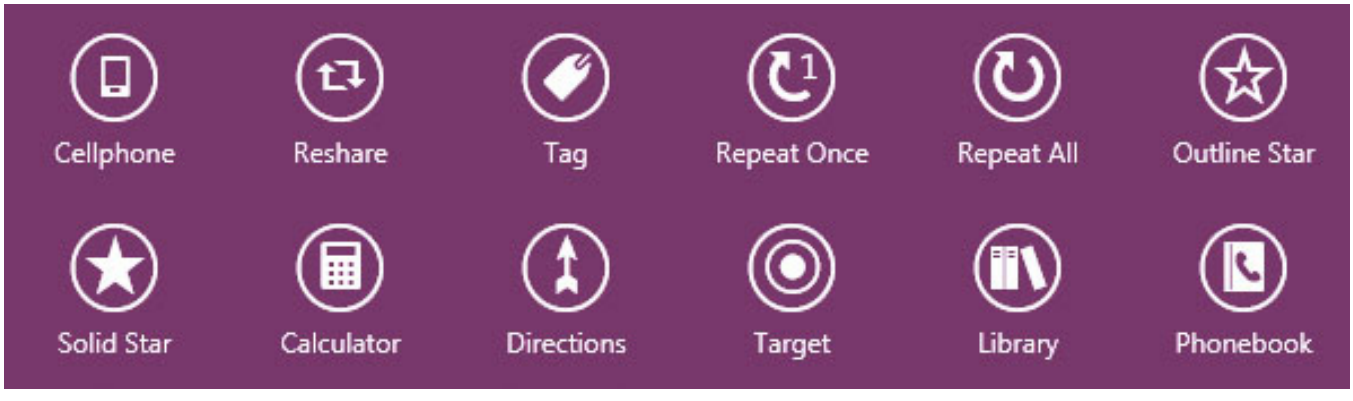


Figure 4.11: Buttons Menu

Figure 4.12 displays the chat-based applications.

Figure 4.12: Chat Menu

4.5 Guidelines and Checklist for App Bars

App bars provide the user easy access to commands by showing the commands or options that are particular to the user's different situations. When user need them they can make app bars appear by swiping and can make app bars disappear when they do not need it.

4.5.1 User Experience Guidelines

Place the commands on the canvas instead of app bars which are specific for a user to finish a workflow.

Following guidelines are used to place commands on the app bar:

- Separate the particular command groups on opposite sides of the app bar. Throughout the app keep the commands in a consistent location.
- Follow placement conventions.
- On the right place the New, Add, Create button, and icon.
- Place the view switching buttons together on far left by grouping them.
- To the left of Reject, No, Cancel place Accept, Yes, OK commands respectively.
- Show commands based on the situation when an item is selected on an app bar.
- When the right-handed people select an item on the app page, it shows the relevant commands on the left side of the app bar.
- While displaying contextual commands, set the app bars sending away mode to sticky.
- When there are too many commands then, try to use menus and flyouts.
- When it is difficult to fit all the commands as separate buttons then, group all the commands together.
- Always create commands which are simple, related, and less complex. These types of commands make an app feel simpler and comfort to use.
- Always design the app bar for snap and portrait view.
- In sticky mode reduce the height of the horizontal scrolling area which is at the bottom of the app page.

- Always use the default styles for flyouts, menus, and commands.

- Always use the navigation bar for navigation and the bottom app bar for commands. Do not put critical commands instead use the built-in icons for commands. Instead of putting account management commands in the app bars, create an account which goes in a Settings flyout.

- Clipboard commands should not be placed for text on the app bar. Instead, place editing option (Cut, Copy, and Paste) commands in a context menu.

- App settings should not be placed in the app bar and these settings, instead place it in Settings flyout.

Following guidelines to be followed while designing the controls for app:

- Use mouse button (right) for an important function directly and do not activate any contextual User Interface.

- When the user right-clicks the regions of the Microsoft DirectX surface, show the app bar.

- When the pixels of the top most horizontal row is clicked by user, bottom most horizontal row or both show the app bar.

- Use the class events of MouseDevice and Holding behavior with a mouse, which can be enabled by setting HoldWithMouse in the GestureSettings property.

- When the user press Z key along with the Windows key it displays and develop an app bar or similar context menu.

- In layout panels arrangement, focus on how the panel should be positioned and sizes its child elements.

Here are the differences of the main features of each panel such as `Canvas`, `StackPanel`, `Grid`, and `VariableSizedWrapGrid`.

- **Canvas**

  Control positioning and sizing child elements, positioning can be done by `Canvas.Top` and `Canvas.Left` attached properties.

  Child content can cross the limit of the panel and using the `Canvas.ZIndex` attached property layering can be specified.

Code Snippet 5 demonstrates the concept.

Code Snippet 5:

```
<Canvas Width=”90” Height=”00”>
<Rectangle Fill=”Red”/>
<Rectangle Fill=”Blue” Canvas.Left=”0” Canvas.Top=”0”/>
<Rectangle Fill=”Green” Canvas.Left=”30” Canvas.Top=”30”/>
<Rectangle Fill=”Yellow” Canvas.Left=”60” Canvas.Top=”60”/>
</Canvas>
```

➔ **StackPanel**

Elements can be piled in a vertical or horizontal manner and a child’s elements size stretches to fill the available width. Code Snippet 6 demonstrates the concept.

Code Snippet 6:

```
<StackPanel>
<Rectangle Fill=”Blue”/>
<Rectangle Fill=”Green”/>
<Rectangle Fill=”Red”/>
</StackPanel>
```

➔ **Grid**

Elements arranged in rows and columns are positioned by `Grid.Row` and `Grid.Column` properties.

Using `Grid.RowSpan` and `Grid.ColumnSpan` attached properties, elements can span multiple rows and columns.

Child’s element size stretches to fill the unoccupied space in the grid cell and child content does not cross the limit of the panel as shown in the following example:

Example:

```
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height=”20”/>
<RowDefinition Height=”20”/>
```

```
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
 <ColumnDefinition Width="20"/>
 <ColumnDefinition Width="20"/>
 </Grid.ColumnDefinitions>
 <Rectangle Fill="Grey"/>
 <Rectangle Fill="Blue" Grid.Row="1"/>
 <Rectangle Fill="Green" Grid.Column="1"/>
 <Rectangle Fill="Yellow" Grid.Row="1" Grid.Column="1"/>
 </Grid>
```

➔ **VariableSizedWrapGrid**

Elements are arranged in rows or columns and using `ItemHeight` and `ItemWidth` properties, elements can be sized as specified.

The Orientation property is specified by using rows or columns. Child content doesn't cross the limit of the panel.

Using `VariableSizedWrapGrid`.RowSpan and `VariableSizedWrapGrid.ColumnSpan` attached properties, elements can span multiple rows and columns. Code Snippet 7 demonstrates the concept.

Code Snippet 7:

```
<VariableSizedWrapGridMaximumRowsOrColumns="3" ItemHeight="44"
ItemWidth="44">
<Rectangle Fill="Red"/>
<Rectangle Fill="Blue" Height="80" VariableSizedWrapGrid.RowSpan="2"/>
<Rectangle Fill="Green" Width="80" VariableSizedWrapGrid.
ColumnSpan="2"/>
<Rectangle Fill="Yellow" Height="80" Width="80" VariableSizedWrapGrid.
RowSpan="2" VariableSizedWrapGrid.ColumnSpan="2"/>
</VariableSizedWrapGrid>
```

Note -

1. The ContentControl default style neither defines any visual states nor use any theme resources.
2. The ScrollViewer control can be modified the resources and the default ControlTemplate is to give the control a unique appearance.

Table 4.3 lists the default VisualStates defined.

VisualState Name	VisualStateGroup Name	Description
NoIndicator	ScrollingIndicatorStates	Control is not scrolling.
VisualState name	TouchIndicator ScrollingIndicatorStates	Control is scrolling by touch.
MouseIndicator	ScrollingIndicatorStates	Control is scrolling by mouse.

Table 4.3: Defined Default VisualState Elements

4.6 Theme Resources

These resources are used as a default style.

➔ **Dark theme brushes**

Enter these brushes in App.xaml to change the colors of the control as shown in the example.

Example

```
<SolidColorBrush x:Key="ScrollBarButtonForegroundThemeBrush" Color="#
your color"/>

<SolidColorBrush x:Key="ScrollBarButtonPointerOverBackgroundThemeBrush"
Color="#your color"/>

<SolidColorBrush x:Key="ScrollBarButtonPointerOverBorderThemeBrush"
Color="#your color"/>

<SolidColorBrush x:Key="ScrollBarButtonPointerOverForegroundThemeBrush"
Color="#your color"/>

<SolidColorBrush x:Key="ScrollBarButtonPressedBackgroundThemeBrush"
Color="#your color"/>

<SolidColorBrush x:Key="ScrollBarButtonPressedBorderThemeBrush"
Color="#your color"/>

<SolidColorBrush x:Key="ScrollBarButtonPressedForegroundThemeBrush"
Color="#your color"/>

<SolidColorBrush x:Key="ScrollBarPanningBackgroundThemeBrush"
Color="#your color"/>

<SolidColorBrush x:Key="ScrollBarPanningBorderThemeBrush" Color="#your
color"/>

<SolidColorBrush x:Key="ScrollBarThumbBackgroundThemeBrush"
Color="#your color"/>

<SolidColorBrush x:Key="ScrollBarThumbBorderThemeBrush" Color="#your
color"/>
```

```
<SolidColorBrush x:Key="ScrollBarThumbPointerOverBackgroundThemeBrush"
Color="#your color"/>

<SolidColorBrush x:Key="ScrollBarThumbPointerOverBorderThemeBrush"
Color="#your color"/>

<SolidColorBrush x:Key="ScrollBarThumbPressedBackgroundThemeBrush"
Color="#your color"/>

<SolidColorBrush x:Key="ScrollBarThumbPressedBorderThemeBrush"
Color="#your color"/>

<SolidColorBrush x:Key="ScrollBarTrackBackgroundThemeBrush"
Color="#your color" />

<SolidColorBrush x:Key="ScrollBarTrackBorderThemeBrush" Color="#your
color" />
```

These code changes the colors in the control for the Dark theme brushes. If you paste these codes in App.xml the color specified in the code will be shown for the corresponding control.

➔ **Light theme brushes**

Enter these brushes in App.xaml to change the colors of the control in the light theme as shown in the example.

Example:

```
<SolidColorBrush x:Key="ScrollBarButtonForegroundThemeBrush"
Color="#your color"/>

<SolidColorBrush x:Key="ScrollBarButtonPointerOverBackgroundThemeBrush"
Color="#your color"/>

<SolidColorBrush x:Key="ScrollBarButtonPointerOverBorderThemeBrush"
Color="#your color"/>

<SolidColorBrush x:Key="ScrollBarButtonPointerOverForegroundThemeBrush"
Color="#your color"/>

<SolidColorBrush x:Key="ScrollBarButtonPressedBackgroundThemeBrush"
Color="#your color"/>

<SolidColorBrush x:Key="ScrollBarButtonPressedBorderThemeBrush"
Color="#your color"/>

<SolidColorBrush x:Key="ScrollBarButtonPressedForegroundThemeBrush"
Color="#your color"/>

<SolidColorBrush x:Key="ScrollBarPanningBackgroundThemeBrush"
Color="#your color"/>
```

```
<SolidColorBrush x:Key="ScrollBarPanningBorderThemeBrush" Color="#your color"/>
<SolidColorBrush x:Key="ScrollBarThumbBackgroundThemeBrush" Color="#your color"/>
<SolidColorBrush x:Key="ScrollBarThumbBorderThemeBrush" Color="#your color"/>
<SolidColorBrush x:Key="ScrollBarThumbPointerOverBackgroundThemeBrush" Color="#your color"/>
<SolidColorBrush x:Key="ScrollBarThumbPointerOverBorderThemeBrush" Color="#your color"/>
<SolidColorBrush x:Key="ScrollBarThumbPressedBackgroundThemeBrush" Color="#your color"/>
<SolidColorBrush x:Key="ScrollBarThumbPressedBorderThemeBrush"
Color="#your color"/>
<SolidColorBrush x:Key="ScrollBarTrackBackgroundThemeBrush" Color="#your color" />
<SolidColorBrush x:Key="ScrollBarTrackBorderThemeBrush" Color="#your color" />
```

4.7 ScrollBar Styles and Templates

The resources and the default `ControlTemplate` can be modified to provide the control a special appearance.

➔ **Visual States**

These are the `VisualStates` defined in the controls default style.

Table 4.4 lists the scroll bar control (visual states).

VisualState Name	VisualStateGroup Name	Description
`Normal`	`CommonStates`	Default state.
`PointerOver`	`CommonStates`	Pointer is positioned on the control.
`Disabled`	`CommonStates`	Control is disabled.
`NoIndicator`	`ScrollingIndicatorStates`	Control is not scrolling.

Concepts

VisualState Name	VisualStateGroup Name	Description
TouchIndicator	ScrollingIndicatorStates	Control is scrolling by touch.
MouseIndicator	ScrollingIndicatorStates	Control is scrolling by mouse.

Table 4.4: Scroll Bar Control (Visual States)

Table 4.5 lists the visual states for the scroll bar's RepeatButtonTemplate.

VisualState Name	VisualStateGroup Name	Description
Normal	CommonStates	Default state.

Table 4.5: Scroll Bar's RepeatButtonTemplate (Visual States)

Table 4.6 lists the visual states for the scroll bar's HorizontalIncrementTemplate.

VisualState Name	VisualStateGroup Name	Description
Normal	CommonStates	Default state.
PointerOver	CommonStates	Pointer is positioned over the control.
Pressed	CommonStates	Control is pressed.
Disabled	CommonStates	Control is disabled.

Table 4.6: Scroll Bar's HorizontalIncrementTemplate(Visual States)

Table 4.7 lists the visual states for the scroll bar's HorizontalDecrementTemplate.

VisualState Name	VisualStateGroup Name	Description
Normal	CommonStates	Default state.
PointerOver	CommonStates	Pointer is positioned over the control.
Pressed	CommonStates	Control is pressed.
Disabled	CommonStates	Control is disabled.

Table 4.7: Scroll Bar's HorizontalDecrementTemplate(Visual States)

Table 4.8 lists the visual states for the scroll bar's VerticalIncrementTemplate.

VisualState Name	VisualStateGroup Name	Description
Normal	CommonStates	Default state.
PointerOver	CommonStates	Pointer is positioned over the control.
Pressed	CommonStates	Control is pressed.

VisualState Name	VisualStateGroup Name	Description
Disabled	CommonStates	Control is disabled.

Table 4.8: Scroll Bar's VerticalIncrementTemplate(Visual States)

Table 4.9 lists the visual states for the scroll bar's `VerticalDecrementTemplate`.

VisualState name	VisualStateGroup Name	Description
Normal	CommonStates	Default state.
PointerOver	CommonStates	Pointer is positioned over the control.
Pressed	CommonStates	Control is pressed.
Disabled	CommonStates	Control is disabled.

Table 4.9: Scroll Bar's VerticalDecrementTemplate(Visual States)

Table 4.10 lists the visual states for the scroll bar's `VerticalThumbTemplate`.

VisualState Name	VisualStateGroup Name	Description
Normal	CommonStates	Default state.
PointerOver	CommonStates	Pointer is positioned over the control.
Pressed	CommonStates	Control is pressed.
Disabled	CommonStates	Control is disabled.
Focused	FocusStates	Control has focus.
Unfocused	FocusStates	Control does not have focus.

Table 4.10: Scroll Bar's VerticalThumbTemplate(Visual States)

Table 4.11 lists the visual states for the scroll bar's `HorizontalThumbTemplate`.

VisualState name	VisualStateGroup name	Description
Normal	CommonStates	Default state.
PointerOver	CommonStates	Pointer is positioned over the control.
Pressed	CommonStates	Control is pressed.
Disabled	CommonStates	Control is disabled.

Table 4.11: Scroll Bar's HorizontalThumbTemplate (Visual States)

4.8 Thumb Styles and Templates

The resources and the `defaultControlTemplate` can be modified to give the control a special appearance.

4.8.1 Visual States

Table 4.12 lists the default Visual States.

VisualState Name	VisualStateGroup Name	Description
`Normal`	`CommonStates`	Default state.
`PointerOver`	`CommonStates`	Pointer is positioned over the control.
`Pressed`	`CommonStates`	Control is pressed.
`Disabled`	`CommonStates`	Control is disabled.
`Focused`	`FocusStates`	Control has focus.
`Unfocused`	`FocusStates`	Control does not have focus.

Table 4.12: Default Visual States

4.9 Check Your Progress

1. Name the function of `CommandBar` and `AppBar` controls.

(A)	Helps to display keywords	(C)	Helps to display commands
(B)	Helps to display UI	(D)	Helps to display code

2. Which element helps to present more options in less space and provides interactive controls?

(A)	Menu	(C)	Controls
(B)	Command	(D)	Button

3. Which control is used to add navigate between pages?

(A)	Control	(C)	Commands
(B)	Menu	(D)	Button

4. Which control provides the user with easy access to commands and helps to show commands or options?

(A)	Command bar	(C)	Control
(B)	Menu	(D)	App bar

5. In which side the view switching buttons should be placed?

(A)	Left	(C)	Top
(B)	Right	(D)	Bottom

4.9.1 Answers

1.	B
2.	A
3.	C
4.	D
5.	A

Summary

- Apps can be shown or make it hidden when the user presses `Windows+Z`, right-clicks or swipes from the edge (top or bottom) of the screen.
- `CommandBar` and `AppBar` controls helps to display UI that appears at the edge (top or bottom) of the screen which appears by swiping on an edge.
- Commands can be added to a `CommandBar` which apply when a `CommandBar` between pages are shared or an item is selected.
- Menu helps to present more options in less space and it provides interactive controls.
- Commands are required to add navigate between pages.
- The styles and templates for the `AppBarButton` and the default `ControlTemplate` control can be modified to give the control a unique appearance.
- The StandardStyles.xaml file present in the Common folder contains styles and resources that give a look and feel of a Windows Store app.
- App bars provide the user easy access to commands by showing the commands or options that are particular to the user's different situations.
- Always use the navigation bar for navigation and the bottom app bar for commands. Use the built-in icons for commands and do not put critical commands.

Session - 5
Data Binding

Welcome to the Session, **Data Binding**.

The session will discuss how to achieve data binding. It will also discuss the methods and techniques to achieve various customizations of objects and collection of objects.

In this Session, you will learn to:

- Define data binding
- Describe how to data bind a control to a single object
- Explain the process of Binding a control with a collection of objects
- Explain the use of a data template
- Describe how to add a details view
- Explain the process of data binding with XAML

5.1 Introduction

Data binding based on C#, Visual Basic, or C++ is a method for Windows Store apps to display data and interact with data. The way data displays is different from the way data is managed.

A binding or a connection between the UI and a data object enables exchange of data. When a binding is confirmed and the data starts flowing, the UI elements bound to the data automatically show changes. Similarly, a user can change the UI element and save it in the data object.

5.2 Data Binding of a Control to a Single Object

Data binding includes a target and a source. A target can be a property of a control, and a source is the property of a data object.

Code Snippet 1 shows the XAML code.

Code Snippet 1:

```
<Grid Background="#FFDC6767">
   <Grid x:Name="MyLayoutRoot">
    <Grid.Background>
     <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">
      <GradientStop Color="Black"/>
      <GradientStop Color="#FFB0F8A8" Offset="1"/>
     </LinearGradientBrush>
    </Grid.Background>
    <TextBox x:Name="myTextBox" Text="{Binding}" FontSize="30"
IsReadOnly="True" TextWrapping="Wrap" AcceptsReturn="True"
Margin="141,324,202,342" />
   </Grid>
 </Grid>
```

Code Snippet 2 shows the C# code.

Code Snippet 2:

```
namespace TestDataBinding
{
 public sealed partial class MainPage : Page
 {
  public MainPage()
```

Concepts

```
InitializeComponent();
    //Set the data context to the content that needs to be shown.
    myTextBox.DataContext = new Employee("Andy Stallone", "Washington",
      new DateTime(2012, 6, 12));
  }
  //create an employee object
  public class Employee
  {
    public Employee(){}
    public Employee(string MyEmployeeName, string MyEmployeeAddress, DateTime
MyJoiningDate)
    {
      EmpName = MyEmployeeName;
      EmpAddress = MyEmployeeAddress;
      EmpDateOfJoining = MyJoiningDate;
    }
    public string EmpName { get; set; }
    public string EmpAddress { get; set; }
    public DateTime EmpDateOfJoining { get; set; }
    //Override the ToString method.
    public override string ToString()
    {
      return EmpName + " Lives in " + EmpAddress + " Joined the Company From: " +
EmpDateOfJoining.ToString("d");
    }
  }
 }
}
```

This code helps the developer bind a data to one single object. Here, the target is the text property of the `TextBox` control and source is a simple `Employee` class. XAML is used for the binding and uses the syntax `{Binding}`. The source is set in code by using the `DataContext` property of the text box. You set our employee details using this property. The arguments of the `DataContext` method is the employee details which is passed to the `Employee` class and shown in textbox. The text box created in

XAML code can show the text initialized in C# class through the data binding concept.

Figure 5.1 displays the data binding of a control to a single object.

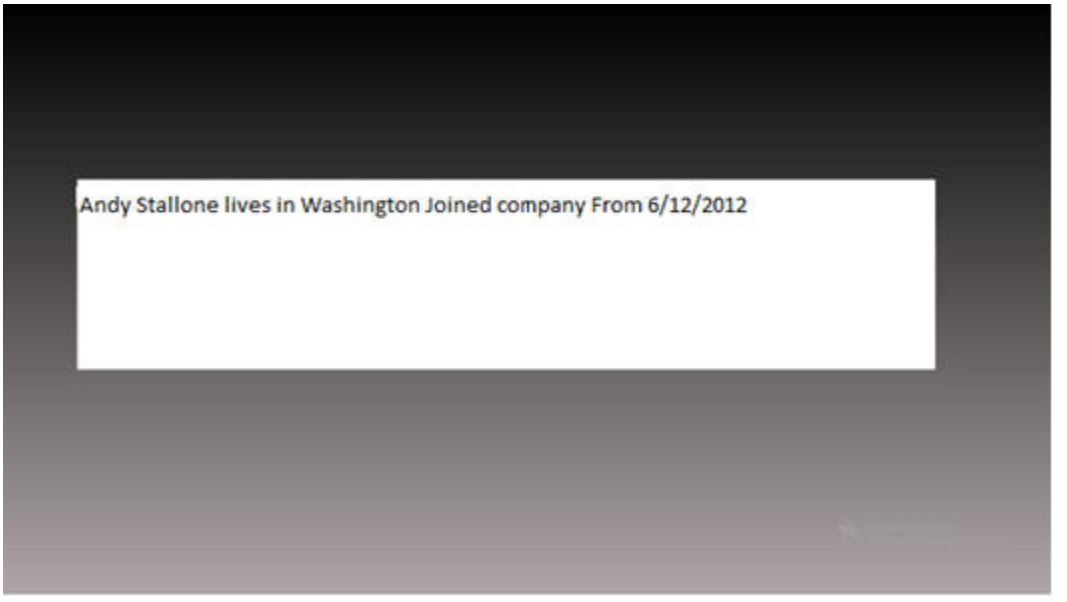


Figure 5.1: Data Binding

5.3 Binding a Control with a Collection of Objects

It is common to bind to a business objects collection. In C# and `Visual Basic`, the generic class `ObservableCollection<T>` has more than one option for data binding and it implements the interfaces: `INotifyPropertyChanged` and `INotifyCollectionChanged`.

If an item changes or if the properties of a list change, the interfaces inform the changes to the bound controls. If the user wants to update the bound controls with the described changes, then, the user should implement the interface `INotifyPropertyChanged`.

Code Snippet 3 demonstrates the binding of a control to a collection of objects. The XAML code defines a combo box control called `cb01` in a grid. The `ItemSource` property of the combo box is mapped to binding. The source of the binding is the combo box data. This is called from the `MyAlbum` collection defined in the C# code in Code Snippet 4. Here the target is the `ItemSource` property of the combo box and source is the `MyAlbum` collection.

Code Snippet 3:

```
<Grid x:Name="myGrid" Grid.Row="1" Margin="12,0,12,0">

  <ComboBox x:Name="cb01" ItemsSource="{Binding}" Foreground="Black"
FontSize="30" Height="50" Width="780"/>

</Grid>
```

Concepts

```
public ObservableCollection<Album> MyAlbum = new
ObservableCollection<Album>();
public Page()
{
InitializeComponent();
 //Adding items to a collection.
 MyAlbum.Add(new Recording("Unknown 01", "Title X", new DateTime(2013, 5, 1)));
 MyAlbum.Add(new Recording("Unknown 02","Title Y", new DateTime(2013, 5, 2)));
 MyAlbum.Add(new Recording("Unknown 03","Title Z", new DateTime(2013, 5, 3)));
 //Set the data context for a combo box.
 cb01.DataContext = MyAlbum;
}
```

In Code snippet 3, C# code a simple `MyAlbum` collection is defined with its individual records and assigned to the `DataContext` property of the combobox control. Here the `MyAlbum` is the source of the binding.

Figure 5.2 shows the output.

Figure 5.2: Combo Box

5.4 Using a DataTemplate to Display Items in a Control

By using the `ToString` method of an item, the user can display the items as a list. More commonly, the user can use `DataTemplate` to provide a custom display of data bound items. `DataTemplate` enables the user to decide how the list items will display in a control. The user can use the `ContentTemplate` property in a content control to customize the data template. The user can also customize the `ItemTemplate` property in an items control.

Here is an example that uses a data template to list the recordings that are bound to a combo box.

Concepts

Remember, a combo box is an `ItemsControl`. It means that the user can set the `ItemTemplate` property to a data template to establish a data template for each item.

Code Snippet 4 demonstrates the binding of a control to a collection of objects.

Code Snippet 4:

```
<ComboBox x:Name="cb01" ItemsSource="{Binding}"
 Foreground="Black" FontSize="30" Height="50" Width="450">
 <ComboBox.ItemTemplate>
  <DataTemplate>
   <StackPanel Orientation="Horizontal" Margin="2">
    <TextBlock Text="Artist:" Margin="2" />
    <TextBlock Text="{Binding Artist}" Margin="2" />
    <TextBlock Text="CD:" Margin="10,2,0,2" />
    <TextBlock Text="{Binding Name}" Margin="2" />
   </StackPanel>
  </DataTemplate>
 </ComboBox.ItemTemplate>
</ComboBox>
```

5.5 Adding a Details View

The user can display the details of an item after selecting it from a collection. They should develop the relevant UI, bind the UI with the data they want it to display. Additionally, if the user wants to bind to the current item to enable the details view, they must use the data context, CollectionViewSource.

Here is an example that lists the recordings. The list is the source of data for the instance: `CollectionViewSource`. The user control or the data context for the whole page is set to the collection view source. The combo box and the details view inherit the data context. This binds the combo box to the collection. The same list of items display and the details view binds with the current item, automatically. The current item and the details view need not bind because the collection view source will provide the appropriate level of data.

Code Snippet 5 shows the example for the list.

Code Snippet 5:

```
<!--The UI for the details view-->
<StackPanel x:Name="AlbumDetails">
 <TextBlock Text="{Binding Artist}" FontWeight="Bold" FontSize="30" />
```

Concepts

```
<TextBlock Text="{Binding Name}" FontStyle="Italic" FontSize="30" />
<TextBlock Text="{Binding ReleaseDate}" FontSize="30" />
</StackPanel>
this.DataContext = new CollectionViewSource {Source = MyAlbum};
```

5.6 Converting Data for Display in Controls

The user can use a converter to modify and show a non-string type in a control. An example is a `TextBox` and instead of displaying only a date, the user can display a label with a modified date.

A converter is a class derived from the `IValueConverter` interface. This interface has two methods: `ConvertBack` and `Convert`. The user should implement the `Convert` method to achieve a one-way binding with the data source and the binding target. In this example, the converter is generic. The user can pass this desired string format like a parameter. The converter then, uses the String.Format method to convert. If the format string is not passed, the converter returns the output of the call to `ToString` on the object.

Once the converter is implemented, the user creates an instance of the converter class. This means the bindings are instructed to utilize the instance. Here, in the example, it is done in XAML. The user creates an instance of the converter as a static resource and assigns a key. The key is then, utilized when the property of the converter is mapped with binding.

Code Snippet 6 demonstrates the example by using XAML.

Code Snippet 6:

```
<UserControl x:Class="ConvertingData"
 xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
 xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
 xmlns:local="using: ConvertingData "
 mc:Ignorable="d"
 d:DesignHeight="768" d:DesignWidth="1366">
 <UserControl.Resources>
<local:StringFormatter x:Key="NewStringConverter"/>
 </UserControl.Resources>
```

```
  <Grid x:Name="NewLayoutRoot" Background="#FF0C0C0C">
 <StackPanel Width="750" Height="200"
  VerticalAlignment="Center" HorizontalAlignment="Center">
   <ComboBox x:Name="NewComboBox" ItemsSource="{Binding}"
  Foreground="Black" FontSize="30" Height="50" Width="750">
   <ComboBox.ItemTemplate>
    <DataTemplate>
     <StackPanel Orientation="Horizontal" Margin="2">
      <TextBlock Text="Artist:" Margin="2" />
      <TextBlock Text="{Binding Artist}" Margin="2" />
      <TextBlock Text="CD:" Margin="10,2,0,2" />
      <TextBlock Text="{Binding Name}" Margin="2" />
     </StackPanel>
    </DataTemplate>
   </ComboBox.ItemTemplate>
  </ComboBox>
  <!--The UI for the details view-->
  <StackPanel x:Name="NewRecordingDetails">
   <TextBlock Text="{Binding Artist}" FontSize="30" FontWeight="Bold" />
   <TextBlock Text="{Binding Name}" FontSize="30" FontStyle="Italic" />
   <TextBlock Text="{Binding ReleaseDate,

          Converter={StaticResource StringConverter},
          ConverterParameter=Released: \{0:d\}}" FontSize="30"  />
      </StackPanel>
    </StackPanel>
    </Grid>
</UserControl>
```

Code Snippet 7 demonstrates the C# code that converts the object to string in display:

Code Snippet 7:

```
public class StringFormatter : IValueConverter


{
 // This converts the value object to the string to display.
 // This will work with most simple types.
 public object Convert(object value, Type targetType,
  object parameter, System.Globalization.CultureInfo culture)
 {
  // Retrieve the format string and use it to format the value.
  string NewFormatString = parameter as string;
  if (!string.IsNullOrEmpty(NewFormatString))
  {
   return string.Format(culture, formatString, value);
  }
  // If the format string is null or empty, simply
  // call ToString() on the value.
  return value.ToString();
 }
 // No need to implement converting back on a one-way binding
 public object ConvertBack(object value, Type targetType,
  object parameter, System.Globalization.CultureInfo culture)
 {
  throw new NotImplementedException();
 }
}
```

5.6.1 Connecting UI Elements with Data

Every binding consists of the following:

➔ **Binding Source**: An object with data that the user has to render. The source can be one of the

two options described here:

- **Common Language Runtime (CLR) Object**: The CLR object can include the target element and other elements of the UI. The UI element can be the source on the user applies the template in case the target is included within a data template. The classes that the user defines in Visual Basic and C# result in CLR objects; hence, these can be bound by default.
- **Windows Runtime Object**: An object that implements `ICustomPropertyProvider` or has a `BindableAttribute`. The user defines classes in C++ that produce Windows Runtime objects and they require one of these methods to be bindable.

➔ **Binding Target**: It is a `DependencyProperty` of a `FrameworkElement` to render the data. The target can be a `DependencyProperty` of a `FrameworkElement`.

➔ **Binding Engine**: The binding engine procures information from the Binding object about the source, target objects, direction of data flow, and other settings.

- The source
- The target objects
- The direction of data flow: the user can set the Binding.Mode property to control the direction
- The value converter, if available: the user can specify a value converter and then map the Converter property to a class instance that will implement `IValueConverter`
- Other settings: `FallbackValue` and `TargetNullValue`. For all the properties, refer to the `Binding` class. For instance, the `Foreground` property of a `TextBox` and `SolidColorBrush` can be bound. The color of the text may change based on this data. Here, the `Foreground` property is the target. The `SolidColorBrush` object is the binding source

Code Snippet 8 shows the related code snippet for this section.

Code Snippet 8:

```
<TextBox x:Name="ColorTextBox" Text="Text" SelectionBrush="{Binding
HighlightBrush}"/>

//Create an instance of the MyColors class

//that implements INotifyPropertyChanged.

MyColors selectioncolor = new MyColors();

//HighlightBrush is set to be a SolidColorBrush with the value Green.

selectioncolor.HighlightBrush = new SolidColorBrush(Colors.Green);

//Set the DataContext of the TextBox ColorTextBox.

ColorTextBox.DataContext = selectioncolor;
```

Concepts

In the code example, the highlighted selection will appear in Green color. The SelectionBrush property of the textbox is bound to SolidColorBrush. Whenever the user uses the textbox named ColorTextBox in a code example, the highlighted selection will be in Green color.

The binding is created in XAML and uses the syntax {Binding ...}. The source is set in code by customizing the DataContext property for TextBox.

The Data context is inherited. If the user sets the data context on a parent element, the children will also use this data context. The relevant child element can set the Source property on its binding object to override this. A child element can also override by setting its DataContext. This override rule is applicable to the children of the described mentioned child element also.

The user should set the data context when they want more than one binding that uses the same source. To map the source for a single binding, they have to enable the property: Source on the Binding object.

The user can use the properties ElementName or RelativeSource to set the binding source. The property: ElementName is useful when the user is binding to other elements in the app. For example, a user adjusts the width of a button by using a slider. The RelativeSource property is useful when the user specifies the binding using a ControlTemplate. Refer to RelativeSource markup extension and the Binding markup extension.

By setting the Binding.Path property, the user can bind to a property in the source object. The property Path will support many syntax options to bind to attached properties, nested properties, and string and integer indexers. To get more information, refer to the Property-path syntax. The result of binding to dynamic properties and not having to implement the ICustomPropertyProvider is achieved by binding to string indexers.

The data binding engine converts the value to a string if the user binds a text control to a non-string value. If this value is a reference type, then the data binding engine calls either ICustomPropertyProvider.GetStringRepresentation or IStringable.ToString if available else calls Object.ToString to retrieve the string value.

The user should remember that the binding engine ignores the implementation of ToString that hides the base-class implementation. Typically, the Subclass implementations override the base class ToString method. Similarly, in unmanaged languages, ICustomPropertyProvider and IStringable are implemented by all managed objects.

All calls to IStringable.ToString and GetStringRepresentation get routed to Object.ToString or an override. The calls are never routed to a new implementation of ToString that masks the base-class implementation.

Windows Runtime in C++ treats all managed objects as ICustomPropertyProvider implementations. Despite the fact that the earlier example enable the binding engine to create a binding. This is a one-way binding by default. The binding connects the Brush1 property of the textcolor object with the Foreground property of the Text Box.

5.6.2 Creating Bindings in Code

The user can also use procedural code instead of XAML to connect the UI elements to data. To do this, the user has to create a new Binding object and set the appropriate properties. Then, the user has to call `FrameworkElement.SetBinding` or `BindingOperations.SetBinding`. Remember that creating bindings using code is useful when, during run time, the user wants to choose the binding property values. This is also useful when the user has to share a single binding with multiple controls. Note, that the user cannot change the binding property values after calling `SetBinding`.

Code Snippet 9 shows the code that helps set the binding for the `ColorTextBox` example.

Code Snippet 9:

```
//create binding and associate it with text box
Binding binding = new Binding() { Path = new PropertyPath("HighlightBrush") };
ColorTextBox.SetBinding( TextBox.SelectionBrush, binding );
```

Here the syntax `PropertyPath` is built into the MSDN class and helps define the property value for the control.

5.6.3 Direction of the Data Flow

The Mode property in every binding controls the data flow. The Windows Store apps that use C++, C#, or `Visual Basic` typically use these three types of bindings:

- **OneTime bindings**: When the binding is created, the user can use these to update the target with the source data.
- **OneWay bindings**: This is a default mode. When the binding is created or any time the data changes, the user updates the target with the source data.
- **TwoWay bindings**: If the source or the target changes, the user can update them. After every keystroke, the modifications to `TextBox.Text` are not updated in a bound source. Only when the `TextBox` loses focus, the changes get updated.

The user can change the behavior of `TwoWay` bindings to avoid copying of values to the source automatically. The user can decide and set the time. The user can change the Binding.UpdateSourceTrigger property and make it explicit. The syntax `GetBindingExpression` is used for the target. This action results in a `BindingExpression` object. The user then, calls `BindingExpression.UpdateSource` to update the data source in code.

For instance, if the user wants to update a data source they can use the event handler called `TextBox.TextChanged`. This handler is bound to a `TextBox.Text` property.

5.6.4 Notification of Change

If the user wants the source object modifications to reflect in the target, they must ensure that the source object implements the `INotifyPropertyChanged` interface.

The `INotifyPropertyChanged` contains the `PropertyChanged` event, which informs the binding engine about the change in the source. The binding engine updates the target value.

For Visual Basic or C#, the user must implement `System.ComponentModel.INotifyPropertyChanged`.

For Visual C++ component extensions (C++/CX), the user must implement `Windows::UI::Xaml::Data::INotifyPropertyChanged`. Code Snippet 10 demonstrates the example for this section.

Code Snippet 10:

```
//Create a class that implements INotifyPropertyChanged.
public class MyColors : INotifyPropertyChanged
{
 private SolidColorBrush _Brush1;
 //Declare the PropertyChanged event.
 public event PropertyChangedEventHandler PropertyChanged;
 //Create the property that will be the source of the binding.
 public SolidColorBrush Brush1
 {
get { return _Brush1; }
  set {
   _Brush1 = value;
   //Call NotifyPropertyChanged when the source property
   //is updated.
   NotifyPropertyChanged("Brush1");
  }
 }
get { return _Brush1; }

set {
            _Brush1 = value;
            // Call NotifyPropertyChanged when the source property
            // is updated.
```

```
        NotifyPropertyChanged("Brush1");
    }
  }
// NotifyPropertyChanged will fire the PropertyChanged event, passing the source
property that is being updated.
  public void NotifyPropertyChanged(string propertyName)
{
    if (PropertyChanged != null)
    {
      PropertyChanged(this,
        new PropertyChangedEventArgs(propertyName));
    }
  }
}
```

The user can run the event: `PropertyChanged` to inform the change in all the object properties. For this, for the `PropertyName` property of the `PropertyChangedEventArgs` the user uses `String.Empty`. The user cannot use a null value for such a thing in Microsoft Silverlight and Windows Presentation Foundation (WPF).

Note- For Visual Basic or C#, the user uses `System.ComponentModel.PropertyChangedEventArgs`.

For `C++/CX`, the user uses `Windows::UI::Xaml::Data::PropertyChangedEventArgs`.

The user can also use the change notification option with indexer properties. They can run the `PropertyChanged` event and by using `PropertyName` show that the indexer properties on the object have changed. They can set the Item [indexer] value for special indexers. In this, for all indexers the indexer is an index value or Item [] value. At present, C++ will not support data binding with indexers.

5.6.5 Binding to Collections, Folder, and File Lists

The user can treat a binding source object either like a single object. The object's properties includes data, or as a collection. For example, they can display a list, such as telephone bills. The user can utilize `ItemsControl` and use the `DataTemplate` to show each item in a collection.

Code Snippet 11 shows the code snippet for this section.

Code Snippet 11:

```
<Grid.Resources>
 <DataTemplate x:Name="dataTemplate">
```

```
<Grid>
 <Grid.ColumnDefinitions>
  <ColumnDefinition />
  <ColumnDefinition />
 </Grid.ColumnDefinitions>
 <TextBlock Grid.Column="0"
  Text="{Binding Month, Converter={StaticResource Converter1}}"/>
 <TextBlock Grid.Column="1" Text="{Binding Total}"/>
</Grid>
</DataTemplate>
</Grid.Resources>
<ItemsControl x:Name="IC1" ItemsSource="{Binding}"
ItemTemplate="{StaticResource dataTemplate}"/>
```

In Visual Basic and C#, to display an unmodified object collection, the user can update and bind to List (of T). If the items are modified in the collection at run time, and they want the target to update the ItemsSource at that time, they have to use ObservableCollection(Of T) instead.

If using C++ code, for both changing and unchanging collections, they can populate and bind to the Vector<T> instances.

5.6.6 Loading Data Incrementally

The user can use incremental loading to bind the list controls to large sources of data, and get high performance. To avoid loading the image results at the same time, they can bind the list controls to the image query results from Bing. Instead, they can load only a few results and load other results later. The user can support incremental loading and implement ISupportIncrementalLoading on a data source that will support the change notification in the collection. When the binding engine of the data asks for more data, the user can ensure that the data source makes the required requests. Then, they should integrate the results before sending the required notifications for updating the UI.

5.6.7 Grouping Data and Tracking the Current Item

To show the collections of collections, the user can also bind to the class instances called CollectionViewSource. They can also keep track of the current item in several views. They must define a CollectionViewSource as a XAML resource. They must use the StaticResource markup extension to bind to it. They can then, set the Source property programmatically at the backend to a supported collection type.

The user must remember that any controls that they bind to one CollectionViewSource will retain the same current item. Bindings with the Path settings will automatically bind with the properties of

the current item. The user accesses the current item through code using the `ICollectionView.CurrentItem` property. This is connected to `CollectionViewSource.View` property value.

The user can display the collections as groups within a bigger collection if the items in a collection are collections, or are objects containing collections. For this, the user must set the `IsSourceGrouped` property as true.

If the items include collections, the user should map the property `ItemsPath` to the collection `PropertyName`. If the user binds with a `CustomerList Customer` objects collection and if there is a Customer class with an Orders property, which is a collection of Order objects, the user can group orders by customer. The user can use the code for this.

Code Snippet 12 shows the relevant code.

Code Snippet 12:

```
<local:CustomerList x:Key="CustomerData"/>

<CollectionViewSource x:Name="Customers" Source="{StaticResource
CustomerData}" IsSourceGrouped="True" ItemsPath="Orders"/>
```

The user can develop a group header template to bind to the properties of the Customer class. The user can create an item template to binds to the properties of the Order class. To access, they can use the property `CollectionViewSource.CollectionGroups`.

This example shows how to bind a `ListBox` control with the results of a grouping `LINQ` query. Here, user can sort a collection of teams and show the name of a specific city in the group header. This example is shown by the property path 'Key' with reference to the Key value group. Refer to the data binding sample for the list of code.

Code Snippet 13 shows the code example.

Code Snippet 13:

```
<Grid>
 <Grid.Resources>
  <CollectionViewSource x:Name="groupInfoCVS" IsSourceGrouped="true"/>
 </Grid.Resources>
 <ListBox x:Name="lbGroupInfoCVS"
  ItemsSource="{Binding Source={StaticResource groupInfoCVS}}">
  <ListBox.GroupStyle>
   <GroupStyle>
    <GroupStyle.HeaderTemplate>
<DataTemplate>
 <TextBlock Text="{Binding Key}" />
```

Concepts

```
    </DataTemplate>
   </GroupStyle.HeaderTemplate>
  </GroupStyle>
 </ListBox.GroupStyle>
  <ListBox.ItemTemplate>
   <DataTemplate>
<Border Background="{Binding Color}"
    Width="200" CornerRadius="10" HorizontalAlignment="Left">
    <TextBlock Text="{Binding Name}"
     Style="{StaticResource DescriptionTextStyle}"
     HorizontalAlignment="Center" FontWeight="Bold"/>
</Border>
  </DataTemplate>
 </ListBox.ItemTemplate>
</ListBox>
</Grid>
Teams myTeams = new Teams();
var result =
 from team in myTeams
 group team by team.City into myGroup
 orderby myGroup.Key
 select myGroup;
groupInfoCVS.Source = result;
```

The user can also show these levels like a list series when they bind to multi-level data like categories with subcategories. If the user selects an item in one list, it will affect the contents of all the subsequent lists. The user can bind the `CollectionViewSource` instances together in a chain and each list to its own CollectionViewSource to synchronize all the lists.

5.6.8 Data Conversions

The user may have to display data in a different format. This format may be different from the way it was originally stored. Here are some examples:

- Storing color as an RGBA value; but displays it as a string name.
- Storing the number like a floating point value; but displays it like currency value.
- Storing the date as `DateTime`, but displays it within a calendar.
- Storing the null value; but displays a friendly default value. In Windows 8.1, the user can set the `TargetNullValue` property and for null backing values display a friendly default value.

Remember, that the user sets a converter on a data binding. To modify the converter for every instance, the user has to develop a class and implement the interface: `IValueConverter`.

If the Converter parameter is set for a binding, the binding engine calls the methods: `Convert` and `ConvertBack`. When data passes from a source, the binding engine calls `Convert`. Then, the engine passes this returned data to the target. When this data is passed from a target, the binding engine calls `ConvertBack`. Then, the engine passes this returned data back to the source.

The converter has an optional parameter, `ConverterLanguage`. This allows the user to specify the language for the conversion. The `ConverterParameter` allows a parameter for the logic of conversion. For example, refer to `IValueConverter` for using the converter parameter.

Do not throw an exception in case there is a conversion error. To stop the transfer of data, return `DependencyProperty.UnsetValue`. Customize the `FallbackValue` property, and display a default value if the user cannot resolve the binding source. It will be useful to manage conversion and any errors in formatting. It is also helpful to bind to properties of source that may not be available for all objects within a bound collection.

5.6.9 Debugging Data Bindings

The user can use Microsoft Visual Studio to debug the data bindings. When the user runs his/her apps with the debugger attached, the binding errors are listed in the Output window in Visual Studio. The user will find this feature useful when they rename properties in their data classes, but do not update the binding expressions of XAML.

5.6.10 Data Display in the Designer

The user should utilize the designer and develop some sample data with data-bound UI. The user can view layout sizes accurately and real outputs to size automatically.

For displaying data within the designer, the user should declare it in the XAML and need not set the `DataContext` property at the backend in code. This is important as and when the user opens a page or user control using the designer, this is not visible. The designer will parse the XAML. However, the designer will not run its code at the backend.

5.6.11 Creating a Master-Details Binding

The user can bind list controls to instances of `CollectionViewSource`. The user can create several levels of master-details view containing data in a hierarchy.

The user can use the Windows Store apps to navigate to different details pages while making a selection in a master list. This feature will be useful when the user wants at every level in a hierarchy, richer visuals for each item. In Visual Studio, the Grid Application and Split Application templates show the user how to use it.

The user can also display, on a single page, several data levels. The technique is useful, when the user wants to display simple lists that allow other users to quickly drill down to an item. This topic uses multiple `CollectionViewSource` instances to explain the implementation of this technique. For each level, these instances keep a check on the current selection.

5.7 Check Your Progress

1. _____ allows Windows Store Apps to display and interact with data.

(A)	Data binding	(C)	Data connection
(B)	Data object	(D)	Visual Basic object

2. In C# and Visual Basic, the generic class _____ includes a good collection choice for data binding.

(A)	INotifyPropertyChanged	(C)	ContentTemplate
(B)	ToString	(D)	ObservableCollection<T>

3. Which of these does a user need if they wish to display items in a list?

(A)	DataTemplate	(C)	ContentTemplate
(B)	ToString method	(D)	ItemsControl

4. Match the items with their descriptions.

Item		Description	
(a)	DataTemplate	1.	Is a class derived from the IValueConverter interface
(b)	ContentTemplate	2.	An object with data that the user has to render
(c)	Converter	3.	Provide a custom display of data bound items
(d)	Binding source	4.	Used to customize the data template

(A)	a-1, b-2, c-3, d-4	(C)	a-3, b-4, c-1, d-2
(B)	a-2, b-3, c-4, d-1	(D)	a-4, b-2, c-3, d-1

5. Which of these is the result when there is an error in conversion?

(A)	Return DependencyProperty.UnsetValue	(C)	Return ConverterParameter
(B)	Return FallbackValue	(D)	Return ConvertBack

Concepts

6. The target can be a _____ of a FrameworkElement.

(A)	Windows Runtime object	(C)	DependencyProperty
(B)	Binding engine	(D)	FallbackValue

7. Which of these is true for the XAML value for Binding?

(A)	`{Binding…}`	(C)	`Property-path`
(B)	`Binding.Path`	(D)	`ToString`

8. Which of these represent the three types of bindings?

(A)	OneTime, OneWay, TwoWay	(C)	OneTime, TwoTime, TwoWay
(B)	OneWay, OneTime, TwoTime	(D)	TwoWay, OneTime, TwoTime

5.7.1 Answers

1.	A
2.	D
3.	B
4.	C
5.	A
6.	C
7.	A
8.	A

Summary

- Data binding based on C#, Visual Basic, or C++ is a method for Windows Store apps to display and interact with data.
- A binding or a connection between the UI and a data object enables exchange of data.
- The binding is done in XAML by using the {Binding} syntax.
- If an item changes or if the properties of a list change, the interfaces inform the changes to the bound controls.
- By using the ToString method of an item, the user can display the items as a list.
- The user can display the details of an item after selecting it from a collection.
- The combo box and the details view inherit the data context. This binds the combo box to the collection.
- The current item and the details view need not bind because the collection view source will provide the appropriate level of data.
- A converter is a class derived from the IValueConverter interface. This interface has two methods: ConvertBack and Convert.
- The Data context is inherited. If the user sets the data context on a parent element, the children will also use this data context.

Onlinevarsity

your e-way to learning

BLOG

Balanced Learner-Oriented Guide

for enriched learning available @

www.onlinevarsity.com

Session - 6

Accessing Data from Files

Welcome to the Session, **Accessing Data from Files**.

The session will discuss about App Data and Files. It will also discuss how to access data and files and read or write a file data.

In this Session, you will learn to:

- Explain about App Data and Files
- Describe how to access data and files
- Explain how file pickers works
- Define programmatically access files
- Explain and list types of App Data
- Describe the read or write a file data

6.1 Introduction to App Data and Files

Number of places are there to store data for Windows Store apps. The right place to store data depends upon the situation and in which language app is written. If it is written in HTML5 and JavaScript, or XAML using Visual C++ component extensions (C++/CX), C#, Microsoft Visual Basic, or Microsoft DirectX with C++, then, you need to overview the complete storage mediums.

6.1.1 Types of App Data

Apps frequently manage or interact with two kinds of data and they are as follows:

➔ **App Data**

This is the first type of app data which creates and manages the app itself and its data is mutable to the particular configuration and internal functions of a specific app. It includes reference content, user preferences, and runtime state. In this app data, data is tied to the origin of the app which is only useful to that app.

➔ **User Data**

This is the second type of app data in which the user makes and manages while using an app. It includes content created by the user, for example, database records holding, document or media files, communication transcripts or e-mail. User data is more useful and using this, user can transmit or manipulate document.

6.2 Accessing Data and Files

To store their files, desktop apps use the Program Files folder and registry to store their settings. It is recommended that for Windows Store apps use app data stores. Make sure that, the data are kept separately from other users and other apps, through that the data store for an app is managed by system.

➔ **Customize vs. Template Data**

Code that obtains the data required for the app present in the `SampleDataSource.cs/vb/cpp` file for the Grid App, Split App, and Hub App project templates. For the app, this represents a sample data source and from a static JSON file (`SampleDataSource.json`), reads `dataSampleDataSource` file.

6.3 File Access Using File Pickers

File pickers provides files and folders which are required for the app. The app's capability declarations can be minimized and potentially simplify the review process when it is not required.

Figure 6.1 displays the file pickers.

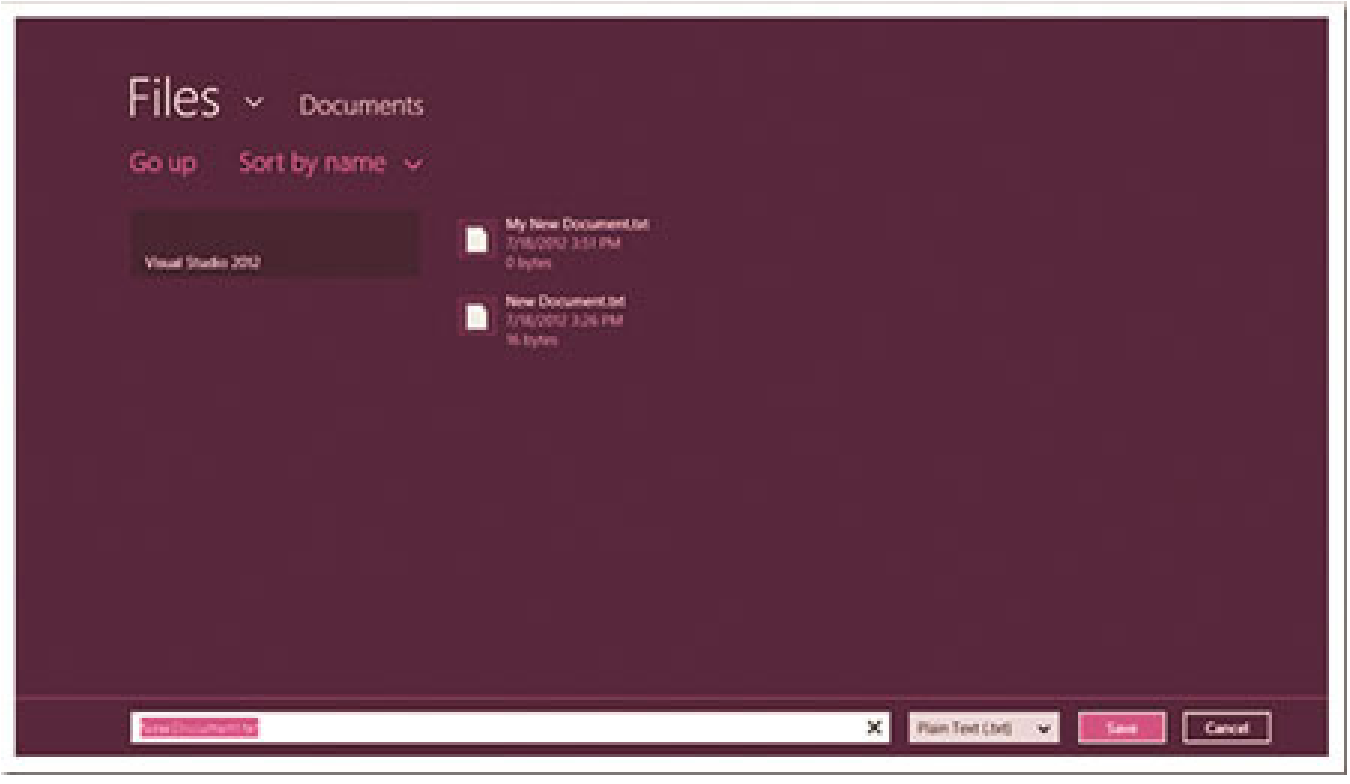


Figure 6.1: File Pickers

- Upper left of the screen is the current location.
- Bottom of the screen, a basket of items are present that the user can select.
- The down-caret in the upper-left are present in a drop-down list of locations from which the user can browse.

Figure 6.2 displays the working of the file picker.

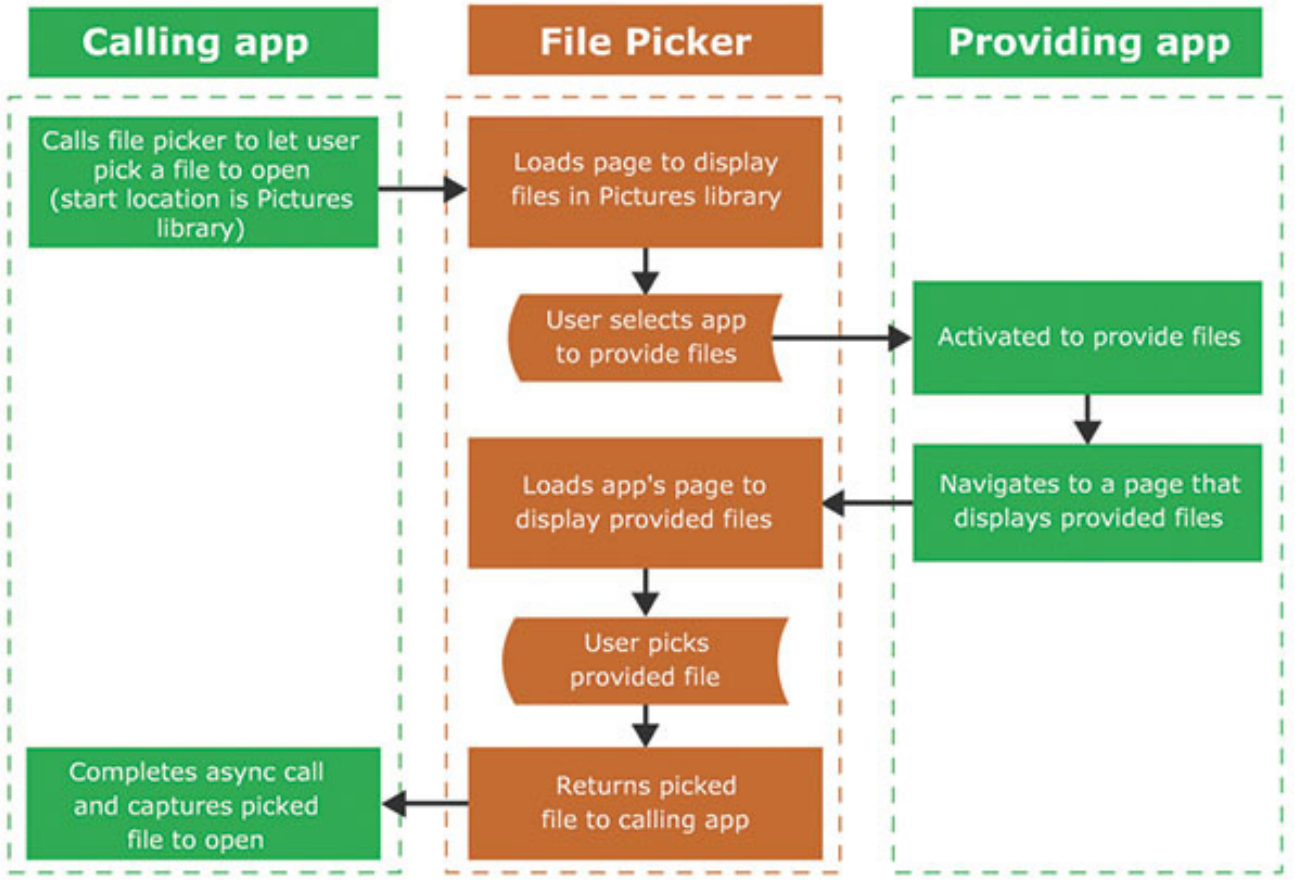


Figure 6.2: Working of File Picker

By calling file picker, the user can select files or folders and browse their system. When the user picks files or folders, the app receives the picked files or folders as `StorageFile` and `StorageFolder` objects. Using these objects, app will then, operate on picked files and folders. Following is the File picker sample code to pick single file:

Code Snippet 1 demonstrates the file picker code to pick a single file.

Code Snippet 1:

```
if(rootPage.EnsureUnsnapped())
{
FileOpenPicker myOpenPicker = new FileOpenPicker();
myOpenPicker.ViewMode = PickerViewMode.Thumbnail;
myOpenPicker.SuggestedStartLocation = PickerLocationId.PicturesLibrary;
```

Concepts

```
myOpenPicker.FileTypeFilter.Add(".jpg");
myOpenPicker.FileTypeFilter.Add(".jpeg");
myOpenPicker.FileTypeFilter.Add(".png");

StorageFile f = await myOpenPicker.PickSingleFileAsync();
if(f != null) {
OutputTextBlock.Text = "Selected Image: " + f.Name;
 }else {
OutputTextBlock.Text = "Canceled."
}
}
```

6.4 Programmatic Access to Files

In the following example, the `StorageFolder.GetFilesAsync` method uses to get all the files in the `PicturesLibrary`:

Example:

```
StorageFolder pf = KnownFolders.PicturesLibrary;
StringBuilderoutputText = new StringBuilder();
IReadOnlyList<StorageFile>fl = await pf.GetFilesAsync();
outputText.AppendLine("File Name:");

foreach (StorageFile f in fl){
outputText.Append(f.Name + "\n");
}

IReadOnlyList<StorageFolder>fl= awaitpicturesFolder.GetFoldersAsync();
outputText.AppendLine("Folder name:");
foreach (StorageFolder f in fl){
outputText.Append(f.DisplayName + "\n");
}
```

In the code the `GetFoldersAsync()` method is used to get all the folders in the `PicturesLibrary`, once the name of each file is listed.

To get both the files and the folders in a particular location, alternatively use the `GetItemsAsync` method. Following example uses the `GetItemsAsync()` method to get all files and folders in the `PicturesLibrary`:

Example:

```
StringBuilderoutputText = new StringBuilder();
IReadOnlyList<IStorageItem>itemsList = await picturesFolder.GetItemsAsync();
foreach (var item in itemsList)
{
if (item is StorageFolder)
 {
outputText.Append(item.Name + " folder\n");
 }
else
 {
outputText.Append(item.Name + "\n");
 }
}
```

Code Snippet 2 demonstrates the use of `GroupByMonth`.

Code Snippet 2:

```
StorageFolder pf = KnownFolders.PicturesLibrary;
StorageFolderQueryResult qr = pf.CreateFolderQuery(CommonFolderQuery.
GroupByMonth);
IReadOnlyList<StorageFolder>fl = await qr.GetFoldersAsync();
StringBuilder txt = new StringBuilder();

foreach (StorageFolder f in fol)
{
IReadOnlyList<StorageFile>fl = await f.GetFilesAsync();
txt.AppendLine(f.Name + " (" + fl.Count + ")");
foreach (StorageFile fi in fl) {
```

Concepts

```
txt.AppendLine(" "+fi.Name);
     }
}
```

In this code, it also lists the name of each file or `StorageFolderpicturesFolder = KnownFolders.PicturesLibrary`. The files are available in the `PicturesLibrary` and group them by the month. In the example, `StorageFolderQueryResult` object created a calling `StorageFolder.CreateFolderQuery`. Then, the `CommonFolderQuery.GroupByMonth` value is passed to the method and then, the example calls `StorageFolderQueryResult.GetFoldersAsync`. `GetFoldersAsync` are virtual folders (`StorageFolder` objects) which are based on the `CommonFolderQuery` value. Into the virtual folders, the files in the `PicturesLibrary` and its sub folders are grouped.

6.5 Access to App Data and File

For the app data, the system provides own place to store data such as settings and files during app installation time. The system completely manages the physical storage and just uses the app data API to make it easy for work.

Following are the few data stores:

1. Temporary
2. Local
3. Installed app data that exists on all devices
4. Persistent data which occur on the current device
5. Roaming

At any time Data could be taken out by the `system.once` the app is removed, then its related data stores are deleted.

➔ **App Settings**

When app data API uses the stored app data registry, the access becomes transparent. App has its root container for settings to which the app can add settings and new containers to the root container. New containers can be created to organize settings. Each container can be nested up to 32 levels.

One can use composite settings to easily handle atomic updates of interdependent settings. These are optimized for small amounts of data, and can have high performance in small data sets.

Some of the Windows Runtime data types that are supported for app settings are listed here:

- `UInt8, Int16, UInt16, Int32, UInt32, Int64, UInt64, Single, Double`
- `Boolean`

- `Char16, String`
- `DateTime, TimeSpan`
- `GUID, Point, Size, Rect`
- `ApplicationDataCompositeValue`

Since there is no binary type, use an app file if it is required to store binary data.

Other types of objects cannot be directly assigned to app data. Data can be produced to one of the supported data types.

Usually the system checks that the allotted data size matches with the size of the data type, but it does not validate the data. So developer must check if the data has changed by the time, it is written to the time read it back.

➔ App Files

App files are stored in the file system. File system uses the file in the similar way as in the app data. Each app can add new files and new directories to the root directory which system has defined.

The app files added to the local data store can be local or roaming. To the installed app, the system automatically synchronizes files to the roaming data.

➔ Local App Data

If any information to be preserved between app sessions, then, local app data can be used. Also can be stored if any data which is not suitable in that devices. There is no size limit on local data stored and location is available in `localFolder` property.

➔ Temporary App Data

The temporary app data store works same as a cache and its files that do not roam. It can be deleted at any time and by the user, also disk cleanup can be done. Temporary information can be stored during an app session using temporary app data via `temporaryFolder` property location is available.

App Data Programming Interfaces

Table 6.1 lists the Programming Interfaces.

Programming Interfaces	Description
`Windows.Storage`	`Namespace`
`Windows.Storage.Application`	`Data` class
`Windows.Storage.Application`	`DataCompositeValue` class
`Windows.Storage.Application`	`DataContainer` class

Programming Interfaces	Description
`Windows.Storage.Application`	`DataContainerSettings` class
`WinJS.Application`	`Namespace`

Table 6.1: Programming Interfaces

6.6 Reading or Writing File Data

Every Windows Store app has following three folders:

1. Local folder,
2. Roaming folder and
3. Temp folder.

➔ In Local folder, assets and application specific folder can be stored locally.

➔ In MSDN, files can be accessed in the local app data store, using the "ms-appdata:///local/" protocol.

➔ In the app package, files can be accessed using `Windows.ApplicationModel.Package.Current.InstalledLocation`.

➔ **MSDN**: Sync engine has some restrictions on the file name conventions to make the roaming folder to roam.

➔ Leading whitespace should not be in file and folder names.

➔ The sync engine limits the total size of settings and files that can roam. Using the "ms-appdata:///roaming/" protocol, files can be accessed in the roaming app data store.

```
<img src="ms-appdata:///roaming/myFile.png" alt="" />
```

Temp is a throw-away location and every time the application is launched that will be cleaned, potentially.

MSDN: Using "ms-appdata:///temp/" protocol, the files can be accessed in the temporary app data store.

Following example shows how to write or read from file in Windows Store Apps in step-wise procedure in C#:

Example:

Step 1: Include the namespace in `.cs` file as follows:

```
Using Windows.Storage;
```

Step 2: You will be using the `DocumentsLibrary` folder where you create a `sample.txt` file and save the object returned as given here:

```
StorageFolder newStorageFolder = KnownFolders.DocumentsLibrary;
StorageFile newSampleFile = await newStorageFolder.GetFileAsync("sample.txt");
```

Note - You are using the `await` keyword to create a file. This implies that you need to use the `await` keyword in the declaration of this method in which the code is used.

Step 3: You write to the file as given here:

```
await Windows.Storage.FileIO.WriteTextAsync(newSampleFile, "Hello World");
```

You use the `FileIO` method `WriteTextAsync` to write to a file which takes two arguments, the first one is `StorageFolder` argument and the second one is the text to be written.

Step 4: You can access the readable files as given here:

```
StorageFile newSampleFile = await
newStorageFolder.GetFileAsync("sample.txt");
```

Step 5: You read the file by using the `ReadTextAsync` method as given here:

```
string fileText = await Windows.Storage.FileIO.ReadTextAsync(newSampleFile);
```

In this example, the code demonstrates the core functionality of writing and reading to Isolated Storage, which is a simple Unit Test. Isolated Storage in the code or namespaces cannot be mentioned, but takes place in Isolated Storage.

➔ **Read Project File**

Following are the steps need to be followed to read a file in the project:

Step 1: File to be added in your project.

Step 2: Files to be changed to Build Action to Content. Always change Copy to Output Directory to Copy.

Step 3: Read your project file such as this:

Example:

```
[TestMethod]
public void ProjectFileTest()
```

Concepts

```
[TestMethod]
public void ProjectFileTest()
{
ProjectFile();
}

privateasync void ProjectFile()
{
 //settings
 //same as (ms-appx:///MyFolder/MyFile.txt)
var _Folder = Windows.ApplicationModel.Package.Current.
InstalledLocation;
 _Folder = await _Folder.GetFolderAsync("MyFolder");

 //acquire file
var _File = await _Folder.GetFileAsync("MyFile.txt");
Assert.IsNotNull(_File, "Acquire File");

 //write content
var _WriteThis = "Hello World";
awaitWindows.Storage.FileIO.WriteTextAsync(_File, _WriteThis);

 //read content
var _ReadThis = await Windows.Storage.FileIO.ReadTextAsync(_File);
Assert.AreEqual(_WriteThis, _ReadThis, "Contents correct");
}
```

In this example, the code how to read a file that is included in the project's folder structure which is a simple Unit Test. Unique Path and Installed Location folder are the two most important parts.

➔ **Read Local files w/a Picker**

If it is required to read a file in the `DocumentsLibrary`, then, user must select a file using a picker.

Concepts

The following example demonstrates how to read from a file in DocumentsLibrary using a Picker.

Code Snippet 3 demonstrates how to read from a file in DocumentsLibrary using a picker.

Code Snippet 3:

```
//create a picker instance
FileOpenPicker myPicker = new FileOpenPicker();


//view mode is set to thumbnail to view the image. There only other options it
supports is list view
myPicker.ViewMode = PickerViewMode.Thumbnail;


//view files from the windows pictures library
myPicker.SuggestedStartLocation = PickerLocationId.PicturesLibrary;


// add filters for the type of files that needs to be open from pictures library
myPicker.FileTypeFilter.Add(".jpg");
myPicker.FileTypeFilter.Add(".jpeg");
myPicker.FileTypeFilter.Add(".png");

//read the file
StorageFile myFile = await myPicker.PickSingleFileAsync();


if (myFile!= null)
{
  //Application now has read/write access to the picked file

  // FilePickButton is the name of the button on which click event the C# code
is used
   Filepickbutton.Content = "Picked photo: " + myFile.Name ;
}
else
{
  Filepickbutton.Content = "Operation cancelled.";
 }
```

In this example, the FileOpenPicker is initialized and then, using the picker PickSingleFileAsync() is invoked. Now a StorageFile is obtained as a technique in the example. Here, a MessageBox ((i.e. MessageDialog) is added to show some details.

➔ **Read Local files w/o a Picker**

A file can be read without using the file picker, but it requires a lot of effort. To request the Document Library Access capability, it is required to update the applications of AppXManifest.

By declaring which file type(s) have to access, you can update your AppXManifest.

Concepts

Without file and user interaction file should not be accessed which may lead user to confusion.

Each file has to write and read in order to have a reliable test. Similar way in the following example, also first, the `HelloWorld.txt` file created and then, read it. At the end of the method, file is deleted since the folder should not get polluted.

Following is an example of reading a file without a picker. It can be used in a button click event or any object overriding of our choice:

Example:

```
    //a reference to the windows storage folder
    var newOpener = new Windows.Storage.Pickers.FolderPicker();

    //add filter for the desired file type
    newOpener.FileTypeFilter.Add(".txt");

    //a folder reference
    StorageFolder myFolder = await newOpener.PickSingleFolderAsync();

    if(myFolder != null)
    {
     StorageApplicationPermissions.FutureAccessList.AddOrReplace("Pick
edFolderToken", myFolder);

     //acquire the file
     IReadOnlyList<StorageFile> files = await myFolder.GetFilesAsync();

     if(files != null)
     {
      foreach(StorageFile newFile in files)
      {
       //noteslist is an ObservableCollections object
        noteslist.Add(new Note()
          {
```

```
            name = newFile.DisplayName,
            file = newFile
          });


      }      // list is the name of the ListView created in the maybe for
example in the XAML file
      list.ItemsSource = noteslist;      }    }
```

In this example, it is easy to read TXT files from the Documents Library folder in the `AppXManifest` file as it is a simple `WinRTStorageFile` .

Some of the storage file methods are as follows:

`CopyAsync()`

`DeleteAsync()`

`MoveAsync()`

`OpenAsync()`

`RenameAsync()`

It is important that the files which are returned to a folder are automatically filtered to the file types. As a developer, one has limitations to ultimately improve the user's experience and safety.

6.7 Check Your Progress

1. Which kind app data and user data belongs to?

(A)	App data	(C)	Data flickers
(B)	User data	(D)	Template data

2. The code that obtains the data required for the app present in _____.

(A)	`PickerLocationId.PicturesLibrary`	(C)	`openPicker.ViewMode`
(B)	`SampleDataSource.json`	(D)	`SampleDataSource.cs/vb/cpp`

3. What provides user to access files and folders?

(A)	App data	(C)	File pickers
(B)	File	(D)	User data

4. Temporary, local belongs to type of _____.

(A)	App data	(C)	File flickers
(B)	Data stores	(D)	Temporary data

5. `UInt8`, `Int16`, `UInt16`, `Int32`, `UInt32`, `Int64`, `UInt64`, `Single` , `Double` belongs to _____.

(A)	Windows Runtime data types	(C)	App data types
(B)	File picker data types	(D)	Template data types

6.7.1 Answers

1.	A
2.	D
3.	B
4.	B
5.	A

Summary

- App data is created and managed by the app itself whereas in user data the user makes and manages while using an app.
- Desktop apps use the Program Files folder and registry to store the settings.
- File pickers provides user to access files and folders which are required the app to work with.
- By calling file picker, the user can browse their system and select files or folders to access and save.
- The system provides own place to store data for app data such as settings and files during app installation time.
- At any time data could be taken out by the system. Once the app is removed, then its related data stores are deleted.
- To easily handle atomic updates of interdependent settings, you can use composite settings.

Onlinevarsity

your e-way to learning

GROWTH

RESEARCH

OBSERVATION

UPDATES

PARTICIPATION

www.onlinevarsity.com

Session - 7

Managing Application Data

Welcome to the Session, **Managing Application Data**.

The session will discuss how to manage application or app data. The session will teach you about the data stores, types of app data, and the various settings related to app data.

In this Session, you will learn to:

- Explain how app data is stored
- Describe the app settings and file settings
- Explain the use of local app data
- Explain the use of roaming app data
- Describe how to use temporary app data

7.1 Storage of Application Data

The Windows Store apps may store app data in many places. It depends on the language that is used to write the code. The app can be coded using these languages:

- JavaScript and HTML5
- Microsoft Visual Basic
- C++ and Microsoft DirectX
- Visual C++ component extensions (C++/CX) with XAML
- C#

Typically, the designer creates a skeletal framework for all the mediums of storage and highlights the best instances for each medium to perform optimally.

Storage of app data becomes essential in these cases:

- Data in the data store is being preserved by the system when the user updates the corresponding app
- Data are removed from the store when the application is uninstalled
- Data in the data store are isolated from other apps and users thereby enhancing security

Normally, apps use two types of data:

- App data
- User data

- **App data**

  A type of data that an app creates and manages. The user can customize this type of data for the relevant technical configurations or internal functions. This type of data includes:

  - **State during runtime**: A state when the application program is executed. During this state, the application program sends instructions to the computer processor, and will be able to access computer's RAM and other resources. The data involved in this process is, App data.
  - **Preferences of the user**: These are the user desired preferences for some application related settings such as language, display settings, keyboard settings, and so on.
  - **Content reference**: This is the content specific to an application, for example, a dictionary application that has dictionary definitions stored.

Users must remember that the app data is related only to the app.

Concepts

Following is the list of app data that are located in different places. Listed are the features, usage suggestions, and location details.

Table 7.1 demonstrates the details for the app data.

Location of data	Availability	Features	
Windows Runtime app data APIs	➔ Apps that use C# ➔ Apps that use C++ with DirectX	➔ Roaming, temporary, and local data store available ➔ Supports structured content, non-structured files, and composite data types ➔ Synchronous API used by the settings and Asynchronous API used by files ➔ Users can accommodate either 8K or 64K per composite settings ➔ Users save the bandwidth and life of the battery when they are roaming	➔ Use for settings and unstructured data files
Extensible Storage Engine(ESE)	➔ Any app	➔ Technology for storage: Indexed Sequential Access Method (ISAM) is a file management system that allows records to be accessed sequentially or randomly. It facilitates faster data retrieval ➔ Cursor navigation: Sequential or Indexed ➔ Consistent data despite system crash ➔ Size scalability	➔ Storage of indexed data ➔ Storage of structured data

Table 7.1: Details of the App Data

- **Storage of App Data**

  When a user installs an app, the system provides the app with a customized data store for its settings and files. They need not know about the location of the app data. The system keeps track of the location of the data and they need to use the API for the app data.

  There are three local data storage locations:

  - **Local**: This is for data that exists only on the currently available device.
  - **Roaming**: This is for the data that is available on all the devices that the user has installed the app on.
  - **Temporary**: This data is temporary and the system can delete it at any time.

Remember that with the uninstallation of the app, the data storages are deleted. The user can also create versions of the app data for future use. This method will ensure that there are no compatibility issues for future versions of the app.

➔ **User data**

The user creates and manages this type of data while using the application or app. This type of data contain documents, media, mails, transcripts of communication, and so on. They must remember that customizations done by the user, will constitute app data and not user data. This type of user data will be useful to more than a single app. They modify or update the user data to create a separate entity.

There are specific data stores made available by the system, which are customized for the user and their apps. The user can store their app data for the Windows Store apps in these data stores. These app data are stored individually and are separately kept from other users and their apps.

If the user installs any update to their apps, the system saves the data stores. Likewise, when they uninstall the app, the system clears the data store. They should ensure that they do not store the data of their apps in locations reserved for user data.

The user must also remember that the data stores are live as long as the apps are available. If the user saves user data such as media files or any documents in the data stores, they may lose these files. They should save the data in a library or use a software such as SkyDrive.

Data related to the user's app, constraints, and the suggested use of data are available. In the Windows Store, access to a few apps that use JavaScript is restricted. The access depends on the context. The constraints on the storage size is ruled by how big the medium of storage is.

Table 7.2 lists the features and ways to use the various data options.

Location of Data	Availability	Features	Usage suggestions
HTML5 File API	➔ Apps that use `JavaScript` (local context and Web context) ➔ Apps that use C++, `Visual Basic`, or C# ➔ Apps that use C++ with `DirectX`	➔ Follows the Web standards	➔ When the user wants to upload or download files ➔ If the user wants to transfer any files when the app is not in use, they have to use `Windows Runtime backgroundTransfer`
Libraries (Windows file pickers and `RuntimeStorage` `File` access these libraries)	➔ Apps that use `JavaScript` ➔ Apps that use C++, `Visual Basic`, or C# ➔ Apps that use C++ with `DirectX`	➔ Files saved, loaded to user libraries ➔ APIs are synchronous in nature ➔ No limit on the size ➔ Allow the user to select or create a file by using the folder and file picker option ➔ Requirement of a suggested type of file ➔ In case of a downloaded file, the app can overwrite	➔ For user data that the user will manage ➔ For user data that will remain active for the entire life of the app

Concepts

Location of Data	Availability	Features	Usage suggestions
		➔ Specific extension is needed for files that can be downloaded	
SkyDrive	➔ Apps that use `JavaScript` ➔ Apps that use `C++`, `Visual Basic`, or `C#` ➔ Apps that use `C++` with `DirectX`	➔ User can store data on the cloud ➔ User data is available to many devices on several platforms ➔ API based on A REST/ JavaScript Object Notation (JSON) ➔ Supports a specific set of formats for files ➔ Other users can access shared data ➔ Size is based on user's account	User can use this when they want to access their data from multiple locations and devices/ platforms

Table 7.2: Features to Use the Various Data Options

When there is an interaction between the app, Web site, and a Web service a few data locations, are used.

Table 7.3 lists the details for the server data.

Location of Data	Availability	Features	Usage suggestions
Cookies	Apps that use `JavaScript` Apps that use `C++`, `Visual Basic`, or `C#` Apps that use `IXMLHTTPRequest2`	This is a Web-based standard Disjoint from the browser's cache memory based on user and the apps used 4000 size limit for each cookie Limited number for the cookies Cookie sharing rules apply	Data that needs to be accessed when a request is sent to a Web server.
HTML5 Application Cache	Apps that use `JavaScript`	It is a standard for the Web. Web-based scripts, style sheets can be fetched in advance Application Cache can be used for the context data that is local The user cannot code it using script	Use when content is read-only, delivered from a Web server, and needs to be accessible while the device is offline. While the user can use the HTML5 Application Cache to deliver custom content, it is used as a cache memory to give offline access to data.
Cloud services	Any app	Complete cloud-based solution Allows trigger based on server or processing periodically Features depend on the adoption of the cloud-based services	Used for high volume data

Table 7.3: Details for the Server Data

Concepts

Code Snippet 1 allow the user to create a document in the documents folder.
Code Snippet 1:

```
StorageFolder myStorageFolder = Windows.Storage.KnownFolders.
DocumentsLibrary;

StorageFile myDocumentFile = await myStorageFolder.CreateFileAsync("sample.
txt",

      CreationCollisionOption.ReplaceExisting);

await FileIO.WriteTextAsync(f myDocumentFile , "My sample document file created
through C# code");
```

The output of the Code Snippet 1 will be a `sample.txt` file in the user's document folder with the text 'My sample document file created through C# code'.

7.2 Understanding Settings and Files for Apps

The registry stores the app data store settings. For the user that uses the API for the app data, the access to the registry is transparent. Every app contains a root folder that hosts the app settings. The user app is able to modify the settings and also add containers to this root folder or container. The user can, add containers as per their convenience, and to better organize their customizations. The user can have upto 32 levels for containers.

The system retains the custom settings when a user is roaming or if they access the app from multiple locations. Such settings help the user manage smaller updates of other interdependent settings.

App settings can be of two types that are as follows:

- Local
- Roaming

Figure 7.1 shows the settings and files for the app.

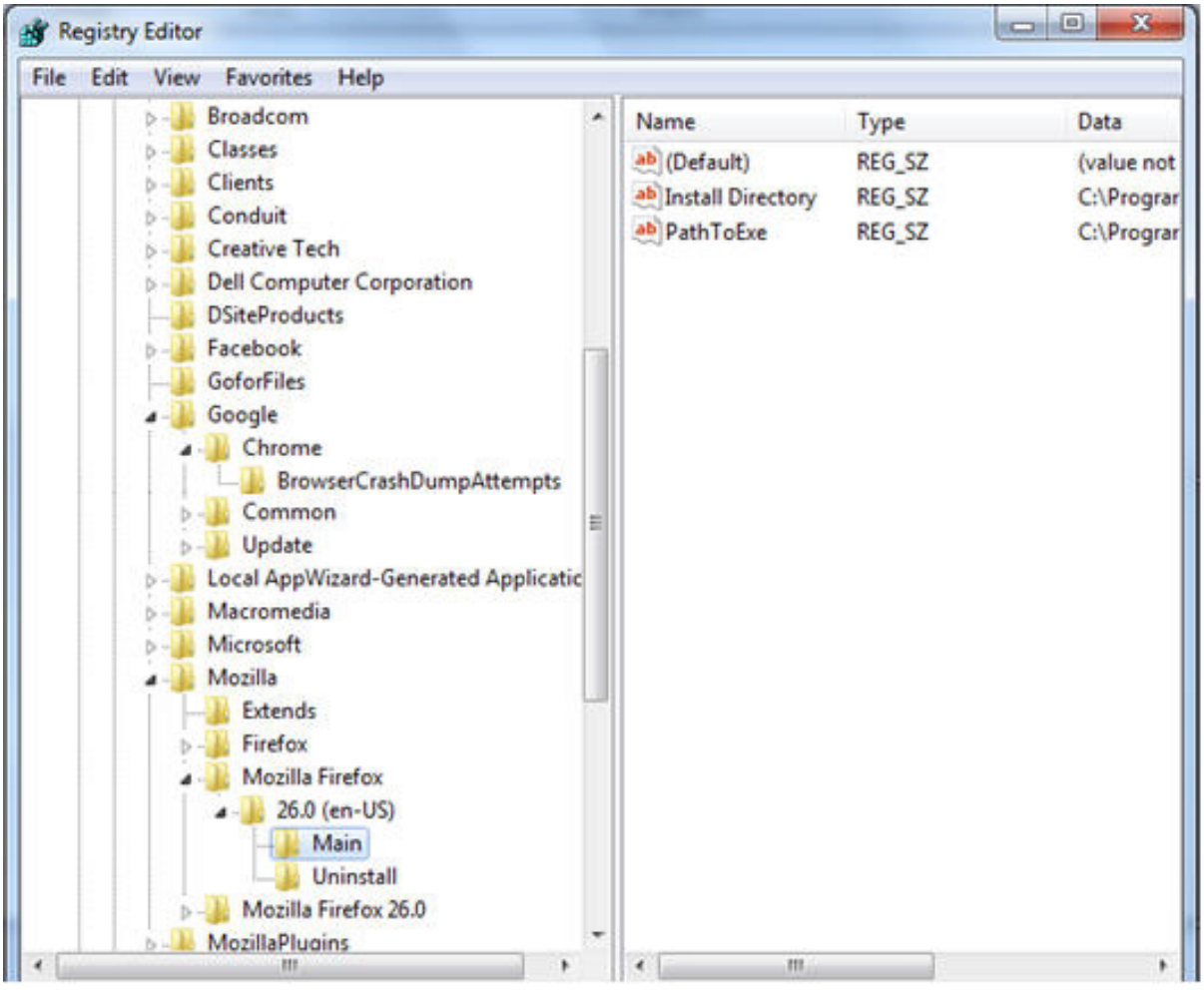


Figure 7.1: Understanding Settings and Files for the App

The app settings added by the app to the user's data storage is available in the currently used device. When the user is roaming, the app settings and the local store data get synchronized.

Following is a list of the data types for runtime on a Windows machine supported by app settings:

- `UInt8, Int16, UInt16, Int32, UInt32, Int64, UInt64, Single, Double`
- `Boolean`
- `Char16, String`
- `DateTime, TimeSpan`
- `GUID, Point, Size, Rect`
- `ApplicationDataCompositeValue`

Concepts

If the user wants to store data for the binary data type, they should get an app file. The system takes care of the size match between the data and the data type. The system will not check the data. The user should only check for changes in data.

➔ **Files Related to an App**

File system contains all the files meant for the app data store. An app can use these files easily. In the data storage location, there is a directory for the local data files, a directory for roaming data files, and a directory for any temporary files. The user's app is allowed to update the root directory.

7.3 Local App Data

The user can use local app data for data they want to use later. Local app data is also useful for data access between each session. The user can save other types of data that are not applicable for any other device here. They should remember that there is no size restriction for the local app data. The property `localFolder` contains the data storage location. For roaming data and for large sets of data, the user can use the data store for local data.

Scenario:

Local data is the system and user specific details, typically large size files, stored on a computer. For example, the browser's cache and history are good example of a local app data.

The following syntax users can use to access their files:

Syntax:

```
<img src="ms-appdata:///local/myFile.png" alt="" />
```

7.4 Roaming App Data

The user can sync their data in all the devices they use. If they install the app in all the devices they use, Windows helps the user to keep the data synchronized.

For example, consider a organization with 1000 workstations. It is required that any employee will work in any workstation. In this case, a roaming profile with the user specific details such as documents, bookmarks, and so on is required so that the user can log in from any workstation. The user profile is synchronized to a server and it is made available to all the workstations in the organization. Here, the user profile is the roaming data.

The user can continue performing a task left half done on one device and pick it up from another device.

Figure 7.2 shows the itunes roaming data in app data.

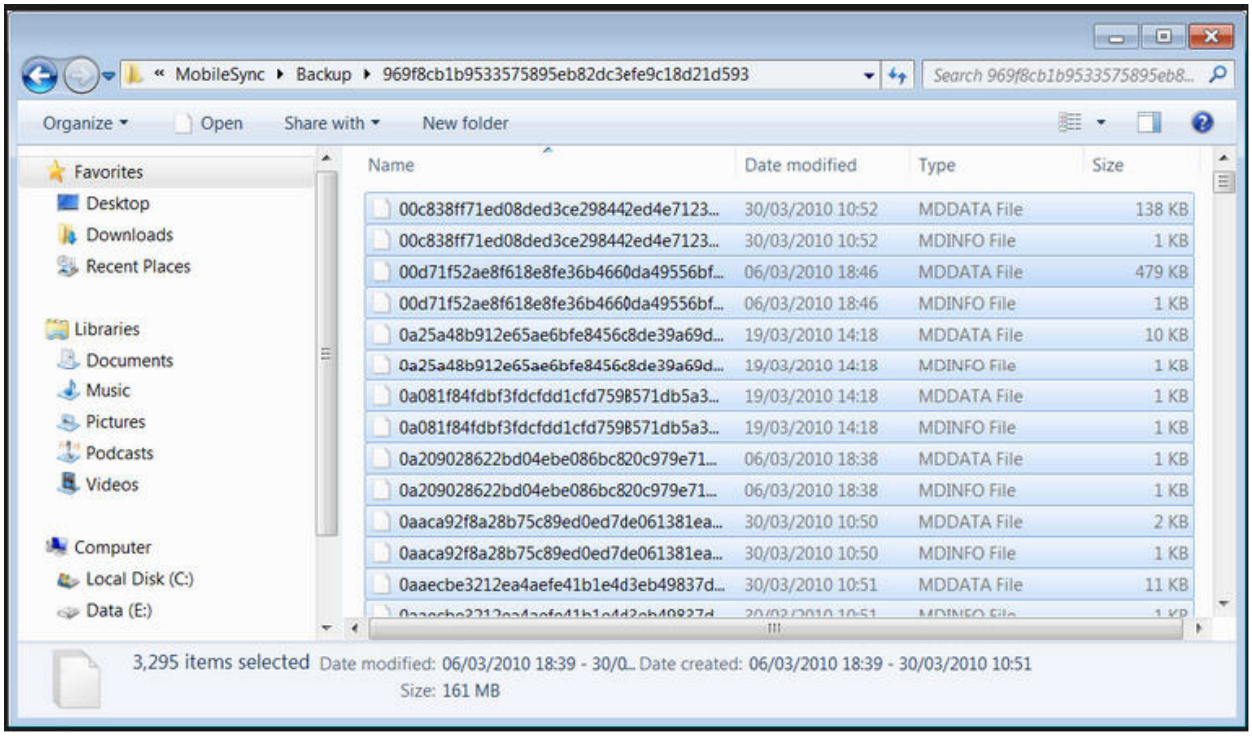


Figure 7.2: Roaming Data in App Data

Windows helps the users and duplicates their relevant data on the devices when they require it. There is a limit to the size of the data of the app for each roaming-enabled app. The syntax is as follows:

Syntax:

```
ApplicationData.RoamingStorageQuota | roamingStorageQuota
```

If the app reaches this limit, the OS will not copy any data on any other device. The user should remember to limit the use of the app while roaming so that they do not cross the size limit.

When the data of a roaming app changes, the app registers the changes with the syntax as follows: `DataChanged | datachanged`

This event occurs whenever there is a change in the roaming data for the app.

If the user updates the app and installs a newer version, the app data gets copied to a cloud server. When the user installs the updated version, the updated app data gets copied. If the user does not access the roaming data for a long time, the data can get deleted. Also, if the user uninstalls the app, the data is retained on the cloud. If the user reinstalls the app within 30 days, the data gets synchronized with the device. The property `roamingFolder` contains the location of the data. It is not necessary that as soon as the app is connected, the data gets synchronized. Many times, roaming of data can get delayed.

➔ **Rules for App Data Roaming**

The OS automatically synchronizes app data between the current device which the user is using. The developer need not do any extensive programming. The end user's experience is enhanced when they install the app on more than one device and use roaming app data.

- **Enhancing the user's experience**: If the user customizes their initially used device with certain settings, Windows helps them by synchronizing the data on the second device they use. The user's experience gets enhanced as their work to customize each device they using is minimized. In the future also, whatever changes they do gets synchronized automatically.
- **Design rules**: If the user uses roaming app data, they need not touch the code for any updates or changes. It is advised to use roaming app data for data that have a size limitation and user preferences related settings. The user needs to keep the points in mind for using the feature optimally.

Table 7.4 lists the rules for app data roaming.

Favorable Actions (Dos)	Unfavorable Actions (Don'ts)
Use of roaming app data for any custom content	Avoid using roaming app data that resides on the currently used device
Any app data that the user needs to use frequently: media apps, background colors, and so on	Roaming app data should not end up messing the end user experience
Use of roaming data to continue tasks seamlessly over multiple devices	Roaming should not be used to transfer large pieces of data
Allow users to transfer their custom content across devices	Roaming should not be used for immediate synchronization of data

Table 7.4: Rules for App Data Roaming

Some of the considerations for working with roaming application data:

➔ **Pre-requisites**: The pre-requisite is that the user needs to use a Microsoft account to access their device while roaming and using the app data. Data can be transferred across devices that use the same account.

➔ **Privileges**: Users who have admin rights or are part of the group policy, admin team can turn off the app data roaming function on the device. If the roaming is off, and if the user is not using their Microsoft account, then, the data they save cannot be synchronized with any other device.

➔ **Device Trust**: Before a device can be used for data synchronization, the user must ensure the

Concepts

device is labeled as trusted.

➔ **Resolve conflicts**: If the user changes a particular data in two devices, a conflict will arise during synchronization. The app will use the latest information.

➔ **Apt time to write data**: The user must write data on the device at different times, as the life span of an app setting is very short. They can copy slowly changing app data faster as compared to app data that changes frequently.

➔ **Version**: App data can be saved as per different versions. The app data can be upgraded from a lower data structure to another by using the latest version. The version number is unique and can be set by the developer. The user should remember to use increasing version numbers for ease of use. Data can be synchronized only between the same versions of an app. For example, a device with the Version 1 of an app can transfer data to Version 1 only, and not for Version 2.

➔ **Protection against extra usage**: To safeguard excess use of a particular application, there are measures in place. If the app data is not synchronizing as expected, the user should wait for sometime and then try again.

➔ **Using the `DataChanged` syntax**: App data on roaming mode can change anytime. The developer needs to use the syntax DataChanged and apply the rules to update the current app data.

7.5 Temporary App Data

The temporary app data storage is same as the cache memory, which is a temporary storage. The files available in this location cannot roam and these files can be deleted at any time. The syntax `System Maintenance` task can automatically delete the files present at anytime. The user can also delete the files by using the Disk Cleanup function. Temporary data required for a particular app session is stored in this location.

7.6 Check Your Progress

1. _____ is not used to code the Windows Store apps.

(A)	Fortran	(C)	JavaScript and HTML5
(B)	Visual Basic	(D)	Visual C++

2. Up to how many levels of containers can the user add?

(A)	42	(C)	31
(B)	24	(D)	32

3. When the user installs an app the _____ provides a customized data store for the settings and files.

(A)	System	(C)	Developer
(B)	Device	(D)	Operating System

4. _____ contains documents, media, email messages, transcripts of communication, and so on.

(A)	User data	(C)	App data
(B)	DataChanged event	(D)	Windows Store App

5. Which of the following syntax deletes files available in the temporary app data store?

(A)	DataChanged syntax	(C)	System Maintenance task
(B)	DataChanged | datachanged syntax	(D)	ApplicationData syntax

Concepts

7.6.1 Answers

1.	A
2.	D
3.	A
4.	A
5.	C

Summary

- Windows Store apps may store app data in many places. It depends on the language used to write the code.
- Temporary app data storage is same as a temporary storage, the cache memory.
- When the user installs an app, the system provides the app with a customized data store for its settings and files.
- User can also create versions of the app data for future use.
- User creates and manages this type of data while using the application or app.
- Registry stores the app data store settings. For the user that uses the API for the app data, the access to the registry is transparent.
- Settings on the app while the device is in roaming mode are synchronized with the local store data.
- The file system contains the entire files meant for the app data store.

Session - 8

Application Lifecycle

Welcome to the Session, **Application Lifecycle**.

The session will discuss about Application lifecycle, App deployment, execution state, launch, and activation. It will also discuss how to activate the apps, reasons for termination and actions to take when app is closed.

In this Session, you will learn to:

- Write the introduction to Application lifecycle
- Describe the app deployment process
- Illustrate the app execution state
- Describe about the app launch
- Write the different methods of app activation
- Explain about the app suspension
- Describe about the app visibility
- Write about the app resume
- Recall the app removal
- Discuss about the app close
- Write reasons for app crash

8.1 Application Lifecycle

Application lifecycle is a specific sequence of processes from start to finish of the application. In Windows 8, Microsoft's effective application lifecycle management makes sure that the apps do not affect neither the system performance nor the battery life. Usually the user has multiple apps open in their system, so that they can quickly navigate between them when required, and it might affect the battery life and slow down other apps in the system. This is not the case in Windows 8, as the system can suspend or terminate apps running in the background.

A good app can be suspended, terminated, or re-launched as and when required and it would seem as though the application is running the whole time. When an application is suspended its state is saved and when it is re-launched again it is open in the same state it was earlier. Figure 8.1 shows the application lifecycle.

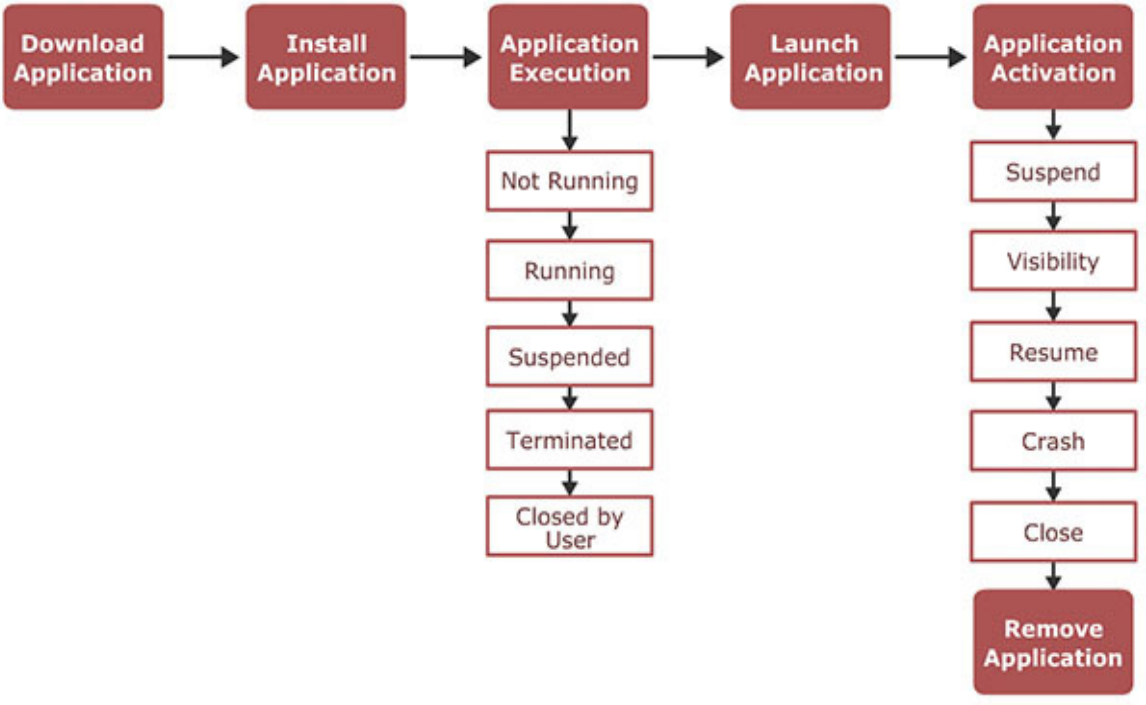


Figure 8.1: Application Lifecycle

Following are the processes involved in an application lifecycle. The following stages are explained briefly:

➔ **Download Application**

The application is available as a package for download, to the user, in the Windows store and this package has the extension .appx. The package has the manifest and application files to build the application.

The manifest file is an XML document that contains information about the system requirements to deploy, display, or update the corresponding application. Each package will contain one package manifest.

Concepts

➔ **Deploy Application**

The downloaded application needs to be installed on the system to facilitate the usage of the application in the system. The application comes with an installer and the installation is done using the details in the manifest file.

➔ **Application Execution**

The installed application is ready to be executed. The Windows Store App has a unique sequence to execute an application as shown in the white boxes in the diagram. Windows 8 programming interface provides an enum, ApplicationExecutionState, and its states are explained in detail in the App Execution State.

➔ **Launch application**

The application is launched by displaying the splash screen which denotes the application's initial UI is about to be displayed. Actions involving data are not performed in this stage.

➔ **Application activation**

The application becomes active when the user responds to specific events like navigating a UI. The following are the stages involved:

1. **Suspend** - It is done when the App is idle.
2. **Visibility** - The UI involved in the App.
3. **Resume** - It happens when the user returns to the App.
4. **Crash** - A software crash related to the App.
5. **Close** - User closes the App.

➔ **Removing the application**

This stage occurs when the user uninstalls the application from the system. Here again the manifest file is used for removing the application.

The access based on program to connected devices or protected resources, required for the App is specified by the package manifest. Until an app gets approval for accessing it cannot access system files in their libraries.

8.2 App Deployment

In a single package the Windows Store App model provides installation, updates data and instructions for an App. It is a declarative state-driven process in which deployment operations are reliable. The files present in the package cannot modify since they are delivered to the computer.

A new version of the App is downloaded and installed to the user's profile once the update process takes place. Later the old version is removed from the computer. Without affecting any other user's apps on the computer apps can be installed, updated, and removed. In Windows Installer, there will not be any patch files or any other files that are used to deploy a Windows Store App. Each user has complete command over their Windows Store apps since Windows Store apps are installed into a user's profile.

8.3 App Execution State

Transitions between App execution states are shown in the following illustration.

`ApplicationExecutionState` doc gives more information about each state transition that is occurred in response to what an App should perform. The following section describes these states and events.

Transition between App execution states

Figure 8.2 displays the App execution state.

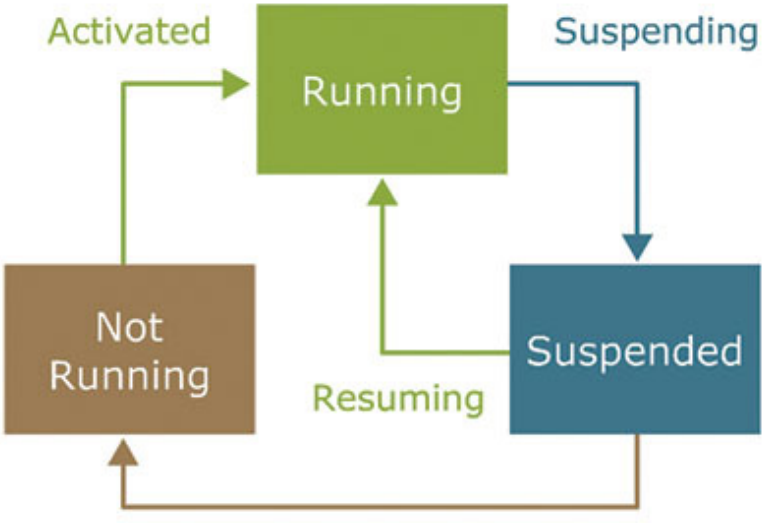


Figure 8.2: App Execution State

The execution states are defined as follows:

➔ **Not Running**

This condition occurs in several situations such as when the user is installing the application, clicks End Task from the Task Manager, restarts the computer, logs off, and back on. This state allows the application to display its initial UI and perform initialization tasks.

Concepts

➔ **Running**

When the application is launched it is still in the Not Running state. Running state occurs only when the user performs any simple action on the application. It then, responds to the activation event triggered by the user.

➔ **Suspended**

This state occurs when the user switches between the applications or when the system is in low power state. Windows waits for 10 seconds, if the user does not return to the application then, it enters into the suspended state. The state of the application is maintained in this stage.

➔ **Terminated**

Windows terminates the suspended App if the system is on low resource. Game based application can consume lot of resource even though they are idle.

➔ **Closed by User**

This stage occurs when the application is closed by close button or keyboard shortcuts.

8.4 App Launch

Whenever an app is activated by the user it is launched. However, the process will be in the Not Running state because it was just deployed, or suspended, or crashed, but cannot be stored in memory. Windows displays a splash screen when it is launched.

The basic tasks for the App is to set up custom UI and register event handlers which are required for loading. These processes should be completed when an App needs to retrieve huge data from disk or request data from the network. Till the App waits for running operations to get over, it can use its own custom loading UI or an extended splash screen. Once the App activation is complete, it is in running mode. There are specific ways to show an activation of an app completion process such as displaying a window, returning from the activation handler, and completing a suspension.

For example, consider an application that is used to connect Internet provided by its service provider. When you double-click the application icon, a splash screen is shown with a progress bar which is shown in figure 8.3.

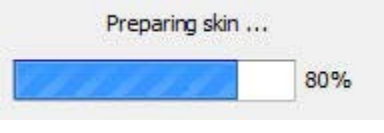


Figure 8.3: Progress Bar

8.5 App Activation

User can activate the App through a variety of contracts and extensions. A contract is nothing but an agreement between apps to participate in certain Windows interactions. For example, the file save picker help users to save files directly to the app while they are using other apps. When user switch back to the app it allows them to save files to the app.

An extension is nothing but an agreement between the app and Windows. It helps to extend or customize a Windows feature for using our own app. For example, Contact Picker extension allows apps to be registered to access contact data. When the user needs to access the system contacts, our app is listed in the set of apps so that they can access contact data. User App must be registered to receive the `Activated/activated` event to participate in activation. Table 8.1 lists the activation types.

Activation Type	Description
`Cached file`	Provides to save content management.
`Camera`	Attaches camera to capture photos or videos.
`Contact picker`	Chooses contacts.
`Device`	Handles AutoPlay.
`File`	Specifies the type of file which, App is registered to handle.
`File open picker`	Selects files or folders that are provided by the App.
`File save picker`	Saves a file and has picked the App.
`Launch`	Launches or is tapped by the user.
`Print task`	Handles print tasks.
`Protocol`	Launches a URL whose protocol an app is registered to handle.
`Search`	Searches with the app.
`Share target`	Targets for a share operations.

Table 8.1: App Activation Types

App is used to activate the restore previously saved data. In this event the Windows terminates the app and then, the user re-launches it. Windows terminates the app once it has been hanged for a different of reasons. The reasons may be the user would have closed the app manually or signed out from app or the system might be running short on resources. When the user launches an app and at the same time if the Windows has terminated app, then, it receives an activated event. Until the app is activated the user can view the splash screen. This event can be used to determine whether the app needs to restore the data or whether it must load the apps by default data. This `PreviousExecutionState` property includes the activated event arguments. It tells that in which state your app was in before it was activated. Table 8.2 lists the possible reasons for termination.

Reasons for Termination	Previous Execution State Property Value	Action to Take
Terminated by the system due to resource constraints	Terminated	Restore session data
Closed by the user	ClosedByUser	Default data to be Started
When the user started session, then, unexpectedly app has been terminated or app has stopped running	NotRunning	Start with default data

Table 8.2: Reasons for Termination

`PreviousExecutionState` might have a value of `Suspended` or `Running`, but in this case you need not care about restoring data as the app was not previously terminated. It should be noted that in Windows 8, the apps will not be activated when you log in using the built-in administrator account.

8.6 App Suspend

App could be suspended when Windows enters a low power state or when the user switches away from it. When the user switches away from them apps stop running.

When the user places an app to the background, then, the Windows waits for few second to see whether the user again switches back to the app immediately. If the user does not switch back immediately, it suspends the app.

Event handler should be called before the app is suspended when `Suspending/suspending` event handler is registered by an app. To save relevant app and user data for constant storage event handler can be used. It is recommended that one can use the APIs application data for this purpose. To know more information about this refer to application data.

Usually app saves its state and releases its exclusive resources. It happens when the `suspending` event is received and usually takes about a few seconds to do so. Windows terminates the app assuming that the app has stopped responding or does not return from the suspending event within few seconds.

Suspended apps are stored in Windows memory and it keeps suspended apps in memory as much as possible. Therefore, it ensures that users can rapidly and consistently switch between suspended apps. Suppose, if there are not enough resources to store the app in memory, Windows can stop the app. Also apps don't receive notification that app got terminated. Therefore, during suspension the app's data can be saved.

Some apps run continuously to complete some background tasks. For example, the app can play audio continuously in the background.

8.7 App Visibility

When the user switches from one app to another app, until Windows suspends it. The app remains in the running state but no longer visible. The app remains in the running state, before Windows can suspend or switches back to it. When the user switches between two apps, the app that is not visible will be in running state for about 10 seconds and then, the Windows suspend it. If the user returns to the previous app before it could be suspended (i.e before 10 seconds) it will still continue to be in the running state.

For example, assume that the user opens two apps one after another. The first app will be in running state for 10 seconds and then, it will be in suspended state. Suppose, if the user returns to the first opened app within 10 seconds then, it will continue to be in the running state.

When the app's visibility changes it does not receive an activation event since it is still in running state (for 10 seconds). If you want to do anything in our app as soon as the app's visibility changes you need to use the `VisibilityChanged/msvisibilitychange` event handler.

8.8 App Resume

When the user switches to the app or Windows comes out of a short of power state then, a suspended app is resumed. When an app is restarted from the `suspended` state, it continues from where it was suspended. It was stored in memory and during this time no application data is lost. So when they are resumed most apps need not to do anything. App can be suspended for some hours or even for days, so if app content or network connections have gone stale, by the time app resumes these should be refreshed.

When a suspended app is activated to take part in an app contract, for example, the user tries to save a file to this app (that is suspended) from another app that the user is currently working, an Activated/activated event is first occurred followed by the `Resuming/resuming` event.

Whenever an app is resumed it should check its network status, since in suspended state these events would have missed as they are not queued to the app that is in a `Suspended` state.

For example, when the app is attached to the Visual Studio debugger, you can send it a **Resume** event. The **Debug Location toolbar** should be shown and then, click the drop-down to the **Suspend** icon. In the list choose **Resume**. You can see the debug process is resumed again.

Here are some more example situations when the application needs to be resumed once the user returns back to access the application.

1. Web browsing session
2. Shopping cart
3. Incomplete e-mail
4. Paused movie or game

8.9 App Removal

When the user deletes a sub app, along with all its local data, the app is removed. Deleting an app will not affect the user's data (files in the Documents or Pictures libraries). When a Windows Store app is installed in the user's profile and hence, the user can have full control over the Windows Store apps. The user can install, update, and remove any store apps without affecting the apps of other users installed in the same system. Consider User A and User B working on the same computer. User A wishes to install a new Windows store app named Skype. User B already has Skype installed on his profile, but later User A uninstalls the app. App will still remain for User B to access in his profile.

8.10 App Close

Users are not required to close the app as Windows manage them. Developer need not provide any UI in the app to enable the user to close it. Users can close an app by pressing **Alt+F4**, or by selecting close gesture. However, there is no particular occurrence to specify that the user has closed an app.

Whereas, in Windows 8, once the user closes the app, it enters the `NotRunning` state after it is suspended and terminated. Whereas, in Windows 8.1, once the user closes the app, then, without terminating the app is removed from the screen.

When the app is suspended it calls the event handler for the `Suspending/suspending` event which is registered and can be used to save relevant application and constantly store user data.

It is recommended that the developer should analyze how the app behaves when it is activated when the user has closed it. It is better to use the activation event handler to conclude whether the app got ended by the user or by Windows. For the `ApplicationExecutionState` enumeration refer to `ClosedByUser` and Terminated states in the docs.

In normal application execution, it should not close programmatically. If done Windows will treat it as an app crash because the app enters into the `NotRunning` state and remains there until the user activates it. The user is not going to activate app since it has been closed by the app programmatically and it will result in a system crash.

8.11 App Crash

During app crash the system simply returns back to the Start screen. This experience is designed for users to get back to what they were doing as quickly as possible. Here, it is not required to provide other notification or a warning dialog box because that will cause a delay for the user. Through the vanishing of the app, user comes to know that something might have gone wrong.

Windows asks the user for permission to send a problem report to Microsoft if the app stops responding or crashes or generates an exception. Microsoft provides a list of error data in the report to improve the app. *T* is available in your app's Quality page present in Dashboard in the Windows Dev Center for Windows Store apps.

Once the app is activated and when it crashes, the activation event handler receives an `ApplicationExecutionState` value of `NotRunning`.

8.12 Check Your Progress

1. App packages contain the _____ files extension.

(A)	.doc	(C)	.appx
(B)	.txt	(D)	.cpp

2. In which state the app process will run when it is launched and app is activated by the user?

(A)	NotRunning state	(C)	PreviousExecutionState
(B)	Running State	(D)	ApplicationExecutionState

3. Which type of activation is used to choose contacts?

(A)	Cached file	(C)	Contact picker
(B)	Device	(D)	File save picker

4. Which type of activation is used to launch a URL whose protocol an app is registered to handle?

(A)	Protocol	(C)	Share target
(B)	Search	(D)	Share target

5. Which action should be taken to restore session data?

(A)	Closed by the user	(C)	Start with default data
(B)	Terminated by the system due to resource constraints	(D)	NotRunning

8.12.1 Answers

1.	C
2.	A
3.	C
4.	A
5.	B

Summary

- App packages contain the files extension .appx that comprises the app.
- In a single package the Windows Store app model provides installation, updates data, and instructions for an app.
- The basic tasks for the app is to set up custom UI and register event handlers which are required for loading.
- App receives an activated event if the Windows has terminated app and then, the user launches the app.
- In Windows 8, the apps will not be activated when you log in using the built-in administrator account.
- Developer should analyze how the app behaves when it is activated by the user that has closed it.
- When an app is in running state it doesn't receive an activation event but app visibility changes.