Windows 8
Design and coding guidelines

Updated for Windows 8.1

Get recommendations from the experts. These guidelines can help you design great Windows Store and Windows Phone Store apps. They cover a variety of essential topics, including layout, controls, accessibility, user interactions, text, and animation.

For the online version of these guidelines, see the Design Guidelines section of the Windows Dev Center.



This document is provided “as-is”. Information and views expressed in this document, including URL and other Internet Web site references, may change without notice.

Some examples depicted herein are provided for illustration only and are fictitious. No real association or connection is intended or should be inferred.

This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes.

© 2014 Microsoft. All rights reserved.

Microsoft, Windows, Windows Media, Windows Phone, Windows Vista, the Windows logo, Bing, Calibri, Cambria, DirectX, Halo, Hotmail, InPrivate, IntelliSense, Internet Explorer, Internet Explorer logo, MSN, Nirmala, OneDrive, OneDrive design 2012, OpenType, Outlook, PowerPoint, Segoe, Silverlight, Skype, Skype logo, Visio, Visual Basic, Visual Studio, Wonderwall, Wordament, and Xbox are trademarks of the Microsoft group of companies. All other trademarks are property of their respective owners.

Table of Contents

Introduction 7
Microsoft design principles 7
Animation 11
Guidelines for add and delete animations 12
Guidelines for content transition animations 13
Guidelines for drag animations 14
Guidelines for edge-based UI animations 16
Guidelines for fade animations 17
Guidelines for page transition animations 18
Guidelines for pointer click animations 21
Guidelines for reposition animations 22
Guidelines for pop-up UI animations 23
Guidelines for swipe animations 24
Contracts and charms 25
Guidelines for app settings 26
Guidelines for developing audio-aware apps 30
Lens design guidelines (Windows Phone Store app) 33
Guidelines for the camera UI 41
Guidelines for devices that access personal data 42
Guidelines for file picker contracts 51
Guidelines for file pickers 57
Guidelines for geofencing apps 61
Guidelines for location-aware apps 64
Guidelines for printing 72
Guidelines for print UI design 73
Guidelines for Proximity 79
Guidelines for sharing content 81
Controls 86
Guidelines for app bars 87
Guidelines for back buttons 93
Guidelines for buttons 94
Guidelines for check boxes 97
Guidelines for context menus 101

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for date pickers 106
Guidelines for drop-down lists 108
Guidelines for flip view controls 110
Guidelines for flyouts 113
Guidelines for hub controls (Windows Store apps) 119
Hub control design guidelines (Windows Phone Store apps) 131
Label (or text block) 136
Guidelines for links 141
Guidelines for list boxes (or select) 144
Guidelines for list and grid view controls 147
Guidelines for Maps 155
Guidelines for message dialogs 156
Guidelines for Pivots (Windows Phone Store apps) 161
Guidelines for progress controls 165
Guidelines for radio buttons 176
Guidelines for the rating control 181
Guidelines for search 184
Guidelines for scroll bars 191
Guidelines for semantic zoom 193
Guidelines for sliders 200
Guidelines for time pickers 205
Guidelines for toggle switch controls 207
Guidelines for tooltips 210
Guidelines for Web views 213
Guidelines for files, data, and connectivity 215
Guidelines for connection usage data 216
Guidelines for creating custom data formats 219
Guidelines for file types and URIs 224
Guidelines for login 229
Guidelines for roaming app data 233
Guidelines for accessing OneDrive from an app 237
Guidelines for single sign-on and connected accounts 241
Guidelines for thumbnails 246
Guidelines for user names and account pictures 251
Guidelines for files, data, and globalization 255

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for app resources ... 256
Guidelines for globalization and localization ... 257
Guidelines for help and instructions ... 261
Guidelines for app help ... 262
Guidelines for designing instructional UI ... 263
Guidelines for launch, suspend, and resume ... 267
Guidelines for app suspend and resume ... 268
Guidelines for splash screens ... 270
Guidelines for layout and scaling ... 276
Guidelines for advertising ... 277
Guidelines for multiple windows ... 280
Guidelines for projection manager ... 283
Guidelines for resizing to narrow layouts ... 285
Guidelines for scaling to pixel density ... 290
Guidelines for supporting multiple screen sizes ... 292
Guidelines for text and input ... 299
Guidelines for clipboard commands ... 300
Guidelines for find-in-page ... 302
Guidelines for fonts ... 304
Guidelines for form layouts ... 314
Guidelines for password boxes ... 326
Guidelines for Segoe UI Symbol icons ... 328
Guidelines for spell checking ... 338
Guidelines for text input ... 341
Guidelines for typography ... 346
Guidelines for tiles and notifications ... 355
Lock screen design guidelines (Windows Phone Store apps) ... 356
Guidelines for periodic notifications ... 362
Guidelines for push notifications ... 364
Guidelines for raw notifications ... 366
Guidelines for scheduled notifications ... 368
Guidelines for secondary tiles ... 369
Guidelines for tiles and badges ... 371
Guidelines for toast notifications ... 386
Guidelines for interactions ... 389

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for designing accessible apps ... 390
Guidelines for cross-slide ... 394
Guidelines for optical zoom and resizing ... 401
Guidelines for panning ... 404
Guidelines for rotation ... 411
Guidelines for selecting text and images (Windows Runtime apps) ... 415
Speech design guidelines (Windows Phone Store apps) ... 421
Guidelines for targeting ... 427
Guidelines for touch interactions ... 431
Guidelines for touch keyboard ... 436
Guidelines for visual feedback ... 440

© 2014 Microsoft. All rights reserved. | May 13, 2014

Introduction

Microsoft design principles



Here are five principles for building great Windows Store apps. Use these principles when you plan your app, and always ensure that your design and development choices live up to them.

Pride in craftsmanship



Engineer the experience to be complete, thorough, and polished at every stage. Devote time and energy to small things that are seen often by many of your users.

- Sweat the details.
- Make using apps safe and reliable.
- Use balance, symmetry, and hierarchy.
- Align your app layout to the grid, the new layout for apps.
- Make your app accessible to the widest possible audience, including people who have impairments or disabilities.

© 2014 Microsoft. All rights reserved. | May 13, 2014

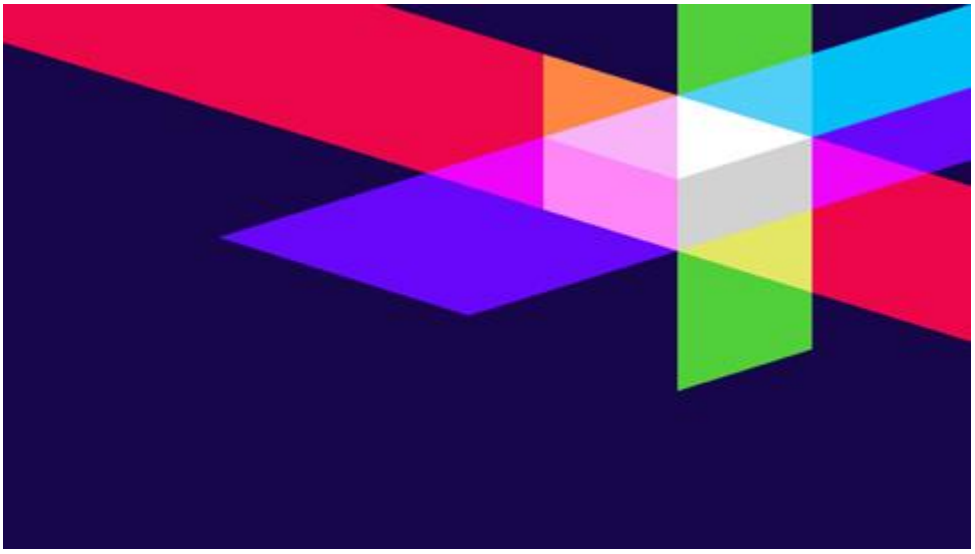
Be fast and fluid



Let people interact directly with content. Respond to actions quickly with matching energy. Bring life to the experience by creating a sense of continuity and telling a story through meaningful use of motion.

- Be responsive to user interaction and ready for the next interaction.
- Design for touch and direct interaction.
- Delight your users with motion.
- Smoothly connect to what comes before and after.

Authentically digital



© 2014 Microsoft. All rights reserved. | May 13, 2014

Exemplify the capabilities of hardware and software. Take full advantage of the digital medium. Remove physical boundaries to create experiences that are more efficient and effortless than reality. Being authentically digital means embracing the fact that apps are pixels on a screen. It means designing with colors and images that go beyond the limits of the real world.

- Be dynamic and alive with communication.
- Use typography beautifully.
- Use bold, vibrant colors.
- Connect to the cloud so that your users can stay connected to each other.

Do more with less

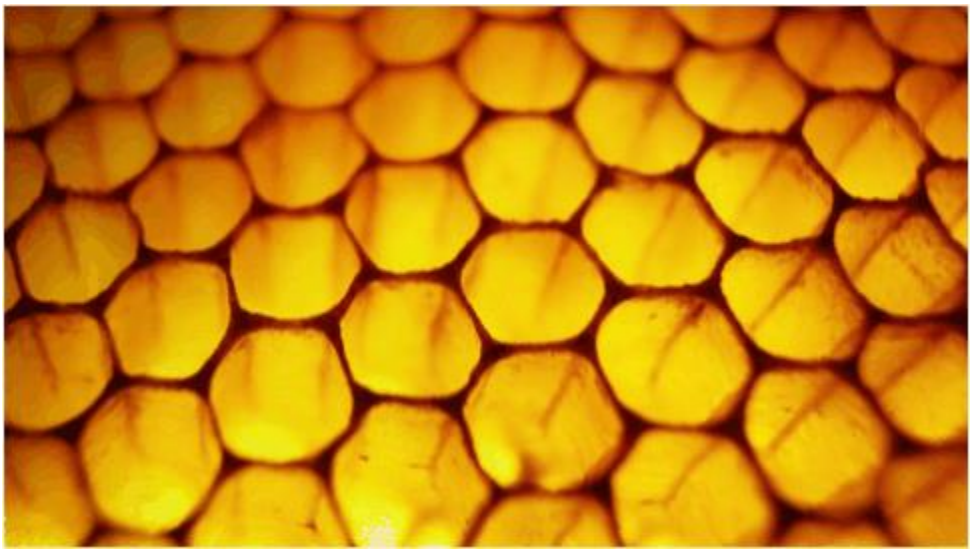


You can do more with less by reducing your design to its essence. Create a clean and purposeful experience by leaving only the most relevant elements on screen so people can be immersed in the content.

- Be great at something instead of mediocre at lots of things.
- Put content before chrome.
- Be visually focused and direct, letting people get immersed in what they love, and they will explore the rest.
- Inspire confidence in users.
- Reduce redundancy in your UI.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Win as one



Work with other apps, devices, and the system to complete scenarios for people. For example, let people get content from one app and share it with another. Take advantage of what people already know, like standard touch gestures and charms, to provide a sense of familiarity, control, and confidence.

- Use the UI model.
- Work with other apps to complete scenarios by participating in app contracts.
- Use our tools and templates to promote consistency.

Following these five Microsoft design principles will help you make the best choices when you design your app.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Animation

Purposeful, well-designed animations bring apps to life and make the experience feel crafted and polished. Help users understand context changes, and tie experiences together with visual transitions.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for add and delete animations



List animations enable you to insert or remove single or multiple items from a collection, such as a photo album or a list of search results.

Unless specified otherwise, these guidelines apply to both list and search list animations.

Dos and don'ts

- Use the list animations to add a single new item to an existing set of items. For example, use them when a new email arrives or a new photo is imported into an existing set.
- Use the list animations to add several items to a set at one time. For example, use them when you import a set of new photos into an existing collection. The addition or deletion of multiple items should happen at the same time, with no delay between the action on the individual objects.
- Use the add and delete list animations as a pair. Whenever you use one of these animations, use the corresponding animation for the opposite action.
- Use the list animations with a list of items to which you can add or delete one element or group of elements at once.
- Use the search list animations when your results will reflow multiple times one after another, such as when you use a word wheel in a search filter.
- Don't use the list animations to display or remove a container. These animations are for members of a collection or set that is already being displayed. Use the pop-up animations to show or hide a transient container on top of the app surface. Use the content transition animations to display or replace a container that is part of the app surface.
- Don't use the list animations on an entire set of items. Use the content transition animations to add or remove an entire collection within your container.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for content transition animations



Content transition animations enable you to change the content of an area of the screen while keeping the container or background constant. New content fades in. If there is existing content to be replaced, that content fades out.

Dos and don'ts

- Use content transitions when there is a set of new items to bring into an empty container. For example, after the initial load of an app, part of the app's content might not be immediately available for display. When that content is ready to be shown, use a content transition animation to bring that late content into the view.
- Use content transitions to replace one set of content with another set of content that already resides in the same container within a view.
- When bringing in new content, slide that content into the view against the general page flow or reading order. For instance, if the animation is to bring new content to a document that flows from left to right, then the incoming content should move in from right to left.
- Introduce new content in a logical manner, for example, introduce the most important piece of content last.
- If you have more than one container whose content is to be updated, trigger all of the transition animations simultaneously without any staggering or delay.
- Don't use content transition animations when the entire page is changing. In that case, use the page transition animations instead.
- Don't use content transition animations if the content is only refreshing. Content transition animations are meant to show movement. For refreshes, use fade animations.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for drag animations



Use drag-and-drop animations when users move objects, such as moving an item within a list, or dropping an item on top of another.

Dos and don'ts

Drag start animation

- Use the drag start animation when the user begins to move an object.
- Include affected objects in the animation if and only if there are other objects that can be affected by the drag-and-drop operation.
- Allow the user to move the object somewhat before you trigger the drag start animation. This will prevent the user from accidentally dragging an object that they only meant to tap or select. The recommended threshold is 20 touch independent pixels (TIPs).
- Use the drag end animation to complete any animation sequence that began with the drag start animation. This reverses the size change in the dragged object that was caused by the drag start animation.

Drag end animation

- Use the drag end animation when the user drops a dragged object.
- When the user drops an object to reorder a list, you must often reposition the other items in the list to make room for the item being dropped. To do so, after the drag end animation is complete, use the list animation for adding an item but with no added item. The item being added is already present. This will animate all elements into their proper positions.
- If the drag source will disappear after being dropped (for example, when the user drops a file onto a folder icon to store the file in that folder), use the fade out animation on the drag source.
- Include affected objects in the drag end animation if and only if you included those same affected objects in the drag start animation.
- Don't use the drag end animation if you have not first used the drag start animation. You must use both animations to return objects to their original sizes after the drag sequence is complete.

Drag between enter animation

- Use the drag between enter animation when the user drags the drag source into a drop area where it can be dropped between two other objects.
- Choose a reasonable drop target area. This area should not be so small that it is difficult for the user to position the drag source for the drop.
- The recommended distance to move affected objects to show the drop area is 40 pixels.

© 2014 Microsoft. All rights reserved. | May 13, 2014

- The recommended direction to move affected objects to show the drop area is directly apart from each other. Whether they move vertically or horizontally depends on the orientation of the affected objects to each other.
- Don't use the drag between enter animation if the drag source cannot be dropped in an area. The drag between enter animation tells the user that the drag source can be dropped between the affected objects.

Drag between leave animation

- Use the drag between leave animation when the user drags an object away from an area where it could have been dropped between two other objects.
- Don't use the drag between leave animation if you have not first used the drag between enter animation.

© 2014 Microsoft. All rights reserved. | May 13, 2014

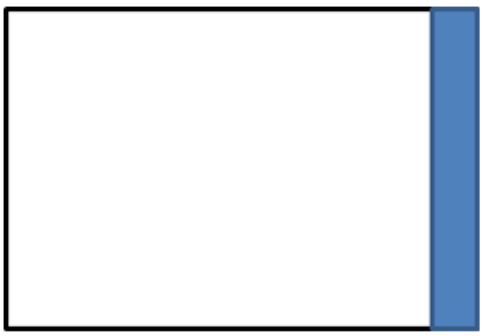
Guidelines for edge-based UI animations



Edge-based animations show or hide UI that originates from the edge of the screen. The show and hide actions can be initiated either by the user or by the app. The UI can either overlay the app or be part of the main app surface. If the UI is part of the app surface, the rest of the app might need to be resized to accommodate it.

Dos and don'ts

- Use edge UI animations to show or hide a custom message or error bar that does not extend far into the screen.
- Use panel animations to show UI that slides a significant distance into the screen, such as a task pane or a custom soft keyboard.
- Slide the UI in from the same edge that it will be attached to.
- Slide the UI out to the same edge that it came in from.
- If the contents of the app need to resize in response to the UI sliding in or out, use fade animations for the resize.
    - If the UI is sliding in, use a fade animation after the edge UI or panel animation.
    - If the UI is sliding out, use a fade animation at the same time as the edge UI or panel animation.
- Don't apply these animations to notifications. Notifications should not be housed within edge-based UI.
- Don't apply the edge UI or panel animations to any UI container or control that is not at the edge of the screen. These animations are used only for showing, resizing, and dismissing UI at the edges of the screen. To move other types of UI, use reposition animations.



Use edge UI animations



Use reposition animation

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for fade animations



Use fade animations to bring items into a view or take them out of a view. There are three types of fade animations: fade in, fade out, and crossfade.

Dos and don'ts

- When your app transitions between unrelated or text-heavy elements, use a fade out followed by a fade in instead of a crossfade. This allows the outgoing object to disappear completely before the incoming object is visible. Crossfading lots of text, in particular, is not recommended.
- Use the crossfade animation to transition between elements of different sizes or when you refresh a large area. In the middle of the animation, both the incoming and outgoing elements will be semitransparent and the background will be visible to the user. Note that programming languages used with Extensible Application Markup Language (XAML) do not contain a dedicated crossfade animation. In those languages, you can achieve the effect by using the fade in and fade out animations with overlapped timing.
- Fade in the incoming element or elements on top of the outgoing elements, if the elements' size remains constant and you want the user to feel that they're looking at the same item. You can fade in the incoming item on top of the outgoing item. Then, once the fade in is complete, the outgoing item can be simply removed. Of course this is only a viable option when the outgoing item will be completely covered by the incoming item. This method also prevents the background from becoming visible during the transition.
- Don't use fade animations to add or delete items in a list. Use the list animations created specifically for that purpose.
- Don't use fade animations to change the entire contents of a page. Use the page transition animations created specifically for that purpose.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for page transition animations



Use page transition animations to display the first page of a newly launched app, or to transition between pages within an app.

Note: When only a portion of the screen's content will change, use content transition animations rather than page transition animations.

Dos and don'ts

- Split your page along natural borders or boundaries into a set of between two and five regions. Apply staggered timings to the regions so that the regions appear sequentially rather than all at once, and offset the regions by 100 pixels for a wide app layout. You can use a smaller offset if your app has a special layout for narrow state.
- Make sure that any content that the outgoing and incoming pages have in common stays in place, without any animations applied to them. For example, if a **Back** button is present on both the outgoing and incoming page, it should not be included in the transition animation.
- If the outgoing page does not have a **Back** button (such as an app's first page) but the incoming page does, the **Back** button should be specified as a separate region and that region should animate into view before any others.
- If your outgoing and incoming pages have different backgrounds, use the fade in animation to show the new background. Start the fade in animation at the same time as the page transition animation.
- If some of the content on the incoming page is not ready to display immediately, use the page transition animation to bring in as much content as is ready at that time. Meanwhile, if necessary, show a progress control while you ready the additional content. Once the additional content is ready to display, animate it into place based on its content area. For a large content area, use the content transition animation. For a small content area or discontinuous content, use the fade in animation.
- Slide the page into the view against the general page flow or reading order. For example, if the content on the incoming page flows from left to right, then the incoming page should slide in from right to left. In apps with right-to-left reading order, the incoming page should slide in from left to right. Similarly, when you split a page into sections as described in the following illustrations, the order that those sections are brought into the view should be the opposite of the reading order.
- Don't run page transition animations when a user resizes an app window. Page transition animations are only for navigating from one page to another within a specific view. The system handles the animation between the old and new layout when the view changes.

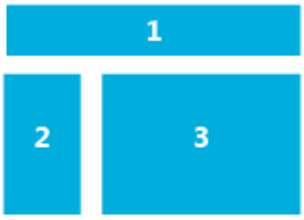
© 2014 Microsoft. All rights reserved. | May 13, 2014

Additional usage guidance

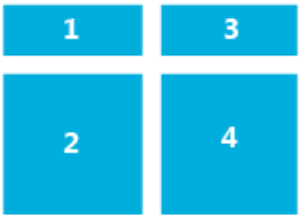
The most common page divisions, including the order they should appear, are shown here:



2 stages: Header, content

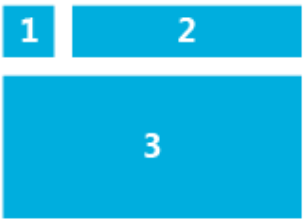


3 stages: Header, left content, right content

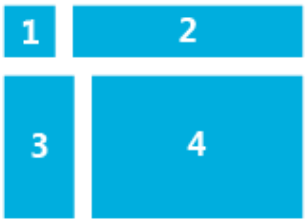


4 stages: Left header, left content, right header, right content

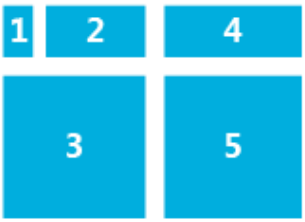
The most common page divisions, with back buttons, are shown here. If the outgoing page does not have a **Back** button (such as an app's first page) but the incoming page does, the **Back** button should be specified as a separate region and that region should animate into view before any others.



3 stages: Back button, header, content



4 stages: Back button, header, left content, right content



5 stages: Back button, left header, left content, right header, right content

The most common page divisions used with an app that is displayed in a narrow width or portrait view, and the order in which the divisions should appear, are shown here. If the **Back** button is already present on the outgoing page, it should stay in place, not included in the animation.

© 2014 Microsoft. All rights reserved. | May 13, 2014



2 stages: Header, content



3 stages: Header, top content, bottom content



3 stages: Back button, header, content



4 stages: Back button, header, top content, bottom content

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for pointer click animations



Use pointer animations to provide users with visual feedback of a tap or click on an item. The pointer down animation is played when a user taps or clicks on an item. The animation slightly shrinks the item to indicate that it is pressed. The pointer up animation is played when the tap or click is released. This animation restores the item to its original size, to indicate that it has been released.

Dos and don'ts

- When you use a pointer up animation, be sure to trigger the animation immediately when the user releases the tap or click. It is critical that the pointer up animation appears quickly. This provides instant feedback to the user that their action has been recognized, even if the action triggered by the tap or click (such as navigating to a new page) is slower to respond.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for reposition animations



Use the reposition animation to move an element or elements into a new position.

Dos and don'ts

- Don't use the reposition animation to show or hide edge-based UI. Edge-based UI is an element or container that is anchored at one edge of the screen. Use the edge-based UI animations created specifically for that purpose.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for pop-up UI animations



Use pop-up animations to show and hide pop-up UI, which includes context menus and flyouts. Pop-up elements are containers that appear over the app's content and are dismissed if the user taps or clicks outside of the pop-up element.

Appropriate use of pop-up animations

- Use pop-up animations to show or hide custom pop-up UI elements, such as a context menu, a flyout, or other contextual UI that not a part of app page itself. The recommended distance for the pop-up element to move as it appears is 50 pixels. The common controls provided by Windows already have these animations built in.
- Don't use pop-up animations for tooltips or dialogs. Use the fade animations to show and hide custom tooltips or dialogs.
- Don't use pop-up animations to show or hide UI within the main content of your app. Use pop-up animations only to show or hide a pop-up container that displays on top of the main app content.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for swipe animations



Use swipe animations when you implement the swipe gesture for selection of an item.

Dos and don'ts

- If the user did not drag the item far enough to toggle the selection state before it was released, use the swipe select animation to move the item back to its rest position without changing the selection state.
- Make sure the direction of the swipe animation matches the direction of the user's swipe gesture.
    - In the case of horizontally scrolling content, the swipe direction is vertical, so the swipe animation sequence should move down and then back up.
    - In the case of vertically scrolling content, the swipe direction is horizontal, so the swipe animation sequence should move horizontally according to the reading order of the app.
        - For an app with a left-to-right reading order, the animation should move right and then left.
        - For an app with a right-to-left reading order, the animation should move left and then right.
- Move the item the recommended distance for swipe animations, which is 15 pixels if moving vertically or 23 pixels if moving horizontally.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Contracts and charms

This section includes guidelines for charms: settings, devices, and sharing. It also includes guidelines for file pickers and file picker contracts.

© 2014 Microsoft. All rights reserved. | May 13, 2014

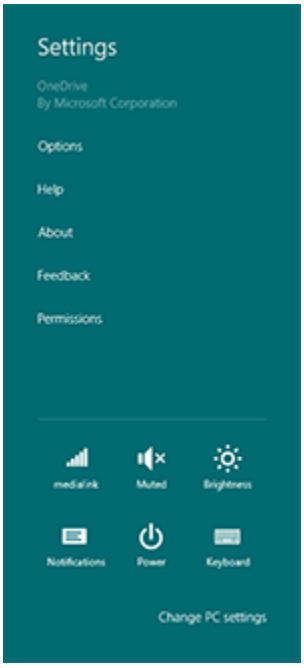
Guidelines for app settings



Windows provides a Settings pane by default for every Windows Store app. The top portion of the Settings pane lists "entry points" for your app settings. Each entry point opens a Settings flyout that displays the settings options themselves. This topic describes best practices for creating and displaying app settings.

Example

Users swipe from the side of the screen to display the charms and then press the Settings charm to show a Settings pane.



Note: Windows adds two default entry points to every Settings pane in a Windows Store app: **Permissions** and **Rate and review**. Note that if your app isn't installed through the Windows Store (if, for example, it's a side-loaded enterprise app), your app's Settings pane won't include a **Rate and review** entry point. The bottom portion of the Settings pane includes system settings like volume, brightness, and power.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Should my app include app settings?

All Windows Store apps include a Settings pane with **Permissions** and **Rate and Review** sections by default. Following are examples of additional settings that belong in the Settings pane:

- Configuration options that affect the behavior of the app and don't require frequent readjustment, like choosing between Celsius or Fahrenheit as default units for temperature in a weather app, or changing account settings for a mail app.
- Options that depend on the user's preferences, like music, sound effects, or color themes.
- App info that's not needed very often, such as privacy statements, help, app version, or copyright info.
- A link to your Privacy Policy if you have Internet Connection declared in your capabilities. Forgetting to link to this policy is the most common certification blocker!

Commands that are part of the typical app workflow (for example, changing the brush size in an art app) shouldn't be in the Settings pane. Instead, put commonly used commands on the top or bottom app bar.

Do and don'ts

General principles

- Create entry points for all of your app settings in the Settings pane.
- Keep your settings simple. Define smart defaults and keep the number of settings to a minimum.
- If necessary, link from elements of your app's UI to the Settings pane or deep-link to a specific Settings flyout. For example, you can link to your help Settings flyout from a "Help" button in the bottom app bar as well as from a "Help" entry point in the Settings pane.
- When a user changes a setting, the app should reflect the change immediately. Apply settings changes instantly or as soon as a user is done interacting with the flyout.
- Use the Settings flyout controls available for WinJS and XAML. These controls implement the UI design guidelines by default.
- Don't include commands that are part of common app workflow, like changing the brush color in an art app, in app settings. These commands belong on an app bar or on the canvas.
- Don't use an entry point in the Settings pane to perform an action directly. Entry points should open Settings flyouts.
- Don't use the Settings pane for navigation. When the Settings pane closes, a user should be in the same place in the app that they were when they clicked the Settings charm. The top app bar is a more appropriate place for navigation.
- Don't use the Settings flyout classes as all-purpose controls. They're intended only for Settings flyouts launched from entry points in the Settings pane.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Entry points

Entry points are the labels that appear at the top of the Settings pane and open Settings flyouts, where you can display one or more settings options. Once you have a list of settings you want to include, consider these guidelines for the entry points:

- Group similar or related options together under one entry point. Avoid adding more than four entry points to your Settings pane.
- Display the same entry points regardless of the app context. If some settings are not relevant in a certain context, disable them in the Settings flyout.
- Use descriptive, one-word labels for your entry points when possible. For example, for account-related settings, call the entry "Accounts" rather than "Account settings." If you want only one entry point for your settings and the settings don't lend themselves to a descriptive label, use "Options" or "Defaults."
- If an entry point is linking directly to the web instead of a flyout, let the user know with a visual clue, for example, "Help (online)" or "Web forums" styled as a hyperlink. Consider grouping multiple links to the web into a flyout with a single entry point. For example, an "About" entry point could open a flyout with links to your terms of use, privacy statement, and app support.
- Combine less-used settings into a single entry point so that more common settings can each have their own entry point. Put content or links that only contain information in an "About" entry point.
- Don’t duplicate the functionality in the "Permissions" pane. Windows provides this pane by default and you can't modify it.

Design a Settings UI

Note: The Settings flyout controls available in WinJS and XAML follow these guidelines by default, except for those pertaining to header and border color.

- Always launch a Settings flyout from entry points in your Settings pane.
- Use a light-dismiss surface that appears on top of the main app content and disappears when the user clicks outside the flyout or resizes the app. This lets people change a setting quickly and get back to the app.
- Make sure that your Settings flyout appears on the same side of the screen as the charms and Settings pane. Use the **SettingsEdgeLocation** property to determine which side of the screen the Settings charm appears on.
- Slide the flyout in from the same side of the screen as your Settings pane, rather than from the top or bottom of the screen.
- A flyout must be full screen height, regardless of orientation, and should be narrow (346 pixels) or wide (646 pixels). Choose the width that's appropriate for the content and don't create custom sizes.
- The flyout header should include a back button, the name of the entry point that opened the flyout, and the app’s icon.
- The header background color should be the same as the background color of your tile.
- The border color should be the same color as the header, but 20% darker.
- Display Settings content on a white background.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Add settings content to Settings flyouts

- Present content from top to bottom in a single column, scrollable if necessary. Limit scrolling to a maximum of twice the screen height.
- Use the following controls for app settings:
    - **Toggle switches**: To let users set values on or off.
    - **Radio buttons:** To let users choose one item from a set of up to 5 mutually exclusive, related options.
    - **Text input box:** To let users enter text. Use the type of text input box that corresponds to the type of text you're getting from the user, such as an email or password.
    - **Hyperlinks:** To take the user to another page within the app or to an external website. When a user clicks a hyperlink, the Settings flyout will be dismissed.
    - **Buttons:** To let users initiate an immediate action without dismissing the current Settings flyout.
- Add a descriptive message if one of the controls is disabled. Place this message above the disabled control.
- Animate content and controls as a single block after the Settings flyout and header have animated. Animate content using the **enterPage** or **EntranceThemeTransition** animations with a 100px left offset.
- Use section headers, paragraphs, and labels to aid organize and clarify content, if necessary.
- If you need to repeat settings, use an additional level of UI or an expand/collapse model, but avoid hierarchies deeper than two levels. For example, a weather app that provides per-city settings could list the cities and let the user tap on the city to either open a new flyout or expand to show the settings options.
- If loading controls or web content takes time, use an indeterminate progress control to indicate to users that info is loading.
- Don't use buttons for navigation or to commit changes. Use hyperlinks to navigate to other pages, and instead of using a button to commit changes, automatically save changes to app settings when a user dismisses the Settings flyout.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for developing audio-aware apps



Windows app: the default transport controls of a media element

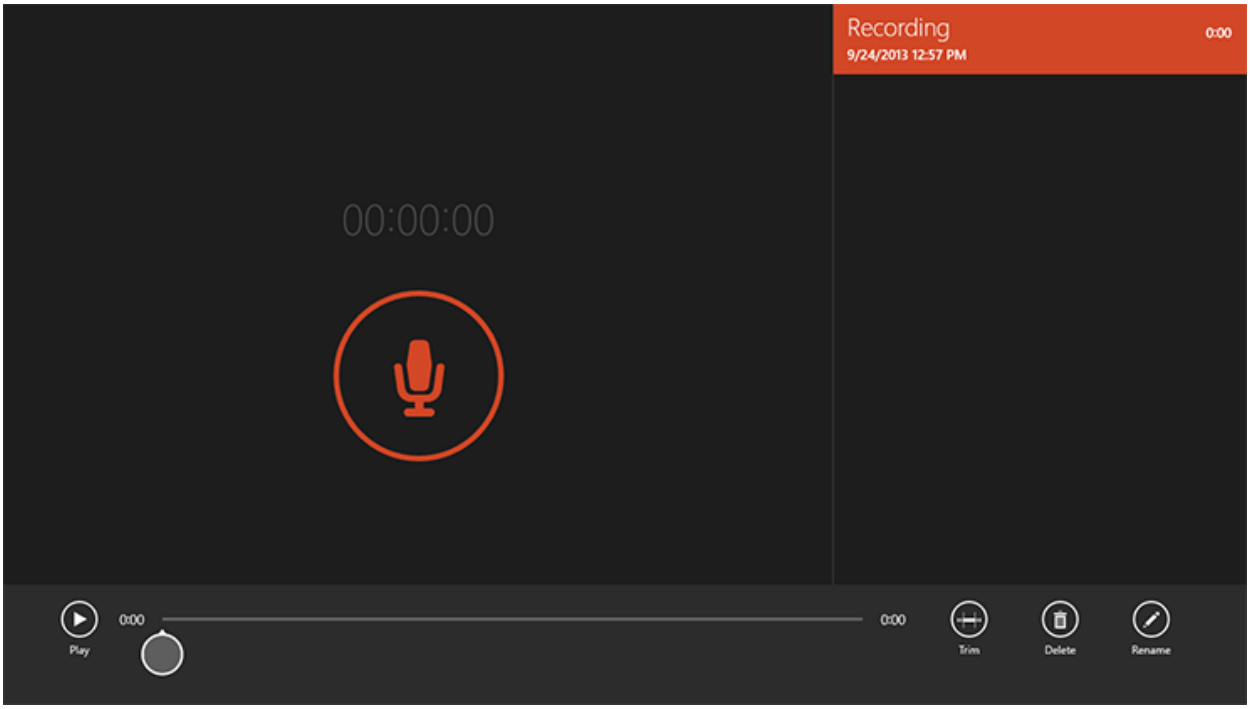
Description

This topic describes best practices for developing audio-aware applications, including things to consider for designing media elements, using playback manager, and managing call control.

Windows Phone: There are several media-related certification requirements that you should become familiar with when you develop any media app. For more info, see sections 6.4 and 6.5 of Additional requirements for specific app types for Windows Phone.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Example



Windows app: custom transport controls

Dos and don'ts

Media elements

- Use a media elements to show video and audio to the user, either full-screen or alongside other visuals. You can show more than one media element on the screen together, and you can programmatically control which is one enabled if you don't want them to be played back at the same time.
- Place media elements—and their transport controls—in a consistent location to help users interact with them confidently and seamlessly. The media element presents frames of video. By default, transport controls are not displayed, but they can be turned on and off with a simple Boolean property. Transport controls—including a full-screen button—appear when the media element is initially displayed, and when it's tapped, and then they fade out.
- If you create your own custom transport controls, show them when the media element is initially displayed. They can then fade out to give a clear view of the media, but should return when the user taps the media element. Play/pause should be a toggle button with visuals that indicate the action that happens when tapped. The button should not show the current playing/paused state of the media. Try to use established media playback symbols when showing skip, stop, play, pause, and seek operations in audio and video.
- Design your button-press event handlers to respond with the least amount of delay possible. This will make sure that the user immediately gets verification of their button

© 2014 Microsoft. All rights reserved. | May 13, 2014

input. Long delays in response cause the user to press buttons multiple times, resulting in unexpected app behavior.
- Make sure that the media buttons are used in their standard ways so that the user has a familiar experience with the use of the media buttons.
- Don't use track and artist name strings that are longer than 127 characters or an error occurs. If you do not handle this error properly, it could cause the app to stop functioning.

Playback manager

- Use the **msAudioCategory**/**MediaElement.AudioCategory** assignment only if you need audio to play in the background. Audio playback drains the battery, so unless there is a clear need for background audio (media playback designed for longer term listening, for example) do not declare an audio category. Alternatively, you can use the "Other" category. Otherwise, your application will be muted and then suspended.
- Use low-latency audio only when it is needed for specific applications that require it (including multi-track recorders and low-latency video capture). Low-latency audio is automatically invoked when you select the "Communications" audio category. For any other category, consider leaving low-latency settings at their default (which is OFF). Low-latency buffers use significantly more CPU and battery resources, and are typically reserved for foreground applications on which the user is focused.
- If you want to mute the playback of background media, choose ForeGroundOnlyMedia from the **msAudioCategory**/**MediaElement.AudioCategory** for this soundtrack. If you develop a game that plays its own audio soundtrack while the user is playing the game, then the soundtrack will be muted if the user was already playing an audio track in the background when the game started. If you believe that the game's audio soundtrack is essential for the function of the game, you can mute the currently playing background audio. "SoundEffects" that are mixed in with background media will still be heard in either case.

  "ForeGroundOnlyMedia" can also be selected for video apps that should stop background media when the video starts, and should not run at all when they are in the background.

Call control

The following table presents the practices recommended for managing call control on the default Bluetooth communications device.

- Make call control functionality predictable for communications applications. This ensures that the user has an experience that is familiar and seamless. If your app uses call buttons in an unfamiliar way, make it very obvious to the user.
- Track call tokens carefully. Make sure you end the call correctly for the device, and also end the audio/video stream. Doing this helps you make sure that when the call is completed, you send an "endcall" notification back to the device, with the appropriate call token. That way the user has an indication from the device that the call has ended.

© 2014 Microsoft. All rights reserved. | May 13, 2014

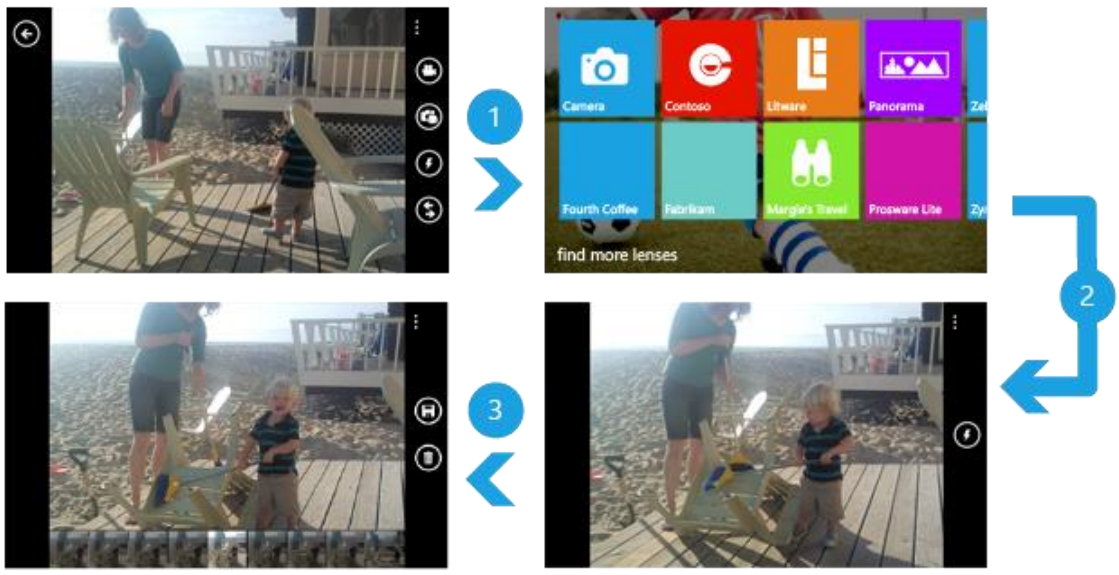
Lens design guidelines (Windows Phone Store app)

In this topic, we'll cover best design practices for using camera lenses and implementing rich media to create consistent, amazing lens app experiences on the Windows Phone. For info about developing lens apps, see Lenses for Windows Phone 8.

User experience guidelines for lenses

A lens app is intended to complement the built-in camera, to feel like a natural extension of both the camera and photo viewing experiences. Lenses are useful because they're a way to map real-world scenarios to an app that specializes in that scenario, such as panorama or group photography. Lenses achieve this by fulfilling two core functions:

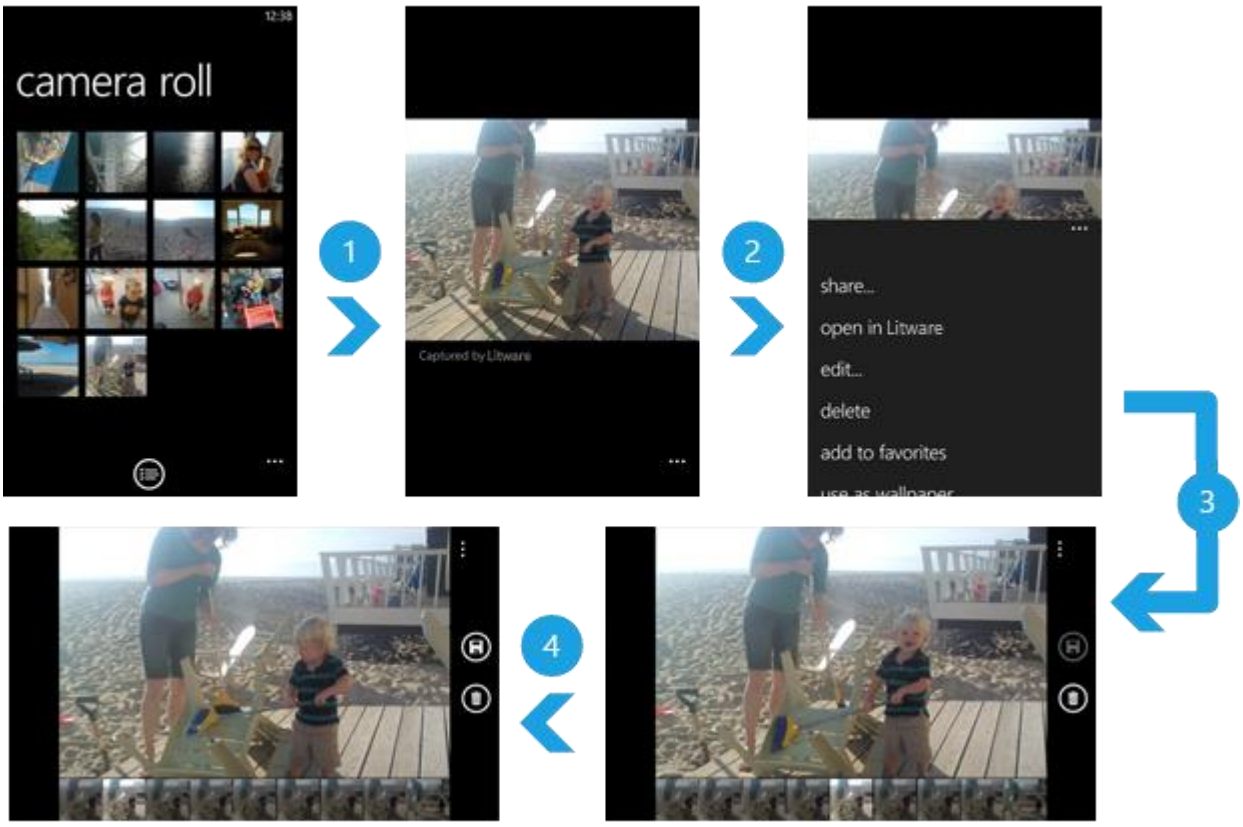
- **Capture:**



UI example: (1) tap the Lens switcher button, (2) choose a lens and capture the moment, (3) confirm and save the picture to the camera roll.

- **View and experience:**

© 2014 Microsoft. All rights reserved. | May 13, 2014



UI example: (1) all photos appear in your camera roll, (2) reopen the photo in the app that created it, (3) experience or edit, (4) save a new photo.

Important Note: When creating lens apps, it's imperative that you keep these fundamental points in mind:

- Lenses jump into a viewfinder-driven experience.
- Lenses save photos to the camera roll.
- Lenses that provide an enhanced viewing or editing experience re-launch that experience from the built-in photo viewer.

The launch experience

Lenses are primarily camera apps, and they launch in the context of the built-in camera experience. Although the camera experience on the device supports both portrait and landscape orientation, it's important to understand that your lens app will most likely launch when the user is holding their device like a camera (landscape orientation). Therefore, we recommend that your launch screen and the default orientation of the app are set to the landscape orientation.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Lenses are a viewfinder-driven experience. That means that a user who is launching a viewfinder-specific app should immediately land on an experience that makes use of viewfinder properties. There are exceptions to this rule, like if your app requires the user to input credentials or get legal consent from the user to use some aspect of the app.

For more info integrating your app with the built-in camera experience, see Lens extensibility for Windows Phone 8.

The capture experience

Generally speaking, the lens capture experience should be consistent with the built-in camera user experience unless there is a specific need to do otherwise. Consider these points to give your lens experience the consistency it needs:

- Gestures (specifically, Swipe left) and the experience should be created with respect to device orientation.
- Your app should support a left-pointing arrow icon—the indicator that additional photos are available—with respect to device orientation.
- Stock animation for Save and Capture should be consistent.
- Your app should support tap-to-capture and a camera hardware button.
- Support half-press to focus.
- Flash iconography and states should be available where relevant.
- Focus brackets should work like they do with the basic camera where relevant.

If a user can take a picture from within your lens app, the picture should immediately be saved in the user's camera roll. If the app takes multiple pictures during a capture, additional pictures (backing data) should be saved in the app's local folder, and a representation of these images should be saved in the camera roll.

Capture methods

Because lenses are applicable to a wide array of camera apps, it's important to distinguish the different types of capture methods available and the unique design guidance that applies to each.

Traditional capture

This type of lens app saves your photos directly to the camera roll and then immediately takes you back to the viewfinder.

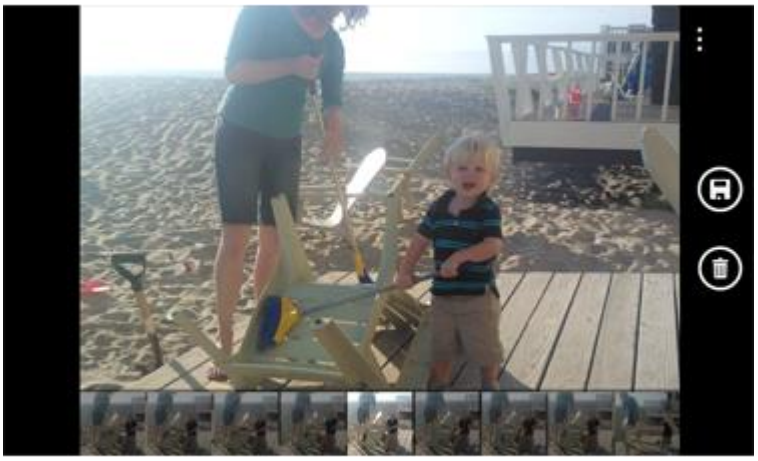
© 2014 Microsoft. All rights reserved. | May 13, 2014



A traditional capture app

Capture and confirm

This type of lens app requires the user to parse and accept captured images before saving them to the camera roll.



A capture and confirm app

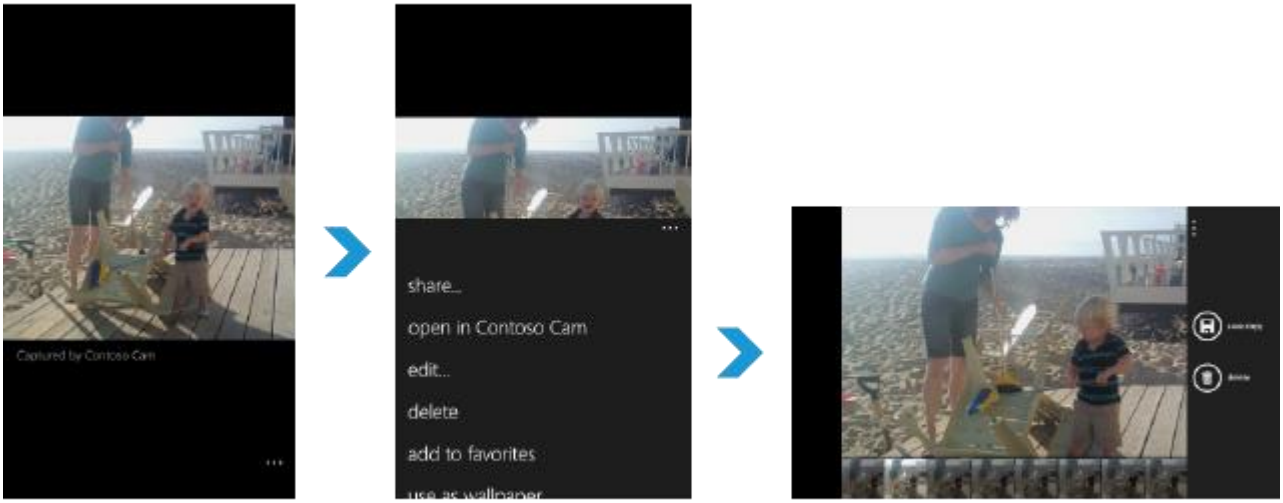
Capture and confirm apps should use a consistent set of icons (Save and Delete) along with animation for confirming and cancelling the storage of the item. Cancel and Save both need to return the user to the viewfinder. These icons are included with the Windows Phone SDK.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Linking back from the viewfinder experience

Although many lens apps will simply store a photo in the user's camera roll, Windows Phone lenses can capture content with far greater complexity than a traditional photo. Rich media lenses incorporate data from the local folder or from the web to provide a richer and deeper way for the user to engage with the images they have captured.

A rich media lens can store a photo that links back to their app. The photo stored in the camera roll can be shared or edited like any other photo in the camera roll. When viewing photos that represent rich media items in the built-in photo viewer, users can link back to the rich media experience associated with that item.



Rich media open link

The open link should launch an experience tailored to viewing or editing a selected item. It shouldn't be seen as a generic app launch point. Be sure to enable Save copy only when the user has made a change to the image. For more info about integrating your rich media lens with the built-in photo viewer, see Rich media extensibility for Windows Phone 8.

With rich media apps, it shouldn't be assumed that the stored image in the camera roll will be there. Users can delete items that are stored in the camera roll, so ensure that your rich media lens can recreate the experience without the representative image that was stored in the camera roll.

Users can share or edit items stored in the camera roll, so avoid using branding elements; let users share their images without dealing with unnecessary visual distractions.

Backing data associated with the captured photos can accumulate in the app's local folder. A rich media app cannot remove images from the camera roll, but it can clear out data from its local folder. These apps should provide the ability to navigate to any photo captured by the app and give users

© 2014 Microsoft. All rights reserved. | May 13, 2014

the ability to delete the backing data associated with the photos. If your rich media lens is creating a new copy of an item from the app, the action should not be Save, but rather, Save copy.

Here are some tips for navigating within a rich media experience:

- If you launch into a viewing or editing experience, Back should take you back to the camera roll.
- When launching into an editing experience, Save copy should keep the user in the app to show the confirmed changes. Delete should remove the backing data associated with the image.

If your app doesn't provide a rich media experience, don't declare a rich media extension in the app's WMAppManifest.xml file.

Note: Apps that don't store rich media items shouldn't offer a Delete option. Instead, they should show items captured in the current session.

Other design considerations

Although lenses are powerful apps, their functionality is limited. You can't delete photos from the user's camera roll, enumerate other lenses the user has installed, or launch built-in editing experiences in a lens app. These limitations are in place to protect the app user's personal information and data. Lenses shouldn't try to mimic every feature of the phone's built-in photo viewer.

Providing icons for the lens picker

The lens picker requires icons that are a different resolution than the icon that represents the app itself. Your app must provide three icons in the Assets folder, one for each of the possible phone resolutions. The following table describes the names and resolutions for each of these icons.

To learn more about creating icons for each resolution, download the Icon templates for Windows Phone 8.

Phone resolution	Icon size (pixels)	Folder	File name
WVGA	173 x 173	Assets	Lens.Screen-WVGA.png
HD720p	259 x 259	Assets	Lens.Screen-720p.png
WXGA	277 x 277	Assets	Lens.Screen-WXGA.png

For more info about phone resolutions, see Multi-resolution apps for Windows Phone 8.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Summary of recommendations

All lenses jump into a viewfinder-driven experience and save photos to the camera roll. The following points summarize additional points to keep in mind.

Launch experience:

- Lens splash screen displays in landscape orientation.
- Lens icons support WVGA, HD720p, and WXGA resolutions.

Capture experience:

- Be consistent with the default camera user experience.
    - Gesture support: swipe left to preview.
    - Support portrait and landscape orientations.
    - Button behavior:
        - Half press.
        - Hardware capture.
        - Touch to capture (with focus).
    - Flash icons and states for On, Off, Auto, and Front-Facing Camera where relevant.
    - Focus brackets.
- One picture per capture saved to the camera roll.
- If more than one JPG image is created through capture, the additional backing data should be saved in the app's local folder.

Capture and confirm apps:

- Use a consistent set of icons for Save, Save copy, and Delete.
- Delete and Save must both return to the viewfinder.

Rich media lenses:

- If your app stores additional data for editing or later viewing of a photo, you should consider implementing a rich media experience.
- The open link directs users to an experience tailored to viewing or manipulating a selected item.
- Check to see if an image exists in the camera roll before opening it (the user could have deleted the image), and gracefully handle the situation if it is missing.
- Apps that provide a rich media experience should be able to handle the case when a user links from an item in the camera roll where the data has been deleted in the app.
- Rich media lens apps should enumerate any content captured by the app based on backing data in their local folder, not the camera roll.
- Rich media lens apps should give users the ability to delete backing data from the device.
- If you launch into an editing experience, the save functionality should be called Save copy. Keep the user in the app to show the confirmed changes.
- Navigation from the open link:

© 2014 Microsoft. All rights reserved. | May 13, 2014

    - If you launch into a viewing or editing experience, pressing Back should bring you back to the camera roll.

If your app doesn't provide a rich media experience:

- Apps that don't store rich media shouldn't offer a Delete option. Instead, show items captured in that current session.
- If your app doesn't use rich media, don't declare a rich media extension in the app's WMAppManifest.xml file.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for the camera UI



The camera dialog is a touch-optimized, full-screen experience for capturing photos and videos from an embedded or attached camera. The full-screen dialog handles the work of presenting the camera UI. The dialog enables you to capture a photo or video with one method call to the **Windows.Media.Capture.CameraCaptureUI.captureFileAsync** API. As part of the capture experience, users can crop their captured photos and trim their captured videos before they return to the calling application. In addition, users can also adjust some of the camera settings such as brightness, contrast, and exposure before capturing a photo or a video. The camera dialog is intended for live photo and video capture.

Dos and don’ts

- Use the camera UI if your application requires live photo or video. For instance, an application that requires a profile picture can ensure that the profile picture is recent by presenting the camera dialog to start a live capture.
- Don’t use the camera UI if you plan to have real-time feedback or control over the image that is being captured. For example, a barcode reader app might provide real-time feedback to the user as they scan a barcode using the camera, to let the user know whether the barcode is readable. In this case the camera dialog would not be the right option, because it does not provide any direct control over the captured video stream. You should instead use the MediaCapture API.
- Don’t use the camera dialog if you need to add custom controls to the user interface. You should instead use the MediaCapture API, if you need to add UI customizations beyond what the camera dialog provides.
- Don’t turn on cropping or trimming in the camera dialog if your application provides them. If your application is a video or photo editing application, or provides some photo or video editing capabilities, you should use the camera dialog with trimming and cropping turned off. Then, the trimming and cropping function in your application will not be redundant with what the camera dialog provides.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for devices that access personal data



Microphones, cameras, location providers, and text messaging services can access the user's personal data or cost the user money, so they are considered *sensitive devices*. Windows Store apps have features to ensure the user has control over which apps may access these sensitive devices. This topic describes guidelines for designing Windows Store apps to account for how a user may enable and disable device access.

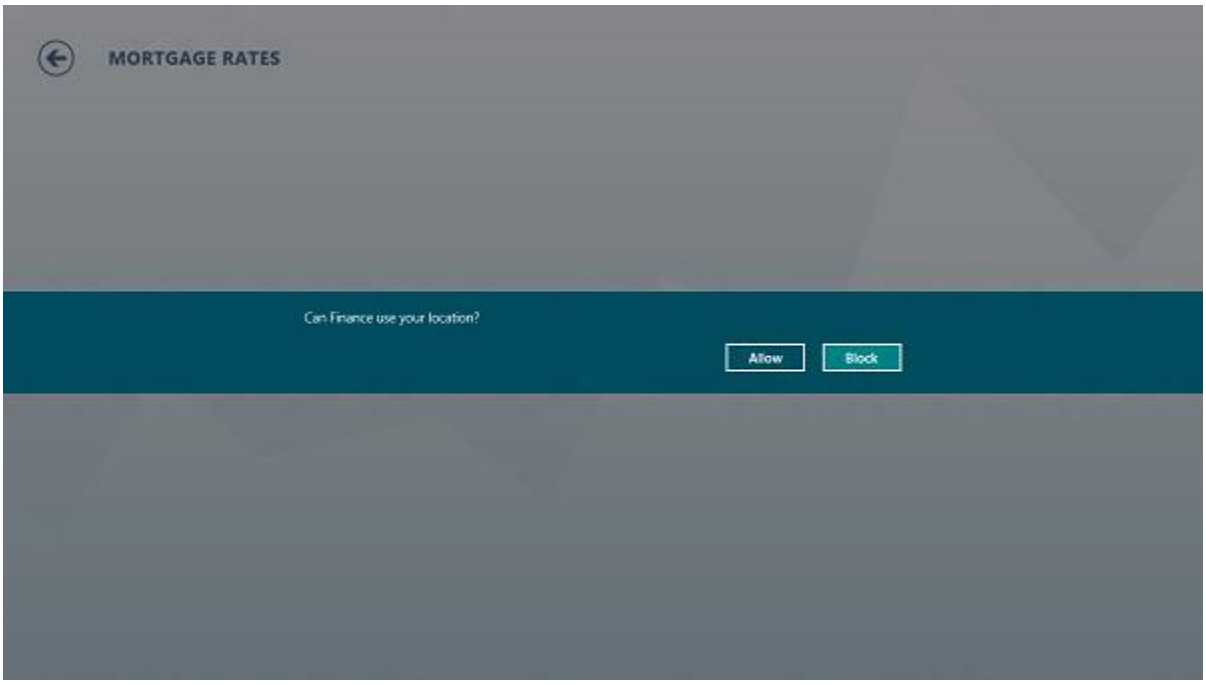
Examples

Permissions for Windows Store apps to use sensitive devices are controlled on a per-app, per-user level. Users control permissions using the consent prompt, or by using the Settings charm. The following images show the UI associated with consent prompts and the **Permissions** section of the Settings pane.

The consent prompt

Here's the consent prompt in a Windows Store app. The prompt appears the first time an app tries to access a sensitive device. It gives the user options to block or allow the app's access to the device capability. Windows remembers the response, so the user isn't prompted again for the same app.

© 2014 Microsoft. All rights reserved. | May 13, 2014



The Settings charm

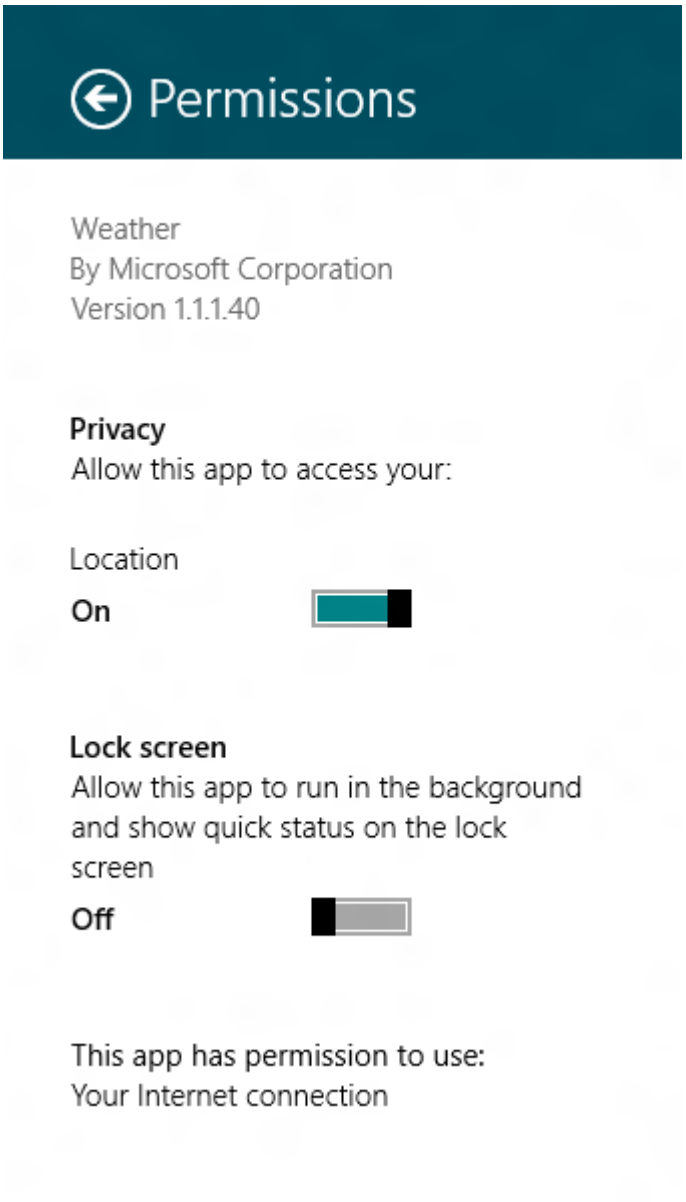
Each app's access to sensitive devices can also be set using the Settings charm. The user taps on the Settings charm to open the Settings pane (**SettingsPane**). The Settings charm is pictured here on the right side of the app:

© 2014 Microsoft. All rights reserved. | May 13, 2014



Windows provide the Permissions flyout by default for all Windows Store apps. When users click Permissions from the Setting pane, the Permissions flyout appears and they can then enable or disable the app's access to sensitive device capabilities. Here is the Permissions flyout from a weather app:

© 2014 Microsoft. All rights reserved. | May 13, 2014



Dos and don'ts

- The first call to start using a sensitive device must be made on the main UI thread so the consent prompt can be shown to the user. The user won't be able to grant device access to the app unless the consent prompt is shown. Ensure that:
    - You app doesn't use a background task for the first use of the device.
    - Your app using JavaScript and HTML doesn't first use the object that accesses the device in the activation handler for the app.
    - Your app using C#/VB/C++ and XAML first uses the object that accesses the device in MainPage.xaml.cs, not App.xaml.cs.

© 2014 Microsoft. All rights reserved. | May 13, 2014

- If use of the sensitive device is not essential for all tasks in your app, don't access it until the user wants to complete a task that requires it. For example, a social networking app with buttons for "Check in with my location" and "Take a profile picture" shouldn't access location or the camera until the user clicks the corresponding button.
- If your app requires sensitive device access for its primary function, access the device when the app starts. For example, an app that captures live videos requires the camera for its main purpose. In this case, it's appropriate to immediately request device access.
- Don't programmatically launch the **Permissions** page in the Settings pane.

If access to a device is turned off

There are three ways access to a device can be disabled: if a user denies consent when prompted, if a user blocks access in the Permissions flyout in the Setting pane, or if the device is not present in the system. Follow these recommendations if your app can't access a sensitive device:

- Notify users of an unavailable device capability when they attempt to use it. The user should be aware of the loss of functionality.
- Make the disabled device notification clearly visible to the user.
- Use a flyout or inline message to notify the user if the capability is not essential to the app's primary function.
- Handle the error from the API that will occur when your app attempts to access a disabled device capability. The Reference section of this topic provides more information on the method calls that may return errors that indicate that the app doesn't have access to a device.
- Display a message to users that informs them that the device capability is disabled, and tells them how they can re-enable it from the Permissions flyout in the Settings pane. For examples of how to notify a user of a disabled device, see Additional usage guidance.
- Provide UI for the user to re-initiate access to the device if the device is re-enabled. Reinstantiate or reinitialize the object that accesses the device by using this UI. For example, a mapping app may provide a button for refreshing the current location. The button must instantiate a new **Geolocator** object.
- Don't use notifications to inform the user of an unavailable device capability.
- Don't show an error message for a device capability that has not yet been requested by the user. For example, if a social networking site has an option to include location when the user posts messages, but the user has not chosen to share location, don't show an error message when posting messages.

Additional usage guidance

Sample error messages

Reason the device is disabled	Sample error message format
The user blocked access using the consent prompt or the Settings charm.	"Your *<device capability>* is currently turned off. To change your *<device capability>* setting, open the settings charm and tap

© 2014 Microsoft. All rights reserved. | May 13, 2014

	permissions. Then <*enable action*> to start using <*device capability*> again. ” • Replace <*device capability*> with webcam, microphone, location, or text messaging. • Replace <*enable action*> with the action the user needs to take in the UI to reinitialize access the capability, like clicking a button.
The device capability isn't present on the system.	“You do not have the required <*device capability*> present on your system.”

The UI you use to present the message about a disabled device capability depends on whether the device capability is essential to the app. The following examples show how to display device capability messages.

Displaying a flyout or inline text if the device is non-essential

If the sensitive device is not essential to your app, display the message in a flyout at the point of invocation, or in unobtrusive inline text.

For example, if a mapping app has a button for showing the current location, and the user clicks the button when the required device capability is disabled, the app should show the error message in a flyout near the button, or in inline text.

The following screen shot shows an app that displays an error message in flyout. The message reads: "Your location cannot be found. Change your permissions in Settings to allow Maps to access your location. Then restart the app."

© 2014 Microsoft. All rights reserved. | May 13, 2014

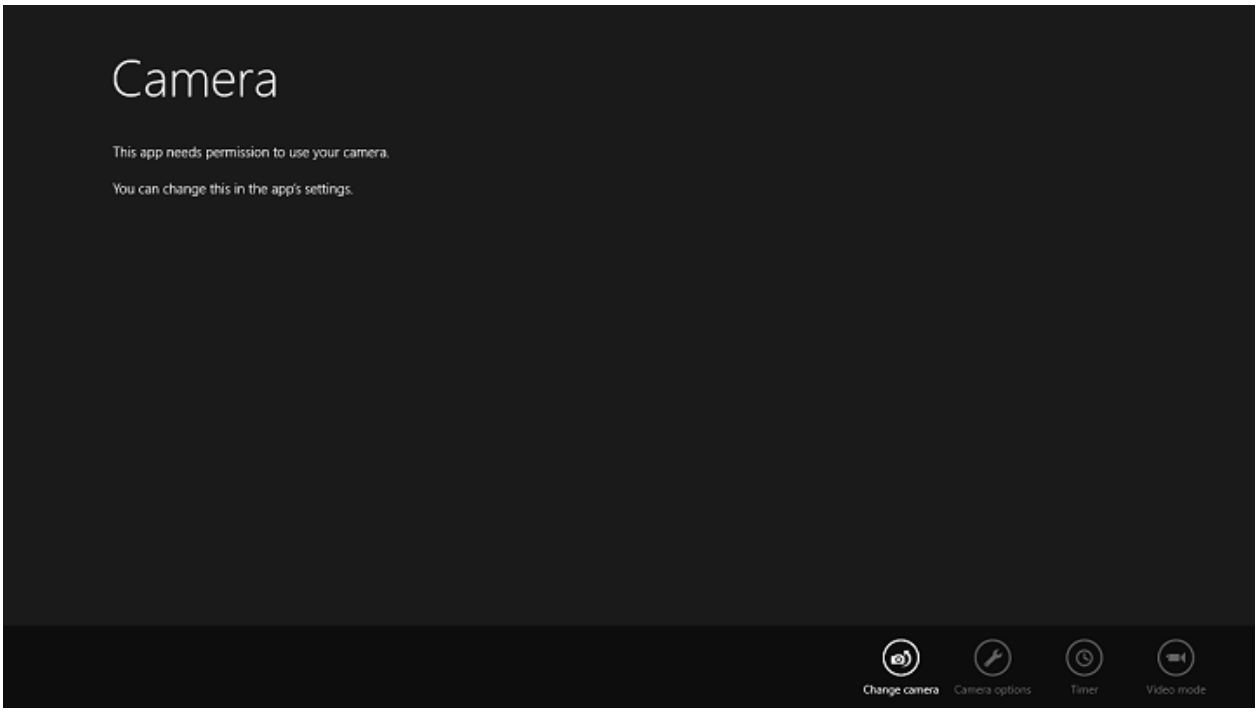


Displaying a dialog if the device capability is essential

If an app requires device access for its main function, display an error message using a **MessageDialog**. For a full sample, see Message dialog sample.

In the screen shot that follows, the Camera app requires access to the webcam and microphone capabilities for its main function, so it displays a message instructing the user to enable the camera capability.

© 2014 Microsoft. All rights reserved. | May 13, 2014



Reference

This table lists API info for disabling and reenabling sensitive device capabilities.

Capability	Info on enable and disable
Location	Either the **GetGeopositionAsync** method or an event handler for the **PositionChanged** event will trigger a consent prompt.
Webcam or Microphone	Apps that use **Windows.Media.Capture.CameraCaptureUI.CaptureFileAsync** to capture photos or videos from a camera should note the following: • Calling **Windows.Media.Capture.CameraCaptureUI.CaptureFileAsync** triggers a consent prompt the first time the app is run. **Windows.Media.Capture.CameraCaptureUI.CaptureFileAsync** does not return an error if the webcam capability is turned off. Instead, the Camera Capture UI displays a message that indicates that the webcam capability is turned off. Apps that use **Windows.Media.Capture.MediaCapture** to preview or capture audio, video, or photos should address the following issues: • The error handlers for the asynchronous methods of **MediaCapture** receive an E_ACCESSDENIED error if the user did not grant permission, and HRESULT_FROM_WIN32(ERROR_FILE_HANDLE_REVOKED), if permission is revoked. • Call **InitializeAsync** again to access the camera, if the user re-enables access to the webcam after revoking it. For example, if the error handler

© 2014 Microsoft. All rights reserved.

	for a HRESULT_FROM_WIN32(ERROR_FILE_HANDLE_REVOKED) error instructs the user to re-enable the webcam using the settings charm and then to tap a button to restart a video preview. The code behind the button should call **InitializeAsync** before making any other calls. Apps using the **IAudioClient2** interface should note that a call to **ActivateAudioInterfaceAsync** will trigger a consent prompt.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for file picker contracts



Follow these guidelines to customize the file picker for apps that participate in the File Open Picker contract, the File Save Picker contract, or the Cached File Updater contract in order to provide other apps with access to the app's content, a save location, or file updates (respectively).

Dos and don'ts

- Provide files. Integrating with the File Open Picker contract lets your app provide users and other apps access to your app's content through the file picker.
- Provide a save location. Integrating with the File Save Picker contract lets your app provide users and other apps with a save location through the file picker.

  If your app provides a save location, you should also provide access to your app's content by integrating with the File Open Picker contract.

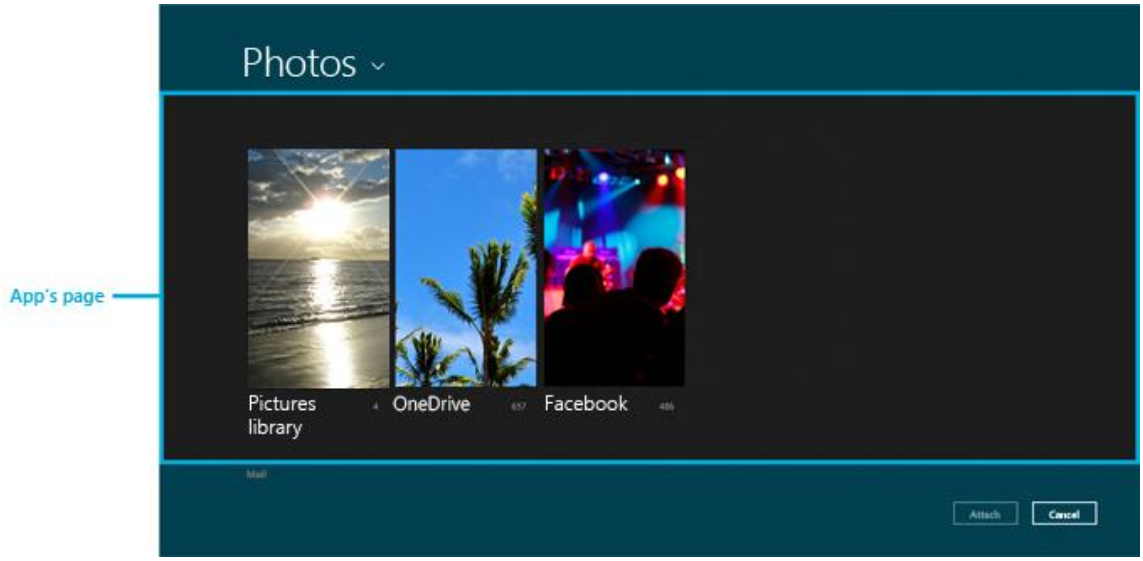
- Provide real-time file updates. Integrating with the Cached File Updater contract lets your app perform updates on files in your app's repository and provide updates to local versions of the files in your repository. From the users' perspective, this lets them to operate on a remote file that your app maintains in its repository as though that file were local. For example, the user could use a text editor app to edit a file and Microsoft OneDrive could update the version of that file in its repository).

  If your app updates files, it should also provide a save location and access to files by integrating with the File Save Picker contract and the File Open Picker contract, respectively.

Additional usage guidance

If your app provides files, a save location, or file updates through file pickers, you need to design a page for your app that displays files (or other UI) to the user. This page will be displayed in the center area of the file picker.

© 2014 Microsoft. All rights reserved. | May 13, 2014



This screen shot has been modified to emphasize and label the center area of a file picker window to show where your app's page (the file picker page) will be loaded.

- **Design your file picker page to adapt to windows of all sizes.**

  Windows 8.1 displays file pickers in windows as narrow as 320 pixels. To make the most of a narrow display, consider reducing the left margin of your file picker page to 20 pixels and using vertical scrolling when loading in a window fewer than 500 pixels wide. Note that the file picker item template in Microsoft Visual Studio supports vertical scrolling in windows fewer than 500 pixels wide and horizontal scrolling in larger windows.

- **Design the page to display in the file picker (your file picker page) based on an existing page that your app uses to display files.**

  If your app is providing files for the user to pick through a file picker, your app should have an existing page that lets users view files. We recommend that you design your file picker page so that it is consistent with this existing file-view page. Making these two pages consistent with each other helps users feel comfortable and familiar with how your app displays files in the file picker.

  To further ensure that users feel comfortable with your app's file picker page, use the same (or similar) navigation UI and error reporting for your file picker page as you use for your existing file-view app page. Especially in the case of navigation, users expect similar commands and locations using to be available in both the file picker page and the existing file-view page.

- **Design your file picker page around your user's current task.**

© 2014 Microsoft. All rights reserved. | May 13, 2014

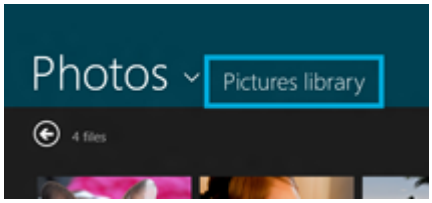
Keep the UI for your file picker page focused on the user's current task, like helping users pick, save, or update files, by stripping out UI that is not directly related. This helps make sure that using the file picker is a quick, in-and-out experience that gets users back into the app they were using (the calling app or caller).

For example, if a file picker is being used to access files that are provided by your app, remove UI that supports complex and/or detailed navigation, search, or information that cannot be picked.

If you want to let the user perform other tasks like consumption, modification, and file management, add controls or other UI for those tasks to your main app.

- **Set the title of the file picker to the name of the user's current location.**

  This gives users a predictable way to orient themselves as they use your app from the file picker. The title, which is highlighted in this screen shot, appears in top bar of the file picker letterbox.

  In this screen shot, the title is Pictures Library, which lets the user know where they are in their system. You should update this title whenever the user navigates to a different location.

- **All file locations that are accessible to your app should be accessible from your file picker page.**

  If your app can normally access files in a particular location, your file picker page should also give access to files in that location. Access to locations should also be consistent across all file picker pages, if your app has more than one page. This ensures that users have predictable access to files and locations.

- **Use the UI templates and controls available in Microsoft Visual Studio.**

  Visual Studio has built-in templates that you can use to create the file picker view for your Windows Store apps.

- **Keep sign-in and setup interactions simple when users launch your app through the file picker.**

  If your sign-in or setup tasks are simple (one-step) you should let users complete those tasks through the file picker so that users don't have to change context. You should avoid asking users to complete multi-step interactions through the Share charm. Instead, tell users to open your app directly to complete more complex interactions. When you use your main app to complete complex interactions you ensure that you have the space to organize those tasks clearly and effectively.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Additional UX guidelines: File Open Picker contract

- **Display files in your file picker page in a unique and relevant way.**

  Organize and display files in a way (or ways) that is unique to your app and ensures that your page is both convenient and relevant to users. This should still be consistent with what users see in the view that your app uses to display files within the app.

- **Display files on your file picker page that are not accessible by using Windows or other apps.**

  Differentiate your app from Windows and other apps by providing access to files in locations that are not accessible from other apps or from Windows, like your app's storage folders or remote servers.

- **Design the UI of your file picker page to respond to the selection mode of the calling app.**

  When an app calls a file picker to access files, the calling app specifies whether the user can pick a single item or multiple items. We recommend that you design your app page to indicate selected files appropriately and differently for each selection mode. For example, if the user is trying to select a profile picture (a single item selection) from files provided by your app, they might tap or click more than one photo while they try to decide which to pick. In this situation, your app UI would only allow one item to be selected at a time. Otherwise, if the user is trying to select multiple files to share with their friends (multiple item selection), your app UI would allow multiple items to be selected simultaneously.

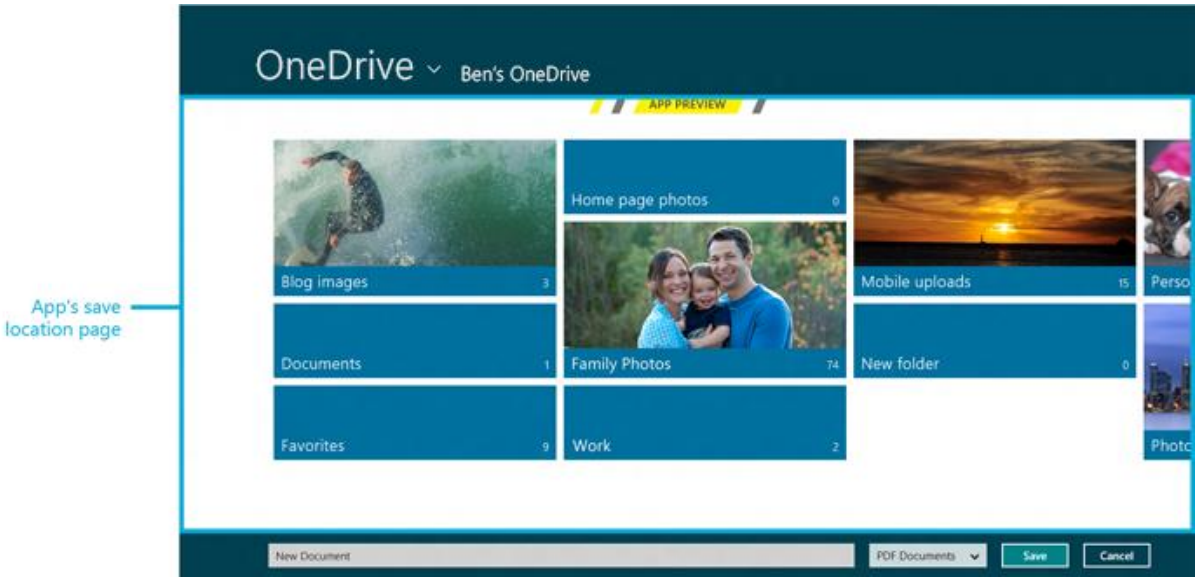
- **For webcam, photography, and camera apps, design the UI of your file picker page around taking pictures.**

  Make sure users can get back into the app they were using (the calling app or caller) by simplifying your app's UI for your file picker page. Limit the controls you provide on your file picker page to controls that let the user take a picture, and let the user apply a few pre-processing effects (like toggling the flash and zooming).

  All available controls must be visible on your file picker page because your app bar is not accessible to the user from the file picker. We recommend that you organize these controls on your file picker page similarly to the way they are organized in your app bar, and position them on your file picker page as close as possible (at the top/bottom of the page) to where they appear in your app bar.

Additional UX guidelines: File Save Picker contract

© 2014 Microsoft. All rights reserved. | May 13, 2014



This modified screen shot emphasizes the center area of a file picker window where the page that displays your app's save location will be loaded.

- **Provide save locations that are not accessible to users through Windows or other apps**.

  Let users save files to locations that are not easily accessible through Windows or other apps, like your app's storage folders or remote storage locations.

- **Change the files displayed on your file picker page based on the file type selected**.

  If the user changes file type in the file picker's file-type drop-down list, you should update your view to display only files that match the selected file type. Filtering displayed files by type provides the user with an easy, consistent method of identifying the types of files they're interested in.

- **Allow the user to replace a file easily by selecting the file in your app's file picker page**

  If the user selects a file in your file picker page, you should automatically replace the file name in the file picker file name box so users can easily replace existing files.

Additional UX guidelines: Cached File Updater contract

- **Provide a repository that can track and update files for users.**

  If users use your app as a primary storage location where they regularly save and access files, you may want your app to track some files to provide real-time updates for users.

© 2014 Microsoft. All rights reserved. | May 13, 2014

- **Design your app and file picker page to present a robust repository.**

  If users use your app as a primary storage location for their files, design your app and your associated file picker view to protect against data loss, which could be caused by frequent file updates or conflicting file versions.

- **Let users resolve issues encountered during updates.**

  To help ensure a successful update, your app should to notify users in real time (using **UIRequested**) when a file is being updated or saved and user intervention is needed to resolve an issue effectively. It is especially important that your app help users resolve issues with credentials, file version conflicts, and disk capacity. The UI you create should be lightweight and focused specifically on resolving the issue. If more than one step is required (like login), all the steps should be handled in your app's file picker page. Once complete, your app can enable the file picker commit UI. In addition, your app should update the file picker title to gives users context about where they are.

  If the problem cannot be solved in real time by the user, or if you simply need let the user know what happened (perhaps an error occurred that the user can't resolve), we recommend that you notify the user of the problem the next time your app is launched instead of immediately when the problem occurs via **UIRequested**.

- **Provide additional information about update and save operations from your normal app pages.**

  Your main app UI should let users manage settings for in-progress and future operations, get information about in-progress and previous operations, and get information about any errors that have occurred.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for file pickers



The file picker allows an app to access files and folders and to save a file.

Dos and don'ts

- Add a control to your app that calls the file picker to let the user pick files for your app to operate on.
- Add a control to your app's UI that calls the file picker so that the user can specify the name, file type, and/or save location (like another app) of the file to save.
- Set the file types to ensure that users can pick or save only file types that your app can handle.
- When accessing files or folders, set the view mode based on the kinds of items that the user is picking from.
- Set the commit button text to match the user's current task.
- Set the suggested start location to the most relevant location possible based on the user's current task.
- When accessing files, let the user pick a single file or multiple files based on the current task.
- When saving files, set a default file name for the file to save.
- Don't use the file picker to explore, consume, or manage file content.
- Don't use the file picker to save a file if a unique, user-specified file name or location is not needed.

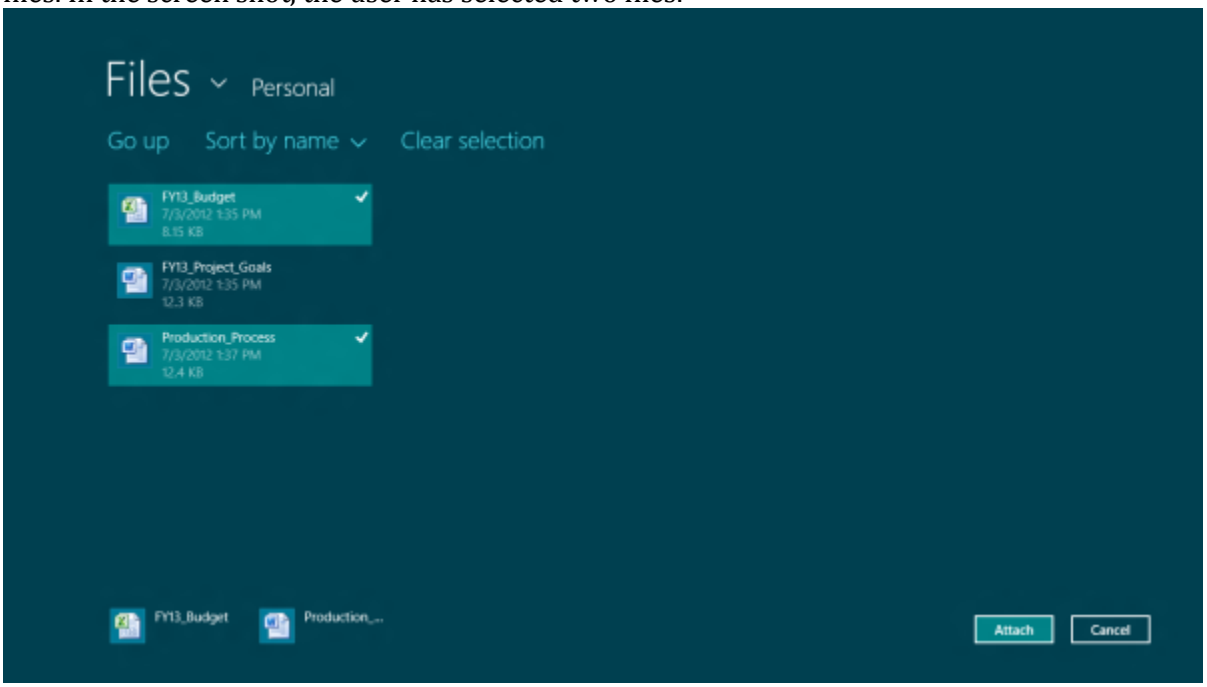
Additional usage guidance

- **Access files and folders.**

  Add a control to your app that calls the file picker to let the user pick files for your app to operate on. The user can then pick files through the file picker's UI as shown in the screen shot.

© 2014 Microsoft. All rights reserved. | May 13, 2014

For example, this screen shot shows a file picker that was called to let the user choose some files. In the screen shot, the user has selected two files.

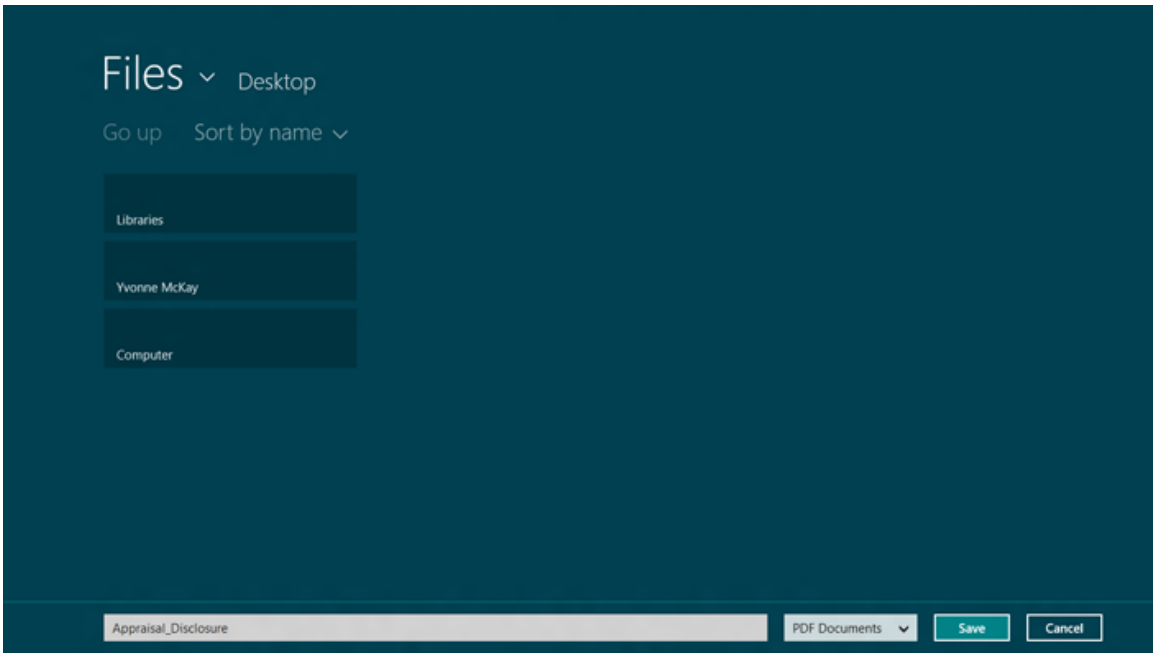


The user can pick files from any location (including from other apps) that is listed in the drop-down list at the upper left in the file picker.

- **Add "save as" to your app.**

  Add a control to your app's UI that calls the file picker so that the user can specify the name, file type, and/or save location (like another app) of the file to save. The user can then navigate and save their file through the file picker's UI, as shown in the screen shot.

© 2014 Microsoft. All rights reserved. | May 13, 2014



We recommend that you let users explore, consume, and/or manage file content by creating dedicated pages and UI in your app. This helps users focus on their current task and helps ensure that when users pick files, their experience is uncluttered by unnecessary functionality.

For example, a photo gallery app should provide a customized, dedicated page and UI that lets users organize and view picture files within the app. The app can then customize this UI to best suit the user's needs. When the user wants to add files to the gallery, it would call the file picker which provides an experience that is specialized for picking.

If the user doesn't have to specify a file name, file type, or location to save to, we recommend that your app save the file automatically in the background (without launching a file picker). This helps eliminate unnecessary user interaction, making the process of saving a file faster and less intrusive.

User experience guidelines: accessing and saving files and folders

- Whether picking or saving files and folders, customize the file picker to display only the file types that your app supports and that are relevant to the user's current task. For example, if the user is picking or saving a video, set the file types so that the user can select or save only a video file that uses a format that your app can handle.

  This also applies to folder picking where the user is using files displayed in the file picker to help them determine which folder to select. By filtering the view to the proper file type, you help the user identify the correct folder faster.

- If the user is picking pictures or videos, set the view mode to **Thumbnail**. If the user is picking any other kind of files or folders, set the view mode to **List**.

© 2014 Microsoft. All rights reserved. | May 13, 2014

In some cases, the user may want to pick a picture/video or any other kind of file (for example, if the user is picking a file to attach to an email or to send via IM). In this case, you should support both view modes by adding two UI controls to your app. One control should call the file picker using the **Thumbnail** view mode so the user can pick pictures and videos and the other should call the file picker using the **List** view mode so that the user can pick other kinds of files. For example, a mail app would have two buttons: **Attach Picture or Video** and **Attach Document**.

- Whether picking or saving files and folders, customize the file picker by setting the commit button text appropriately for the user's current task. For example, if the user wants to pick a set of files to upload to your app, set the commit button text to "Upload".
- Whether picking or saving files and folders, customize the file picker to suggest the most relevant start location possible based on the user's current task and the list of possible start locations provided by the **PickerLocationId** enumeration. For example, if the user is picking pictures, you may want to set the suggested start location to the user’s Pictures.
- If the user is picking a profile picture, call the file picker for picking a single file. If the user is picking photos to send to a friend, call the file picker for picking multiple files.
- If the user accepts the default file name that you provide, they won't have to take the time to enter a different name and they can complete the "save as" task faster. You can use the **FileSavePicker.SuggestedFileName** property to set the default file name.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for geofencing apps



Follow these best practices for **geofencing** in your Windows Runtime app.

Dos and don'ts

- If your app will need internet access when a **Geofence** event occurs, check for internet access before creating the geofence.
    - If the app doesn't currently have internet access, you can prompt the user to connect to the internet before you set up the geofence.
    - If internet access isn't possible, avoid consuming the power required for the geofencing location checks.
- Ensure the relevance of geofencing notifications by checking the time stamp and current location when a geofence event indicates changes to an **Entered** or **Exited** state.
- Create exceptions to manage cases when a device can't access location info, and notify the user if necessary. Location info may be unavailable because permissions are turned off, the device doesn't contain a GPS radio, the GPS signal is blocked, or the Wi-Fi signal isn't strong enough.
- In general, it isn't necessary to listen for geofence events in the foreground and background at the same time. However, if your app needs to listen for geofence events in both the foreground and background:
    - Call the **ReadReports** method to find out if an event has occurred.
    - Unregister your foreground event listener when your app isn't visible to the user and re-register when it becomes visible again.
- Don't use more than 1000 geofences per app. The system actually supports thousands of geofences per app, you can maintain good app performance to help reduce the app's memory usage by using no more than 1000.
- Don't create a geofence with a radius smaller than 50 meters. If your app needs to use a geofence with a small radius, advise users to use your app on a device with a GPS radio to ensure the best performance.

Additional usage guidance

Checking the time stamp and current location

When an event indicates a change to an **Entered** or **Exited** state, check both the time stamp of the event and your current location. Various factors, such as the system not having enough resources to launch a background task, the user not noticing the notification, or the device being in standby (on Windows), may affect when the event is actually processed by the user. For example, the following sequence may occur:

© 2014 Microsoft. All rights reserved. | May 13, 2014

- Your app creates a geofence and monitors the geofence for enter and exit events.
- The user moves the device inside of the geofence, causing an enter event to be triggered.
- Your app sends a notification to the user that they are now inside the geofence.
- The user was busy and does not notice the notification until 10 minutes later.
- During that 10 minute delay, the user has moved back outside of the geofence.

From the timestamp, you can tell that the action occurred in the past. From the current location, you can see that the user is now back outside of the geofence. Depending on the functionality of your app, you may want to filter out this event.

Background and foreground listeners

In general, your app doesn't need to listen for **Geofence** events both in the foreground and in a background task at the same time. The cleanest method for handling a case where you might need both is to let the background task handle the notifications. If you do set up both foreground and background geofence listeners, there is no guarantee which will be triggered first and so you must always call the **ReadReports** method to find out if an event has occurred.

If you have set up both foreground and background geofence listeners, you should unregister your foreground event listener whenever your app is not visible to the user and re-register your app when it becomes visible again. Here's some example code that registers for the visibility event.

C#

```
    Windows.UI.Core.CoreWindow coreWindow;

    coreWindow = CoreWindow.GetForCurrentThread(); // This needs to be set before
InitializeComponent sets up event registration for app visibility
    coreWindow.VisibilityChanged += OnVisibilityChanged;
```

JavaScript

```
 document.addEventListener("visibilitychange", onVisibilityChanged, false);
```

When the visibility changes, you can then enable or disable the foreground event handlers as shown here.

C#

```
private void OnVisibilityChanged(CoreWindow sender, VisibilityChangedEventArgs args)
{
    // NOTE: After the app is no longer visible on the screen and before the app is
suspended
    // you might want your app to use toast notification for any geofence activity.
    // By registering for VisibiltyChanged the app is notified when the app is no
longer visible in the foreground.

    if (args.Visible)
    {
        // register for foreground events
        GeofenceMonitor.Current.GeofenceStateChanged += OnGeofenceStateChanged;
```

© 2014 Microsoft. All rights reserved. | May 13, 2014

```
        GeofenceMonitor.Current.StatusChanged += OnGeofenceStatusChanged;
    }
    else
    {
        // unregister foreground events (let background capture events)
        GeofenceMonitor.Current.GeofenceStateChanged -= OnGeofenceStateChanged;
        GeofenceMonitor.Current.StatusChanged -= OnGeofenceStatusChanged;
    }
}
```

JavaScript

```
function onVisibilityChanged() {
    // NOTE: After the app is no longer visible on the screen and before the app is
suspended
    // you might want your app to use toast notification for any geofence activity.
    // By registering for VisibiltyChanged the app is notified when the app is no
longer visible in the foreground.

    if (document.msVisibilityState === "visible") {
        // register for foreground events

Windows.Devices.Geolocation.Geofencing.GeofenceMonitor.current.addEventListener("geo
fencestatechanged", onGeofenceStateChanged);

Windows.Devices.Geolocation.Geofencing.GeofenceMonitor.current.addEventListener("sta
tuschanged", onGeofenceStatusChanged);
    } else {
        // unregister foreground events (let background capture events)

Windows.Devices.Geolocation.Geofencing.GeofenceMonitor.current.removeEventListener("
geofencestatechanged", onGeofenceStateChanged);

Windows.Devices.Geolocation.Geofencing.GeofenceMonitor.current.removeEventListener("
statuschanged", onGeofenceStatusChanged);
    }
}
```

Sizing your geofences

While GPS can provide the most accurate location info, geofencing can also use Wi-Fi or other location sensors to determine the user's current position. But using these other methods can affect the size of the geofences you can create. If the accuracy level is low, creating small geofences won't be useful. In general, it is recommended that you do not create a geofence with a radius smaller than 50 meters. Also, geofence background tasks only run periodically on Windows; if you use a small geofence, there's a possibility that you could miss an **Enter** or **Exit** event entirely.

If your app needs to use a geofence with a small radius, advise users to use your app on a device with a GPS radio to ensure the best performance.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for location-aware apps



This topic describes performance guidelines for apps that require access to a user's location.

See the Geolocation sample to see how a Windows Store app can detect a user's location.

Dos and don'ts

- Start using the location object only when the app requires location data.

  The app's first access to the **Geolocator** object triggers a consent prompt. This occurs the first time an app calls **getGeopositionAsync** or registers an event handler for the **positionChanged** event. If an app's primary purpose does not require access to location data, the user can find it confusing to see a prompt for permission to use the device as soon as the app starts.

- If location isn't essential to your app, don't access it until the user tries to complete a task that requires it. For example, if a social networking app has a button for "Check in with my location," the app shouldn't access location until the user clicks the button. It's okay to immediately access location if it is required for your app's main function.
- On Windows only, the first use of the **Geolocator** object must be made on the main UI thread, to show a consent prompt to the user. The first use of the **Geolocator** can be either the first call to **getGeopositionAsync** or the first registration of a handler for the **positionChanged** event. The consent prompt is described further in Guidelines for using sensitive devices. This means that in an app using JavaScript, the first use of the **Geolocator** object should not occur in an activation handler.
- Tell the user how location data will be used.
- Provide UI to enable users to manually refresh their location.
- Display a progress bar or ring while waiting to get location data.
- Show appropriate error messages or dialogs when location services are disabled or not available.

  When access to location data is revoked by the user, or when data is not available to the app for other reasons, provide the appropriate error message:

  - We suggest that you use this message on Windows: "Your location services are currently turned off. Use the Settings charm to turn them back on." On Windows Phone, you can use the following message: "Location is disabled on your device. To enable location, go to Settings and select location."
  - Don't let error messages interrupt the app's flow. If location data is not essential to your app, display the message as inline text. Social networking or gaming apps fall into this category.

© 2014 Microsoft. All rights reserved. | May 13, 2014

  - If location data is essential to your app's functionality, display the message as a flyout or a dialog. Mapping and navigation apps fall into this category.
  - Don't try to display the Settings charm programmatically.

- Clear cached location data and release the **Geolocator** when the user disables access to location info.

  Release the **Geolocator** object if the user turns off access to location info through Settings. The app will then receive **ACCESS_DENIED** results for any location API calls. If your app saves or caches location data, clear any cached data when the user revokes access to location info. Provide an alternate way to manually enter location info when location data is not available via location services.

- Provide UI for reenabling location services. For example, provide a refresh button that reinstantiates the **Geolocator** object and tries to get location info again.

  Have your app provide UI for reenabling location services—

  - If the user reenables location access after disabling it, there is no notification to the app. The **status** property does not change and there is no **statusChanged** event. Your app should create a new **Geolocator** object and call **getGeopositionAsync** to try to get updated location data, or subscribe again to **positionChanged** events. If the status then indicates that location has been reenabled, clear any UI by which your app previously notified the user that location services were disabled, and respond appropriately to the new status.
  - Your app should also try again to get location data upon activation, or when the user explicitly tries to use functionality that requires location info, or at any other scenario-appropriate time.

Performance

- Use one-time location requests if your app doesn't need to receive location updates. For example, an app that adds a location tag to a photo doesn't need to receive location update events. Instead, it should request location using **getGeopositionAsync**.

  When you make a one-time location request, you should set the following values.

  - Specify the accuracy requested by your app by setting the **DesiredAccuracy** or the **DesiredAccuracyInMeters**. See below for recommendations on using these parameters
  - Set the max age parameter of **GetGeopositionAsync** to specify how long ago a location can have been obtained to be useful for your app. If your app can use a position that is a few seconds or minutes old, your app can receive a position almost immediately and contribute to saving device power.
  - Set the timeout parameter of **GetGeopositionAsync**. This is how long your app can wait for a position or an error to be returned. You will need to figure out the trade-offs between responsiveness to the user and accuracy your app needs.

© 2014 Microsoft. All rights reserved. | May 13, 2014

- Use continuous location session when frequent position updates are required. Use **positionChanged** and **statusChanged** events for detecting movement past a specific threshold or for continuous location updates as they occur.

  When requesting location updates, you may want to specify the accuracy requested by your app by setting the **DesiredAccuracy** or the **DesiredAccuracyInMeters**. You should also set the frequency at which the location updates are needed, by using the **MovementThreshold** or the **ReportInterval**.

  - Specify the movement threshold. Some apps need location updates only when the user has moved a large distance. For example, an app that provides local news or weather updates may not need location updates unless the user's location has changed to a different city. In this case, you adjust the minimum required movement for a location update event by setting the **MovementThreshold** property. This has the effect of filtering out **PositionChanged** events. These events are raised only when the change in position exceeds the movement threshold.

    On Windows, when you set the **MovementThreshold** property, it doesn't change the frequency at which the source of the location data (such as the Windows Location Provider or an attached GPS device) calculates location. On Windows Phone the **MovementThreshold** property, together with other factors such as estimated speed of movement, is used to tune the frequency at which the location is calculated in the system in order to conserve device power.

  - Use **reportInterval** that aligns with your app experience and that minimizes the use of system resources. For example, a weather app may require a data update only every 15 minutes. Most apps, other than real-time navigation apps, don't require a highly accurate, constant stream of location updates. If your app doesn't require the most accurate stream of data possible, or requires updates infrequently, set the **ReportInterval** property to indicate the minimum frequency of location updates that your app needs. The location source can then conserve power by calculating location only when needed.

    Apps that do require real-time data should set **ReportInterval** to 0, to indicate that no minimum interval is specified. On Windows, when the report interval is 0, the app receives events at the frequency that the most accurate location source sends them. On Windows Phone, the app will receive updates at a rate dependent on the accuracy requested by the app.

    Devices that provide location data may track the report interval requested by different apps, and provide data reports at the smallest requested interval. The app with the greatest need for accuracy thus receives the data it needs. Therefore, it's possible that the location provider will generate updates at a higher frequency than your app requested, if another app has requested more frequent updates.

    Note: It isn't guaranteed that the location source will honor the request for the given report interval. Not all location provider devices track the report interval, but you should still provide it for those that do.

© 2014 Microsoft. All rights reserved. | May 13, 2014

  - To help conserve power, set the **desiredAccuracy** property to indicate to the location platform whether your app needs high-accuracy data. If no apps require high-accuracy data, the system can save power by not turning on GPS providers.
    - Set **desiredAccuracy** to **HIGH** to enable the GPS to acquire data.
    - Set **desiredAccuracy** to **Default** and use only a single-shot call pattern to minimize power consumption if your app uses location info solely for ad targeting.

    If your app has specific needs around accuracy, you may want to use the **DesiredAccuracyInMeters** property instead of using **DesiredAccuracy**. This is particularly useful on Windows Phone, where position can usually be obtained based on cellular beacons, Wi-Fi beacons and satellites. Picking a more specific accuracy value will help the system identify the right technologies to use with the lowest power cost when providing a position.

    For example:

    - If your app is obtaining location for ads tuning, weather, news, etc., an accuracy of 5000 meters is generally enough.
    - If you app is displaying nearby deals in the neighborhood, an accuracy of 300 meters is generally good to provide results.
    - If the user is looking for recommendations to nearby restaurants, we likely want to get a position within a block, so an accuracy of 100 meters is sufficient.
    - If the user is trying to share his position, the app should request an accuracy of about 10 meters.
  - Use the **Geocoordinate.accuracy** property if your app has specific accuracy requirements. For example, navigation apps should use the **Geocoordinate.accuracy** property to determine whether the available location data meets the app's requirements.
- Consider start-up delay. The first time an app requests location data, there might be a short delay (1-2 seconds) while the location provider starts up. Consider this in the design of your app's UI. For instance, you may want to avoid blocking other tasks pending the completion of the call to **GetGeopositionAsync**.
- Consider background behavior. If a Windows Runtime app doesn't have focus, it won't receive location update events while it's suspended in the background. If your app tracks location updates by logging them, be aware of this. When the app regains focus, it receives only new events. It does not get any updates that occurred when it was inactive.
- Use raw and fusion sensors efficiently. Windows 8 supports two types of sensors: *raw* and *fusion*.
  - Raw sensors include the accelerometer, gyrometer, and magnetometer.
  - Fusion sensors include orientation, inclinometer, and compass. Fusion sensors get their data from combinations of the raw sensors.

  The Windows Runtime APIs can access all of these sensors except for the magnetometer. Fusion sensors are more accurate and stable than raw sensors, but they use more power. You should use the right sensors for the right purpose.

© 2014 Microsoft. All rights reserved. | May 13, 2014

 **Connected standby:** Windows only. When the PC is in connected standby state, **Geolocator** objects can always be instantiated. However, the **Geolocator** object will not find any sensors to aggregate and therefore calls to **GetGeopositionAsync** will time out after 7 seconds, **PositionChanged** event listeners will never be called, and **StatusChanged** event listeners will be called once with the **NoData** status.

Additional usage guidance

Detecting changes in location settings

On Windows, the user can turn off location functionality by using the Settings charm or Control Panel. On Windows Phone, the user can disable location in the Settings app.

- To detect when the user disables or reenables location services:
  - Handle the **StatusChanged** event. The **Status** property of the argument to the **StatusChanged** event has the value **Disabled** if the user turns off location services.
  - Check the error codes returned from **GetGeopositionAsync**. If the user has disabled location services, calls to **GetGeopositionAsync** fail with an **ACCESS_DENIED** error and the **LocationStatus** property has the value **Disabled**.
- If you have an app for which location data is essential—for example, a mapping app—, be sure to do the following:
  - Handle the **PositionChanged** event to get updates if the user's location changes.
  - Handle the **StatusChanged** event as described previously, to detect changes in location settings.

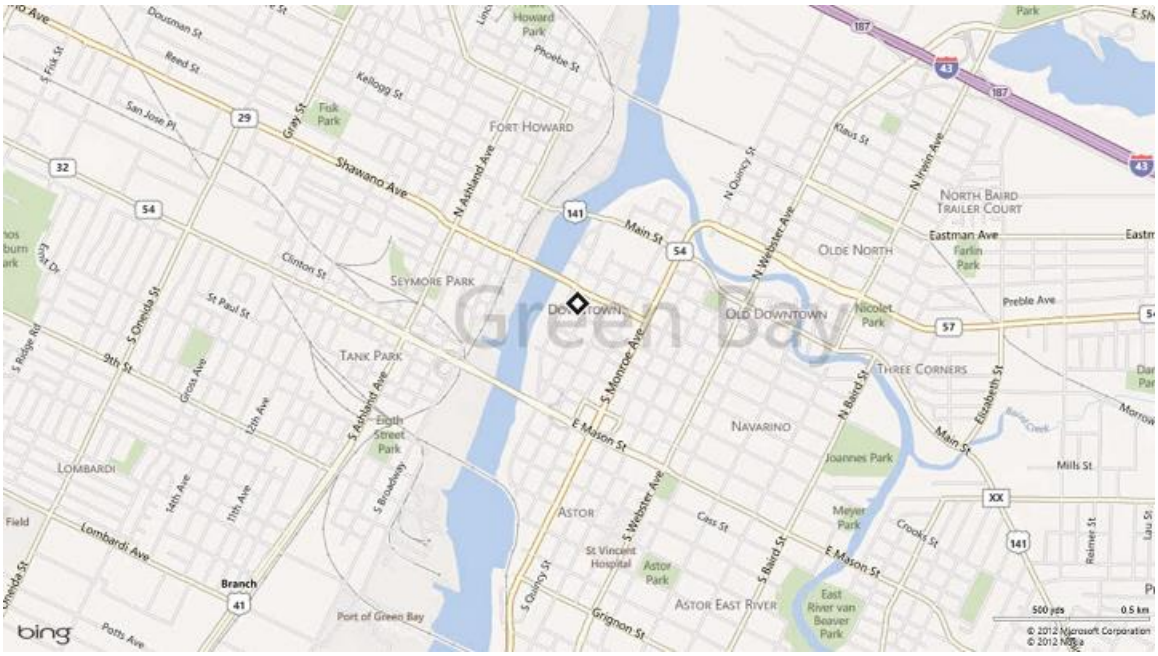
Note that the location API will return data as it becomes available. It may first return a location with a larger error radius and then update the location with more accurate information as it becomes available. Apps displaying the user's location would normally want to update the location as more accurate information becomes available.

Graphical representations of location

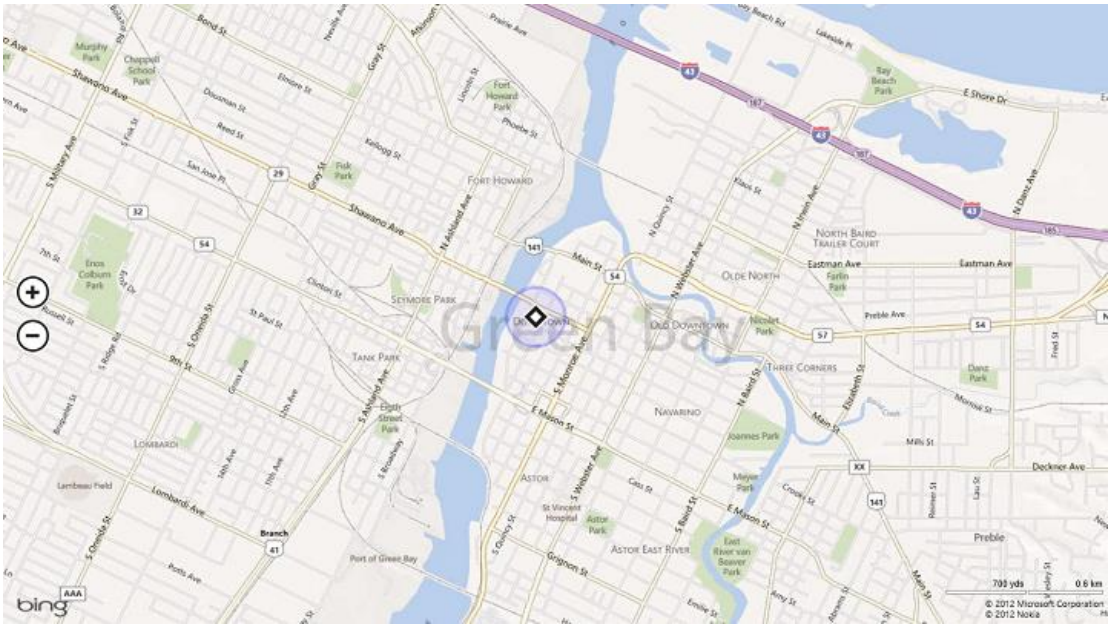
Have your app use **Geocoordinate.accuracy** to denote the user's current location on the map clearly. There are three main bands for accuracy—an error radius of approximately 10 meters, an error radius of approximately 100 meters, and an error radius of greater than 1 kilometer. By using the accuracy information, you can ensure that your app displays location accurately in the context of the data available.

- For accuracy approximately equal to 10 meters (GPS resolution), location can be denoted by a dot or pin on the map. With this accuracy, latitude-longitude coordinates and street address can be shown as well.

© 2014 Microsoft. All rights reserved. | May 13, 2014

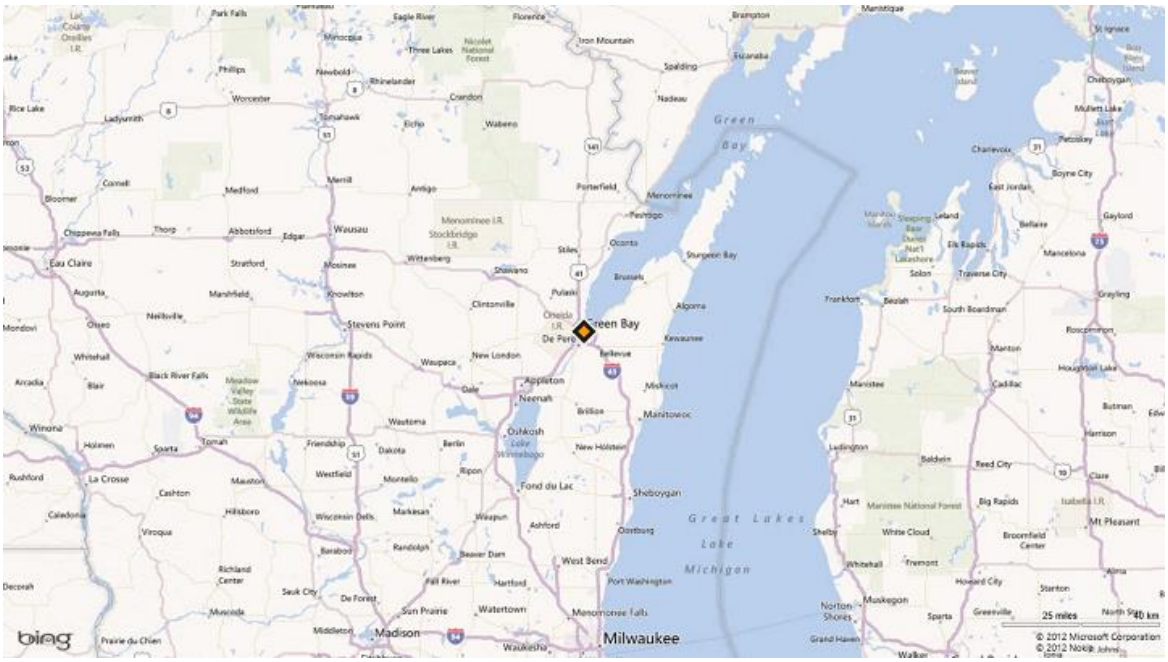


- For accuracy between 10 and 500 meters (approximately 100 meters), location is generally received through Wi-Fi resolution. Location obtained from cellular has an accuracy of around 300 meters. In such a case, we recommend that your app show an error radius. For apps that show directions where a centering dot is required, such a dot can be shown with an error radius surrounding it.



- If the accuracy returned is greater than 1 kilometer, you are probably receiving location info at IP-level resolution. This level of accuracy is often too low to pinpoint a particular spot on a map. Your app should refrain from showing any geographic representation of a pin or

circle and instead just zoom in to the city level on the map, or to the appropriate area based on the error radius (for example, region level).



When location accuracy switches from one band of accuracy to another, provide a graceful transition between the different graphical representations. This can be done by:

- Making the transition animation smooth and keeping the transition fast and fluid.
- Waiting for a few consecutive reports to confirm the change in accuracy, to help prevent unwanted and too-frequent zooms.

Textual representations of location

Some types of apps—for example, a weather app or a local information app—need ways to represent location textually at the different bands of accuracy. Be sure to display the location clearly and only down to the level of accuracy provided in the data.

- For accuracy approximately equal to 10 meters (GPS resolution), the location data received is fairly accurate and so can be communicated to the level of the neighborhood name. City name, state or province name, and country/region name can also be used.
- For accuracy approximately equal to 100 meters (Wi-Fi resolution), the location data received is moderately accurate and so we recommend that you display information down to the city name. Avoid using the neighborhood name.
- For accuracy greater than 1 kilometer (IP resolution), display only the state or province, or country/region name.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Privacy considerations

A user's geographic location is personally identifiable information (PII). The following website provides guidance for protecting user privacy.

- Microsoft Privacy

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for printing



Follow these guidelines when allowing users to print content from your Windows Store apps.

Dos and don'ts

- Display an error message when a user enters invalid info in the print window. Be sure to specify a corrective action and limit error messages to two lines or less.
- Don't use a print button in your app unless it's necessary for a specific task. In general, users should print using the Devices charm. However, in some circumstances, adding a print button might simplify a user's experience. For example, if a user expects to see a button to print a boarding pass after checking in to an airline, using the print charm might complicate the task.
- Don't change the order of the settings shown in the print window. Although the order of the settings shown to the user is customizable, retain the default order of settings to maintain a consistent experience for users. For example, the **Copies** settings are listed first in the default print experience; users expect this listing order in your app's print experience too.
- Don't add more printer settings to the print window unless absolutely necessary. Instead, allow printer manufacturers to handle the addition of printer-specific settings. If the manufacturer provides printer-specific settings, users can click **More Settings** in the print window to display additional settings (assuming that the user has installed the Windows Store device app that enables this display).

For developers

- Do the least amount of work possible to create the print task in the **PrintTaskRequested** event handler. Save more expensive work for when the **PrintTaskSourceRequestedHandler** is called to retrieve the printable content.
- Don't register the **PrintTaskRequested** event on pages of your app that don't support printing. When **PrintTaskRequested** is registered, Window assumes that printing is supported and will let users try to print. For example, if your news app doesn't support printing from its landing page, but does support printing from content pages, **PrintTaskRequested** shouldn't be registered when the landing page is in view.

Note: Blob contents can only be printed in full fidelity when you use a reusable URL. For more information, see Accessing the file system efficiently.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for print UI design

This topic discusses the print UI associated with a Windows Store device app. This type of app provides a device-specific, supplementary experience for the user. When you highlight features that are specific to a particular make and model of printing device, you can provide a much richer user experience. The info in this topic is meant for independent hardware vendors or developers who plan to write apps that communicate directly with their printing device. Follow these guidelines when you design a customized print UI for your Windows Store device app.

If you aren't writing a device app, see **Guidelines for print-capable apps** for more applicable recommendations.

Example

Here's an example of how a Windows Store device app can enhance a user's print experience. This app replaces the default print experience provided by Windows with a customized **More Settings** flyout and a **notification**, which can be used by devices to alert the user or app about device-related issues.

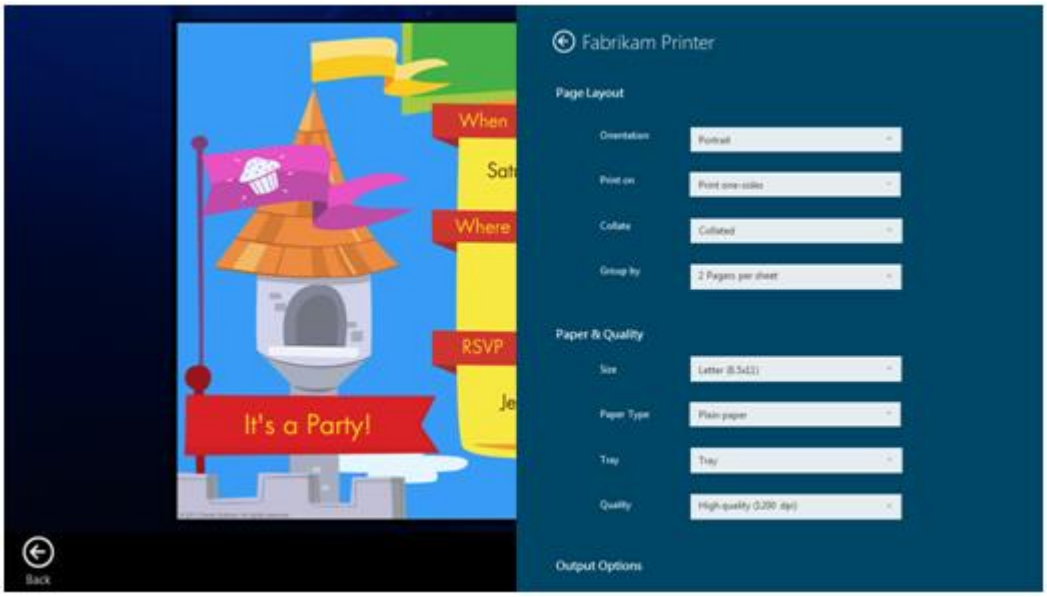


Amelia creates a party invitation, and then selects **Print**.

© 2014 Microsoft. All rights reserved. | May 13, 2014

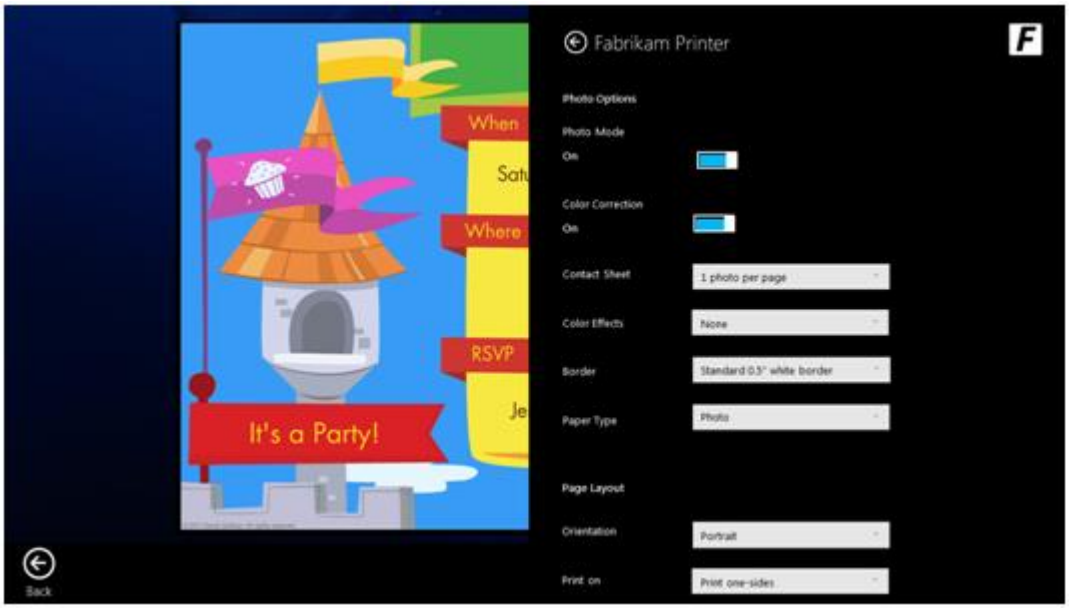


In the print flyout, Amelia selects **More Settings** to see the current printing preferences.

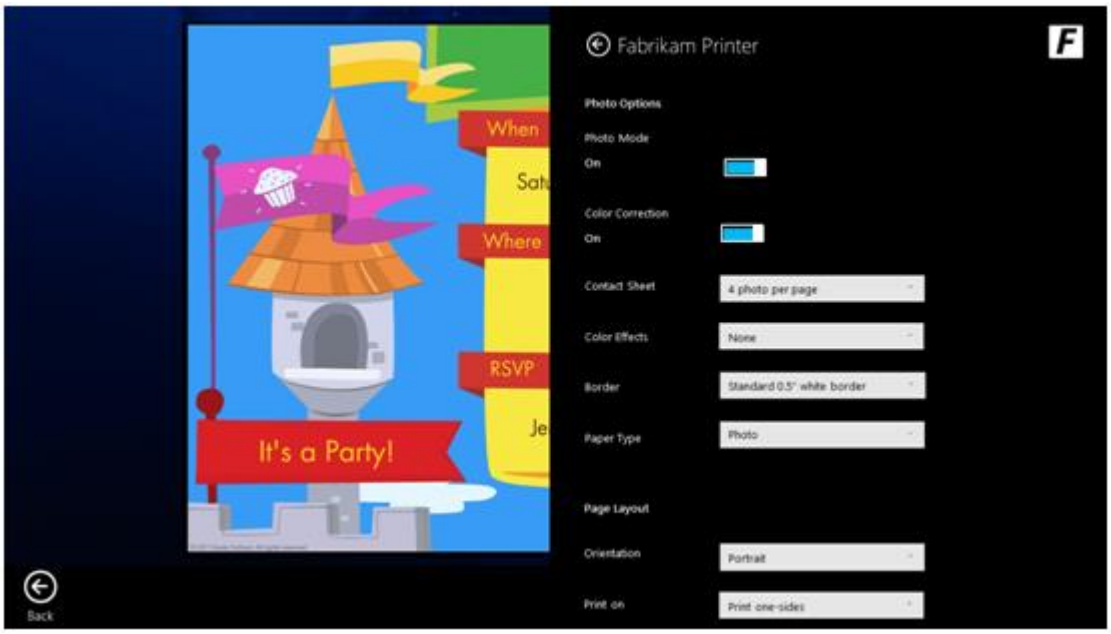


Without customization, this is the Windows-provided default printing preferences window that Amelia would see.

© 2014 Microsoft. All rights reserved. | May 13, 2014

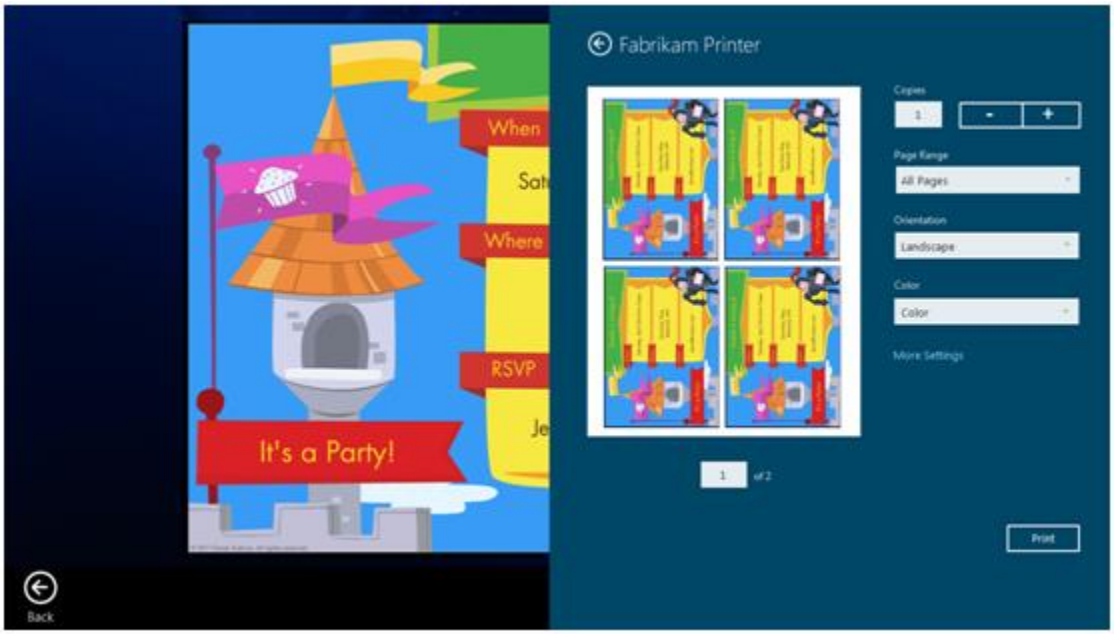


In this customized and branded Windows Store device app for print preferences, Amelia can modify the number of photos per page.



Next, Amelia presses the **Back** button. This automatically saves the new settings or printing preferences, and returns her to the original print flyout.

© 2014 Microsoft. All rights reserved. | May 13, 2014

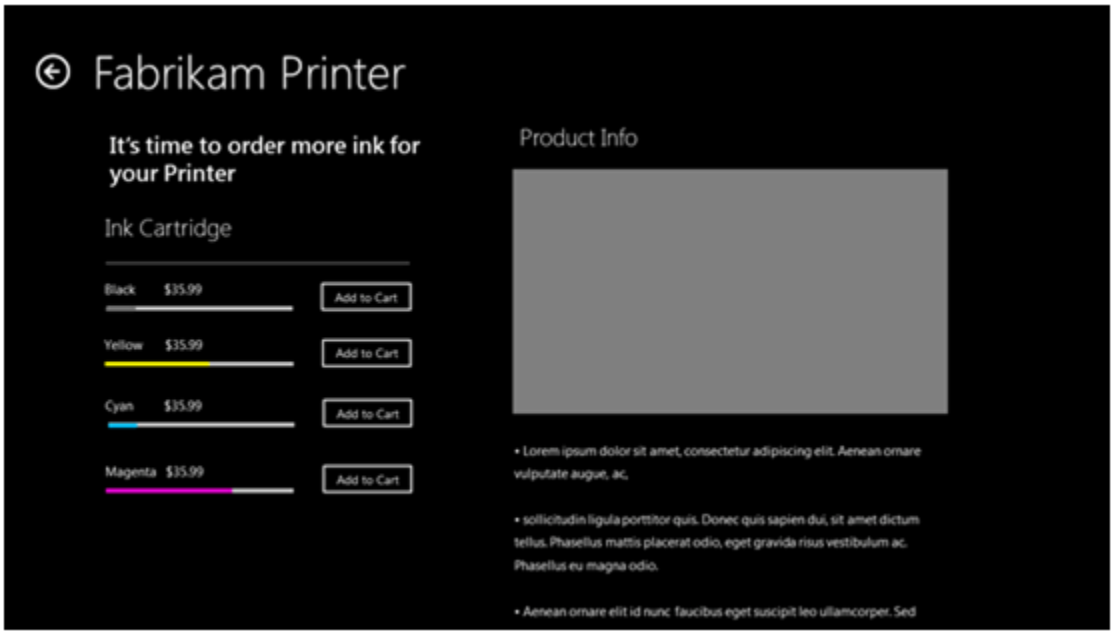


Amelia verifies her new printing preferences, then selects **Print**.

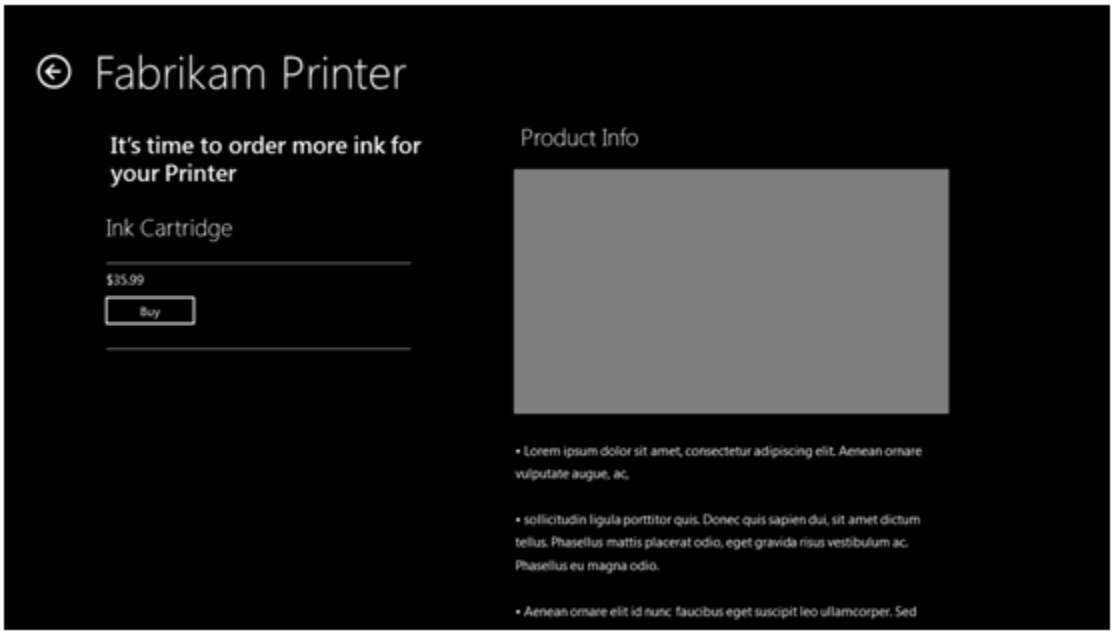


When Amelia selects **Print**, the app she's using receives a notification from the printer indicating that ink levels are low. This is called a toast notification.

© 2014 Microsoft. All rights reserved. | May 13, 2014



Amelia selects (or touches) the toast, and the next customized window shows her which ink cartridges have low ink. She selects **Add to Cart** to order replacements.



When Amelia adds the ink cartridge to her cart, the next window provides more information about the order that she's about to place. She reviews the information and selects **Buy**.

Amelia then selects or touches the **Back** button to return to the Print window, where she can select **Print** to print her invitation.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Should I create a Windows Store device app for my printer?

Use a Windows Store device app for a printer if you'd like to:

- Highlight advanced device capabilities, such as printing multiple photos per page.
- Make device-specific recommendations. For example, you could use your device app to present image management options or provide methods for setting and saving printer-specific defaults.

Dos and don'ts

- After you call window.print(), check for and handle error messages from within the onClick event handler for your app's Print button. This allows your app to abort a print request if, for example, no printer is available.
- Notify the user if printing fails and, if possible, explain the reason for the failure.
- If you plan to customize the print experience, separate this code into a print companion app. This allows you to componentize your code and eases the test and debugging process.
- Don't try to customize your print experience to use the V3 print driver.
- Don't advertise accessories for the print device in your customized print UI.
- Don't show items for sale that aren't related to the reason the Windows Store device app was invoked. For example, it's relevant to show print cartridges for purchase after a user clicks a notification alerting them that ink is low. However, it's not appropriate to also try to sell print cords or photo printing kits in this same scenario.
- Don't redirect the user to your company's website for more product sales.
- Don't present information that isn't relevant to the task of setting printing preferences. For example, don't provide info about how to clean the print heads or how to align and test the print nozzles.

Security considerations

The following articles provide guidance for writing secure C++ code.

- Security Best Practices for C++
- Patterns & Practices Security Guidance for Applications

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for Proximity



This topic describes best practices for using Proximity to connect apps and share content.

Proximity is a great way to create a shared app experience between two instances of your app running on two different devices. In a Proximity-enabled app, app users simply *tap* two devices together to initiate a connection, or a user can browse to find another device in wireless range that is running the app. On a PC, the user can use Wi-Fi Direct to find the app running on other PCs. On Windows Phone, the user can use Bluetooth to find the app running on other Windows Phones.

There are several ways to communicate using **Proximity**:

- **Out-of-band sessions**: You can establish a session using the **PeerFinder** object, which connects devices over an out-of-band transport (Bluetooth, Infrastructure network, or Wi-Fi Direct). Although the range for a tap is limited to 4 centimeters or less, the range for out-of-band transport options is much larger. You may not need to include Proximity in your app to share resources. If Windows supports your sharing scenario, then enable the Sharing contract and use built-in Windows functionality to share resources by using tap gestures.

  Note: Wi-Fi Direct is not supported on Windows Phone.

- **Browse for peers**: You can establish a session by using the **PeerFinder.FindAllPeersAsync()** method. This method finds all remote peers that are running the same app, if they have also called the **PeerFinder.Start()** method to advertise that they are available for a peer session. Browsing for peers does not use a tap gesture, but instead uses Wi-Fi Direct to discover the remote peer establish a connection.

  Note: Browsing for peers takes place over Bluetooth in a Windows Phone app. As a result, your app running on Windows Phone can only find phone peers and your app running on a PC can only find PC peers.

- **Publishing and subscribing for messages**: You can send or receive messages during a tap gesture by using the **ProximityDevice** object.

If an app calls the **ConnectAsync** method to create a connection with a peer, the app will no longer advertise for a connection and will not be found by the **FindAllPeersAsync()** method until the app calls the **StreamSocket.Close** method to close the socket connection.

You will only find peers when the device is within wireless range and the peer app is running in the foreground. If a peer app is running in the background, Proximity does not advertise for peer connections.

© 2014 Microsoft. All rights reserved. | May 13, 2014

If you open a socket connection by calling the **ConnectAsync** method, only one socket connection can be open at a time for the device. If your app, or another app calls the **ConnectAsync** method, then the existing socket connection will be closed.

Each app on a device can have one open connection to a peer app on another device, if that connection was established using a tap gesture. You can open socket connections from one app to a peer app on multiple devices by tapping each device. If you create a connection using a tap gesture, a new tap gesture will not close the existing connection. You must call the **StreamSocket.Close** method of the socket object to create a new connection to the same peer app on the same peer device using a tap gesture.

Note: Proximity only creates **StreamSocket** objects for network connections. If your app requires a different type of connection object than a **StreamSocket** object, you cannot use Proximity to connect.

Dos and don'ts

- When your app browses for other peers that are running your app, do not continuously browse for peers. Instead, let the user initiate the action by providing an option to browse for peers within range.
- Always ask for user consent to start a connected Proximity experience and put an app in multi-user mode. Asking for consent should be straightforward and dismissible while users are running an app. For example, two people each playing a game should be given a chance to provide consent before they decide to play the game together through Proximity. If a tap occurs as the app launches, users should be given a chance to provide consent on the start menu or in the lobby of the app.
- When a user puts an app in multi-user mode, update the UI to display one of these three connection states:
    - Waiting for a tap
    - Connecting devices (show progress)
    - Devices now connected, or connection failed.
- Revert to single-user mode if a connection breaks or can't be established. Provide a message for your user stating that the connection failed.
- Ensure that users can easily leave a Proximity experience.
- Don't continuously browse for other devices that are running the same app. Instead, provide the user an option to browse for peers within Wi-Fi range.
- Don't use Proximity if an app needs constant updates about the connection (for example, updates on bandwidth usage or speed).

© 2014 Microsoft. All rights reserved. | May 13, 2014

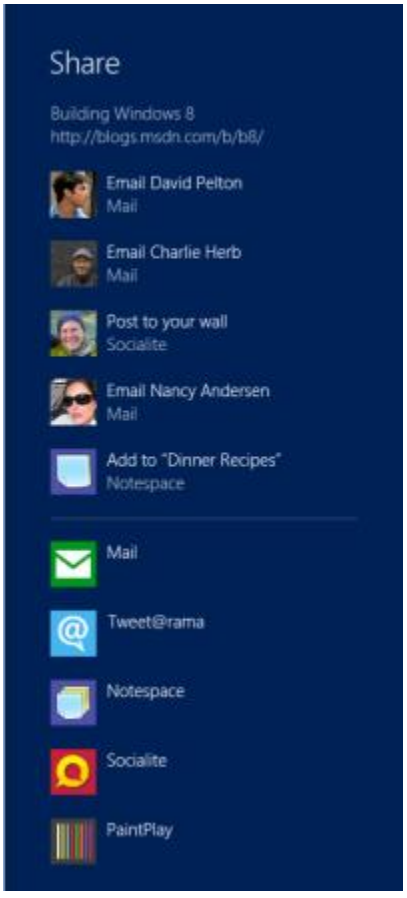
Guidelines for sharing content

Let users to share content from your app or receive shared content from other installed apps with the Share contract. You can register your app as a share source, share target, or both. This topic describes best practices for sharing content between Windows Store apps.

Examples

When people swipe from the side of the screen and tap the Share charm, the Share pane appears with a list of apps people can use to share their content. This list includes any installed apps that are “share targets” for a particular data format.

The links at the top of the image, called QuickLinks, allow users to complete specific share tasks directly. For example, when the user chose to share this content, QuickLinks appeared to let her email the URL to frequently used contacts, post it to the wall of her social media app, or add it to a specific page in a note-taking app.



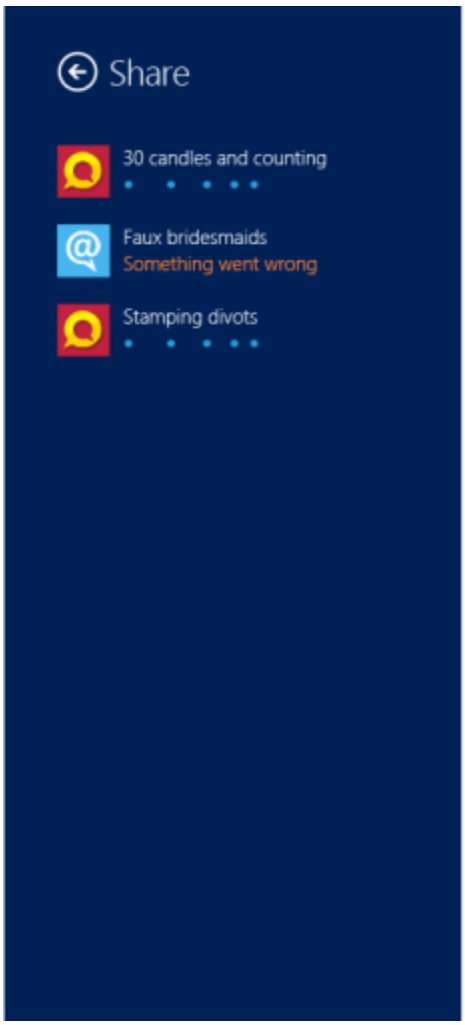
© 2014 Microsoft. All rights reserved.

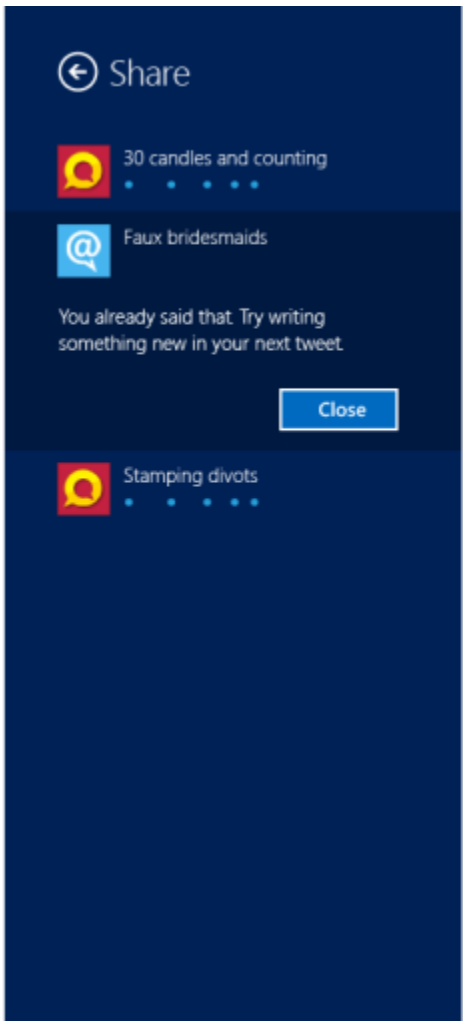
Here's the Share flyout that appears when the user chooses "Tweet@rama" from the list of share targets.



Lengthy share operations

Here's an example of a Share flyout completing lengthy share operations. Note the indeterminate progress bar and informative error message displayed by the target app when sharing failed.





© 2014 Microsoft. All rights reserved. | May 13, 2014

Dos and don'ts

Source apps

- Share content in the format intended by the user. For example, if a user selects a portion of a webpage to share, don't share a link to the entire webpage. Instead share the selected text.
- Provide a description that indicates what the user is sharing. For example, if the user is sharing a webpage, add a sentence that includes the URL of the page. If an image is being shared, include a description or title for the image.
- If a user selects a portion of the displayed app content, preserve this selection after the Share flyout (window) closes. This helps users if they want to modify their selection, or share the same content with multiple targets.
- Share links to online versions of local content rather than sharing a copy of local, downloaded content. For example, suppose a news site creates its own news app but also publishes the same articles on its website. If a user wants to share an article with a social networking site, the app should share links to the online version of the article that the user is currently viewing.
- Inform the user when sharing can't be completed. If a particular sharing operation fails, display a message in the Share flyout that describes the problem and, if applicable, how the user can solve it. The **DataRequest** object supports a **FailWithDisplayText** method that is useful in this circumstance.
- If your app supports a way to copy data in the app, you should also provide a way to share that same data.
- Set properties to supply target apps with useful info about the content a user wants to share. See **DataPackage.DataPackagePropertySet** to learn about the available properties.
- Don't display a message that sharing is not supported by your app. Windows displays a standard message for users if your app doesn't support the Share contract.

Target apps

- Keep the look and feel the same between your target app and your primary app. The UI of your target app should feel familiar to people who use your primary app frequently. Keep fonts, colors, and controls consistent.
- If your setup and sign-in processes only involve one step, let users complete those tasks through the Share charm so they don't have to change context.
- If your app is both source and target for a particular data format, then it appears by default in the list of share targets each time people share from your app. If it doesn't make sense for a user to share content using the same app, display an error message prompting the user to select a different target app.
- Remove links that lead users away from the sharing experience. For example, if your target app has links that lead to other areas of your app (such as to a home page), you should remove or hide them so the user doesn't leave the sharing experience accidentally.
- Previews should match the actual content whenever possible. If your app includes a preview of what the user is sharing, that preview should match what will actually be shared as much as possible.
- Acknowledge user actions. When a user taps the Share charm or invokes the Share UI, let them know that the system is responding to their action—for example, through an inline

© 2014 Microsoft. All rights reserved. | May 13, 2014

message—before closing the share pane. This helps give the user confidence that their share started successfully.
- Take advantage of the **QuickLink class**. QuickLinks are links to your app that are customized for a specific set of user actions (like sending an email to a specific person). In the Examples section above, the links above the list of available target apps are QuickLinks.
- Don't create a back button to navigate between pages in your target app. When a user selects your app to share content, Windows automatically provides a back button for navigating back to list of available target apps. Keep navigation simple in your target app using inline controls and error messages.
- Don't perform time-consuming, complex, or multi-step interactions in your target app. Text formatting, tagging people in photos, or setup tasks like connecting to data sources are best handled outside of the Share charm. If your app requires multi-step sign-in or setup processes, tell users to open your app directly to complete more complex interactions.

Additional usage guidance

Debugging target apps

A share target app can be launched only through the Share charm, and clicking outside the app dismisses it. This means that most local debugging scenarios are impractical for share targets because clicking in the debugger will dismiss the target app. To debug a share target app, you typically need to use the simulator or remote debugging because they operate on another machine, either virtually or physically.

For share target scenarios that can be debugged locally, remember that you can't close the target app after launching it from Visual Studio because the debugger will detach. Instead, leave the full-screen app open, go to the Start screen, and then perform your share scenario.

What to debug

A share target app is activated within a fraction of a second after launching, typically before you can attach a debugger to the running process. To debug code in the activation handler, it's best to debug directly on the local machine or through the simulator.

If you've implemented a long-running share operation that can run even when a user navigates away from your app, errors might occur after the app's UI is dismissed. In this case, the debugger can remain attached to the target app even when the source app is out of view. With the debugger attached, you can catch exceptions, pause on breakpoints, and walk through code without the app UI on screen.

Common debugging issues

- Unhandled exceptions in a target app cause it to immediately terminate and be replaced with an error message. The target app should gracefully handle any expected errors originating from the user, such as invalid input data, and report them to the user.

© 2014 Microsoft. All rights reserved. | May 13, 2014

- If a target app takes too long to respond to an activation event, the system assumes that the app is choosing not to respond and displays an error. Processing data should be moved out of the activation handler whenever possible, typically by storing a **ShareOperation** object and processing it asynchronously.
- Calls to sharing the API can throw exceptions when called too many times or in the wrong order. When you implement a long-running share, be sure to call the share methods in the following order, without calling any single method twice in a row. You can call **ReportError** or **ReportCompleted** at any time to complete the sharing operation.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Controls

This section aggregates design information about all the available controls into a single place for ease of access and quick reference. It describes the standard controls that are available without customization. It also provides an example of usage in a real app for each control.

Using standard controls can save you time when creating common interactions. They work well right out of the box. When working with a developer, you can reference these controls in your design in order to use a common language.

The standard controls are only a starting place. For some interactions customization is not necessary, but for others it provides an opportunity to delight users with unique UI or to infuse branded elements into the app. For example, in some apps, the button control with no changes is the right, fast solution, and in others, customized buttons are a branded element of the app.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for app bars







Description

App bars provide users with easy access to commands when they need them. Users can swipe the top or bottom edge of the screen to make app bars appear and can interact with their content to make app bars disappear. App bars can also be used to show commands or options that are specific to the user's context, such as photo selection or drawing mode. They can also be used for navigation among pages or sections of the app.

If you have commands that are necessary for a user to complete a workflow (such as buying a product), place those commands on the canvas instead of in app bars.

Top app bar

The navigation bar, or top app bar, is the recommended place to put navigation controls that let the user access other areas of the app. By using the top app bar for navigation, you provide a consistent and predictable user experience for navigating Windows Store apps. Consistency gives users the confidence to move around the system and helps them transfer their knowledge of app navigation from app to app.

Users should be able to complete core scenarios just by using the canvas. The navigation bar is a secondary location for navigation controls. The navigation bar can help orient the user to all parts of an app, can provide quick access to the home page, or can encourage users to explore by jumping to different parts of the app.

You can also choose to include other functionality within the top app bar, such as adding a '+' button to create something new or integrating a search box. When you add other functionality, we recommend that you place these on the right side of the app bar.

While you can style items in the nav bar, the default appearance is a simple button. However, another typical visual treatment is the use of button and thumbnail, as shown in the previous image.

© 2014 Microsoft. All rights reserved. | May 13, 2014

You can also divide the nav bar into multiple areas, as the Store app does. Here you can see that the top section is for global navigation and the bottom section is for app categories.



Bottom app bar

The bottom app bar is the recommended place to put commands. By moving commands from the app canvas to the app bar, you deliver the most immersive experience possible to the user.

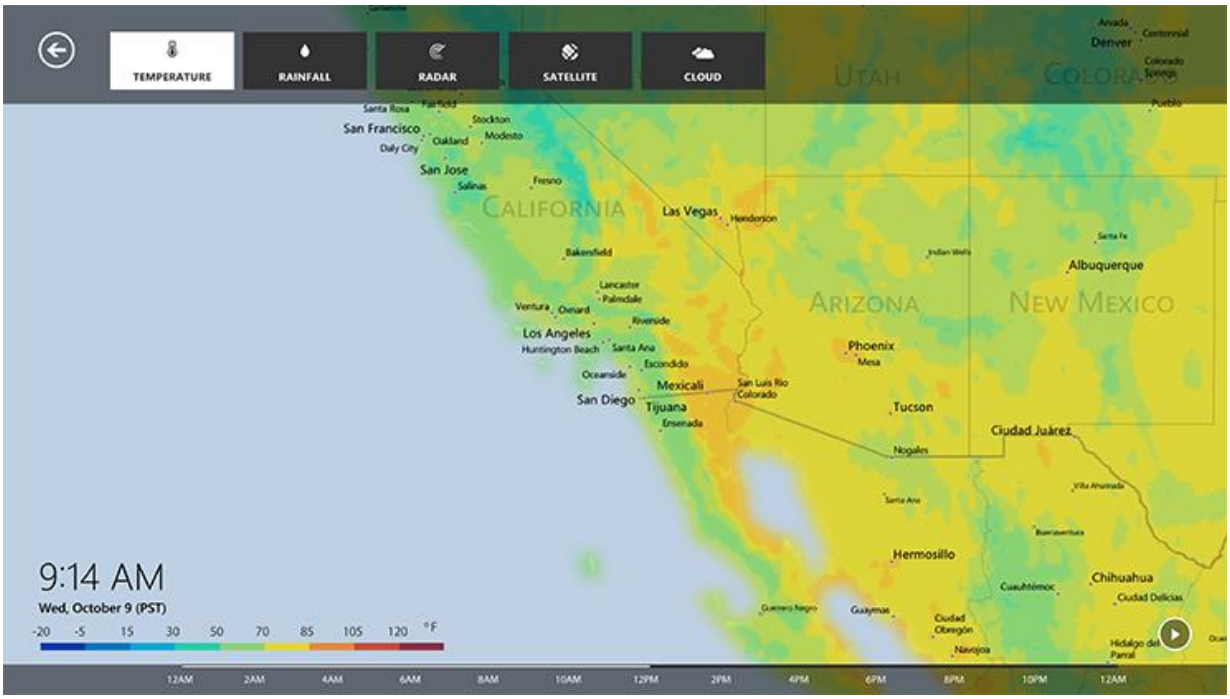
The standard app bar control is intended for the developer who is interested in implementing an app bar with little to no custom work. While it is easy to create an app bar, it is not easy to ensure that it behaves in accordance with Windows guidance and patterns. The **CommandBar class** and **WinJS.UI.AppBar object** are made to align with the intended design and behaviors so the developer doesn't have to think about every little detail, and will be less likely to diverge on the common commanding pattern.

If you want to depart from the standard experience and customize your app bars, use the **AppBar** control instead of the **CommandBar** control in XAML.

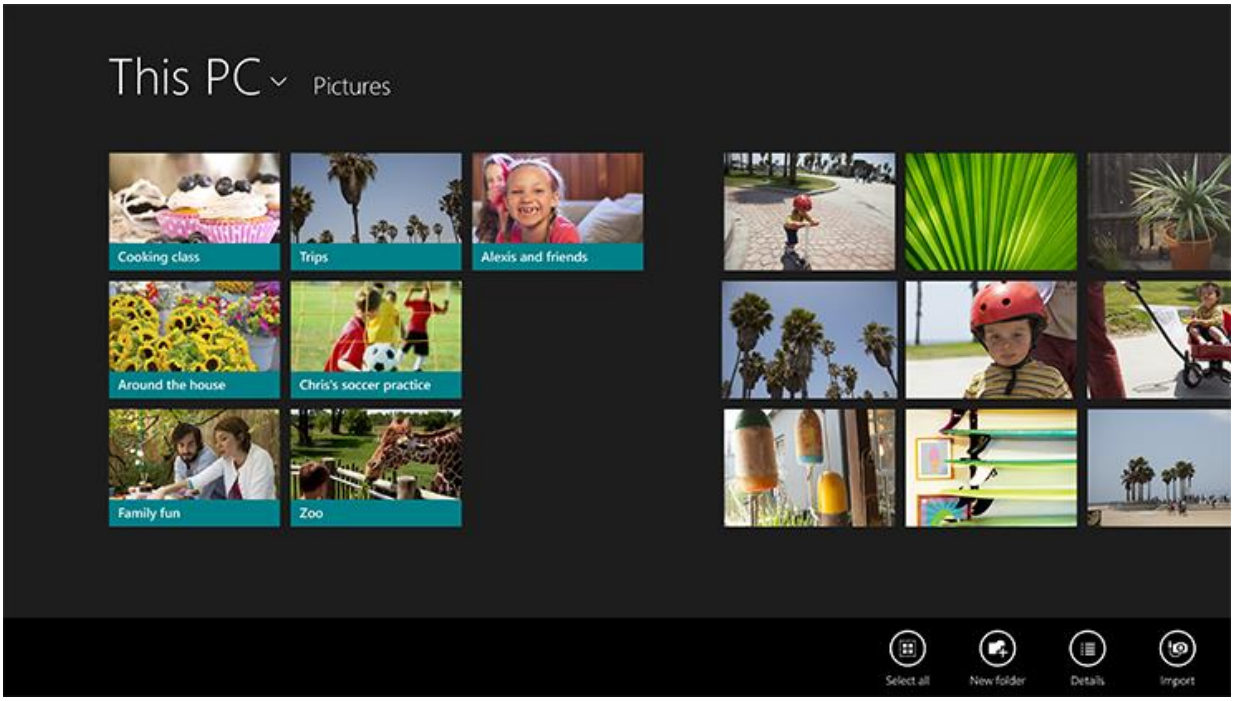
Examples

Top app bar

© 2014 Microsoft. All rights reserved. | May 13, 2014



Bottom app bar



© 2014 Microsoft. All rights reserved. | May 13, 2014

Dos and don'ts

- Separate navigation and commands. Ideally, put commands in the bottom app bar and navigation in the top app bar. Consider providing help the first time that an app runs to let users know the important commands that are in the app bar.
- If you have distinct sets of commands (such as a set for creating new content, and a set for filtering your view), place one set on the right side and one on the left side of the app bar. If you have more than two sets, separate the sets with a separator.
- Place commands in a consistent location throughout the app. While each page should only contain the commands that are relevant to that page, if any commands are shared between them, they should be as close to the same location as possible, so that users can predict where the commands will be.
- Follow placement conventions:
    - Place the New/Add/Create button (+ icon) on the far right.
    - Group view switching buttons together and place them on the far left.
    - Place Accept, Yes, OK commands to the left of Reject, No, Cancel commands.
- Show commands contextually on an app bar when an item is selected, and show the app bar automatically. Most people are right-handed, so when people select an item on the app page, show whatever commands are relevant to that item on the left side of the app bar. That way, their arms or hands won't block their view of the commands.
- If you have contextual commands on an app bar, set the mode to sticky while that context exists so that the bar doesn't automatically hide when the user interacts with the app. Turn off sticky mode when the context is no longer present. For example, you might want to show contextual commands for photo manipulation when a user selects an image, but you want to let the user keep working with the image, like rotating or cropping the image. In this case, the bar stays visible until the user deselects the image or dismisses the app bar with an edge swipe.
- If you are unable to fit all of your commands in an app bar as separate buttons, group like commands together and place them in menus that people can access from app bar buttons. Use logical groupings for the commands, such as placing Reply, Reply All, and Forward in a Respond menu.



- Design your app bar for resized and portrait views. If you have ten commands or fewer, Windows automatically hides labels and adjusts padding for you when people resize your

© 2014 Microsoft. All rights reserved. | May 13, 2014

app or rotate to portrait view, so provide tooltips for all your commands. If you want a more custom view, you can either group commands together into menus or provide a more focused experience that requires fewer commands in resized or portrait view.

In JavaScript apps, it's a good idea to place global commands before contextual commands in the DOM in order to get the best layout when people resize your app.

In C#/C++/VB apps, resizing is handled for you if you use the **CommandBar** control.

- If your app has a horizontal scrolling area at the bottom of the app page, reduce the height of the scrolling area when the app bar appears in sticky mode. Otherwise, the app bar can cover up your scrollbar and people might need to go out of their way to dismiss the app bar in order to keep scrolling. You should try to keep the bottom edge of the scrollbar flush against the top edge of the app bar.
- Don't put critical commands on the app bar. For example, in a camera app, place the "Take a picture" command on the app page rather than in an app bar. You could either add a button to the app page or simply let people tap the preview to take the picture.
- Don’t put login, logout, or other account management commands in the app bar. All account management commands, like login, logout, account settings, or create an account should go in a Settings flyout. If it's critical that people log in on a particular page, provide the login button on the app page.
- Don't put app settings in the app bar. All app settings commands, like defaults and preferences, should go in a Settings flyout. The Settings flyout is also the best place for seldom used management commands, like those for clearing history.

Additional usage guidance

Scaling for different window sizes

When people resize a window with an app bar, the system resizes the commands and might drop labels. When reduced-sized commands no longer fit on a single row in the screen, the system wraps them to a second row.

- Design at least two views of the app bar, one full size and one reduced size (either at 500px or 320px minimum). Most people only use common window sizes, either full screen or 50/50 with another app.
- Group commands when designing smaller views. If you can't meaningfully group commands, place them in a "more" group using the ellipses icon.

Common window resolutions (in pixels)	Number of reduced-sized buttons with no labels in a single row	Number of full-sized buttons with labels in a single row
1366	22	13
1024	16	10
768	12	7
500	8	5
320	5	3

© 2014 Microsoft. All rights reserved. | May 13, 2014

Handling the right mouse button

To keep your app's UI consistent with other Windows Store apps, users must click the right mouse button to trigger the app bar that you provide. If you have an app that must use the right mouse button for another purpose, like secondary fire in a game or a virtual trackball in a 3-D viewer, the app can ignore the events that raise the app bar. But you should still consider the role of the app bar, or a similar context menu, in your game's control model, because it's an important part of the Windows Store apps experience.

Follow these guidelines when designing the controls for your app:

- If your app needs to use the right mouse button for an important function, use it for that function directly. Don't activate any contextual UI or the app bar if it isn't important to workflow.
- Show the app bar when the user right-clicks regions of your app that don't need contextual right-click actions, such as border menus.
- If right mouse button support is needed everywhere, consider showing the app bar when the user right-clicks the top-most horizontal row of pixels, the bottom-most horizontal row of pixels, or both.
- If none of these solutions suffice, use custom user interaction controls to open the app bar with the mouse.
- Use the **MouseDevice** class events, like **MouseMoved**, to implement your own context menu behaviors.
- Remember that the press-and-hold touch gesture is the same as a right button click. Handle both events in a similar way. To handle this event and define a custom behavior for it, register for the **Holding** event. To enable holding, set **Hold** (for touch and pen input) and **HoldWithMouse** in the **GestureSettings** property.
- Don't change the behavior of the Windows logo key + Z combination. Your app should display the app bar or a context menu. Register for the **KeyDown** and **AcceleratorKeyActivated** events to determine when these two keys have been pressed.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for back buttons

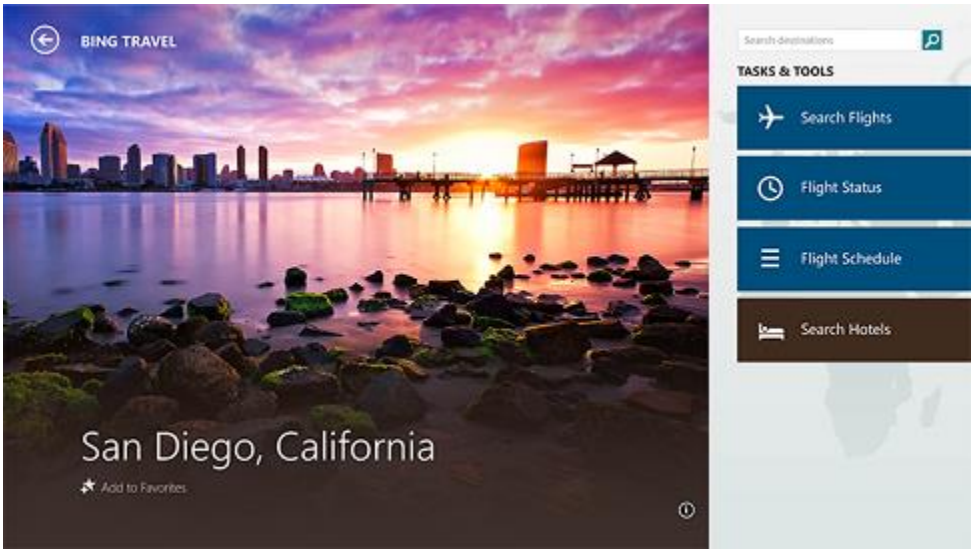




Description

A back button provides backward navigation in the form of a button.

Example



Dos and don'ts

- Use the back button in apps that use a hierarchical navigation pattern.
- When your app is at narrow widths, use a smaller back button size.
- Don't use the back button in apps that use a flat navigation pattern. In flat navigation apps, the user typically moves through pages either through direct links within the content or through the nav bar.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for buttons





Windows app: button states



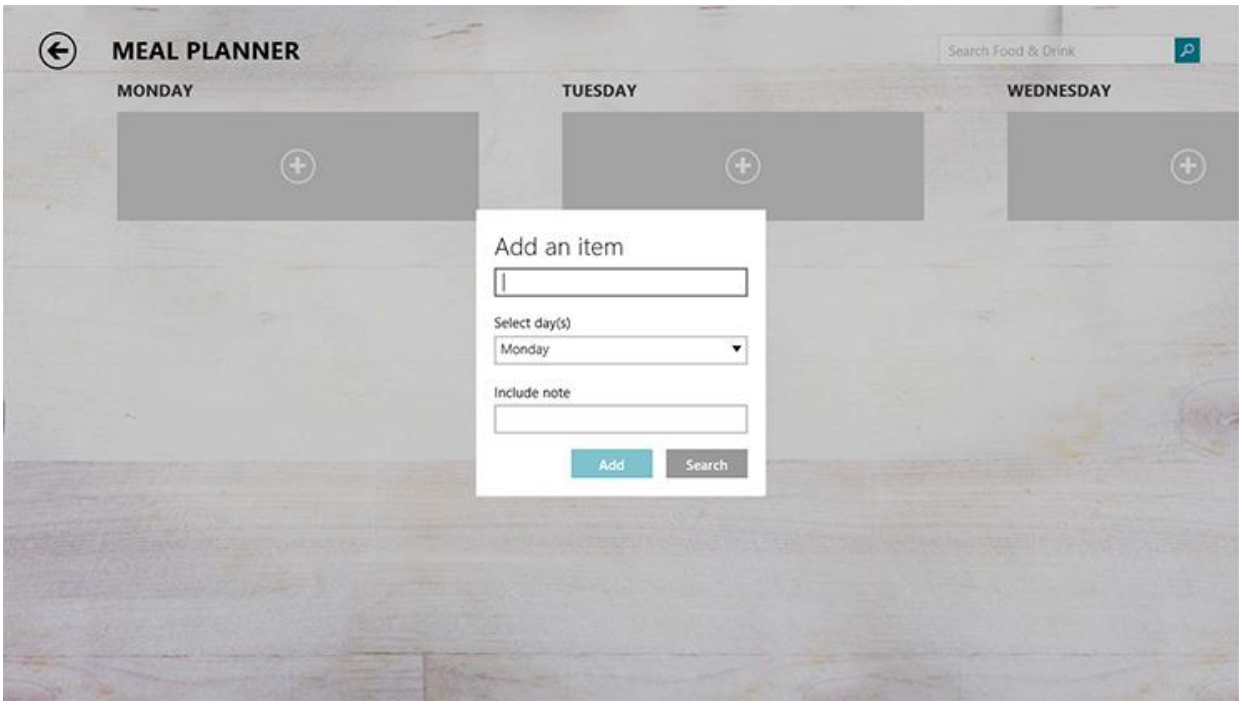
Windows Phone app: button states

© 2014 Microsoft. All rights reserved. | May 13, 2014

Description

A button, also known as a command button, gives the user a way to trigger an immediate action.

Example



Is this the right control?

A button lets the user initiate an immediate action, such as submitting a form.

Don't use a button when the action is to navigate to another page; use a link instead. Exception: For wizard navigation, use buttons labeled "Back" and "Next". For other types of backwards navigation or navigation to an upper level, use a button with the `win-backbutton` style.

Dos and don'ts

- Make sure the purpose and state of a button are clear to the user.
- Use a concise, specific, self-explanatory text that clearly describes the action that the button performs. Usually button text content is a single word, a verb.
- If the button's text content is dynamic, for example, it is localized, consider how the button will resize and what will happen to controls around it.
- For command buttons with text content, use a minimum button width.
- Avoid narrow, short, or tall command buttons with text content.
- Use the default font unless your brand guidelines tell you to use something different.

© 2014 Microsoft. All rights reserved. | May 13, 2014

- For an action that needs to be available across multiple pages within your app, instead of duplicating a button on multiple pages, consider using a bottom app bar.
- When using AJAX to submit a form, use submit and override the form submit function so users can commit by pressing the enter key regardless of where the focus is in the form.
- Expose only one or two buttons to the user at a time, for example, Accept and Cancel. If you need to expose more actions to the user, consider using checkboxes or radio buttons from which the user can select actions, with a single command button to trigger those actions.
- Use the default command button to indicate the most common or recommended action.
- Consider customizing your buttons. A button's shape is rectangular by default, but you can customize the visuals that make up the button's appearance. A button's content is usually text—for example, Accept or Cancel—but you could replace the text with an icon, or use an icon plus text.
- Make sure that as the user interacts with a button, the button changes state and appearance to provide feedback to the user. Normal, pressed, and disabled are examples of button states.
- Trigger the button's action when the user taps or presses the button. Usually the action is triggered when the user releases the button, but you also can set a button's action to trigger when a finger first presses it.
- Don't use a command button to set state.
- Don't change button text unless for localization.
- Don't swap the default submit, reset, and button styles.
- Don't put too much content inside a button. Although it can contain almost any other HTML element, such as tables and check boxes, too much content will confuse users. Make the content concise and easy to understand (nothing more than a picture and some text).

Additional usage guidance

Choosing the right type of button (JavaScript and HTML)

There are three types of HTML **button** controls: **submit**, **reset**, and **button**. Follow these guidelines to choose the right type:

- **Submit**
  Send a user input to a server or perform an action, such as saving form data and going to the next app page.

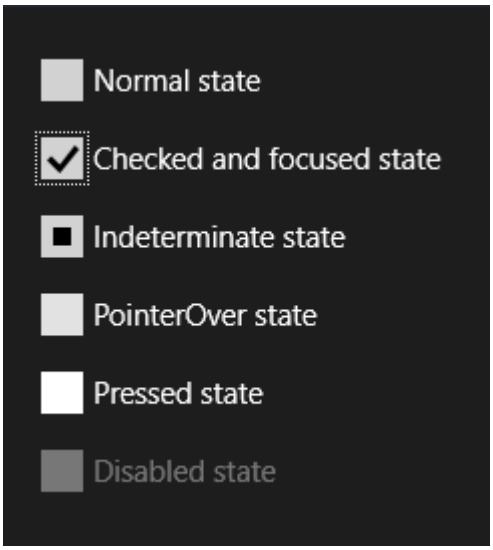
- **Reset**
  Clear a form or page of user input.

- **Button**
  Create a customized command or action.
  Inside a form, a **button** without any attributes acts as a **submit** if it is the first **button** inside the form.

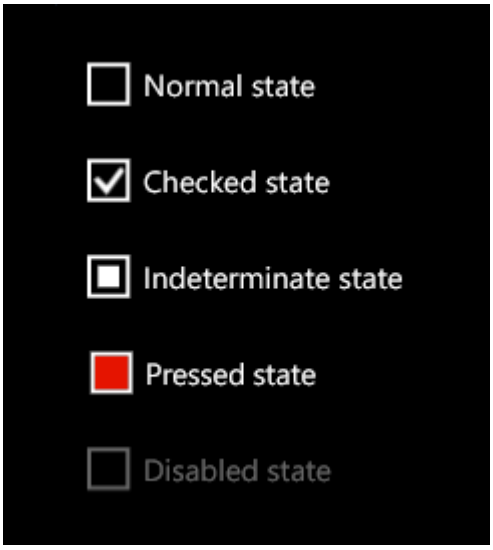
© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for check boxes





Windows app: check box states



Windows Phone app: check box states

© 2014 Microsoft. All rights reserved. | May 13, 2014

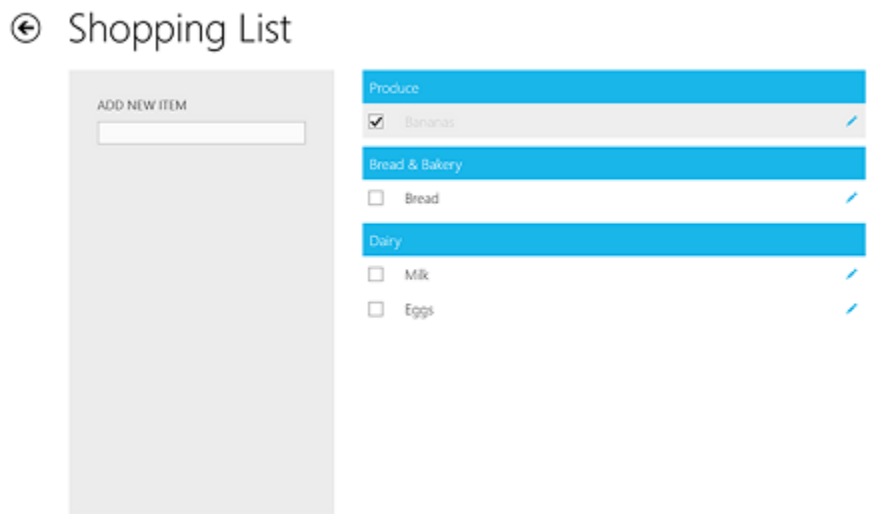
Description

Typically, a check box gives the user a way to register a choice between two discrete and opposite options. For example, the choice may be whether to subscribe to a newsletter, or whether to select each email in a list. In rare cases a check box will be neither unchecked nor checked, but instead it will be in what's called an indeterminate state. For example, a check box's checked state is indeterminate when it represents a collection of sub-choices with values that are mixed.

As the user interacts with it, a check box gives feedback by changing its state and therefore its appearance. Normal, pressed, unchecked, checked, and disabled are examples of check box states.

A user taps the check box to toggle it to the opposite value.

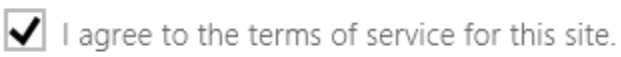
Example



Is this the right control?

Use check boxes to present users with a binary choice, one or more options that are not mutually exclusive, or a mixed choice.

Use a single check box for a binary yes or no choice.

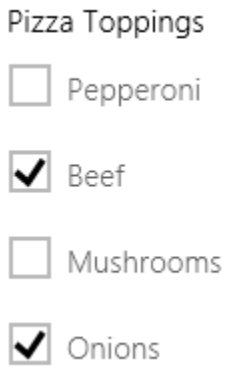


For a binary choice, the main difference between a check box and a toggle switch is that the check box is for status and the toggle switch is for action. You can delay committing a check box

© 2014 Microsoft. All rights reserved. | May 13, 2014

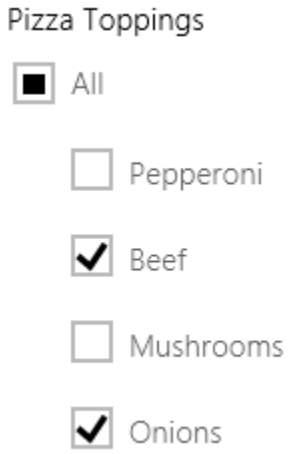
interaction (as part of a form submit, for example) while you should immediately commit a toggle switch interaction. Also, only check boxes allow for multi-selection.

Create a group of check boxes when users can select any combination of options.



Unlike radio buttons, where a group of radio buttons represents a single choice, each check box in a group represents a separate, independent choice. When there is more than one option but only one can be selected, use a radio button instead.

When an option applies to more than one object, you can use a check box to indicate whether the option applies to all, some, or none of those objects. When the option applies to some, but not all, of those objects, use the check box's indeterminate state to represent a mixed choice. One example of a mixed choice check box is a "Select all" check box that becomes indeterminate when a user selects some, but not all, sub-items.

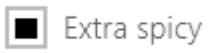


© 2014 Microsoft. All rights reserved. | May 13, 2014

Dos and don'ts

- Verify that the purpose and current state of the check box is clear. Verify that it is clear what is meant by the clearing the check box. For example, if you use a check box for Landscape orientation, it is not clear that clearing the check box means Portrait orientation. You could use two radio buttons instead – Landscape and Portrait.
- Limit check box text content to no more than two lines.
- Word the text content as a statement that the check mark makes true, and the absence of a check mark makes false.
- Use the default font unless your brand guidelines tell you to use another.
- If there are several choices to present, consider using a scroll viewer control with a layout panel inside it.
- Enclose the check box within a label with a check box so that clicking the label toggles the check box. Doing so increases the size of the selection area and makes the check box more accessible to touch users.
- Use the indeterminate state to indicate that an option is set for some, but not all, sub-choices.
- When using indeterminate state, use subordinate check boxes to show which options are selected and which are not. Design the UI so that the user can get see the sub-choices.
- If the text content is dynamic, consider how the control will resize and what will happen to visuals around it.
- Don't put two check box groups next to each other, or users won't be able to tell which options belong with which group. Use group labels to separate the groups.
- Don't use a check box as an on/off control or to perform a command; use a toggle switch instead.
- Don't use a check box to display other controls, such as a dialog box.
- Don't use the indeterminate state to represent a third state. The indeterminate state is used to indicate that an option is set for some, but not all, sub-choices. So, don't allow users to set an indeterminate state directly.

  For an example of what not to do, this check box uses the indeterminate state to indicate medium spiciness:

  Instead, use a radio button group that has three options: Not spicy, Spicy, and Extra spicy.

© 2014 Microsoft. All rights reserved. | May 13, 2014

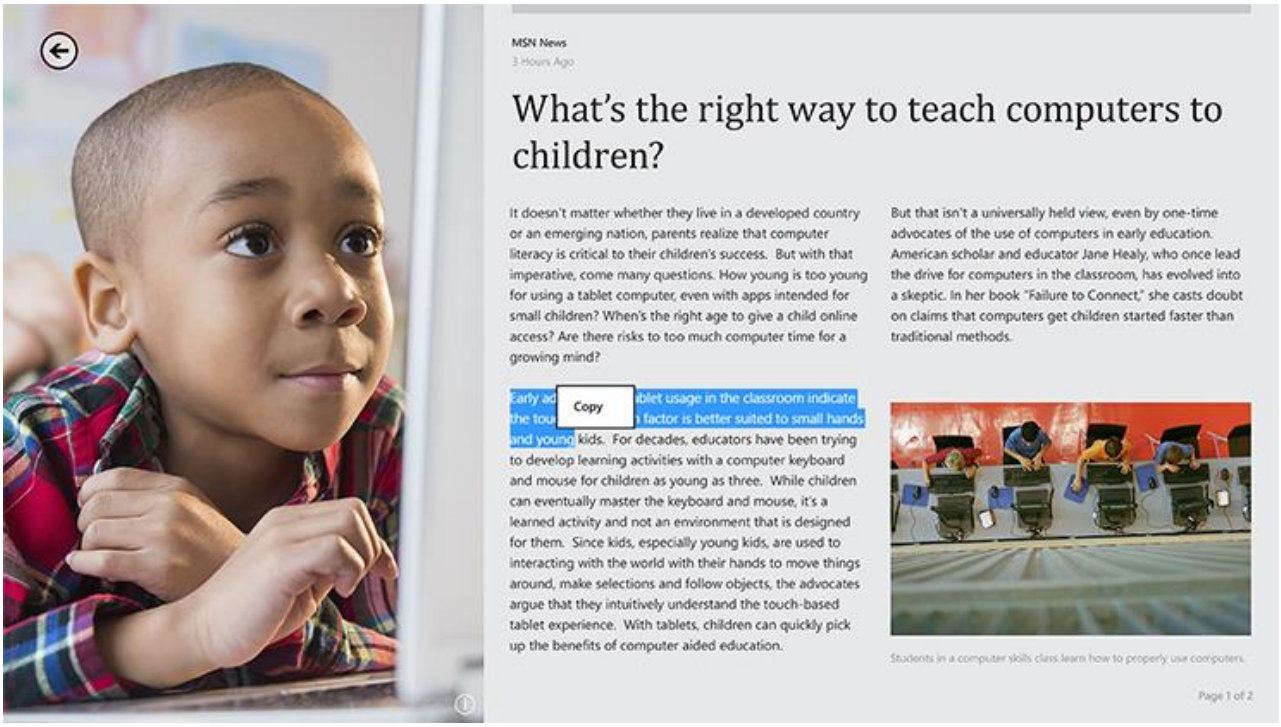
Guidelines for context menus



Description

A context menu is a lightweight menu that gives users immediate access to actions on text (like clipboard commands). The system provides apps with default context menus for text and hyperlinks. You can replace the default context menus with menus that show custom commands for text and hyperlinks.

Example



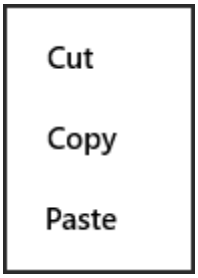
© 2014 Microsoft. All rights reserved. | May 13, 2014

Dos and don'ts

- Show custom commands. The system comes with default context menus for text and hyperlinks. Apps can replace these context menus with their own context menus. A custom command is appropriate to include in a context menu when it is not found in the app's toolbar and cannot be performed using direct interaction (like rotation).



- Use a context menu to show clipboard commands (Cut, Copy, and Paste) for objects such as selected text. By default, the system shows Cut, Copy, and Paste commands for selected text. Common paste menu commands are Select All, Paste, and Undo. You can override these commands by showing a customized context menu. Where possible, mimic system behavior in your app by preserving the default commands.



- Use the context menu to show commands for an object that needs to be acted upon but that can't be selected or otherwise indicated. For example: A chat conversation may not be appropriate to add selection to; in this case a context menu could make commands available for each message in a chat conversation.

© 2014 Microsoft. All rights reserved. | May 13, 2014

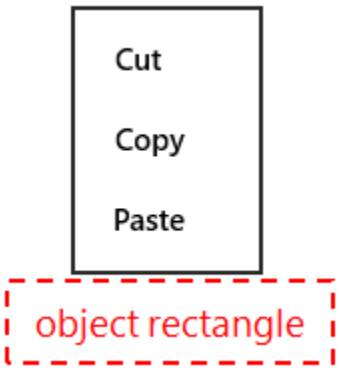


- Keep command names short. The context menu has a maximum width that equates to approximately 50 characters. When command names are too long, they are truncated automatically and an ellipse ("…") replaces the missing characters. Avoid truncation of your commands in the context menu by keeping command names short.
- Use sentence capitalization for each command name. The first character of the command should be uppercase and all of the remaining characters should be lowercase.
- Use a separator to distinguish groups of related commands. Use separators in the context menus to group sets of commands together. Use a separator to divide app-specific or view-specific commands from a predictable set of commands, such as clipboard commands, that are shown together.
- Show the fewest number of commands possible, up to the six-command limit.

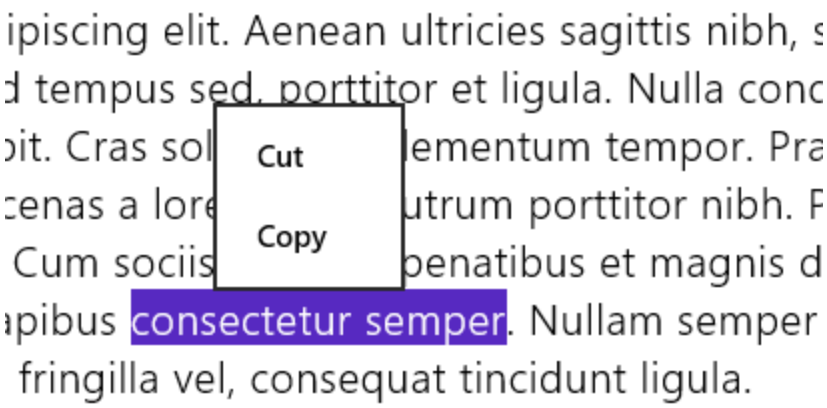
  If you're struggling to fit commands into the menu, ask yourself the following questions:

  - Can this command be represented with direct manipulation?
  - What would your user's experience be like without this command?
  - Is this command accessible in another way? What value, if any, is gained by duplicating the command in the context menu?
  - Does the item have a page that represents it? If so, the command could be in the app bar or on the canvas on that page instead on the context menu in the collection.
  - Does this command always need to be shown, or only in certain contexts?
- Order custom commands in the context menu by importance, with the most important commands at the bottom.
- Order clipboard commands in the standard Cut, Copy, Paste order and place them at the bottom of the menu. If you want to show only two clipboard commands (for example, Cut and Paste) simply omit the unused command and otherwise preserve the order.
- Position the context menu close to the object your user wants to act on. The context menu should be positioned close to the object or selection that it is acting on. When showing the context menu, provide a rectangle for the object that it will be acting on, and the context menu will position itself near it. By default, a context menu should be centered above the object that it is acting on.

© 2014 Microsoft. All rights reserved. | May 13, 2014



- Dismiss the context menu. Programmatically dismiss the context menu when the context it was shown for does not exist anymore. This can be done by using cancel in the standard async pattern.
- Don't use item accelerators. Item accelerators cannot be used with context menu commands. Do not use ampersands (&) at the beginning of command names in your menu.
- Don't show a command in a context menu unless it is contextually relevant to the selection or object. Note that the context menu does not have a disabled state; the context menu doesn't appear if there are no commands in it. For example, if there is no text in the clipboard that can be pasted into text that the user can edit, the default context menu does not show the **Paste** command. Instead, it shows only the **Cut** and **Copy** commands.



- Don't add a command to the context menu when direct manipulation or selection is possible. Users should execute commands primarily by directly manipulating a UI element or by selecting a UI element and using a command that is on the app bar. This helps ensure commands are in predictable and discoverable screen locations. For example, users should be able to rotate a picture by manipulating the image directly with their fingers instead of using a "Rotate" command in the context menu.
- Don't duplicate commands in the context menu. If there is already a clear way to accomplish the action like direct manipulation or an existing app bar command , do not add a context menu command for that same action. Instead of relying on duplication, trust that users find commands by selecting an item or navigating to an item to act on it. However, there is an exception for some actions that can be executed by keyboard shortcuts. If the action can

© 2014 Microsoft. All rights reserved. | May 13, 2014

only be executed by a keyboard shortcut, like CRTL+C to copy, it is okay to duplicate that action by adding a command to the context menu.
- Don't show a context menu for the background of a page or for a large object. Instead, when you have commands that act on the background of a page or on an object that takes up the whole screen, use the app bar or add a command to your app's canvas to act on the page or object.
- Don't show a command that would result in an error. For example, do not show the paste command if the paste location cannot be edited.

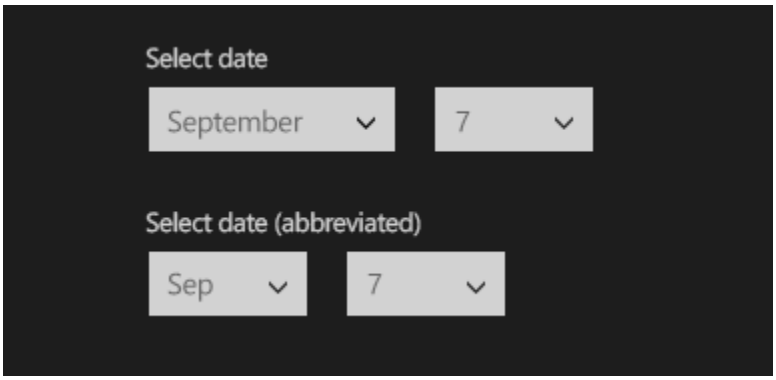
Additional usage guidance

A context menu is typically hidden from view and is an efficient means of conserving screen space. Use a context menu if there is a small set of contextual commands and options that apply to the selected object or window region. Context menus have a specific order: most frequently used items first (primary commands), transfer commands next, and properties last. This order provides efficiency and predictability.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for date pickers





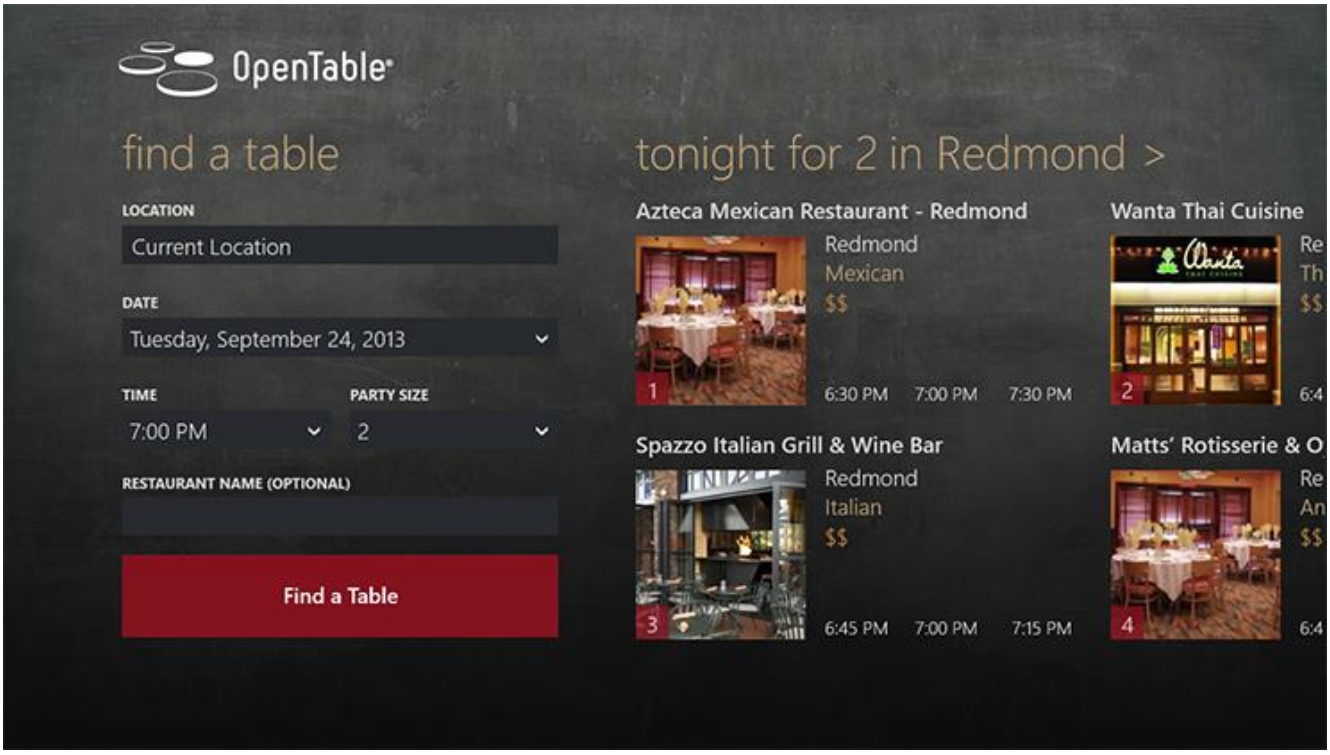
Description

The date picker provides a standardized way to let users pick a localized date by using touch, mouse, or keyboard input.

Use the date picker when a user needs to select a single date. The date picker is a good option for conserving real estate because the screen space used is fixed and independent of the number of choices.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Example

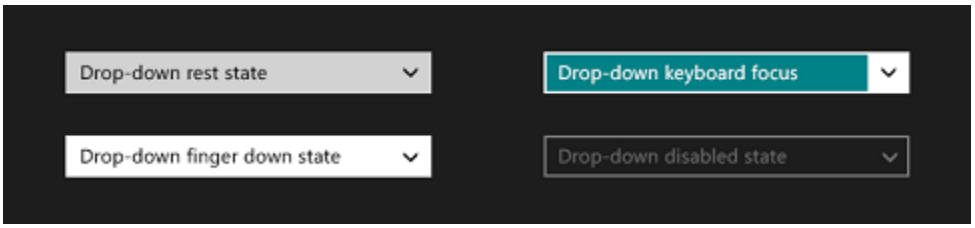


Dos and don'ts

- Use the date picker to display dates on forms or when you need to use space efficiently.
- Don't set the year range to more than 200 years. Instead, define a useful range of years by using the properties for minimum and maximum years.
- Don't use the date picker for selecting a date range.
- Don't use the date picker for performing commands, displaying other windows such as dialog boxes, or dynamically displaying other controls.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for drop-down lists



Description

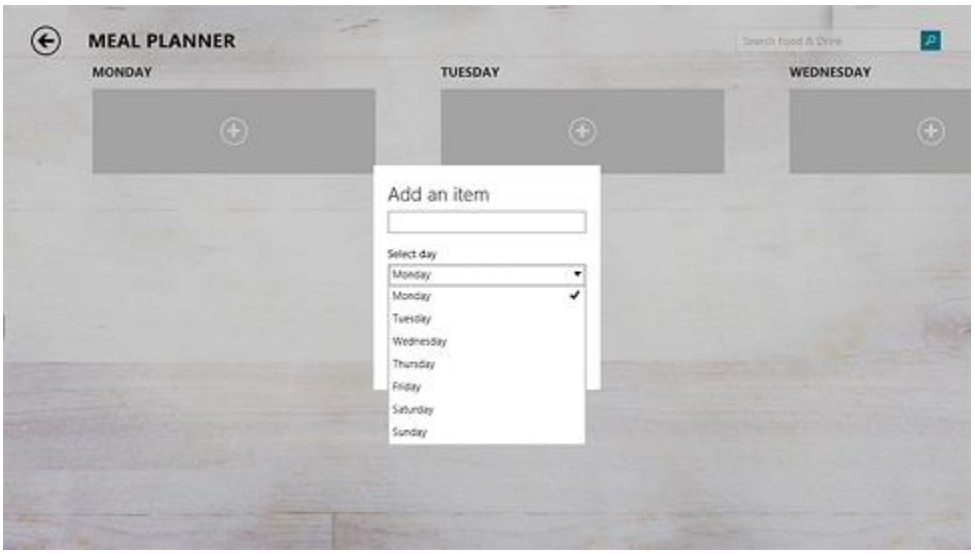
A drop-down list lets users make a choice among a list values. A drop-down list is a list in which the selected item is always visible, and the others are visible on demand by clicking a drop-down button. In HTML, a drop-down list is known as a select control in flyout mode. In XAML, a drop-down list is known as a combo box. With a drop-down list, users can choose one and only one option.

You can also create a list that displays all items without any additional interaction. It supports both single selection and multiple selection. In HTML, this kind of list is known as a select control in inline mode. In XAML, this list is known as a list box.

Examples

Here is an example of a drop-down list. The items in the list are only visible when the user selects the drop-down button.

© 2014 Microsoft. All rights reserved. | May 13, 2014



Here is an example of a list box in which all items in the list are always visible.

Is this the right control?

Use a drop-down list control to let users select a single value from a set of items that can be adequately represented using single lines of text.

Don't use this controls to display items that contain multiple lines of text or images. Use a list view or grid view instead.

When there are fewer than five items, consider using radio buttons (if only one item can be selected) or check boxes (if multiple items can be selected) instead.

Use a drop-down list when the selection items are of secondary importance in the flow of your app. If the default option is recommended for most users in most situations, showing all the items by using a list box might draw more attention to the options than necessary. You can save space and minimize distraction by using a drop-down list.

Dos and Don'ts

- Limit the drop-down list item’s text content to a single line.
- Sort items in a drop-down list in a logical order, such as grouping related options together, placing most common options first, or using alphabetical order. Sort names in alphabetical order, numbers in numeric order, and dates in chronological order.
- Add padding of 27 pixels on the right side of your content to keep the scrollbars from overlapping the content.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for flip view controls

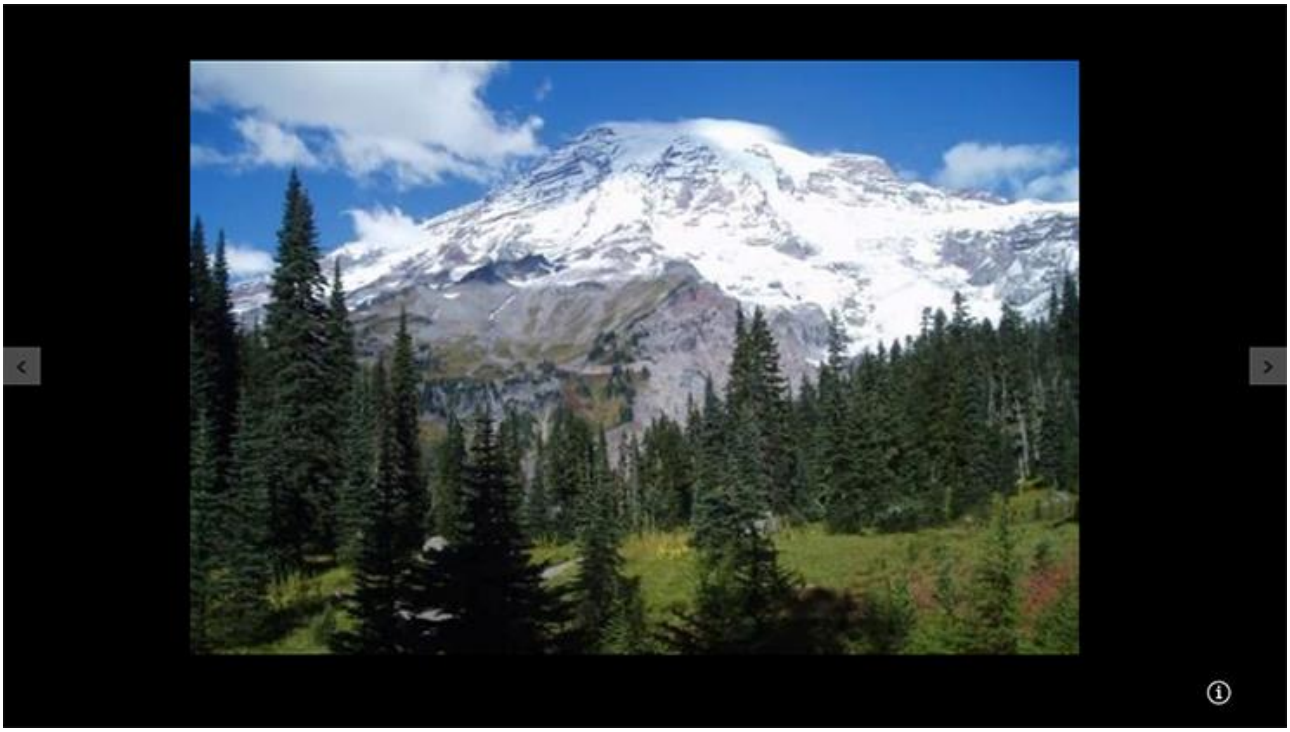


Description

The flip view control lets people flip through views in an app or through a collection of items, like photos in a photo album, one at a time. It offers people a way to look at each individual item while navigating through the collection.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Example



Is this the right control?

The flip view control lets people flip through views or items one at a time. Flip buttons appear on mouse hover and let people flip to the next or previous item.



You can also add a context indicator control so users can jump directly to a particular item.

© 2014 Microsoft. All rights reserved. | May 13, 2014

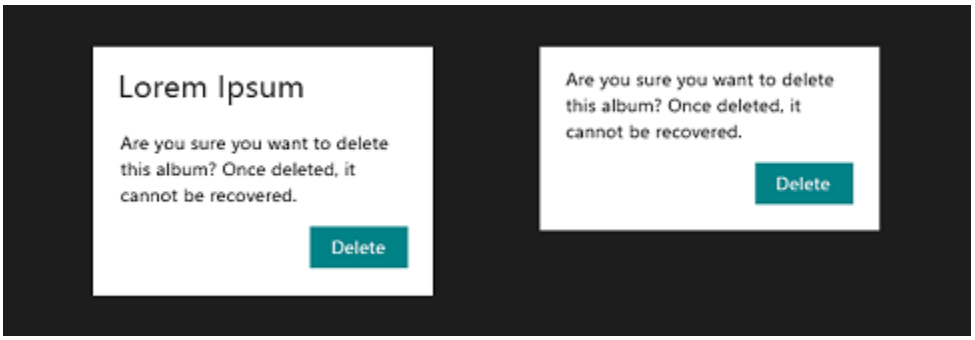


Dos and don'ts

- Use a flip view control to:
    - Flip between items in a small collection (< 10) of related items. For example, in a shopping app, you might want to show several images of a product in a product details page. You can use the context indicator control for a small collection, but this is often not necessary.
    - Flip between items in a medium-sized list (10 to 25) of related items. For example, in a real estate app, you might want to show a number of images of each house, showcasing the rooms and views. For these medium-sized lists, include a context indicator control, like a filmstrip of thumbnails, that lets users to jump to specific photos.
    - Flip between views of an application. For example, an app launcher can have one view that shows a user's favorites in a grid and another that shows the favorites in a list. You can add a flip view control to let users navigate from one view to another.
- Use a context indicator when the items in the collection don't provide enough contextual information to help users know where they are in the collection. A flip view for days of the week would not need a context indicator, whereas a flip view for product images would.
- Give users an indication of where the current item is in the collection. Use a context indicator to help users know how large the collection is and where the current item is in relation to the others in the collection.
- Tailor the indicator for the number of items and the specific scenario. For small collections, you can show the entire collection in the context indicator. For medium-sized collections, you might want to show only 5 at a time, for example. For purely visual collections, you might display thumbnails; for text-based collections, you might prefer to display the alphabet so users can jump to the right letter.
- Allow users to jump to specific items. Make sure you always help users feel in control of where they are and where they want to go.
- Don't use a flip view control for large collections. The repetitive motion of flipping through each item becomes tedious for users. Instead, use a list view or a grid view.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for flyouts

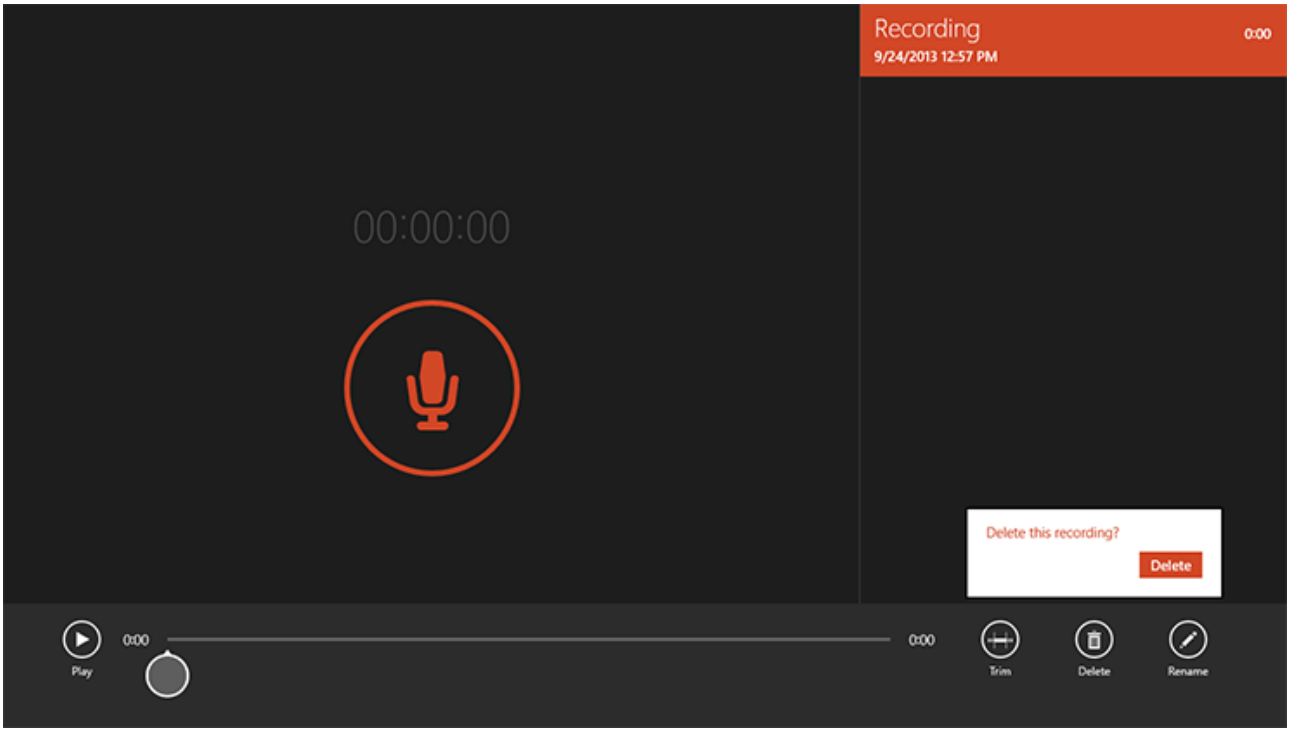


Description

A flyout is a lightweight popup that is used to temporarily show UI that is related to what the user is currently doing. It can be used to show a menu, reveal a hidden control, show more details about an item, or ask the user to confirm an action. You should only show a flyout in response to a user tap or click. A flyout is always dismissed when the user taps outside of it.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Example



Dos and don'ts

- Use flyouts to show UI that you don't want on the screen all the time. A user can dismiss a flyouts at any time by simply tapping or clicking outside of it, or by pressing ESC. When the user makes a selection in the flyout, the flyout should be dismissed.
- Position a flyout next to its point of invocation. If the user taps a button in the app bar to bring up a flyout, then the flyout should appear above or below the app bar button. If showing the flyout above or below the control would obscure important content, then it can be placed to the left or right of it. When you implement a flyout you position it by specifying the object to which you want it anchored and the side of the object on which it should appear. Flyouts should be center-aligned to their anchor unless the anchor is on the very edge of the screen (such as the user-tile flyout in the Start screen).
- Use flyouts for the following:
    - **Collecting information:** If the user selects an action that requires more input, such as choosing an option or typing information, then that UI can be placed in a flyout to keep the user in their original context. For example, let's say that in a map app, users can label the locations they tag. Users tap the location to tag it, and the app presents a flyout so that users can enter their label.

      Example: In the browser, to pin an item to the Start screen, the user taps the Pin icon on the app bar. The user then enters the name of the new tile in a flyout

© 2014 Microsoft. All rights reserved. | May 13, 2014

- **Warnings and confirmations:** Warn the user before they take a potentially destructive action.

  Example: In a photo app, the user presses a delete icon in the toolbar. Next to the toolbar button, a flyout appears that warns the user that the photos will be permanently deleted, and provides the delete command. The user can easily press the revealed delete command that appears, or dismiss the flyout if they pressed the delete icon by accident.

  Note: The only warnings or errors that should go in a flyout are those that can be shown immediately and are a direct result of user action.

- **Drop-down menus:** If a button in an app bar has more than one option, then display a drop-down menu in the flyout to let the user pick the option.

  Example: In an email app, the user presses Respond on the app bar, and a menu is displayed to let the user choose among ways to respond: Reply, Reply All, or Forward. If the user presses the Cancel button on the app bar, then a menu is displayed to let the user choose between the ways to Cancel: Discard or Save Draft.

  Note: Use context menus, not flyouts, for contextual actions on text selections.

- **Displaying more info:** Show more details about an item on the screen that the user is interested in.

  Example: In the browser, while browsing InPrivate, the user selects the InPrivate icon. A flyout then appears to give the user more information about InPrivate mode. Most of the time the browser UI is kept clean, but if requested can provide more detail for users that are interested.

- Identify the order in which users use the TAB key to navigate from one item to another in your flyout. Be sure to set your tab indices to positive values (0 is considered the same as not set) and make sure the range of tab indices within the flyout does not overlap with the ranges of tab indices for elements outside the flyout, or users will jump between the flyout and the main app page.
- Don't dismiss a flyout programmatically unless the user has pressed a command button or selected a menu item in the flyout. A flyout should not be dismissed automatically if the user has simply toggled a setting, for instance.
- Don't use a flyout if the message, error, warning, or other piece of UI is not invoked directly by the user at that moment. Example: Notifications that updates are available, a trial has expired, or the Internet is not available, should not be displayed in flyouts.
- Don't use a flyout if an experience is one that requires prolonged interaction, multiple screens, or lots of UI. Instead, integrate the UI into the canvas of the app. For example:
  - The user is working through a wizard with a lot of text entry.
  - The user is changing a long list of settings.
- Don't use a flyout for the primary list of commands for your app. Use the app bar for that.
- Don't use a flyout if a menu is required solely for commands about a text selection. Use a context menu instead.

© 2014 Microsoft. All rights reserved. | May 13, 2014

- Don't include controls on a flyout that are not necessary for the situation. For example, if there are no actions for the user to take, then don't include any buttons. There is no need for Close or OK buttons; relying on light-dismiss (in which the flyout disappears when the user touches anywhere on the screen outside of the flyout) will do just fine. Similarly, if a title isn't absolutely necessary, then don't include a title.
- Don't add extra padding beyond what is provided by the flyout itself. The flyout should be as small as possible given its content.
- Don't position flyouts in non-contextual places, such as the center of the screen, for the following reasons:
    - When UI is shown in a position disconnected from the action that invoked it, the user needs to go searching for this UI and is slowed down. The overall experience is disrupted, creating less pleasant and fluid UI.
    - The user may not notice that the flyout has appeared and may accidentally dismiss it by continuing to tap, making your app feel unresponsive.
    - Users expect that centered (or other arbitrarily positioned windows) contain a Close or Cancel button, and would use them even if a light-dismiss option was present, undermining your goal of lightweight UI.

Additional usage guidance

Parts of a flyout

A flyout has three components: the title, the main content, and command buttons.

Here are recommendations for when to use each component for each common use of the flyout.

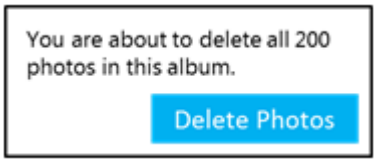
- **Collecting information:**

Title	None
Main content	Include just the controls you need. Keep any instructions or "Learn More" links to a minimum. If the user is changing a setting or toggling an on/off switch, for example, then the change should commit as soon as it is made. Interacting with custom content should not dismiss the flyout; unless there is a command button, the user should be in control of dismissing the flyout manually.
Controls	If a button is just meant to commit the user's changes, then it isn't required and those changes should be committed automatically. If the button begins some action (such as Login, or Save Document), or the user has entered text that they want to commit, then a button is appropriate and the flyout should be dismissed when the user presses the button. But the user can cancel without committing by light-dismissing the flyout. 

-

© 2014 Microsoft. All rights reserved. | May 13, 2014

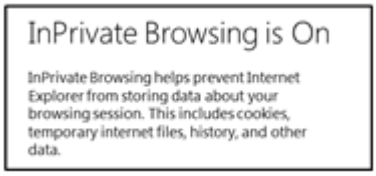
- **Warnings and confirmations:**

Title	None
Main content	State the warning that the user should consider before taking the action. **Do not** phrase it as a question. 
Controls	Include just the action that the user initiated, such as Delete. Do not include the opposite action or a Cancel button; that can be achieved by dismissing the flyout.

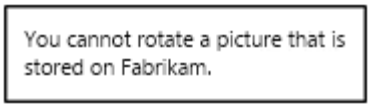
- **Drop-down menus:**

Title	None
Main content	List the menu items that the user can interact with. 
Controls	Use the **MenuCommand** for these buttons.

- **Displaying more info:**

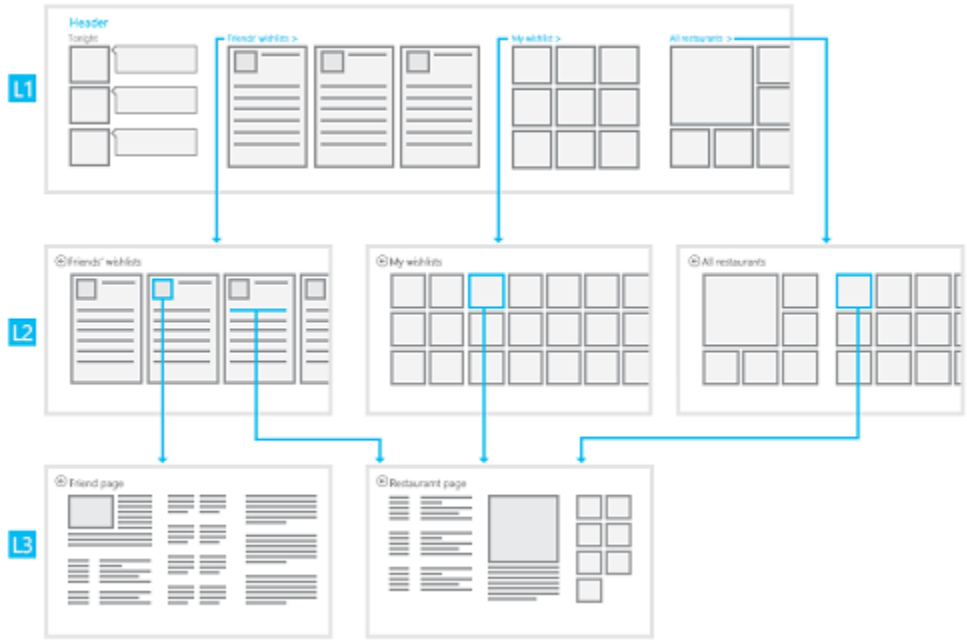
Title	Optional title to relate status, or a description of an icon that was used to invoke it. 
Main content	Include the information.

© 2014 Microsoft. All rights reserved. | May 13, 2014

	
Controls	Put optional buttons to do more with the information in the flyout.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for hub controls (Windows Store apps)



Description

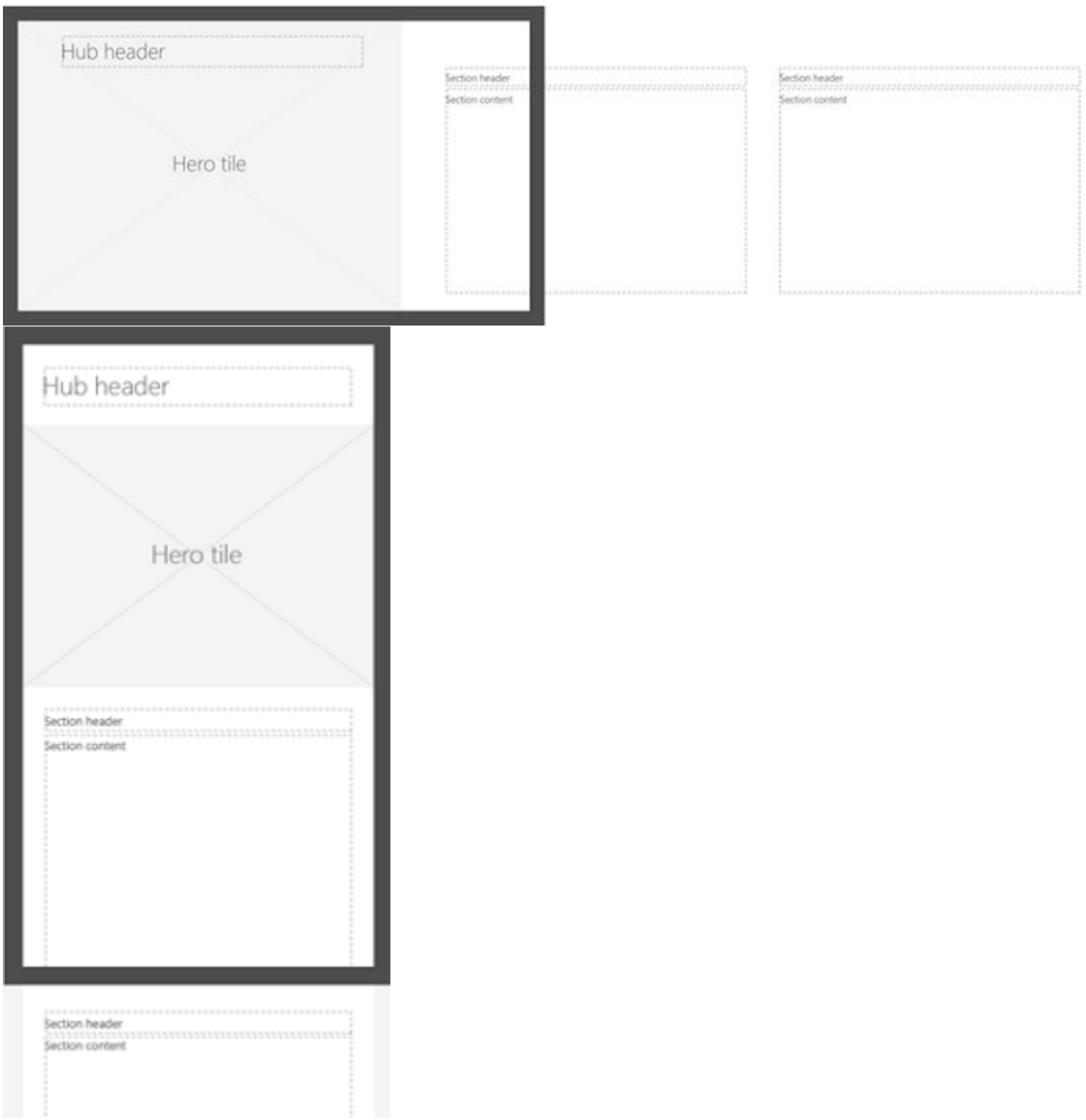
The hub control uses a hierarchical navigation pattern to support Windows Store apps that require a relational information architecture.

This navigation pattern is most useful when app content can be organized into distinct, yet related, sections or categories with varying levels of detail that should be traversed in a preferred sequence or order.

This control is a starting point for designing a visually stunning app. It gives you a framework and the flexibility to customize your app to fit your brand. The hub page should have visual variety and draw users into the app, from the landing page, to the section pages and detail pages.

You can use a hero image or content section for the first group.

© 2014 Microsoft. All rights reserved. | May 13, 2014



Use a large image for the hero that may be cropped both vertically and horizontally without losing the center of interest.

Here is an example of a single hero image and how that image may be cropped for landscape, portrait, and narrow width.

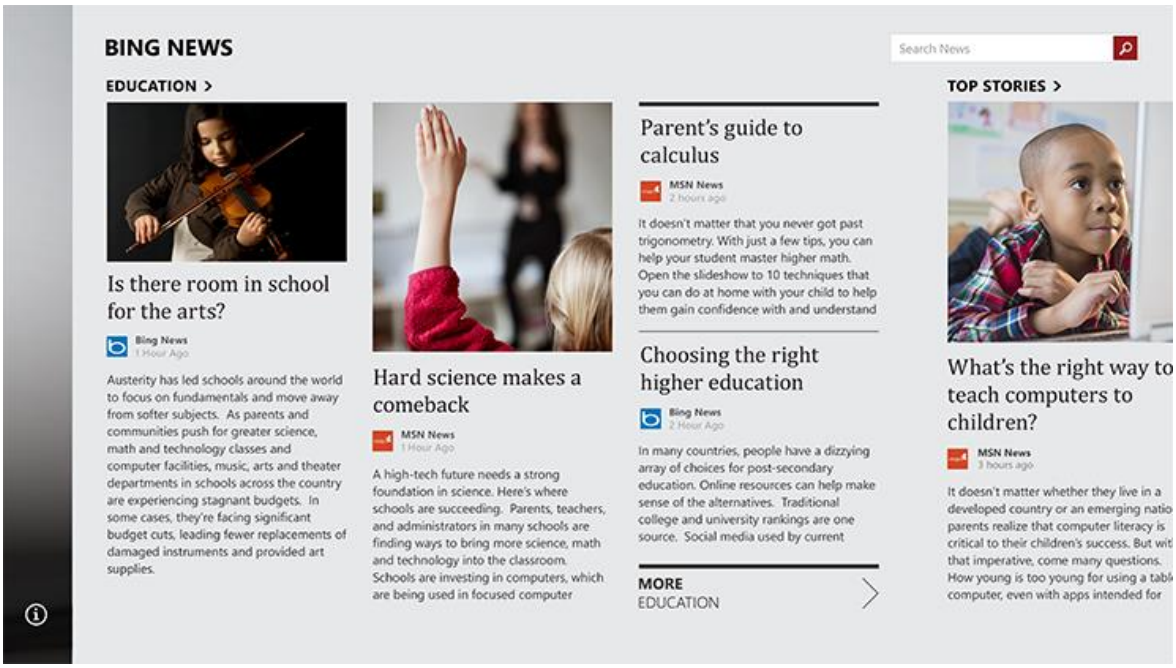
© 2014 Microsoft. All rights reserved. | May 13, 2014



Examples

The hub provides a great deal of design flexibility. This lets you design apps that have a wide variety of compelling and visually rich experiences. The following sections show a nature app that illustrates a range of design and style possibilities, through the use of the color, typography, images, graphics, and composition.

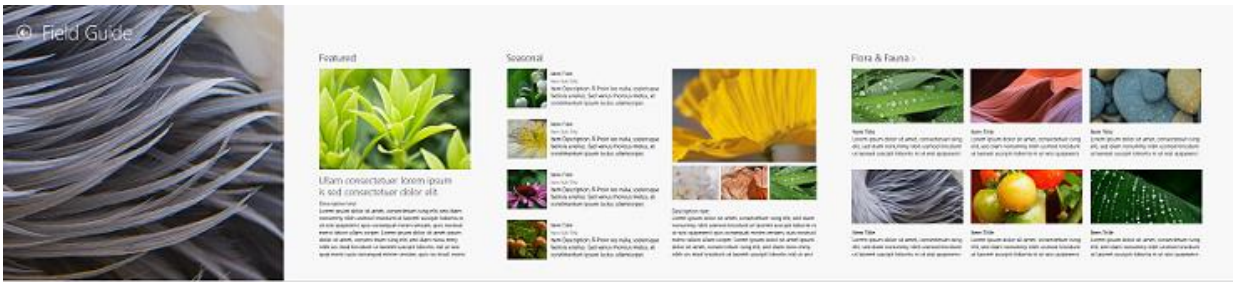
© 2014 Microsoft. All rights reserved. | May 13, 2014



The hub template

The hub template is the starting point. Here's an example of the nature app using the Microsoft Visual Studio hub template.

Landing page:



© 2014 Microsoft. All rights reserved. | May 13, 2014

Detail page:



The content is clean and organized but is not differentiated from all the other apps that also use the same template. You should use the template only as the base of your app, and build upon it to create a truly unique experience.

Variation 1 - Modular

This design is defined by a modular layout with a bold pop of color and graphic cropping of images.



© 2014 Microsoft. All rights reserved. | May 13, 2014

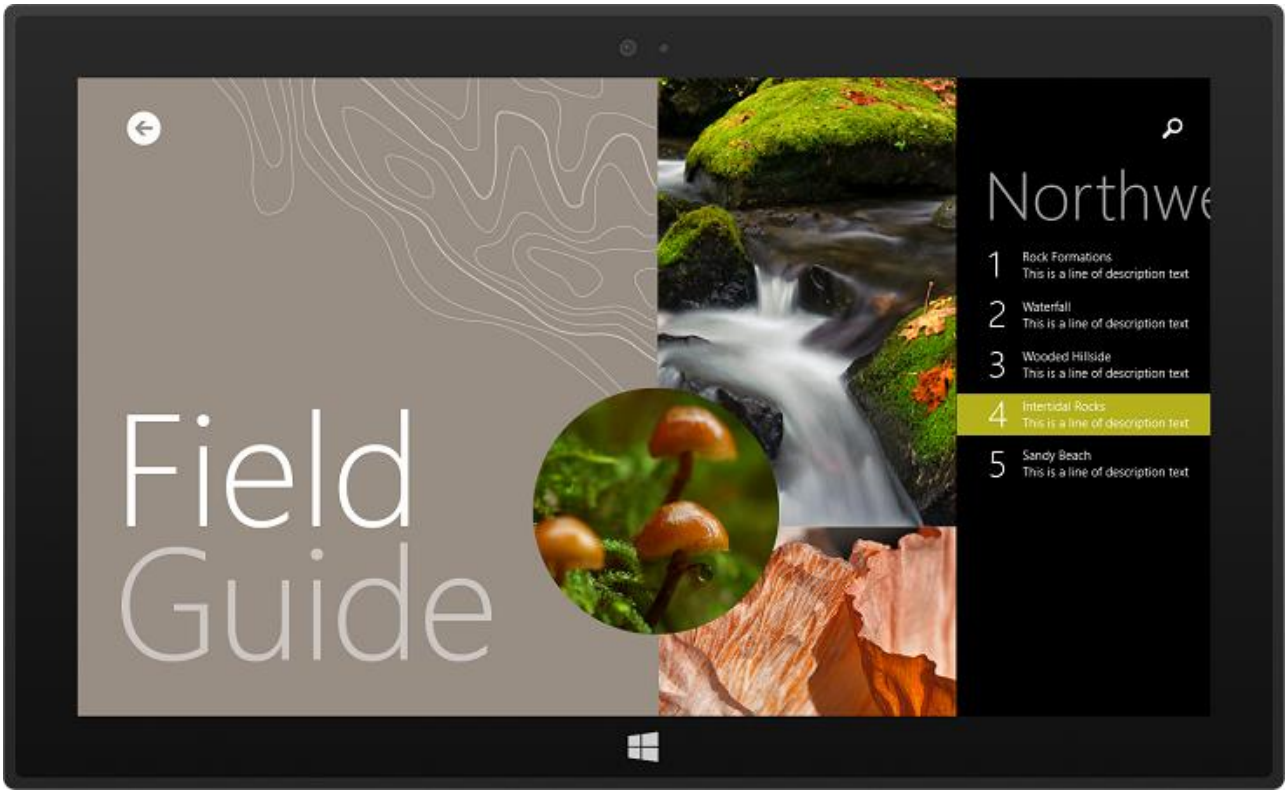
Landing page:


Featured
Seasonal
Flora & Fauna ›
Lorem ipsum dolor
Quisque in porta
Lorem ipsum dolor
Quisque in porta
Lorem ipsum dolor sit amet, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.
Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.
Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi.
Lorem ipsum dolor sit amet, consectetuer adipiscing elit

are tied together through a consistent title treatment and alignment to the underlying grid.

Variation 2 - Edge-to-edge vertical panes

This design juxtaposes an edge-to-edge composition of vertical panes with organic elements to create an earthy yet modern aesthetic.

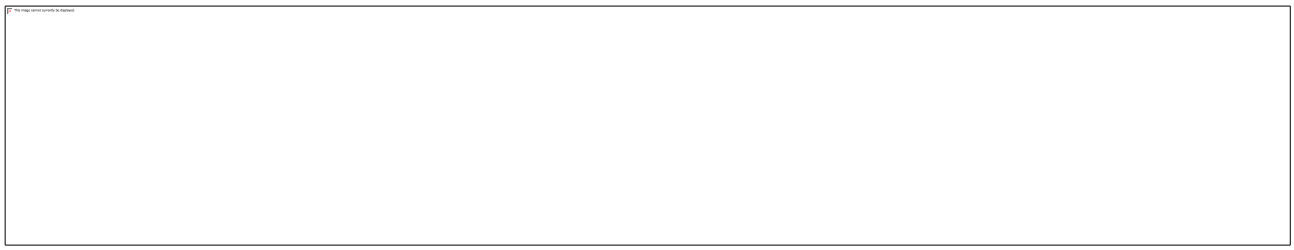


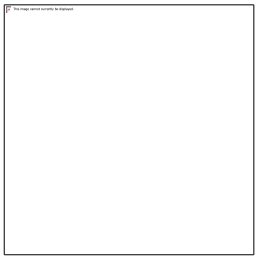
Landing page:



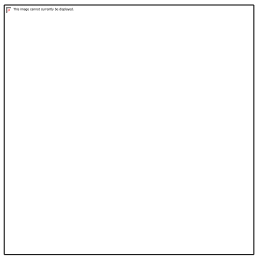
© 2014 Microsoft. All rights reserved. | May 13, 2014

Detail page:

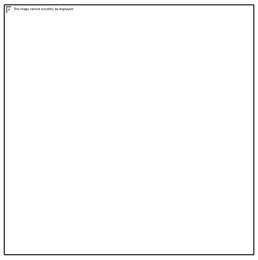




Color: An earthly color palette that consists of green, black, and brown complements the nature images.



Typography: This design uses Segoe UI in light and regular weights and in a variety of sizes. Scale and transparency of the type is used to provide graphic interest in the hero section, group titles, and points of interest on the map.



Graphics: A contour map graphic in the hub hero section reinforces the nature theme and provides visual contrast to the grid organization of the app.

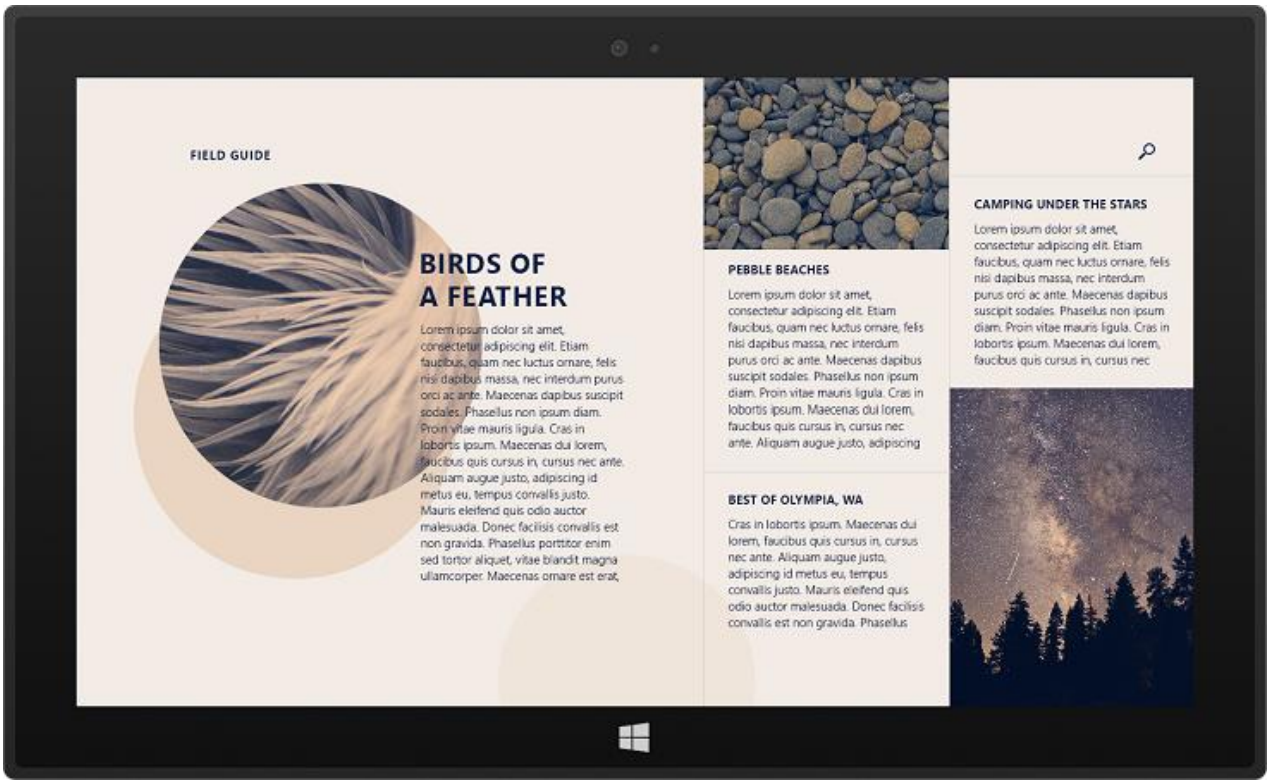


Composition: The app is composed of colored vertical panes of text and images. Elements such as circular accent images, group headers, and pull quotes bridge across the vertical panes to add interest to the composition and draw the eye to key content areas.

Variation 3 - Structured columns

This design employs a structured columnar layout with a soft color scheme and images to create a calming feel.

© 2014 Microsoft. All rights reserved. | May 13, 2014



Landing page:



Detail page:



© 2014 Microsoft. All rights reserved. | May 13, 2014

Color A soft color palette that consists of light neutrals creates a calming feel.



Imagery Neutral or sepia-toned images complement the soft color scheme. Similarly toned images create cohesion across the layout and reduce visual noise.



Graphics Circular, transparent graphics playfully break up the otherwise linear composition.



Composition Delicate 1px gray lines separate different groups within the hub. Small typography composed against liberal amounts of whitespace further reinforce the calm aesthetic of this app.

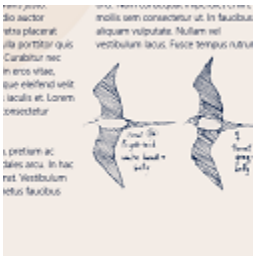


Illustration Drawings or interesting types of media break up large blocks of text. The sketch here gives texture and personalization to the article.

Variation 4 - Angular

This variation features an angular motif to frame the imagery and create background interest.

© 2014 Microsoft. All rights reserved. | May 13, 2014



Landing page:



Detail page:



© 2014 Microsoft. All rights reserved. | May 13, 2014



Color The app uses a light color palette with bold colors for the angular accent shapes.



Typography The design uses Segoe UI for the body text but contrasts it with Bebas Neue for the title and group header text. Using a different typeface is a quick and effective way to create a unique feel to your app.



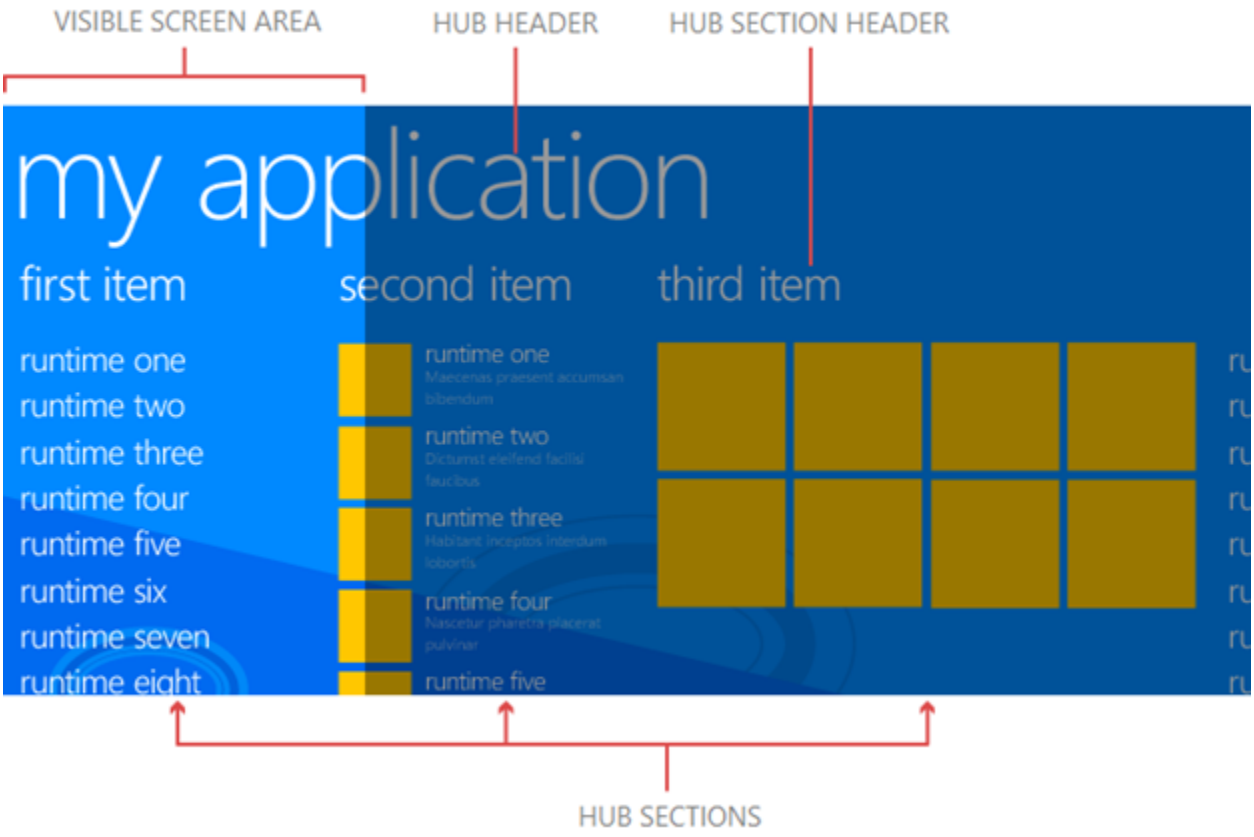
Graphics This app uses a standard hub composition of a hero image followed by separate groups containing lists. However, the addition of angular graphic elements both in the hero and as background images makes this design feel dynamic and distinct.

Dos and don'ts

- Use interactive headers and place the chevron glyph (>) at the end of the header text to indicate that there is more content. .
- Do not create a hub within a hub. Use interactive headers to navigate to another section or page instead.
- Dynamically reflow the content in groups to accommodate different window sizes.
- If you have a lot of sections, consider adding semantic zoom to the hub. This also makes it easier to find sections when the app is resized to a narrow width.
- Make a thoughtful choice of imagery for the main app content that works well with overlaid text.
- Use the hub template as a starting point and customize the layout to best reflect what your app is great at. You can customize the following aspects of the hub control:
    - Number of sections
    - Type of content in each section
    - Placement and order of sections
    - Size of sections
    - Spacing between sections
    - Spacing between a section and the top or bottom of the hub
    - Text style and size in headers and content
    - Color of the background, sections, section headers, and section content
- Avoid using vertically panning sections within a horizontally panning hub. Swipe selection and mouse scrolling will not work as expected.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Hub control design guidelines (Windows Phone Store apps)



Description

The hub control—on the phone, intended to be used in portrait orientation only—displays a series of sections that can be panned side to side. It's a full-screen container and navigation model for an app.

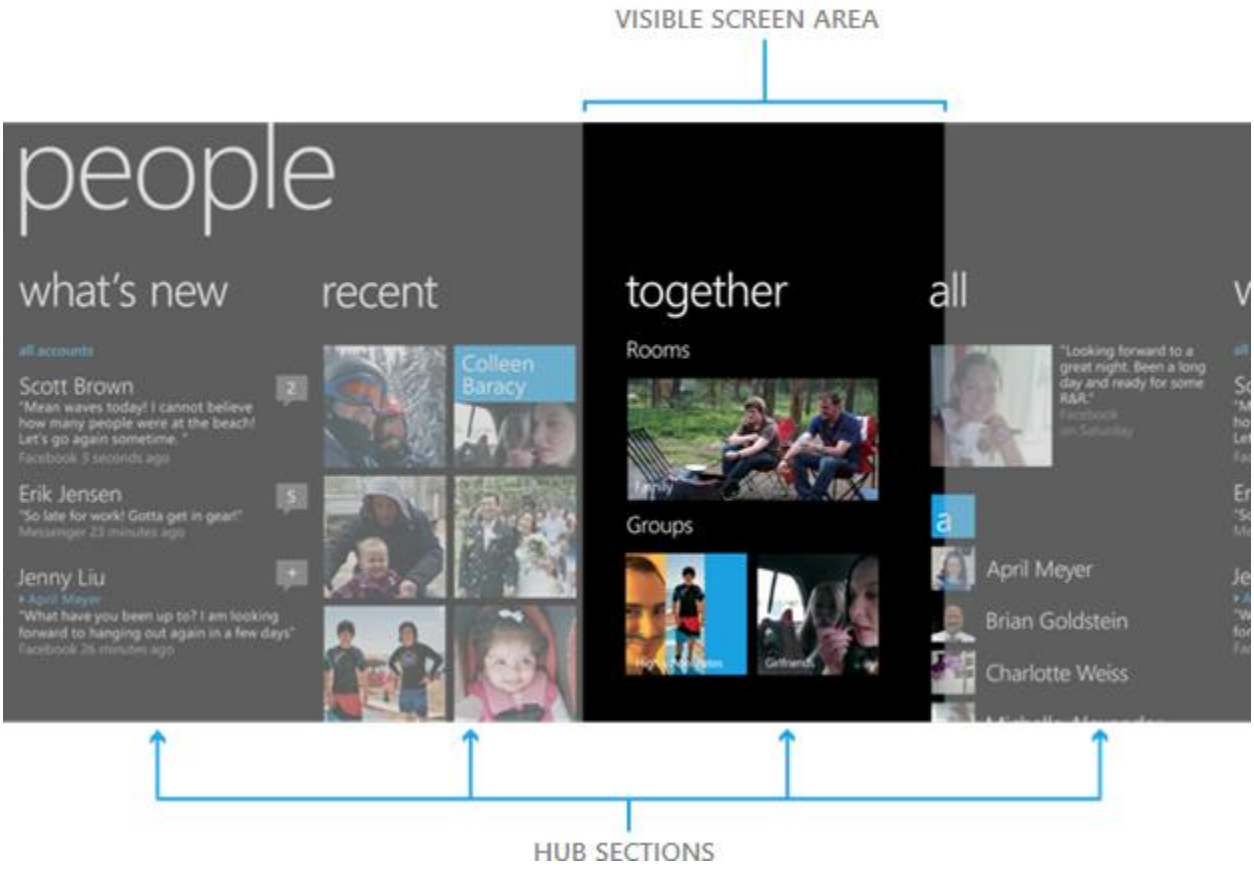
Hub (formerly *panoramic*) experiences are a part of the native Windows Phone look and feel. Unlike an app designed to fit within the confines of the phone screen, a hub app offers a unique way to view controls, data, and services by using a wide, virtual canvas which extends horizontally beyond

© 2014 Microsoft. All rights reserved. | May 13, 2014

the confines of the screen. On Windows Phone, these inherently dynamic views use layered animations and content so that layers smoothly pan at different speeds, similar to parallax effects.

Sections in a hub app serve as the starting point for more detailed experiences. Your goal should be to give users a visually rich content presentation.

Examples



The user interface consists of layers, which move independently: a background color or image, a hub header, hub section headers, and hub sections.

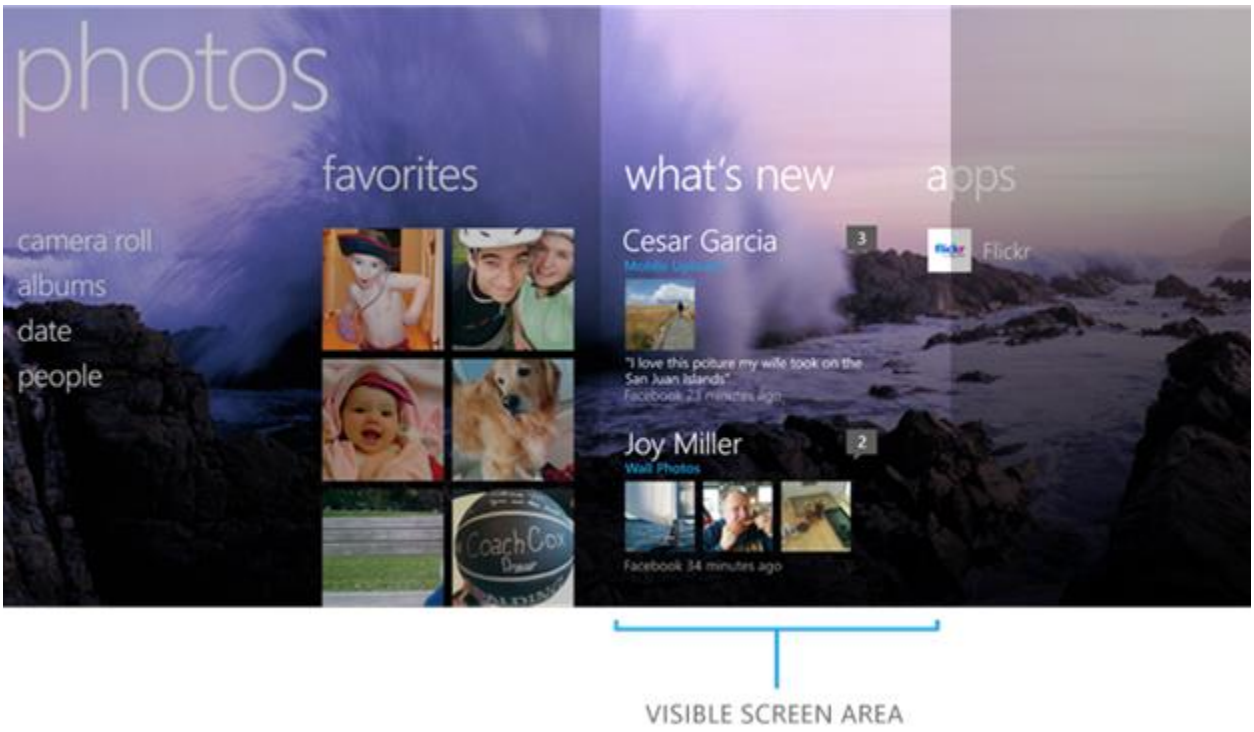
If set, a background image is the lowest layer and is meant to give the hub its rich, magazine-like feel. Usually a full-bleed image, the background image is potentially the most visual part of the app.

The hub header identifies the app and it should be visible no matter how the user enters the app.

Hub sections are the component of the hub app that contain other controls and content. Hub sections move at the same rate as the touch pan or flick. A hub section header is optional for any given hub section.

Thumbnails can be an important element, linking to content or media on other pages.

© 2014 Microsoft. All rights reserved. | May 13, 2014



Usage guidance

Based on the requirements of the app, you can add several hub sections to the hub control—each one offering a distinct functional purpose. For example, one section might contain a series of links and controls, while another is a repository for thumbnail images. A user can pan back and forth through these sections using the gesture support built into the hub control.

Design guidelines

- The hub control's behavior and rendering is supported in portrait orientation only.
- A hub control can be themed, and you can override the default colors, and so on.
- Use the wrap effect of the hub control for Windows Phone to wrap from the last section to the first section and vice versa. A hub that contains 1 or 2 sections will not wrap. A hub that contains 3 or more sections will wrap.
- On Windows Phone, if you use an app bar in your hub, set the app bar's mode to *minimized*. This mode is designed specifically to maximize screen space on a hub page.
- Show a progress bar in the System Tray, or a full-screen "loading" indicator while your hub control is launching.
- Show a progress bar in the System Tray when a section of a hub control is being updated or refreshed but isn't blocking user interaction.
- First visit: the first section shown should have the hub header correctly aligned on the left. There is no standard guidance for which section is the default; it depends on the content being presented.

© 2014 Microsoft. All rights reserved. | May 13, 2014

- For subsequent visits to the same hub control, take the user back to the section where they left off.
- Don't create more than five sections in a hub control. This is for performance reasons and also to limit the number of areas that the user has to navigate through. Use fewer sections when the contents are more complex. Don’t overwhelm the user with lots of sections. With just four or five sections, users can get their bearings as to where they are and what’s to the left and right.
- Don’t use a pivot control inside a hub control or vice versa. You can, however, link an item in a hub section to a different page containing a pivot control.
- Don’t use controls that can pan or scroll inside a hub control. For example, putting a map control inside a hub section can make it difficult to use the hub control. The gesture input gets confused. For example, if you have a slider and try to slide it left and you’re in a section of a hub control, it’s unclear whether you want to scroll the section or move the slider. The solution for a control that needs gesture input is to put it in its own page and navigate to that. You can place a gesture-disabled control in a hub section—perhaps a map. You could overlay a button that activates the map. Pressing the button would navigate to a different page containing just the map. The user could then press the Back button to go back to the hub section.

Hub headers:

- Use either plain text or images, such as a logo. You can also use multiple elements, such as a logo and text (or other UI elements).
- Ensure that the font or image color for the header makes it clearly visible across the entire background image (because the two move independently). Use system fonts and styles unless there’s a need for a specific branded experience that uses a different font, size, or color.
- Avoid animating the header, or dynamically changing its size.
- For the sake of consistency, make the hub header match the launch Tile in Start.
- When laying out your hub control, and designing the headers, avoid occlusion problems with the System Tray or other elements.

Hub section headers:

- Use either plain text or images, such as a logo. You can also use multiple elements, such as a logo and text (or other UI elements).
- Ensure that the font or image color for the header makes it clearly visible across the entire background image (because the two move independently). Use system fonts and styles unless there’s a need for a specific branded experience that uses a different font, size, or color.
- Avoid animating the header, or dynamically changing its size.

Hub sections:

- Keep the beauty of the hub control experience intact by being selective about the text and images included in section content so that the hub doesn’t become overwhelming and busy.
- Take into account orientation if you use vertical scrolling. Vertical scrolling in a hub section is acceptable as long as the section’s width is greater than the screen width.

© 2014 Microsoft. All rights reserved. | May 13, 2014

- Consider hiding hub sections until they have content to display.

Background art

- A dark, soft, and low-contrast background is best. Single-color, or gradient.
- A subtle, unobtrusive photo in the background can make a hub control look visually compelling. Photos with lots of bright colors are to be avoided, though, because they can make the sections hard to read. One practical technique is to use a semi-transparent black or white filter (a rectangle) on top of the photo. You can do that in a bitmap editing tool, and then flatten the image.
- A background image should span the entire hub control. That means its aspect ratio should match that of the hub control, and its size should consider the most common device resolution, the largest device resolution, and performance. The JPEG format is recommended to keep file sizes small.
- You can switch from one background image to another, even while your app is running, but only one image can be displayed at a time.

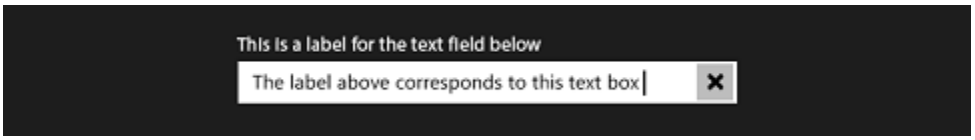
Thumbnails

- Use cropped images that highlight an identifiable subject rather than scaling down an entire image. If the image isn't identifiable without text, up to two lines of text can be used to identify the content.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Label (or text block)





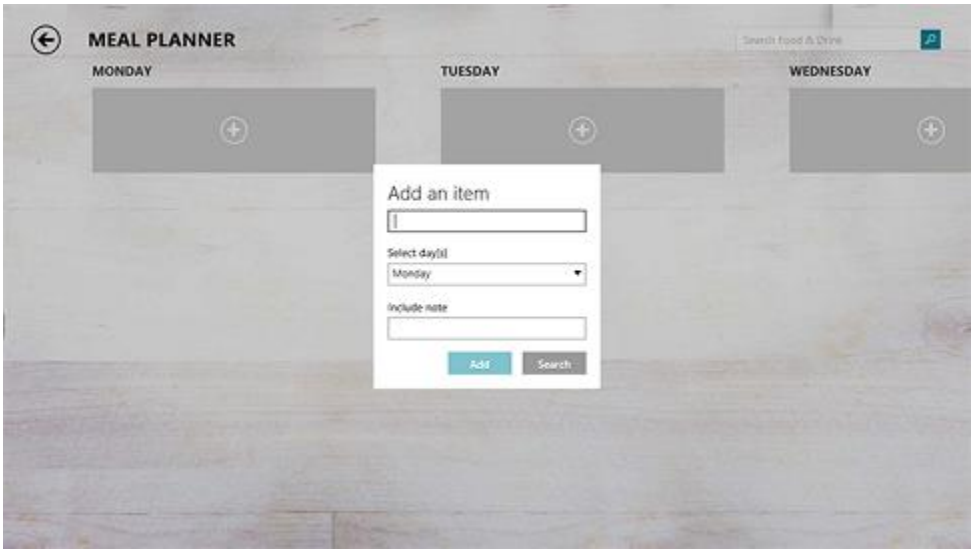
Description

A label is the name or title of a control or a group of related controls.

In XAML, many controls have a built-in Header property that you use to display the label. For controls that don't have a Header property, or to label groups of controls, you can use a **TextBlock** instead.

In HTML, you use the **label element**.

Example



Dos and Don'ts

- Use a label to indicate to the user what they should enter into an adjacent control, if it's not already obvious. For example, 'Name' above a text input box. You can also label a group of related controls, or display instructional text near a group of related controls.

© 2014 Microsoft. All rights reserved. | May 13, 2014

- If you're labeling controls, write the label as a noun or a concise noun phrase, not as a sentence, and not as instructional text. Don't use colons or other punctuation.
- You can be more liberal with the length of instructional text, and you can use punctuation.

XAML style gallery for Windows Store apps

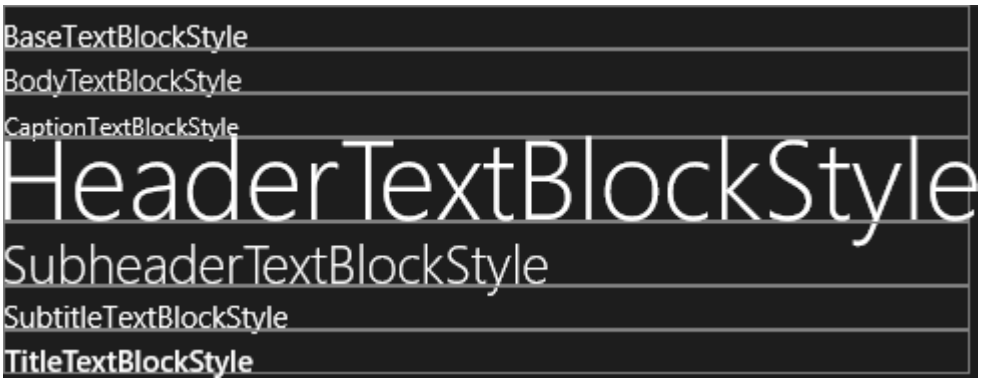


To help you style your app like a built-in app, you can use this XAML snippet in a Windows Store app to make yourself a TextBlock style gallery to refer to.

XAML

```
<StackPanel>
    <Border BorderBrush="Gray" BorderThickness="1">
        <TextBlock Text="BaseTextBlockStyle" Style="{StaticResource
BaseTextBlockStyle}"/>
    </Border>
    <Border BorderBrush="Gray" BorderThickness="1">
        <TextBlock Text="BodyTextBlockStyle" Style="{StaticResource
BodyTextBlockStyle}"/>
    </Border>
    <Border BorderBrush="Gray" BorderThickness="1">
        <TextBlock Text="CaptionTextBlockStyle" Style="{StaticResource
CaptionTextBlockStyle}"/>
    </Border>
    <Border BorderBrush="Gray" BorderThickness="1">
        <TextBlock Text="HeaderTextBlockStyle" Style="{StaticResource
HeaderTextBlockStyle}"/>
    </Border>
    <Border BorderBrush="Gray" BorderThickness="1">
        <TextBlock Text="SubheaderTextBlockStyle" Style="{StaticResource
SubheaderTextBlockStyle}"/>
    </Border>
    <Border BorderBrush="Gray" BorderThickness="1">
        <TextBlock Text="SubtitleTextBlockStyle" Style="{StaticResource
SubtitleTextBlockStyle}"/>
    </Border>
    <Border BorderBrush="Gray" BorderThickness="1">
        <TextBlock Text="TitleTextBlockStyle" Style="{StaticResource
TitleTextBlockStyle}"/>
    </Border>
</StackPanel>
```

© 2014 Microsoft. All rights reserved. | May 13, 2014



XAML style gallery for Windows Phone Store apps

To help you style your app like a built-in app, you can use this XAML snippet in a Windows Phone Store app to make yourself a TextBlock style gallery to refer to.

XAML

```
<StackPanel>
    <Border BorderBrush="Gray" BorderThickness="1">
        <TextBlock Text="BaseTextBlockStyle" Style="{StaticResource
BaseTextBlockStyle}"/>
    </Border>
    <Border BorderBrush="Gray" BorderThickness="1">
        <TextBlock Text="BodyTextBlockStyle" Style="{StaticResource
BodyTextBlockStyle}"/>
    </Border>
    <Border BorderBrush="Gray" BorderThickness="1">
        <TextBlock Text="ComboBoxPlaceholderTextBlockStyle" Style="{StaticResource
ComboBoxPlaceholderTextBlockStyle}"/>
    </Border>
    <Border BorderBrush="Gray" BorderThickness="1">
        <TextBlock Text="ControlContextualInfoTextBlockStyle" Style="{StaticResource
ControlContextualInfoTextBlockStyle}"/>
    </Border>
    <Border BorderBrush="Gray" BorderThickness="1">
        <TextBlock Text="ControlHeaderTextBlockStyle" Style="{StaticResource
ControlHeaderTextBlockStyle}"/>
    </Border>
    <Border BorderBrush="Gray" BorderThickness="1">
        <TextBlock Text="FlyoutPickerTitleTextBlockStyle" Style="{StaticResource
FlyoutPickerTitleTextBlockStyle}"/>
    </Border>
    <Border BorderBrush="Gray" BorderThickness="1">
        <TextBlock Text="GroupHeaderTextBlockStyle" Style="{StaticResource
GroupHeaderTextBlockStyle}"/>
    </Border>
    <Border BorderBrush="Gray" BorderThickness="1">
        <TextBlock Text="HeaderTextBlockStyle" Style="{StaticResource
HeaderTextBlockStyle}"/>
    </Border>
```

© 2014 Microsoft. All rights reserved. | May 13, 2014

```
    <Border BorderBrush="Gray" BorderThickness="1">
        <TextBlock Text="ListViewEmptyStaticTextBlockStyle" Style="{StaticResource
ListViewEmptyStaticTextBlockStyle}"/>
    </Border>
    <Border BorderBrush="Gray" BorderThickness="1">
        <TextBlock Text="ListViewItemContentTextBlockStyle" Style="{StaticResource
ListViewItemContentTextBlockStyle}"/>
    </Border>
    <Border BorderBrush="Gray" BorderThickness="1">
        <TextBlock Text="ListViewItemSubheaderTextBlockStyle" Style="{StaticResource
ListViewItemSubheaderTextBlockStyle}"/>
    </Border>
    <Border BorderBrush="Gray" BorderThickness="1">
        <TextBlock Text="ListViewItemTextBlockStyle" Style="{StaticResource
ListViewItemTextBlockStyle}"/>
    </Border>
    <Border BorderBrush="Gray" BorderThickness="1">
        <TextBlock Text="MessageDialogContentStyle" Style="{StaticResource
MessageDialogContentStyle}"/>
    </Border>
    <Border BorderBrush="Gray" BorderThickness="1">
        <TextBlock Text="MessageDialogTitleStyle" Style="{StaticResource
MessageDialogTitleStyle}"/>
    </Border>
    <Border BorderBrush="Gray" BorderThickness="1">
        <TextBlock Text="SubheaderTextBlockStyle" Style="{StaticResource
SubheaderTextBlockStyle}"/>
    </Border>
    <Border BorderBrush="Gray" BorderThickness="1">
        <TextBlock Text="TitleTextBlockStyle" Style="{StaticResource
TitleTextBlockStyle}"/>
    </Border>
</StackPanel>
```

© 2014 Microsoft. All rights reserved. | May 13, 2014



© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for links

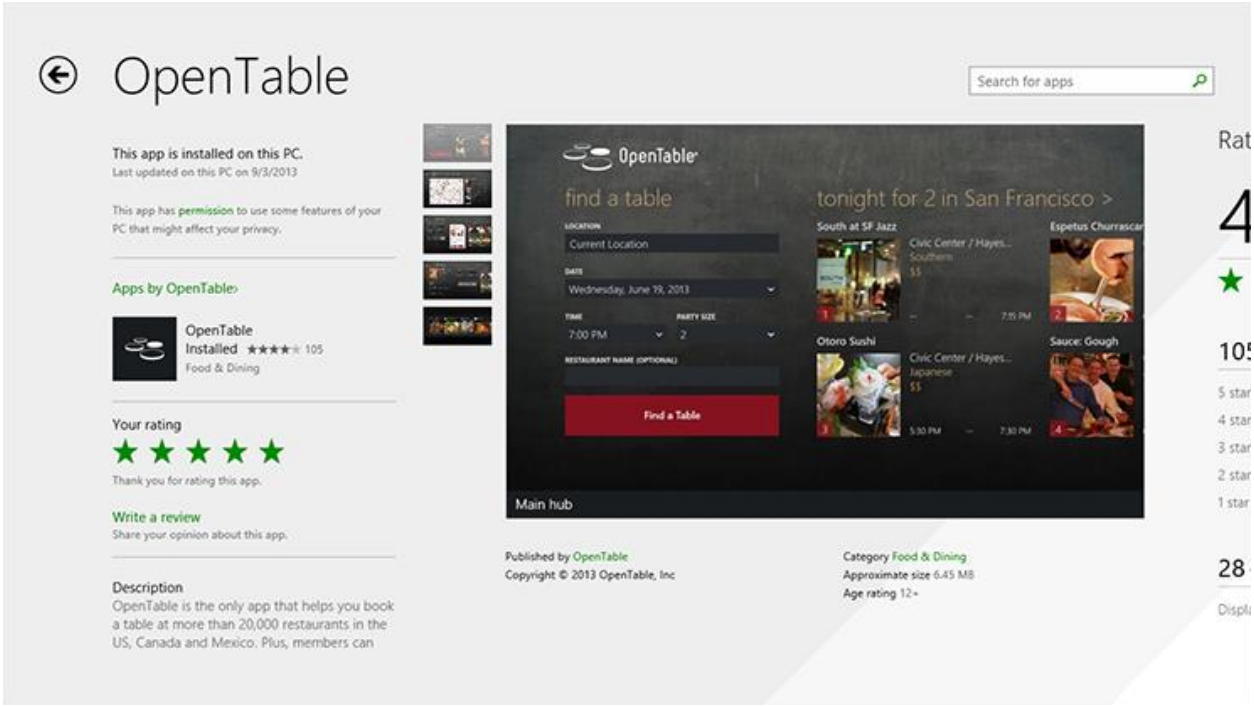


A hyperlink lets you give users a visual hint that certain text links to other content. Read more on the Dev Center. Text in a hyperlink element is treated like the rest of the text, so it participates in line-breaking.

Description

A hyperlink can take the form of either inline text (XAML and HTML) or a hyperlink button (XAML). In either form, a hyperlink is a piece of text that a user can tap to open a web page in a browser, or to navigate to another page—or to another section within the same page—in the current app.

Example



Is this the right control?

Use a hyperlink when hypertext is the right interaction pattern—that is, text that responds when tapped and navigates the user to more information about, or related to, the text that was tapped.

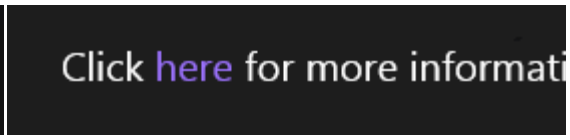
© 2014 Microsoft. All rights reserved. | May 13, 2014

The navigation destination is encoded in a Uniform Resource Identifier (URI) in the **NavigateUri** property (XAML) or the **href** property (HTML).

A hyperlink that appears inline in dynamic text is best accomplished using a **Hyperlink** (XAML) or an **a element | a object** (HTML) so that you'll benefit from automatic line-breaking within the hyperlink. Use a **HyperlinkButton** (XAML) when you don't need line-breaking or if you need to customize the visuals of the hyperlink.

By default, a hyperlink looks and operates like a conventional web hyperlink: text displayed in a distinct color, or an image, that can be tapped to trigger the navigation. But you can customize the visuals that make up a **HyperlinkButton** (XAML). As the user interacts with it, a hyperlink gives feedback by changing its state and therefore its appearance. Normal, pressed, and disabled are examples of hyperlink states.

As shown below, a button embedded into a sentence is a good opportunity for a rethink. If the button navigates the user to more information about, or related to, the text, then consider a hyperlink instead. If the button triggers another action then consider laying out the button in a different place, and reworking the text and the button's content.

Incorrect	**Correct**
	
In this example, a button is not the correct control to use to display additional content to the user, because it takes up too much space and looks out of place.	In this example, a hyperlink works because it takes the same form as the rest of the text within the sentence.

Dos and don'ts

- Keep individual hyperlinks far enough apart from one another that the user can tap them accurately.
- Only use the disabled state for a hyperlink if that state is temporary—such as other system processes are occurring—or if the hyperlink can be enabled by user action.
- Use the default font unless your brand guidelines tell you to use another.
- Put a tooltip on every link. That way, if the link is occluded by the user's finger, the user can still see what it will do.
- When navigating to an external site, put the domain name inside the tooltip and style it with a secondary font color. Adding the domain name to the tooltip lets users know that they're about to navigate to an external site so they won't be surprised when they click the link. It's enough to just show the top-level domain.
- When the user doesn't care whether he or she has visited a link, style the visited state for that link so that the link looks the same regardless of whether the user has visited that page. The default style for a visited link makes it look different than a link that hasn't been visited.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Sometimes the user doesn't care whether a link has been visited. This is usually the case for links that are a part of your app's main navigation.
- Keep the link text concise. If you want to provide additional information, put it inside the link's tooltip.
- Don’t use links to perform actions other than navigating.

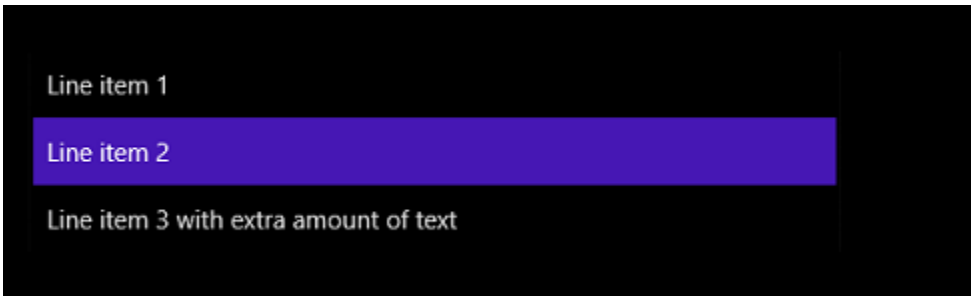
© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for list boxes (or select)





Windows app: enabled and disabled list boxes



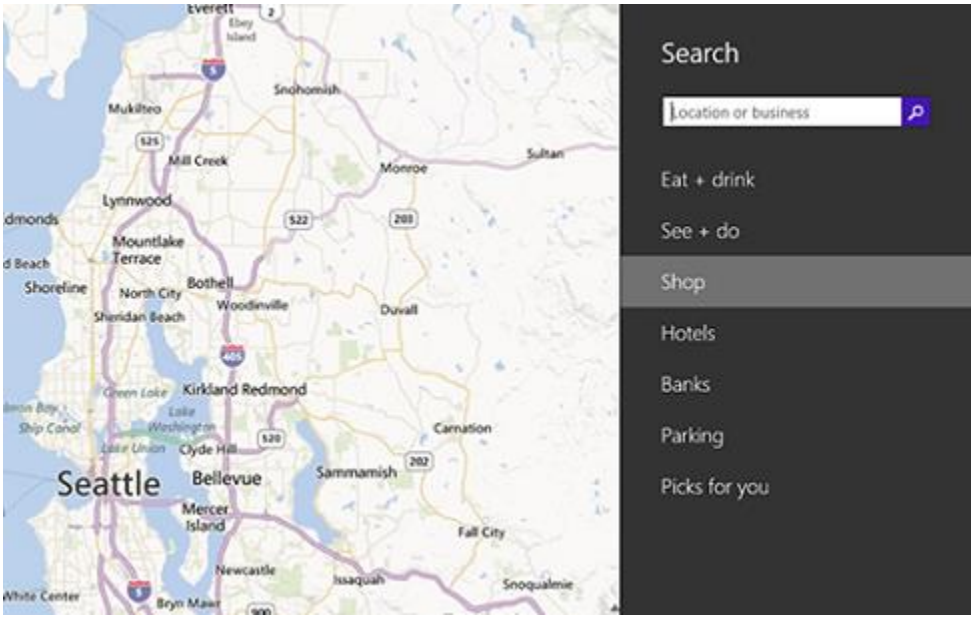
Windows Phone app: list box

Description

A list box (also known as a select) gives the user a way to choose usually one item, but sometimes several, from a list of items. Items in a list box can be scrolled if there isn't space to show all of them.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Example



Is this the right control?

Use a list box when the items are important enough to display prominently. A list box should draw the user's attention to the full set of alternatives in an important choice. If you can, set the size of a list box so that its list of items doesn't need to be panned or scrolled. Between 3 and 9 items is the sweet spot for a list box. And a list box works well when its items can vary dynamically.

Factors that indicate against using a list box include:

- There are a very small number of items. A single-select list box that always has the same 2 options might be better presented as radio buttons. Even consider radio buttons when there are 3, or even 4, static items.
- The list box is single-select and it always has the same 2 options where one can be implied as 'not' the other—such as "on" and "off." Use a single check box or a toggle switch.
- There are a very large number of items. A better choice for long lists are grid view and list view. For very long lists of grouped data, semantic zoom is preferred.
- The items are contiguous numeric values. Consider a slider.
- The selection items are of secondary importance in the flow of your app or the default option is recommended for most users in most situations. Use a drop-down list instead.

Dos and don'ts

- Verify that the purpose of the list box, and which items are currently selected, is clear.
- Reserve visual effects and animations for touch feedback, and for the selected state of items.

© 2014 Microsoft. All rights reserved. | May 13, 2014

- Limit the list box item's text content to a single line. If the items are visuals, you can customize the size. If an item contains multiple lines of text or images, use a grid view or list view instead.
- If the list box is auto-sized and its items are dynamic, consider how the list box will resize and what will happen to visuals around it. A fixed-sized list box with dynamic items will not resize, but will allow scrolling.
- Use the default font unless your brand guidelines tell you to use another.
- Sort items in a list box in a logical order, such as grouping related options together, placing most common options first, or using alphabetical order. Sort names in alphabetical order, numbers in numeric order, and dates in chronological order.
- Add padding of 27 pixels on the right side of your content to keep the scrollbars from overlapping the content.
- Don't use a list box to perform commands or to dynamically show or hide other controls.

Additional usage guidance

Appearance and interaction

A list box is always open (in contrast to a drop-down list). Its items can be text strings, or numeric values, or they can be customized to be any other visuals. You're encouraged to be creative. Users are comfortable interacting directly with the content of an app. So you may choose to show the actual content on offer, perhaps drawings or photos of products. Just remember that an item should always give visual feedback when it's tapped, and it should be clear when an item is selected.

A list box control presents its list of items vertically by default. If you want to present items horizontally—particularly if the items are graphics or photos—then it is possible to customize the layout of list box items. If you want items laid out in a horizontal or vertical wrap grid then you can do that, too. Or just use a grid view which lays out items that way by default.

As the user interacts with them, items give feedback by changing state and therefore appearance. Normal, pressed, unselected, selected, and disabled are examples of list box item states.

Tapping an item selects it and, in multi-select mode, tapping a selected item deselects it. In single-select mode, tapping another item transfers selection to that item. A vertical swipe of a finger causes the list of items to scroll up or down with inertia. A list box has a scroll bar whose position indicates the user's relative location within the list of items, and whose size indicates the proportion of items in view. The scroll bar is only visible while scrolling.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for list and grid view controls



Description

A grid view or list view is a collection of content within a Windows Store app. Grid and list views provide a consistent item viewing experience for users and are optimized for touch.

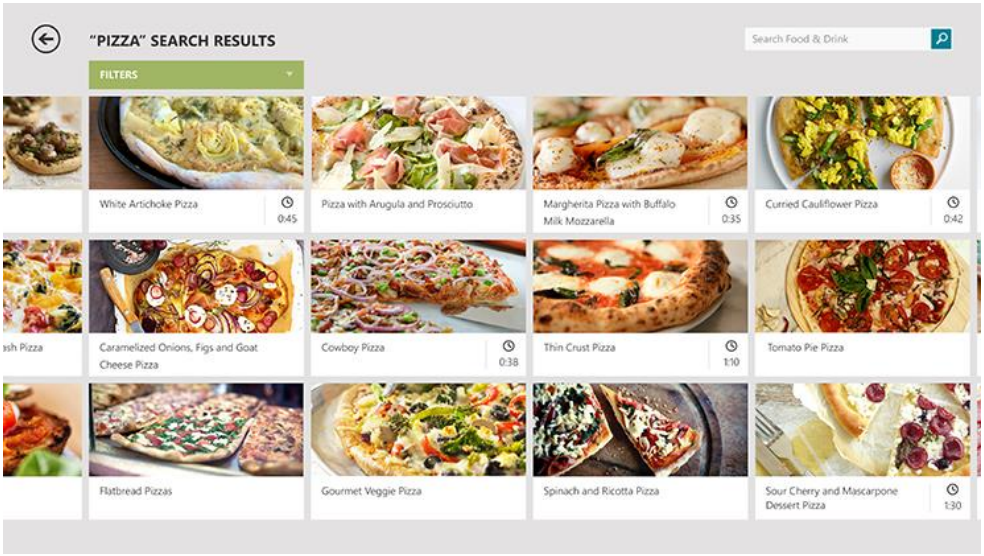
These controls come in multiple formats - ungrouped grid, grouped grid, list (HTML), ungrouped list (XAML), and grouped list (XAML). Lists pan vertically. Grids pan horizontally. The reading order for lists is from top to bottom. Grids read from top to bottom, left to right.

Note: The Windows Library for JavaScript (WinJS) contains a single **ListView** control with list and grid layout options. Apps using C#/VB/C++ and XAML create list and grid layouts with two separate controls, **ListView** and **GridView**. These guidelines apply to all of these control types.

Example

Here is a list of search results displayed in a standard grid view:

© 2014 Microsoft. All rights reserved. | May 13, 2014



Is this the right control?

Use a list or grid view control to present a collection of data as a series of items to users. For example, you can use a list view to present a list of emails, items in a shopping cart, a list of images, and search results.

Use this control when you want to:

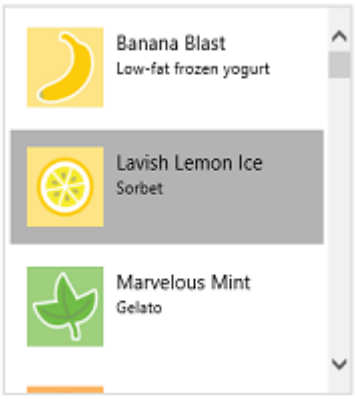
- Present data in groups
- Let users select one or more items
- Customize how your data is loaded
- Provide better keyboarding and selection
- Dynamically edit your list of items and get built-in animations
- Customize your items with templates or give your items their own events
- Display items of different sizes

List or grid?

WinJS provides two basic layouts for **ListView**: a list layout and a grid layout. (You can also create your own custom layouts.) If you use C#/VB/C++ and XAML, you use two separate controls called **ListView** and **GridView** to achieve the same functionality.

List: The list layout displays as a single-column, reads from top to bottom, and pans or scrolls vertically. This layout doesn't display group info, such as group headers or group boundaries.

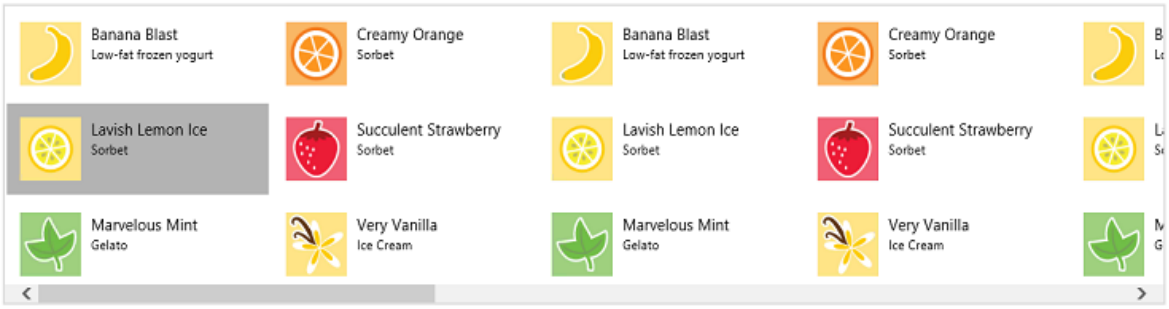
© 2014 Microsoft. All rights reserved. | May 13, 2014



Use a list when:

- Displaying content in a narrow window or in portrait orientation.
- Trying to conserve space; list layouts take up less room than grid layouts.
- Grouping items isn't necessary.
- Creating the master pane in master/detail view. Master/detail view is the format often used in mail apps where one side of the screen shows a list of selectable items and the other side of the screen provides a detailed view of the selected item. See the following section, **Using the right interaction pattern**, for an example of this view.

Grid: The grid layout (or **GridView** in apps using XAML) always pans horizontally, and items are laid out following a top-to-bottom, then left-to-right reading order.



Use a grid when:

- Displaying content libraries. See using the right interaction pattern for an example of this view.
- Your app needs to group items. Grouping helps organize content when the collection of items is large or naturally lends itself to grouping (like a collection of songs organized by album).
- Formatting the two content views associated with semantic zoom.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Using the right interaction pattern

List or grid view controls are typically used in four different ways: to display a content library, to display master/detail data, as a picker, or to display static data. When you use these controls in one of these ways, users expect it follow these interaction patterns:

- **Content library pattern**

  Use this pattern when displaying a collection, or library, of content. It is often used when presenting media such as pictures and videos.

  In a content library, users expect to be able to tap an item to invoke an action. Most content libraries also support the selection of items using the cross-slide gesture. Cross-slide lets your user swipe perpendicular to the panning direction to select an item. For example, if you enable cross-slide in your horizontally panning content library, a user will be able to select an item by dragging it vertically.

  If you're using C#/VB/C++ and XAML, add a **GridView** control formatted with one of these item templates for grid layouts to quickly implement this kind of display.

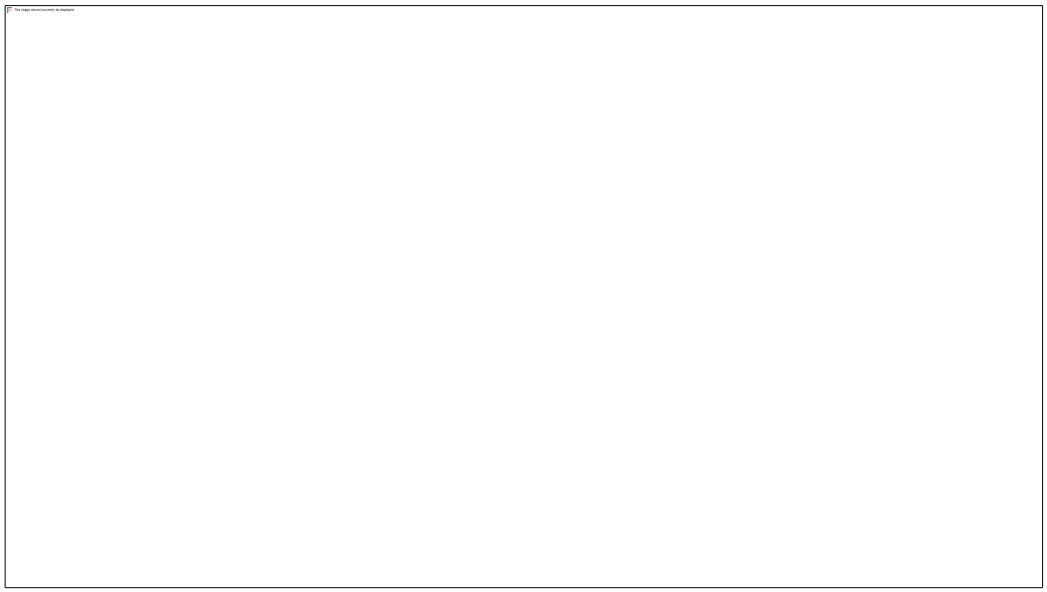
  If you're using JavaScript and HTML, you can make your **ListView** follow the content library pattern by configuring it like this:

  - **selectionMode**: Set to "single" or "multi".
  - **tapBehavior**: Set to "invokeOnly".
  - **swipeBehavior**: Set to "select".

- **Master/detail pattern (also known as split view pattern)**

  When using the master/detail pattern, you can use list view to organize the master pane. The master pane shows a list of selectable items. When a user selects an item in the master pane, additional info about the selected item is displayed in the details pane. List view is only useful for formatting a master pane display; use another control for the details pane.

© 2014 Microsoft. All rights reserved. | May 13, 2014



The XAML image and text list (inbox) template styles a **ListView** to display content in the master pane pattern.

Using HTML? To make your **ListView** behave like the master pane in the master/detail pattern, configure it like this:

- **selectionMode**: "single" or "multi". Use "single" for single-select.
- **tapBehavior**: "directSelect"
- **swipeBehavior**: "none" for single, "select" for multi.

You can chain several list view controls together to create complex master/detail hierarchies.

- **Picker pattern**

  Use the picker pattern when the primary user action is selection and the ability to invoke an item on tap is not important. In this interaction pattern, tap is used for selection in addition to cross-slide.

© 2014 Microsoft. All rights reserved. | May 13, 2014



- **Static pattern**

  Use this pattern when you want to use the list view solely as a way to present content, and most types of interactivity are disabled. This pattern is useful for collections of items that are read-only and can't be activated or navigated into.

See the code associated with each of these interaction patterns in the HTML ListView customizing interactivity sample or the XAML ListView and GridView customizing interactivity sample.

Dos and don'ts

- Items in the same list or grid view should have the same behavior. For example, if one item in a list view performs an action when the user taps it, all items in the list view should perform an action when the user taps them.
- If your list is divided into groups, use semantic zoom. Semantic zoom makes it easier for users to navigate through grouped content.
  - Use the grid layout in the WinJS **ListView** control or the XAML **GridView** control to organize the zoomed-in view. This view displays group headers and individual items.

© 2014 Microsoft. All rights reserved. | May 13, 2014



- Also use a grid layout for the zoomed out view. Note that this view only displays group headers.



- When using interactive group headers in grid view, register for the Ctrl + Alt + G key press and use it to navigate the user into the group for the currently focused item.
- Don't display selection checkmarks when users can only select a single item (when selectionMode is set to "single"). See the HTML ListView customizing interactivity sample or XAML ListView and GridView customizing interactivity sample for examples.
- Don't use a list or grid views as general purpose layout controls. To create a grid or table-like layout, use a **Hub** control, **Grid** control, or a CSS layout, such as grid layout or flexible box ("flexbox") layout.
- Don’t use a list or grid view control to create command toolbars, such as a toolbar of buttons for cut, copy, and paste commands. Instead, use several button controls laid out side-by-side.

Guidelines specific to WinJS ListView control

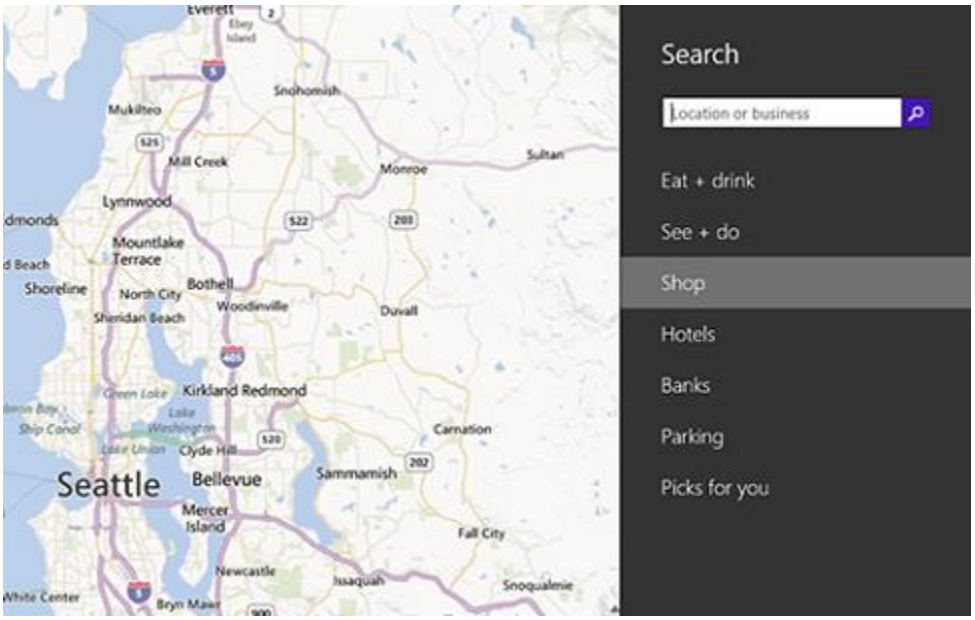
- If your **ListView** uses a grid layout, switch to a list layout when your app is resized to a narrow width or switch to portrait orientation.

© 2014 Microsoft. All rights reserved. | May 13, 2014

- If your **ListView** uses a grid layout and contains groups, display the groups when you switch to a list layout. When the user taps a group, update the **ListView** to display the items in the group and provide a back button that lets the user go back to the group display. You can use the **SemanticZoom** control to provide this functionality.
- Change how the items in the **ListView** are displayed when your app is resized. You can use Media Queries to update your CSS automatically or you can switch to a different **itemTemplate** when the app enters the resized view. If you update your CSS styling, you must call the **ListView** control's **forceLayout** method for it to display the changes correctly.
- Store the **ListView** control's **indexOfFirstVisible** value before the app is resized, and then restore it when the app returns to full screen so that the user doesn't lose their position in the list.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for Maps



Windows app: a map control

Description

The map control can display road maps and aerial views, directions, search results, and traffic.

Dos and don’ts

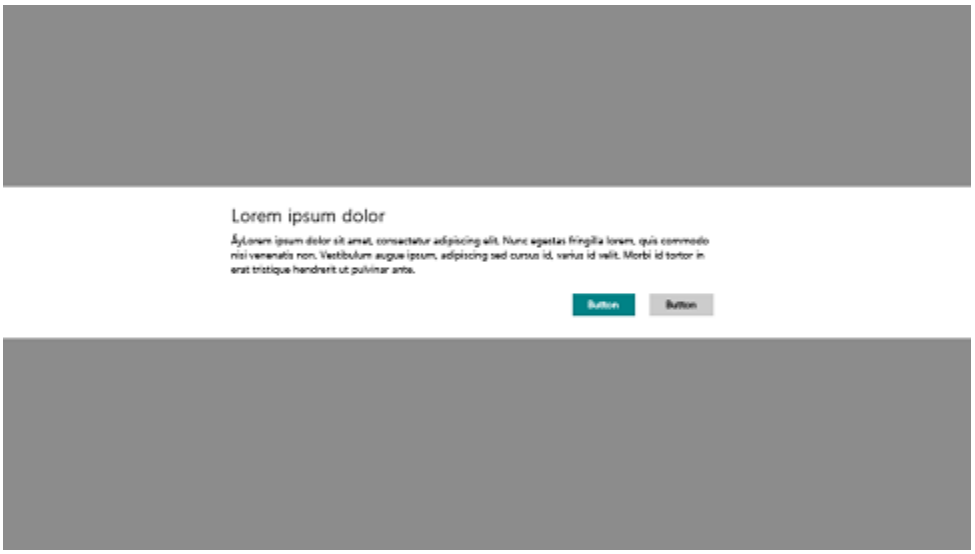
- Use ample screen space or, ideally, the entire screen space to display a map so that users don’t have to pan and zoom excessively to view geographical info; if the control is only used to present a static, informational view, then it’s acceptable to use a smaller map.

Additional usage guidance

Use a map within your app to let users view app-specific or general geographic info without having to leave your app. On a map, you can display the user’s location, directions, and points of interest. A map can also show aerial views, traffic, and local search results.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for message dialogs

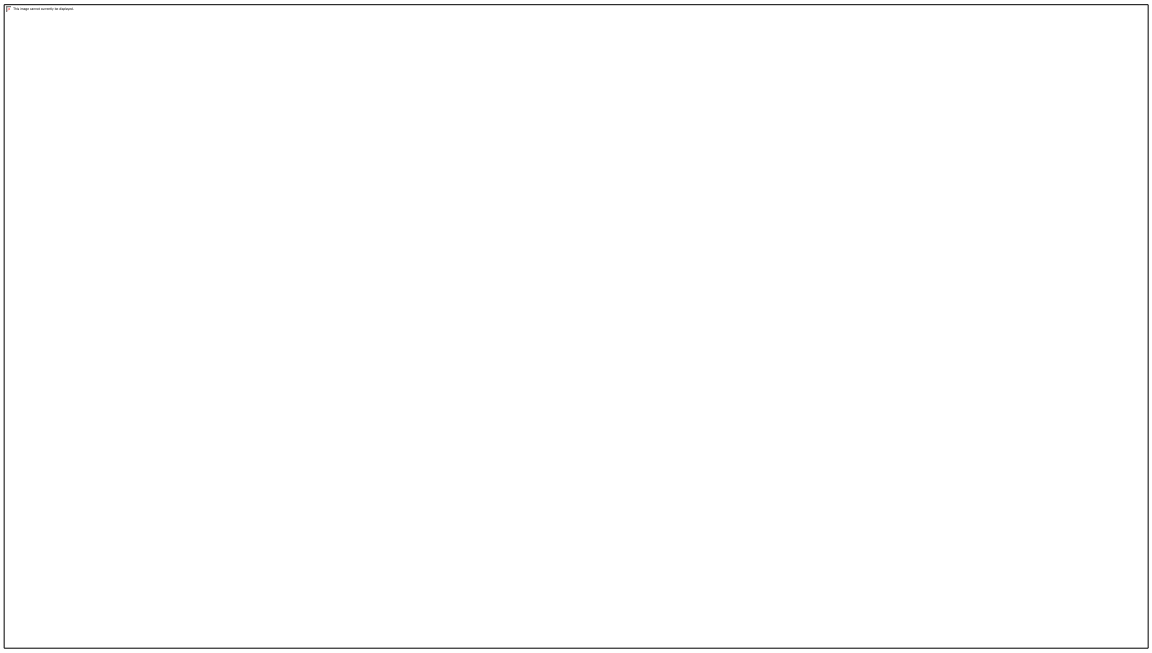


Description

A message dialog is an overlay UI element that provides a stable and contextual surface that is always modal and explicitly dismissed. Message dialogs appear at a consistent location on the screen.

© 2014 Microsoft. All rights reserved. | May 13, 2014

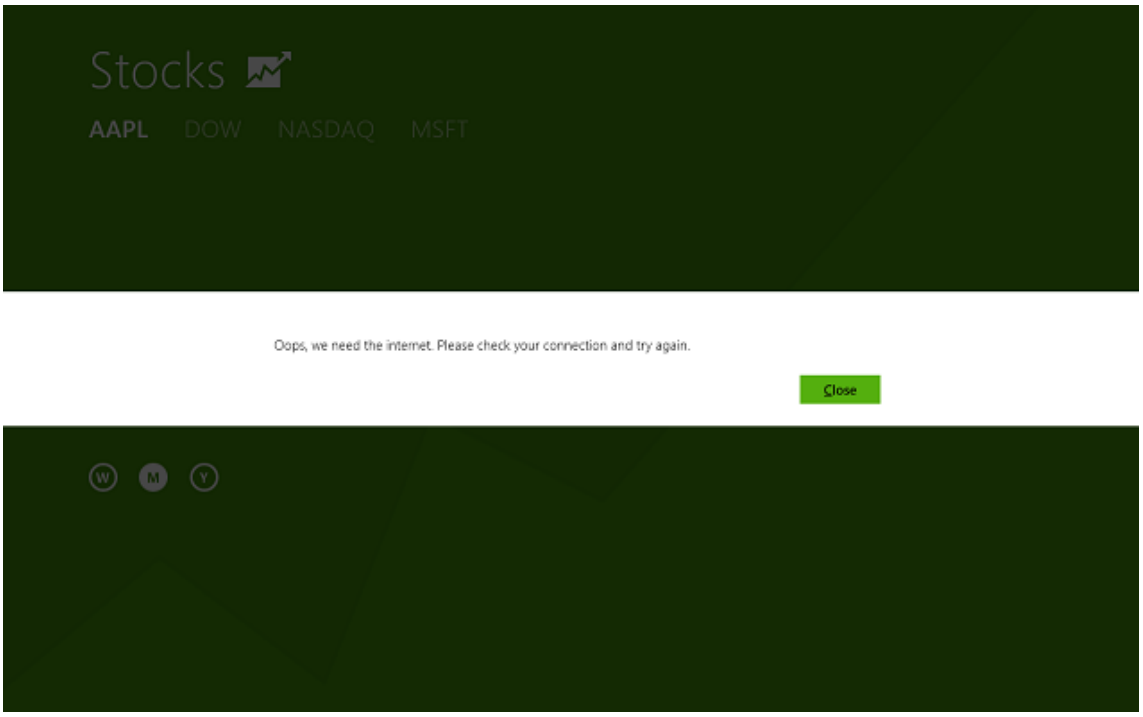
Example



Errors

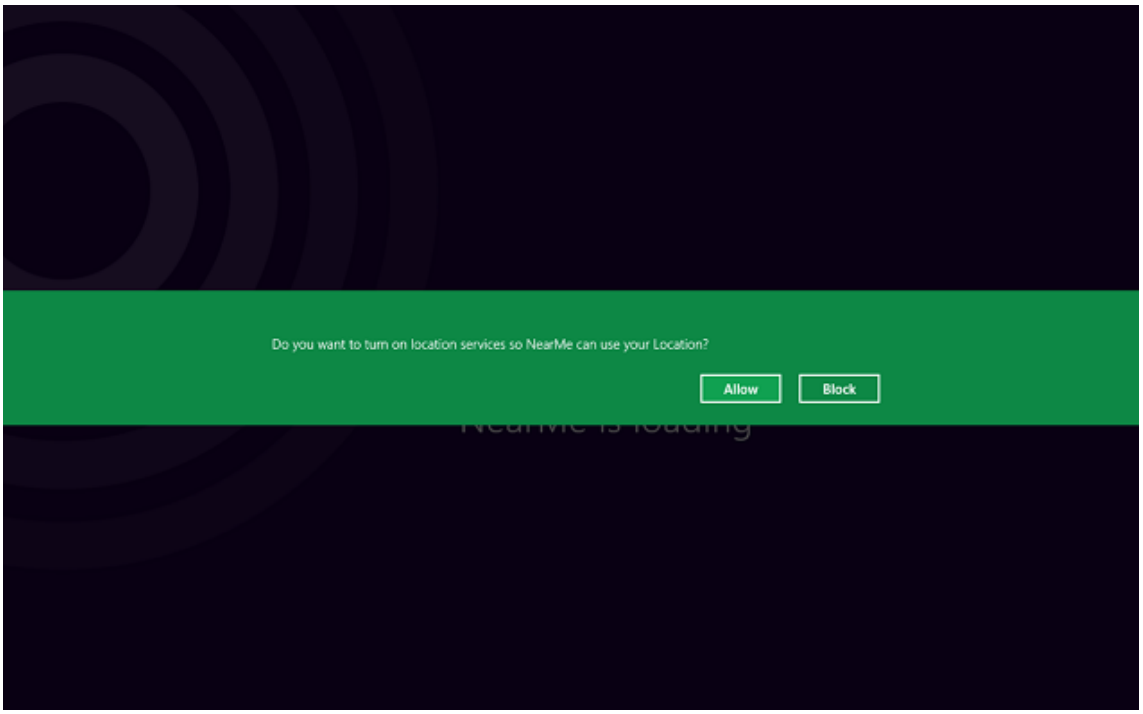
Error messages that apply to the overall app context use message dialogs. These are different than error messages that can be conveyed inline. An appropriate example is a message dialog that shows a connectivity error; this critically affects the value that the user can get from the app:

© 2014 Microsoft. All rights reserved. | May 13, 2014



Questions

Here's an example of a message dialog from the Windows device consent broker asking for consent to use location services:



© 2014 Microsoft. All rights reserved. | May 13, 2014

Dos and don'ts

- Use message dialogs to convey urgent information that the user must see and acknowledge before continuing. An example is, "Your trial period for advanced features has expired."
- Use message dialogs to present blocking questions that require the user's input. A blocking question is a question where the application cannot make a choice on the user's behalf, and cannot continue to fulfill its value proposition to the user. A blocking question should present clear choices to the user. It is not a question that can be ignored or postponed.
- Use message dialogs to ask for explicit action from the user or to deliver a message that is important for the user to acknowledge. Examples of usages of dialogs are the following:
    - The user is about to permanently alter a valuable asset
    - The user is about to delete a valuable asset
    - The security of the user's system could be compromised
- Use custom dialogs when the app or the system must invest a significant amount of time in the ensuing actions such that an accidental dismiss would be detrimental to the user's confidence.
- All dialogs should clearly identify the user's objective in the first line of the dialog's text (with or without a title).
- Don't use message dialogs when the app needs to confirm the user's intention for an action that the user has taken. Instead, a flyout is the appropriate surface.
- Don't use message dialogs for errors that are contextual to a specific place on the page, such as validation errors (in password fields, for example), use the app's canvas itself to show inline errors.

Additional usage guidance

All message dialogs should clearly identify the user's objective in the first line of the dialog's text. The following guidelines explain how to use the "title" and "content" fields of the message dialog to convey information effectively.

- **Title (main instruction, optional)**
    - Use a short title to explain what people need to do with the dialog. Long titles do not wrap and are truncated.
    - If you're using the dialog to deliver a simple message, error or question, you can optionally omit the title. Rely on the content text to deliver that core information.
    - Make sure the title relates directly to the button choices.
- **Content (descriptive text)**
    - Present the message, error, or blocking question as simply as possible without extraneous information.
    - When a title is used, use the content area to provide more detail or define terminology. Don't repeat the title with slightly different wording.
- **Buttons**
    - Use buttons with text that identifies specific responses to the main instruction or content. An example is, "Do you want to allow AppName to access your location?", followed by "Allow" and "Block" buttons. Specific responses can be understood more quickly, resulting in efficient decision making.
    - Avoid using generic patterns such as "OK/Cancel".

© 2014 Microsoft. All rights reserved. | May 13, 2014

    - Specify the default button, which should be the action you most want the user to take. In the above example, "Allow" is the default choice.
        - If not specified, the default is the leftmost button.
        - Put the safest, most conservative choice on the rightmost position. In the above example, "Block" is the rightmost choice as it is more conservative.
- **Color**
    - Message dialog backgrounds are always white. The primary colors of the app that owns the dialog are used for controls.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for Pivots (Windows Phone Store apps)



Windows Phone app: a pivot control with pivot items

Description

The pivot control is a full-screen container and navigation model that also provides a quick way to move between different pivots (views or filters), typically in the same set of data. For example, an email app using a pivot control might list all emails in the first pivot item (or view), and then filter the same list into unread, flagged, and urgent emails in the other pivot items.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Dos and don'ts

- Use themes to override the default appearance of a pivot control.
- The pivot control wraps from the last pivot item to the first and vice versa. Use this effect to properly design the flow of your app.
- Use no more than four or five pivot items in a pivot control for performance reasons and also to limit the chance of the user becoming lost. Use pivot controls sparingly and limit the use of pivot items to scenarios where it's appropriate for the experience.
- Use the pivot control to only display objects or data of similar type (for example, filtered views of the same data).
- Limit pivot item header text to a maximum of two words, both to provide the user a visual clue about the existence of the next pivot pane and to help with localization.
- Don't use the pivot control for task flow (exposing radically different tasks). Different pivot items should flow seamlessly (look and feel), and moving between them shouldn't change the user's activity drastically (for example, one page to filter mail and another to view pictures).
- Don't remove an empty pivot item while user action could cause info to be added to it. For example, if there are currently no unread emails, don't remove the Unread email pivot item, because synchronizing could cause some to appear. Instead, show placeholder content such as 'No unread messages'.
- Don't use a pivot control inside a hub control or vice versa. And don't place a pivot control inside another pivot control. You can, however, have an object inside a hub section link to a pivot control, and vice versa.
- Don't use controls that can pan or scroll inside a pivot control. For example, putting a map control inside a pivot item can make it difficult to use the pivot control. The gesture input gets confused. For example, if you have a slider and try to slide it left and you're in an item of a pivot control, it's unclear whether you want to move to the neighboring item or move the slider. The solution for a control that needs gesture input is to put it in its own page and navigate to that. You can place a gesture-disabled control in a pivot item—perhaps a map. You could overlay a button that activates the map. Pressing or tapping the button would navigate to a different page containing just the map. The user could then press the Back button to go back to the pivot item.
- Never use a text input box within a pivot item. Doing this interferes with the left-to-right flick and pan gesture interactions.

Additional usage guidance

Use the pivot control for filtering large data sets, viewing multiple data sets, or switching app views. Your app can look and react like the integrated Windows Phone pivot experiences.

Appearance and action

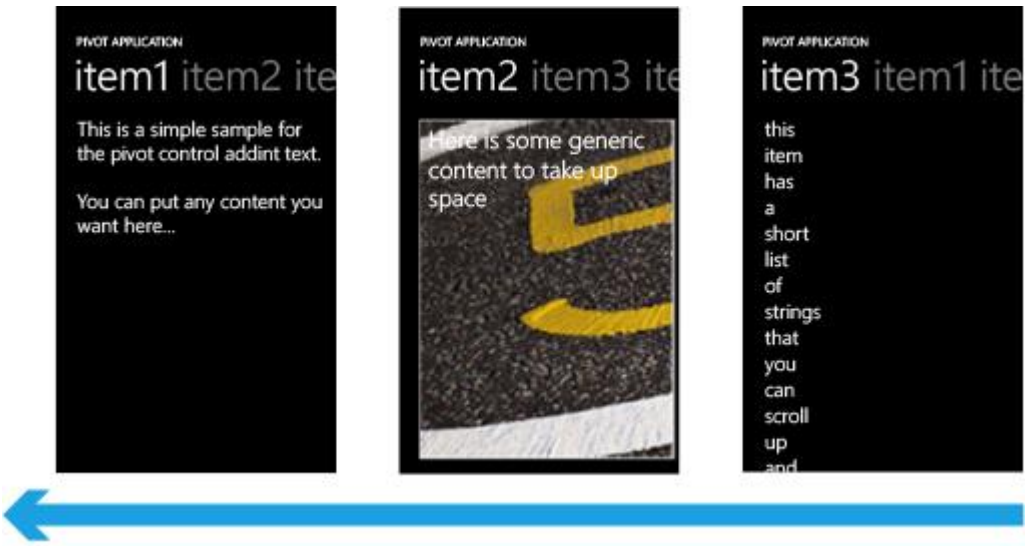
The pivot control hosts a set of pivot items (or views) arranged horizontally next to each other. The control enables the user to slide or flick horizontally to advance to the next or previous pivot item.

The pivot has built-in support for the following touch interactions:

© 2014 Microsoft. All rights reserved. | May 13, 2014

- Horizontal slide (touch and drag left/right)
- Horizontal flick (touch and flick quickly left/right)

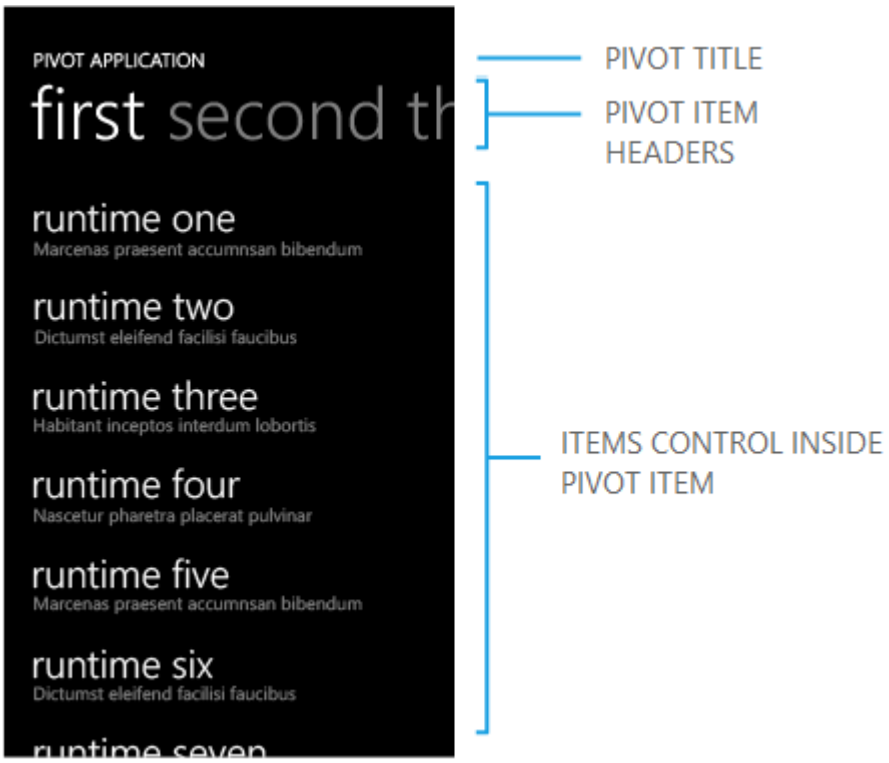
Controls hosted in a pivot item continue to be interactive, as usual—for example, links can be tapped and lists can be scrolled vertically.



Control description

The pivot control is a host, or container, for pivot items. Each pivot item can in turn contain any content such as layout panels, controls, and links. For more info about pivot control architecture, see Pivot control architecture for Windows Phone.

© 2014 Microsoft. All rights reserved. | May 13, 2014



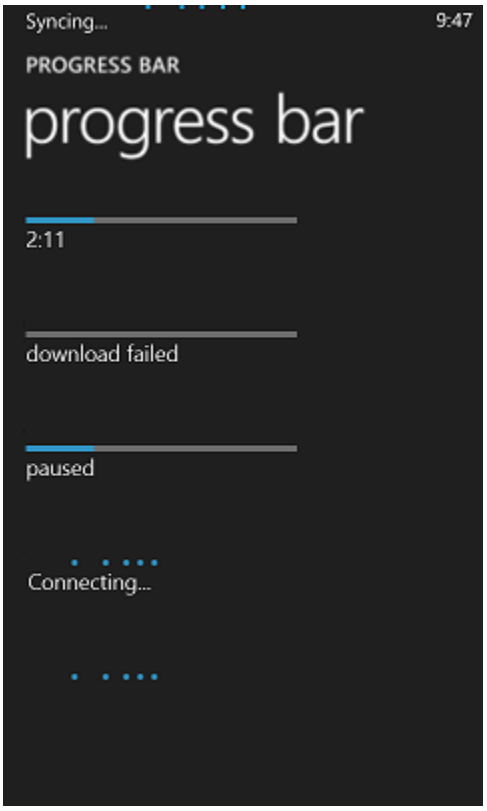
Parts of the pivot control (pivot title, pivot item headers, and so on)

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for progress controls



Windows app: indeterminate progress bar, progress ring, and determinate progress bar



Windows Phone app: status bar progress indicator and progress bars

Description

A progress control provides feedback to the user that a long-running operation is underway. A *determinate* progress bar shows the percentage of completion of an operation. An *indeterminate* progress bar, or a progress ring, shows that an operation is underway.

© 2014 Microsoft. All rights reserved. | May 13, 2014

A progress control is read only; it is not interactive.

Examples

Here is an example of a progress ring control on a splash screen.



A progress bar is also a good indicator of state or position. A progress bar used for a music track corresponds to the timeline of the song: the bar's value is the song position; the paused state indicates that playback is paused.



Is this the right control?

It's not always necessary to show a progress control. Sometimes a task's progress is obvious enough on its own or the task completes so quickly that showing a progress control would be distracting. Here are some points to consider when determining whether you should show a progress control.

- **Does the operation take more than two seconds to complete?**

  If so, show a progress control as soon as the operation starts. If an operation takes more than two seconds to complete most of the time, but sometimes completes in under two seconds, wait 500ms before showing the control to avoid flickering.

© 2014 Microsoft. All rights reserved. | May 13, 2014

- **Is the operation waiting for the user to complete a task?**

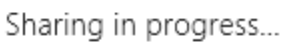
  If so, don't use a progress bar. Progress bars are for computer progress, not user progress.

- **Does the user need to know that something is happening?**

  For example, if the app is downloading something in the background and the user didn't initiate the download, the user doesn't need to know about it.

- **Is the operation a background activity that doesn't block user activity and is of minimal (but still some) interest to the user?**

  Use text and ellipses when your app is performing tasks that don't have to be visible all the time, but you still need to show the status.

  Use the ellipses to indicate that the task is ongoing. If there are multiple tasks or items, you can indicate the number of remaining tasks. When all tasks complete, dismiss the indicator.

- **Can you use the content from the operation to visualize progress?**

  If so, don't show a progress control. For example, when displaying images loaded from the disk, images appear on the screen one-by-one as they are loaded. Displaying a progress control would provide no benefit; it would just clutter the UI.

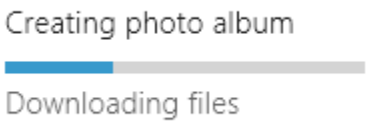
- **Can you determine, relatively, how much of the total work is complete while the operation is progressing?**

  If so, use a determinate progress bar, especially for operations that block the user. Use an indeterminate progress bar or ring otherwise. Even if all the user knows is that something is happening, that's still helpful.

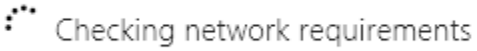
Dos and don'ts

- Use the determinate progress bar when a task is determinate, that is when it has a well-defined duration or a predictable end. For example, if you can estimate remaining amount of work in time, bytes, files, or some other quantifiable units of measure, use a determinate progress bar. Here are some examples of determinate tasks:
  - The app is downloading a 500k photo and has received 100k so far.
  - The app is displaying a 15 second advertisement and 2 seconds have elapsed.

© 2014 Microsoft. All rights reserved.



- Use the indeterminate progress ring for tasks that are not determinate and are modal (block user interaction).



- Use the indeterminate progress bar for tasks that are not determinate that are non-modal (don't block user interaction).

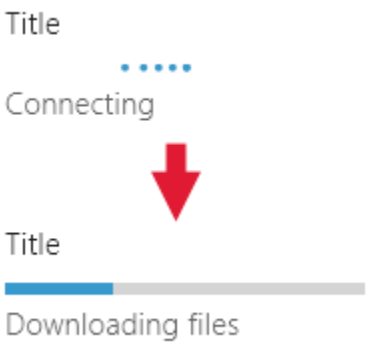
- Treat partially modal tasks as non-modal if the modal state lasts less than 2 seconds. Some tasks block interaction until some progress has been made, and then the user can start interacting with the app again. For example, when the user performs a search query, interaction is blocked until the first result is displayed. Treat tasks such as these as non-modal and use the indeterminate progress bar style if modal state lasts less than 2 seconds. If modal state can last more than 2 seconds, use the indeterminate progress ring for the modal phase of the task, and use the indeterminate progress bar for the non-modal phase.
- Consider providing a way to cancel or pause the operation that is in progress, particularly when the user is blocked awaiting the completion of the operation and has a good idea of how much longer the operation has left to run.
- Don't use the "wait cursor" to indicate activity, because users who use touch to interact with the system won't see it, and those users who use mouse don't need two ways to visualize activity (the cursor and the progress control).
- Show a single progress control for multiple active related tasks. If there are multiple related items on the screen that are all simultaneously performing some kind of activity, don't show multiple progress controls. Instead, show one that ends when the last task completes. For example, if the app downloads multiple photos, show a single progress control, instead of showing one for every photo.
- Don't change the location or size of the progress control while the task is running.

Guidelines for determinate tasks

- If the operation is modal (blocks user interaction), and takes longer than 10 seconds, provide a way to cancel it. The option to cancel should be available when the operation begins.
- Space progress updates evenly. Avoid situations where progress increases to over 80% and then stops for a long period of time. You want to speed up progress towards the end, not slow it down. Avoid drastic jumps, such as from 0% to 90%.

© 2014 Microsoft. All rights reserved. | May 13, 2014

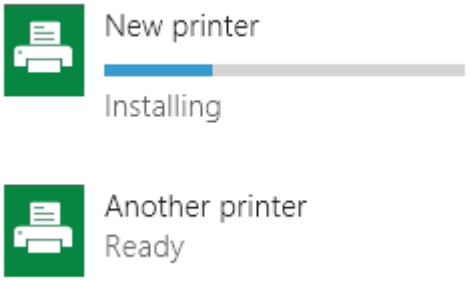
- After setting progress to 100%, wait until the determinate progress bar finishes animating before hiding it.
- If your task is stopped (by a user or an external condition), but a user can resume it, visually indicate that progress is paused. In JavaScript apps, you do this by using the win-paused CSS style. In C#/C++/VB apps, you do this by setting the ShowPaused property to true. Provide status text under the progress bar that tells the user what's going on.
- If the task is stopped and can't be resumed or has to be restarted from scratch, visually indicate that there's an error. In JavaScript apps, you do this by using the win-error CSS style. In C#/C++/VB apps, you do this by setting the ShowError property to true. Replace the status text (underneath the bar) with a message that tells the user what happened and how to fix the issue (if possible).
- If some time (or action) is needed before you can provide determinate progress, use the indeterminate bar first, and then switch to the determinate bar. For example, if the first step of a download task is connecting to a server, you can't estimate how long that takes. After the connection is established, switch to the determinate progress bar to show the download progress. Keep the progress bar in exactly the same place, and of the same size after the switch.



- If you have a list of items, such as a list of printers, and certain actions can initiate an operation on items in that list (such as installing a driver for one of the printers), show a determinate progress bar next to the item.

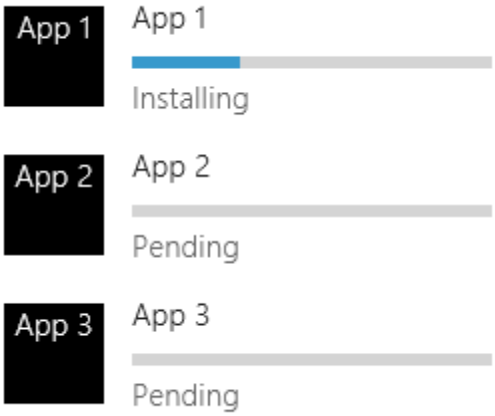
  Show the subject (label) of the task above the progress bar and status underneath. Don't provide status text if what's happening is obvious. After the task completes, hide the progress bar. Use the status text to communicate the new state of an item.

© 2014 Microsoft. All rights reserved. | May 13, 2014



- To show a list of tasks, align the content in a grid so users can see the status at a glance. Show progress bars for all items, even those that are pending.

  Because the purpose of this list is to show ongoing operations, remove operations from the list when they complete.



- If a user initiated a task from the app bar and it blocks user interaction, show the progress control in the app bar.

  If it's clear what the progress bar is showing progress for, you can align progress bar to the top of the app bar and omit the label and status; otherwise, provide a label and status text.

  Disable interaction during the task by disabling controls in the app bar and ignoring input in the content area.

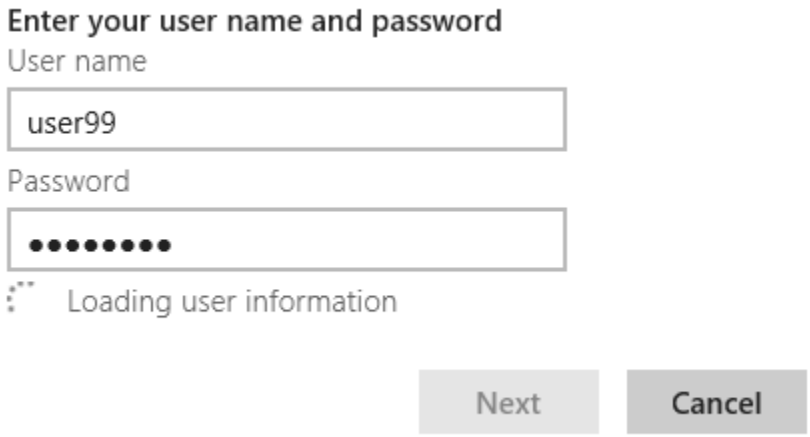
- Don’t decrement progress. Always increment the progress value. If you need to reverse an action, show the progress of reversal as you would show progress of any other action.
- Don’t restart progress (from 100% to 0%), unless it’s obvious to the user that a current step or task is not the last one. For example, suppose a task has two parts: downloading some data, and then processing and displaying the data. After the download is complete, reset the progress bar to 0% and begin showing the data processing progress. If it’s unclear to users

© 2014 Microsoft. All rights reserved. | May 13, 2014

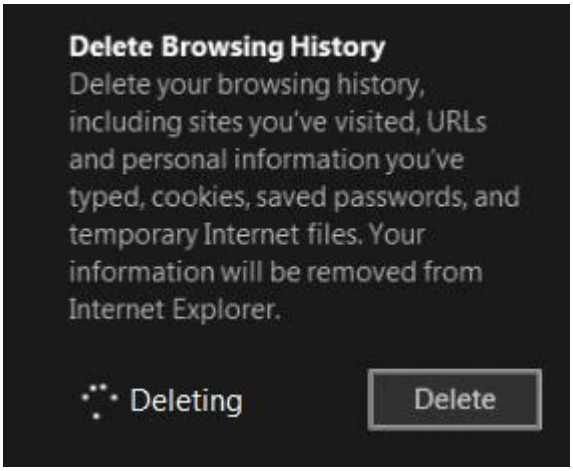
that there are multiple steps in a task, collapse the tasks into a single 0-100% scale and update status text as you move from one task to the next.

Guidelines for modal, indeterminate tasks that use the progress ring

- Display the progress ring in the context of the action: show it near the location where the user initiated the action or where the resulting data will display.
- Provide status text to the right of the progress ring.
- Make the progress ring the same color as its status text.
- Disable controls that user shouldn't interact with while the task is running.
- If the task results in an error, hide the progress indicator and status text and display an error message in their place.
- In a dialog, if an operation must complete before you move to the next screen, place the progress ring just above the button area, left-aligned with the content of the dialog.

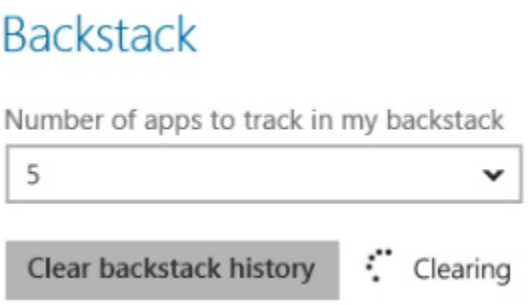


- In an app window with right-aligned controls, place the progress ring to the left or just above the control that caused the action. Left-align the progress ring with related content.

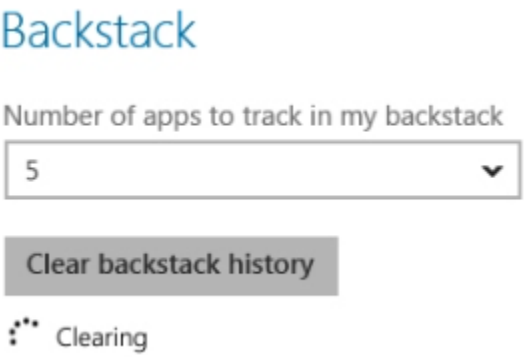


© 2014 Microsoft. All rights reserved. | May 13, 2014

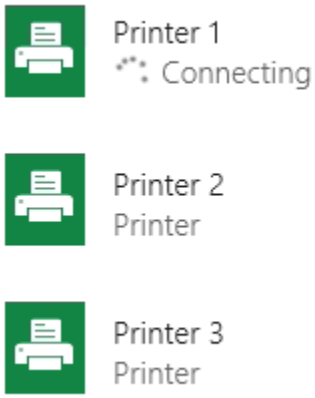
- In an app window with left-aligned controls, place the progress ring to the right or just under the control that caused the action.



- or -



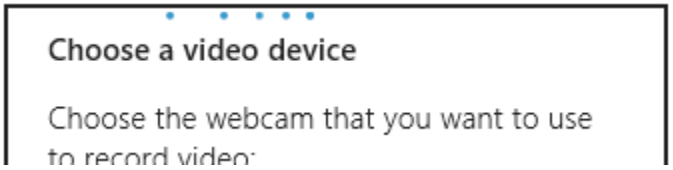
- If you are showing multiple items, place the progress ring and status text underneath the title of the item. If an error occurs, replace the progress ring and status with error text.



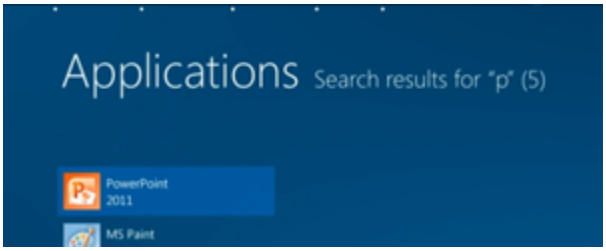
© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for non-modal, indeterminate tasks that use the progress bar

- If you show progress in a flyout, place the indeterminate progress bar at the top of the flyout and set its width so that it spans the entire flyout. This placement minimizes distraction but still communicates ongoing activity. Don't give the flyout a title, because a title prevents you from placing the progress bar at the top of the flyout.



- If you show progress in an app window, place the indeterminate progress bar at the top of the app window, spanning the entire window.



Guidelines for status text

- When you use the determinate progress bar, don’t show the progress percentage in the status text. The control already provides that info.
- If you use text to indicate activity without a progress control, use ellipsis to convey that the activity is ongoing.
- If you use a progress control, don't use ellipsis in your status text, because the progress control already indicates that the operation is ongoing.

Guidelines for appearance and layout

- A determinate progress bar appears as a colored bar that grows to fill a gray background bar. The proportion of the total length that is colored indicates, relatively, how much of the operation is complete.
- An indeterminate progress bar or ring is made of continually moving colored dots.
- Choose the progress control's location and prominence based on its importance.

  Important progress controls can serve as a call-to-action, telling the user to resume a certain operation after the system has done its work. Some built-in Windows Phone apps use a status bar progress indicator at the top of the screen for important cases. You can do this, too, and configure it to be determinate or indeterminate.

© 2014 Microsoft. All rights reserved. | May 13, 2014

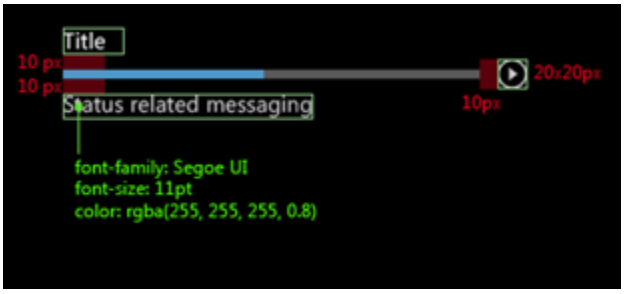
Cases that are less critical, such as during downloading, appear smaller and are restricted to one view.

- Use a label to show the progress value, or to describe the process taking place, or to indicate that the operation has been interrupted. A label is optional, but we highly recommend it.

  To describe the process taking place, use a gerund (an –ing verb), e.g. 'connecting', 'downloading', or 'sending'.

  To indicate that progress is paused or has encountered an exception, use past participles, e.g. 'paused', 'download failed', or 'canceled'.

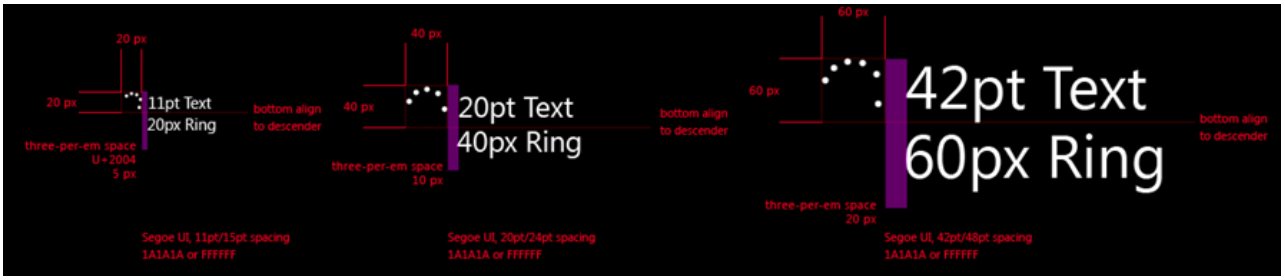
- Determinate progress bar with label and status



- Multiple progress bars

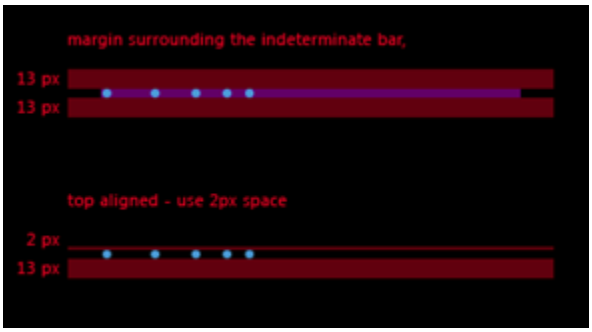


- Indeterminate progress ring with status text



- Indeterminate progress bar

© 2014 Microsoft. All rights reserved.



Additional usage guidance

Decision tree for choosing a progress style

- **Does the user need to know that something is happening?**

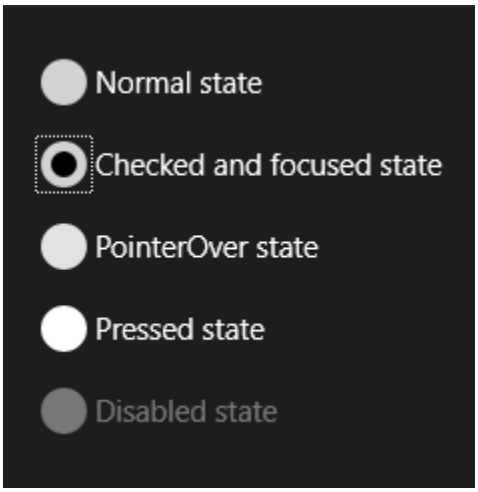
  If the answer is no, don't show a progress control.

- **Is info about how much time it will take to complete the task available?**
  - **Yes: Does the task take more than two seconds to complete?**
    - **Yes:** Use a determinate progress bar. For tasks that take longer than 10 seconds, provide a way to cancel the task.
    - **No:** Don't show a progress control.
  - **No: Are users blocked from interacting with the UI until the task is complete?**
    - **Yes: Is this task part of a multi-step process where the user needs to know specific details of the operation?**
      - **Yes:** Use an indeterminate progress ring with status text horizontally centered in the screen.
      - **No:** Use an indeterminate progress ring without text in the center of the screen.
    - **No: Is this a primary activity?**
      - **Yes: Is progress related to a single, specific element in the UI?**
        - **Yes:** Use an inline indeterminate progress ring with status text next to its related UI element.
        - **No: Is a large amount of data being loaded into a list?**
          - **Yes:** Use the indeterminate progress bar at the top with placeholders to represent incoming content.
          - **No:** Use the indeterminate progress bar at the top of the screen or surface.
      - **No:** Use status text in an upper corner of the screen.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for radio buttons





Windows app: radio buttons



Windows Phone app: radio buttons

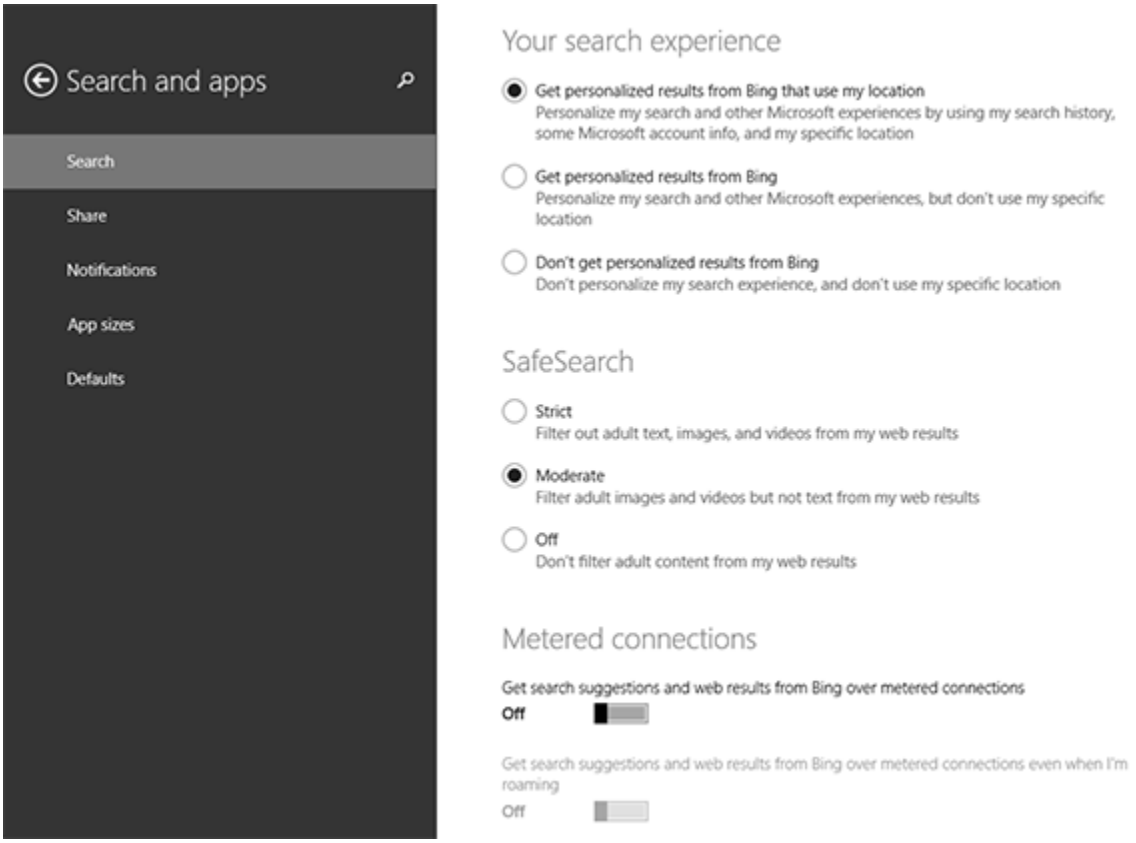
Description

Radio buttons let users select one option from two or more choices. Each option is represented by one radio button; a user can select only one radio button in a radio button group.

Radio buttons are named for the channel preset buttons on a radio.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Example



Is this the right control?

Use radio buttons to present users with two or more mutually exclusive options, as here.

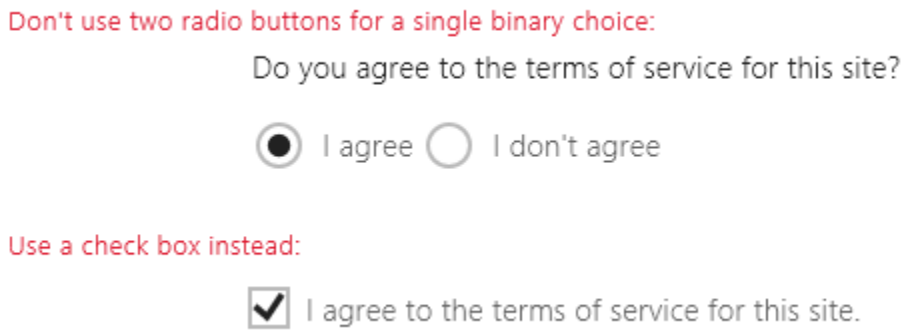


Radio buttons add clarity and weight to very important options in your app. Use radio buttons when the options being presented are important enough to command more screen space and where the clarity of the choice demands very explicit options.

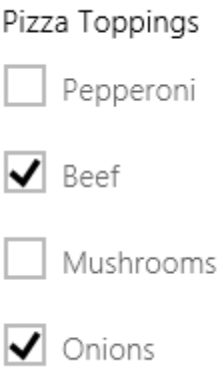
Radio buttons emphasize all options equally, and that may draw more attention to the options than necessary. Consider using other controls, unless the options deserve extra attention from the user. For example, if the default option is recommended for most users in most situations, use a drop-down list instead.

© 2014 Microsoft. All rights reserved. | May 13, 2014

If there are only two mutually exclusive options, combine them into a single checkbox or toggle switch. For example, use a checkbox for "I agree" instead of two radio buttons for "I agree" and "I don't agree."



When the user can select multiple options, use a checkbox or list box control instead.



Don't use radio buttons when the options are numbers that have fixed steps, like 10, 20, 30. Use a slider control instead.

If there are more than 8 options, use a drop-down list, a single-select list box, or a list view instead.

If the available options are based on the app's current context, or can otherwise vary dynamically, use a single-select list box instead.



A group of radio buttons behaves like a single control when accessed via the keyboard. Only the selected choice is accessible using the Tab key but users can cycle through the group using arrow keys.

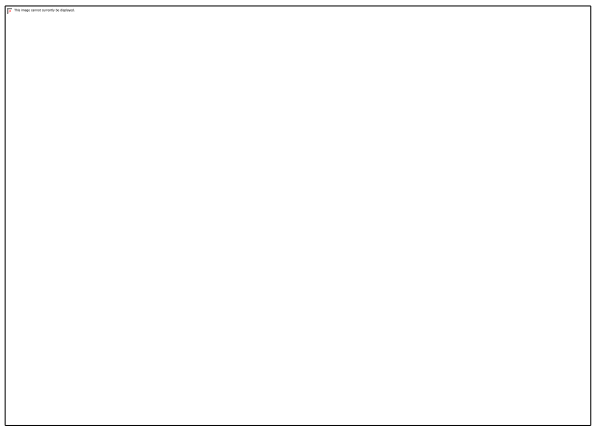
© 2014 Microsoft. All rights reserved. | May 13, 2014

Dos and don'ts

- Make sure that the purpose and current state of a set of radio buttons is clear.
- Always give visual feedback when the user taps a radio button.
- Give visual feedback as the user interacts with radio buttons. Normal, pressed, checked, and disabled are examples of radio button states. A user taps a radio button to activate the related option. Tapping an activated option doesn’t deactivate it, but tapping another option transfers activation to that option.
- Reserve visual effects and animations for touch feedback, and for the checked state; in the unchecked state, radio button controls should appear unused or inactive (but not disabled).
- Limit the radio button’s text content to a single line. You can customize the radio button’s visuals to display a description of the option in smaller font size below the main line of text.
- If the text content is dynamic, consider how the button will resize and what will happen to visuals around it.
- Use the default font unless your brand guidelines tell you to use another.
- Enclose the radio button in a label element so that tapping the label selects the radio button.
- Place the label text after the radio button control, not before or above it.
- Consider customizing your radio buttons. By default, a radio button consists of two concentric circles—the inner one filled (and shown when the radio button is checked), the outer one stroked—and some text content. But we encourage you to be creative. Users are comfortable interacting directly with the content of an app. So you may choose to show the actual content on offer, whether that’s presented with graphics or as subtle textual toggle buttons.
- Don't put more than 8 options in a radio button group. When you need to present more options, use a drop-down list, list box, or a list view instead.
- Don't put two radio button groups next to each other. When two radio button groups are right next to each other, it's difficult to determine which buttons belong to which group. Use group labels to separate them.

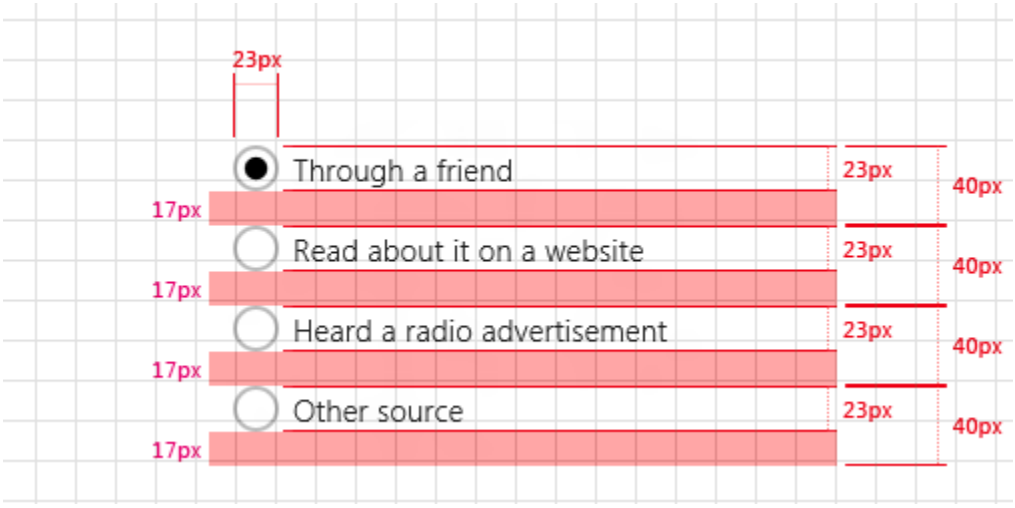
Additional usage guidance

This illustration shows the proper way to position and space radio buttons.



© 2014 Microsoft. All rights reserved. | May 13, 2014

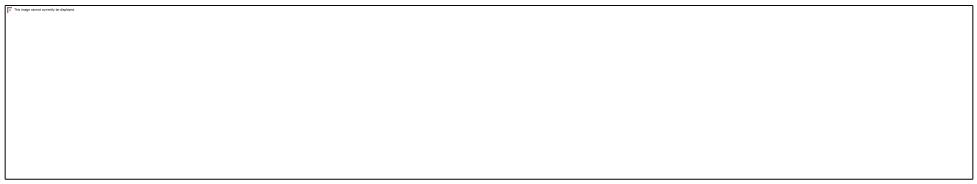
This diagram highlights the sizing and spacing used in the previous illustration.



© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for the rating control





Description

The rating control lets users rate something by clicking an icon that represents a rating. It can display three types of ratings: an average rating, a tentative rating, and the user's rating.



Note The rating control is available only in HTML. There is no XAML rating control.

Example



© 2014 Microsoft. All rights reserved. | May 13, 2014

Is this the right control?

Use the rating control to show the degree to which users like, appreciate, or are satisfied with an item or service. For example, use the rating control to let users rate a movie. Don't use it for other types of data that have a continuous range, such as screen brightness. (Use a slider for that).

Don't use the rating control as a filter control. For example, if you want to let users filter search results to show restaurants with five stars, don't implement the filter with a rating control. Using the ratings control could mislead users into thinking that they are giving a new rating to the restaurant. You could use a drop-down list instead.

Don't use a one-star rating control as a like/dislike control. You should use a check box instead. The rating control is not designed for a binary rating, for example, tapping on the control doesn't toggle the star on and off. For an example of how to use a check box as a like/dislike control in Windows Store apps using JavaScript, see the Common HTML controls and everyday widgets sample.

Dos and don'ts

- Use tooltips to give users more context. You can customize the tooltip to show more meaningful words for each star, like "excellent, "very good," or "not bad," as shown here:



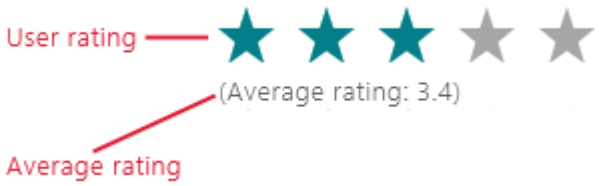
- Disable the rating control when you want to prevent the user from adding or modifying the rating. A disabled rating control continues to display the rating (if one is set), but doesn't allow the user to add or modify it. Suppose you want to restrict ratings to logged-in users. You can disable the rating control, and when users tap the control, you can send them to a log-in page.

  If the control cannot be enabled (it is read-only for the life of the app), make it smaller than other rating controls by setting the **class** attribute of the control host element to "win-small". Making the control smaller helps distinguish it from the other controls and also discourages interaction.

© 2014 Microsoft. All rights reserved. | May 13, 2014



- Show the average rating and user rating at the same time. After the user provides a rating, the rating control displays the user's rating instead of the average rating. Whenever it's meaningful to your users, show them the average user rating in addition to their rating. Here are two ways you can display average rating:
  - Display the average in an accompanying text string (such as "average: 3.5").
  - Use two rating controls together, one to show the user rating, and one that shows the average rating and doesn't allow user input.



- Don't change the default number of stars (the max rating) unless you must. By default, the rating control has 5 stars, with 1 being the lowest or worst rating and 5 being the highest or best. If your app follows this convention, it will be easy for users to interpret what the rating means.
- Don't disable the "Clear your rating" feature unless you must prevent users from deleting their ratings.

© 2014 Microsoft. All rights reserved. | May 13, 2014

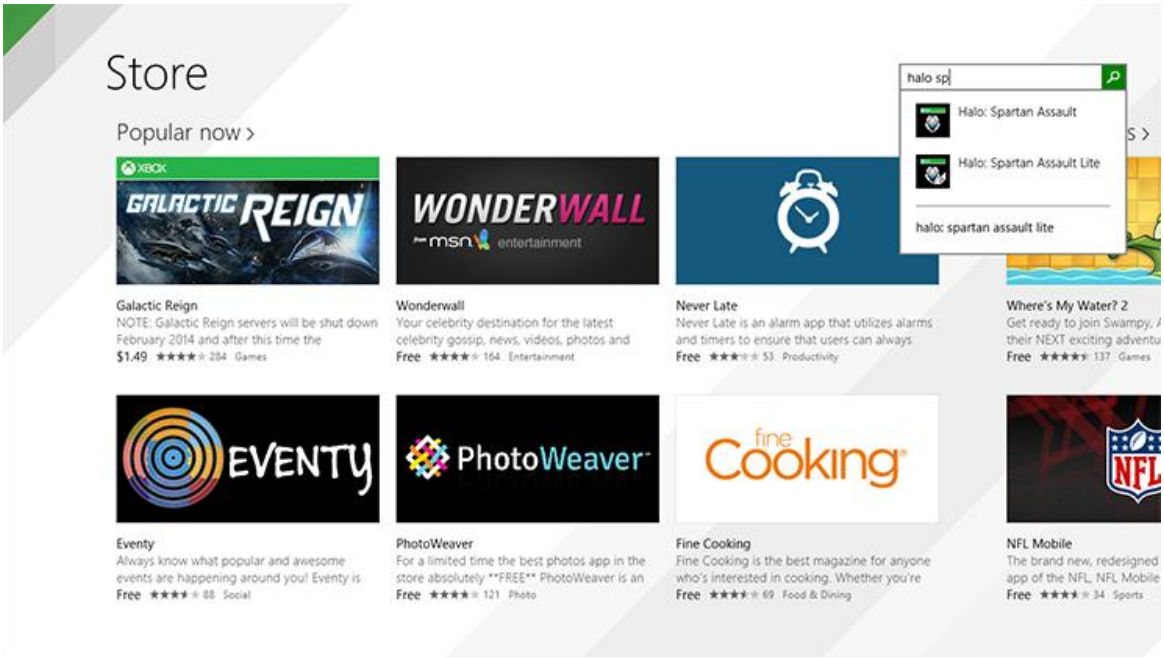
Guidelines for search



Description

Adding search to your app lets users find what they're looking for by entering queries. To add search to your app, you add a search box for entering queries and a search results page for display search results. (Alternatively, you can use the search pane and the search contract instead of the search box to handle queries.)

Example



© 2014 Microsoft. All rights reserved. | May 13, 2014

Should my app include search?

If your app contains content the user might want to search for, then add search to your app. For example, if your app plays media files, users will expect to be able to search for a specific song or video; if your app is a cooking app, users will expect to search for specific recipes or ingredients.

Not every app needs to provide search. For example, most games and simple productivity apps like calculators and timers don't contain content that the user will want to search for.

Do's and don’ts

- Implement search if your app has content the user might want to search for.
- If search is one of the main ways people will get to content in your app, add a search box to the app canvas.
-

Search box guidelines

The search box is a control that can be used to enter search query text.

Consider adding a search box to your app canvas if search is one of the main ways people will get to content in your app. A search box is a great way for users to know where to start. If space is a concern for your layout, use an icon that expands to reveal a search box. Also, you can put your search box in the app bar. Consider putting the search box on every page in your app. Users may be confused if some pages have a search box and others don't, but this decision depends on context and the purpose of your app.

- Use the **Windows.UI.Xaml.Controls.SearchBox** control for Windows Store apps using C++, C#, or Visual Basic and the **WinJS.UI.SearchBox** control for Windows Store apps using JavaScript.
- Regardless of where your app's content is located, for example on the local file system, or through a web service, you can use the search box to respond to user queries and display search results in an app page of your own design.
- A search box is a great way for users to know where to start. If space is a concern for your layout, use an icon that expands to reveal a search box. Also, you can put your search box in the app bar.
- If you need search UI in the app bar, place it in the upper app bar. In Windows Store apps using JavaScript, you can place search UI in the nav bar.
- Consider putting the search box on every page in your app. Users may be confused if some pages have a search box and others don't, but this decision depends on context and the purpose of your app.

Search box sizing

© 2014 Microsoft. All rights reserved. | May 13, 2014

- When using the in-app search box control, keep in mind the various window sizes and the control's position relative to the title.
- Avoid clipping the title.
- At smaller window sizes, you may want to collapse the search box to a button with the search icon, depending on the contents of your title bar. When users click, expand the search box and hide the elements that it would occlude, like the title.
- Don't occlude the back button, since it's a core piece of navigation and it's important for users to have access to it at all times.
- The minimum height of the suggestions flyout is 200 pixels.
- The default maximum height is 300 pixels, and you can change this.
- The minimum width is 270 pixels. You can change this, but the Input Method Editor (IME) window won't have a smaller width than 270 pixels.
- The maximum placeholder text length is 128 characters. Maximum query text length is 2048 characters. Maximum length of all of the textual fields in a suggestion (text, detailtext, image alt text) is 512 characters.
- On results pages, pre-populate the query in the search box. The results should persist if the user navigates away and back. Clear the query text on forward navigation, even from a results page.

Customizing suggestions and placeholder text in the search box

When people start typing a query into the search box, the app supplies search suggestions in a suggestions flyout beneath the search box.

The search box takes a maximum of 25 search suggestions.

An app can provide two types of search suggestions: *query suggestions* and *result suggestions*. You can think of result suggestions as *answers* and query suggestions as *possibilities*.

- **Query suggestions** are frequently auto-completions of the user's query text, and they provide queries that the user might want to search for. For example, let's say the user is searching for an app in the Store app and enters the query "word" in the Store app's search box. "Wordament" and "Wordpress" might appear as result suggestions, and "word games" and "words" might be query suggestions. If the user selects one of the query suggestions, the Store app take the user to a search results page for that query. Although it's usually the case that query suggestions are auto-completions, they don't have to be. For example, if you're in a recipe app searching for “eggs,” quiche is a viable query suggestion.
- **Result suggestions** are strong or exact or matches to the user's query that the user may want to view immediately. For example, the Wordament app is suggested as a result, under the Recommendations label, for the "word" query in the Store app. If the user selects the Wordament result suggestion, the Store app takes the user directly to the Wordament app page in the Store.

Result suggestions, which are usually closer matches, go on top of the suggestions flyout, and query suggestions follow.

© 2014 Microsoft. All rights reserved. | May 13, 2014

When the user enters a search query, the system automatically provides search history suggestions for possible queries in the search box. These suggestions are based on the user's previous interactions with your app's search box, before suggestions that your app provides. You can disable the search history.

Don't use suggestions to filter or scope search results for your app. Filtering and scoping options should be placed with the results in the app's search results page, where they can refine the search results. In contrast, suggestions should be placed in the search box, because they are directly related to the query text that the user enters.

Query suggestions

- In general, consider providing query suggestions to help the user search quickly.
- Use query suggestions as way to auto-complete query text that users can search for in your app. This helps users search quickly by reducing the amount of typing needed to complete a search. Instead of entering the entire query, users can select one of the suggested queries and immediately execute the search.
- If the user selects a query suggestion, navigate immediately to a search results page for the selected query.
- Your app's query suggestions should contain the user's current query text. Because your query suggestions usually are auto-completions that are based on the current query text, they should contain the current query text.
- Your query suggestions should reflect the content of your app and the results it can provide, instead of providing results from a more global context. By searching the content your app can provide, your query suggestions help the user figure out what they can search for in your app. For example, the Weather app automatically completes the user's query to suggest cities for which the app can provide weather reports.

Result suggestions

- If you want to recommend strong or exact matches for the user's query, provide result suggestions.
- Use result suggestions to take the user directly to the details of a particular result without first taking them to a search results page.
- A result suggestion should consist of an appropriate image or thumbnail, a relevant title or label, and a brief description. The image size is fixed at 40x40 pixels.
- The image, title, and description help the user to determine quickly whether the suggested result is what they were searching for.
- With a small result set, consider filtering results on the app's canvas, which means that as the user types, the results update immediately with each keystroke. This is a form of filtering as the user types. For example, the Reading List app filters results this way. The search history should be turned off, and the app updates its canvas as fast as possible. Implementing this feature depends on UI performance and on how quickly you can return a small result set back, usually with no more than ten results. If the user has to type more than three characters to get a set of ten results, consider using suggestions as described previously.

Separators

© 2014 Microsoft. All rights reserved. | May 13, 2014

- Use a separator to define a group of a similar or conceptually aligned results.
- You can add separators to the list of suggestions that your app supplies to the search box, but each separator counts toward the limit of 25 suggestions.
- If you want to supply multiple types of result suggestions, use labeled separators to help users distinguish between results. For example, when you provide more than one suggestion for results with different content types, like movies and TV shows, use a labeled separator to provide a meaningful distinction between the "movies" and "TV shows" content types.

Placeholder text

- Consider using placeholder text in the search box to describe what users can search for in your app. For example, a recipe app could show "Search for recipes."
- Show placeholder text only when the search box is empty. Clear the placeholder text when the user starts typing in the search box.
- Set the placeholder text of the search box to a brief description of the searchable content in your app. For example, a music app that supports searching by album name, song name, or artist name could set the placeholder text to "Album, artist, or song name".
- Use placeholder text to provide context to the user about what is being searched. This is important when a subset of the app is being searched in a given context. For example, use text like "Artist" or "Album" in a music app that searches artists or albums in subsections dedicated to those topics.

Enabling type-to-search

If you're using the search box, consider letting people search simply by typing, without having to click or tap in the search box control first. You can do this by enabling the type-to-search feature of your search box control. This is the same kind of behavior users are familiar with from using the Start screen on Windows 8. Many people use a keyboard to interact with Windows 8, and letting users search by typing makes efficient use of keyboard interaction.

Don't enable type-to-search on every page of your app. Consider where type-to-search would be most useful for your users and where it might be a problem. For example, you'd want to disable type-to-search if other controls that take input, like a text box, get focus. Otherwise, the search box continues to take the characters.

Use the following guidelines to help you figure out how to add type to search to your app so that it creates a positive experience for your users.

- **Enable type to search on your app's main page (or pages).**

  Do this for your app's main page and any app page that displays, directly or indirectly, all of your app's searchable content. Your main page is the page that is displayed when users enter your app. In some cases an app may offer multiple main pages that offer different presentations for similar content. In this case type to search should be enabled on all of these pages.

© 2014 Microsoft. All rights reserved. | May 13, 2014

- **Enable type to search on your app's search results pages.**

  After users get search results they often want to search again. You should enable typing directly into the search box from search results pages, so that keyboard users can perform another search quickly.

- **Disable type to search on most other pages in your app.**

  In most cases, you should disable typing directly into the search box on pages that are not your app's main page or a search results page. For pages with one or more input boxes, your app can disable type to search if a text box gets focus.

Designing a search results page

Craft your search experience around your brand. Search results user experience (UX) should be tailored to your content, similar to how you present content in your app.

When users submit search queries to your app, they see a page that shows search results for the query. Because you design the search results page for your app, you can ensure that the results presented to your user are useful and have an appropriate layout.

Title the search results page, and put the user's query text in the search box to provide context.

Indicate the ranking of search results by ordering and laying them out by relevancy, from best match to weakest.

Indicate why a search result matches the query.

Don't flip the search icon in bi-directional languages.

Highlight the matching search terms to show the part of the suggestion that matches the query. This is valuable in all cases, if the match property is being shown to the user.

Filtering results

- You can improve your app's search results page by letting users set filters and scopes to refine the set of search results.
- If you have a large number of results from a query, consider providing a way for the user to filter the results.
- Let users filter and scope search results from the search results page.
- Indicate the number of results available with each filter or in each scope. This helps users understand whether they are effectively refining their search.
- Provide a way to clear the filters and see all results.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for using the Search contract and the search pane

Use the following guidance if your app uses the Search contract and the search pane to handle queries.

- Add a button with the search icon to your app to invoke the search pane programmatically and scope it correctly. Use the search glyph, Segoe UI Symbol 0xE0094 at 15pt, to invoke the pane.
- A prominent search icon gives users a strong visual cue that tells them where to begin. When users select the icon your app should open the Search charm programmatically so that users can enter their query using the search pane. Using the Search charm in this way helps make your app more intuitive.
- You can't use the search box and the search pane in the same app.

For code that shows how to use the search contract and the search pane, see the Search contract sample.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for scroll bars



Description

Panning and scrolling allows users to reach content that extends beyond the bounds of the screen.

A scroll viewer control is composed of as much content as will fit in the viewport, and either one or two scroll bars. Touch gestures can be used to pan and zoom (the scroll bars fade in only during manipulation), and the pointer can be used to scroll. The flick gesture pans with inertia.

Windows: There are two panning display modes based on the input device detected: panning indicators for touch; and scroll bars for other input devices including mouse, touchpad, keyboard, and stylus.

Example



© 2014 Microsoft. All rights reserved. | May 13, 2014

Dos and don'ts

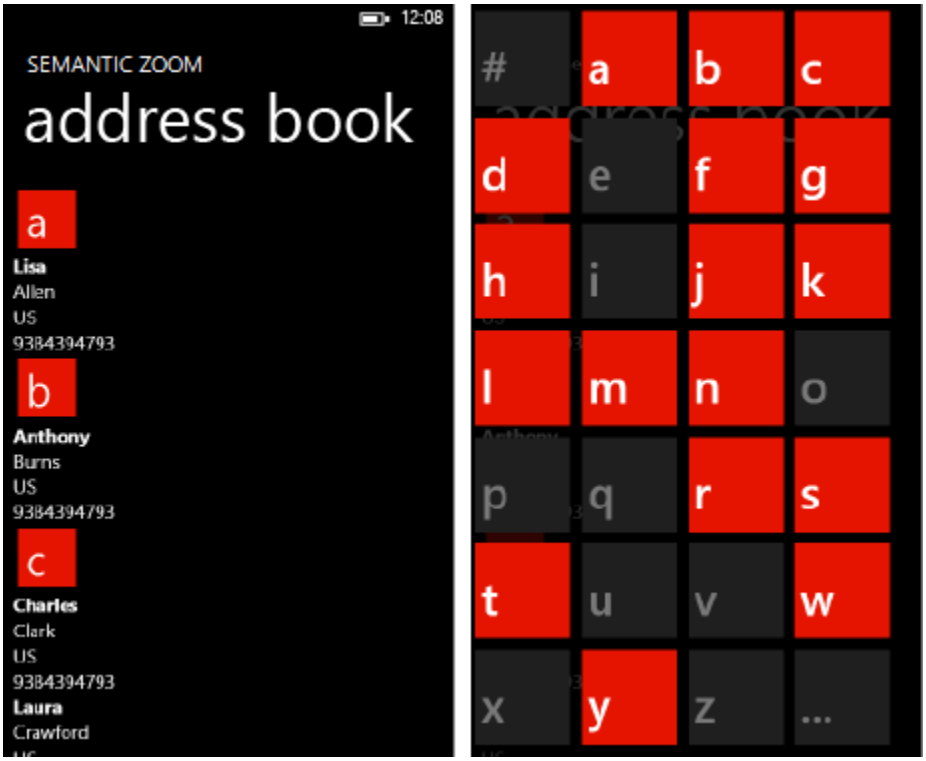
- Display panning indicators and scroll bars to provide location and size cues. Hide them if you provide a custom navigation feature.
- Use one-axis panning for content regions that extend beyond one viewport boundary (vertical or horizontal). Use two-axis panning for content regions that extend beyond both viewport boundaries (vertical and horizontal).
- Use the built-in scroll functionality in the list box, drop-down list, text input box, grid view, list view, and hub controls. With those controls, if there are too many items to show all at once, the user is able to scroll either horizontally or vertically over the list of items.
- If you want the user to pan in both directions around a larger area, and possibly to zoom, too, for example, if you want to allow the user to pan and zoom over a full-sized image (rather than an image sized to fit the screen) then place the image inside a scroll viewer.
- If the user will scroll through a long passage of text, configure the scroll viewer to scroll vertically only.
- Use a scroll viewer to contain one object only. Note that the one object can be a layout panel, in turn containing any number of objects of its own.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for semantic zoom



Windows app: zoomed-in and zoomed-out views of a semantic zoom control



Windows Phone app: zoomed-out and zoomed-in views of a semantic zoom control

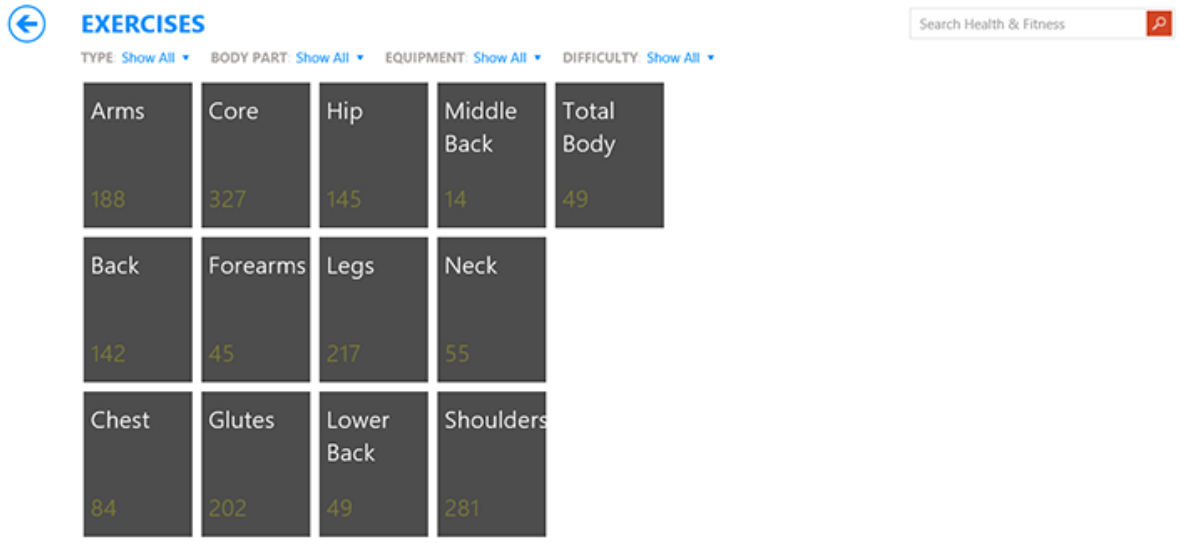
Description

A semantic zoom control allows the user to zoom between two different semantic views of the same data set. One view contains a list of items grouped by some key, and the other view contains a list of group keys. Tapping a group key zooms back into the items in that group.

© 2014 Microsoft. All rights reserved. | May 13, 2014

If the data set is a list of physical exercises grouped by the body part that each one benefits, then the zoomed-in view would be a list of exercises grouped by body part (the items) and the zoomed-out view would be a list of body parts (the group keys). With a large data set, being able to zoom out and see all the group keys on one page really helps the user to quickly access a distant item that would otherwise take lots of scrolling to reach.

Examples



Dos and don'ts

- Use the correct touch target size for elements that are useable or interactive.
- Provide a suitable and intuitive Semantic Zoom region. Users often initiate Semantic Zoom from within the area that surrounds the displayed items. Make the Semantic Zoom region large enough to encompass this area. For example, the Windows Store provides a large amount of white space around the app list where the user can place their fingers and zoom in or out.
- Use structure and semantics that are intrinsic to the view.
    - Use the group names for items in a grouped collection.
    - Use sort ordering (such as chronological ordering for dates or alphabetical ordering for a list of names) for an ungrouped, but sorted, collection.
    - Use pages to represent a document collection.
- Ensure that the item layout and panning direction does not change based on zoom level. Layouts and panning interactions should be consistent and predictable across zoom levels.

© 2014 Microsoft. All rights reserved. | May 13, 2014

- Limit the number of pages (or screens) in the zoomed-out mode to three. Semantic Zoom enables a user to jump quickly to content. Introducing excessive panning destroys this benefit.
- Don't use Semantic Zoom to change the scope of the content. For example, a photo album should never switch to a folder view in File Explorer.
- Don't set a border on the Semantic Zoom control's child controls. If you set borders on both the Semantic Zoom and its child controls, the Semantic Zoom border and the border of the child control that is in view will both be visible. When zooming in or out, the child control's borders are scaled along with the content and won't look good. Set a border only on the Semantic Zoom control.

Additional usage guidance

Optical zooming alters the scale and magnification of content. Semantic zooming switches the semantics (the meaning) of the content presented between items grouped by key, and a list of those keys. Consider using a semantic zoom control whenever you want to display a long list of grouped data (for example, exercises grouped by body part, or names grouped by initial).

Scrolling from names beginning with 'A' to a name beginning with 'Z' may take a large number of swipe gestures. With a semantic zoom control, it can take as few as two taps: one to zoom out to the list of initials, and a second to zoom back into the 'Z's.

The semantic zoom manages zooming between its two views. A view is either a list view or a grid view, containing the items or the group keys as appropriate.

Semantic zoom uses two distinct modes of classification (or zoom levels) for organizing and presenting the content: one low-level (or zoomed in) mode that is typically used to display items in a flat, all-up structure and another, high-level (or zoomed out) mode that displays items in groups and enables a user to quickly navigate and browse through the content.

The semantic zoom interaction is performed with the pinch and stretch gestures (moving the fingers farther apart zooms in and moving them closer together zooms out), or by holding the Ctrl key down while scrolling the mouse scroll wheel, or by holding the Ctrl key down (with the Shift key, if no numeric keypad is available) and pressing the plus (+) or minus (-) key.

Examples of apps that could use semantic zoom include:

- An address book that organizes contacts alphabetically (or by some other means) and presents the data using the letters of the alphabet. The user could then zoom in on a letter to see the contacts associated with that letter.
- A photo album that organizes images by metadata (such as date taken). The user could then zoom in on a specific date to display the collection of images associated with that date.
- A product catalog that organizes items by category.
- Other examples of Semantic Zoom layouts:

Zoomed in	Zoomed out

© 2014 Microsoft. All rights reserved. | May 13, 2014



Navigating with semantic zoom

While navigating content is possible through panning and scrolling alone, powerful navigational and organizational capabilities are possible when paired with semantic zoom.

Panning and scrolling are useful for small sets of content and short distances. However, navigation quickly becomes cumbersome for large sets of content. Semantic zoom greatly reduces the perception of traveling long distances when navigating through large amounts of content and provides quick and easy access to locations within the content.

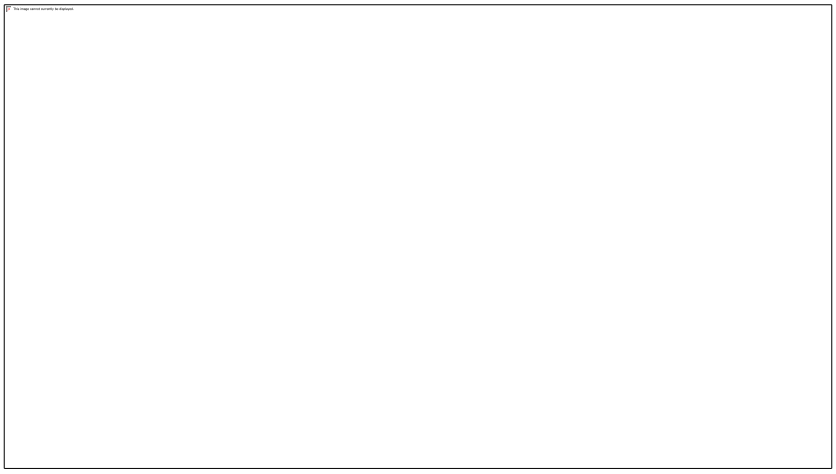
© 2014 Microsoft. All rights reserved. | May 13, 2014

Note

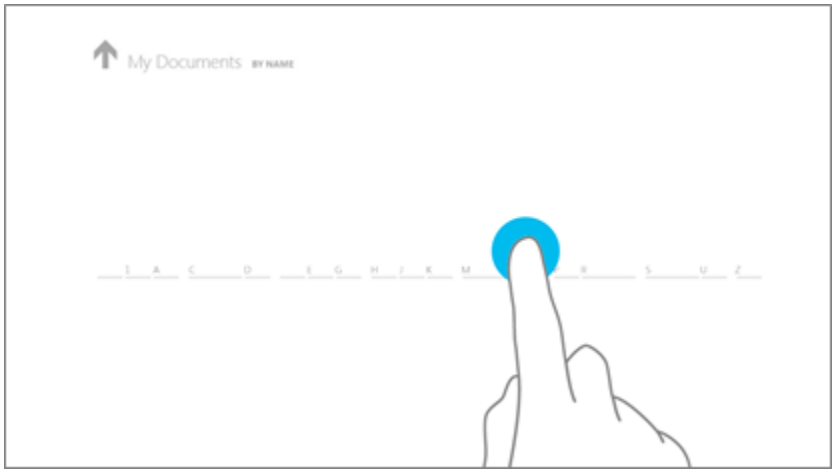
Semantic zoom should not be confused with optical zoom. While they share the same interaction and basic behavior (displaying more or less detail based on a zoom factor), optical zoom refers to the adjustment of magnification for a content area or object such as a photograph.

- **Scroll jump**

  Tapping the content in zoomed-out mode will zoom the view and pan to the tapped point, as shown in these three diagrams.



Zoomed out, the entire content can be a touch target.



A tap on a section of the content.

© 2014 Microsoft. All rights reserved. | May 13, 2014



Zoomed in and panned to the tapped area.

- **Transitions**

  A smooth cross-fade and scale animation is used for the transition from one semantic zoom level to another. This is the default Windows Touch behavior and cannot be customized.

Considerations and recommendations

You are responsible for defining the two semantic levels in your apps.

Consider these questions when designing the zoomed-out mode:

- How should the structure and presentation of the info change based on the zoom level?
- Would hints, or "signposts," be helpful for navigating the data?
- What amount of content provides a useful semantic view while minimizing panning and scrolling?

These considerations are often in conflict with each other. You want rich content with lots of info so that users know where they are jumping to, but you need to balance this info with the total length of the semantic level. If users need to pan a lot in the zoomed-out mode, you lose the main benefit provided by Semantic Zoom—quick navigation.

Hands-on labs for Windows 8

If you'd like to try working with Semantic Zoom and other key Windows 8 features, download the hands-on labs for Windows 8. These labs provide a modular, step-by-step introduction to creating a sample Windows Store app in the programming language of your choice (JavaScript and HTML or C# and XAML).

Appearance and interaction

© 2014 Microsoft. All rights reserved. | May 13, 2014

The appearance of a semantic zoom control depends on which of its views is being shown (each view is either a list view control or a grid view control). When zoomed-in, the semantic zoom appears as a list of items grouped under key headers; when zoomed-out, it appears as a list of keys.

Tapping an item while zoomed-in either selects the item or navigates to its details page. When zoomed-out, tapping a group key zooms in with that group scrolled into view.

Windows: The zoom interaction is performed with the pinch and stretch gestures (moving the fingers closer together zooms out and moving them farther apart zooms in), or by holding the Ctrl key down while scrolling the mouse scroll wheel, or by holding the Ctrl key down (with the Shift key, if no numeric keypad is available) and pressing the plus (+) or minus (-) key.

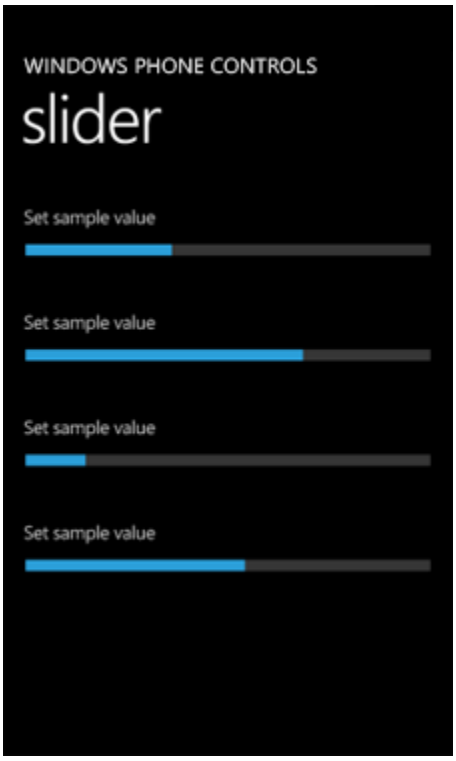
Windows Phone: When zoomed-in, tapping a group key header zooms out.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for sliders



Windows app: slider controls



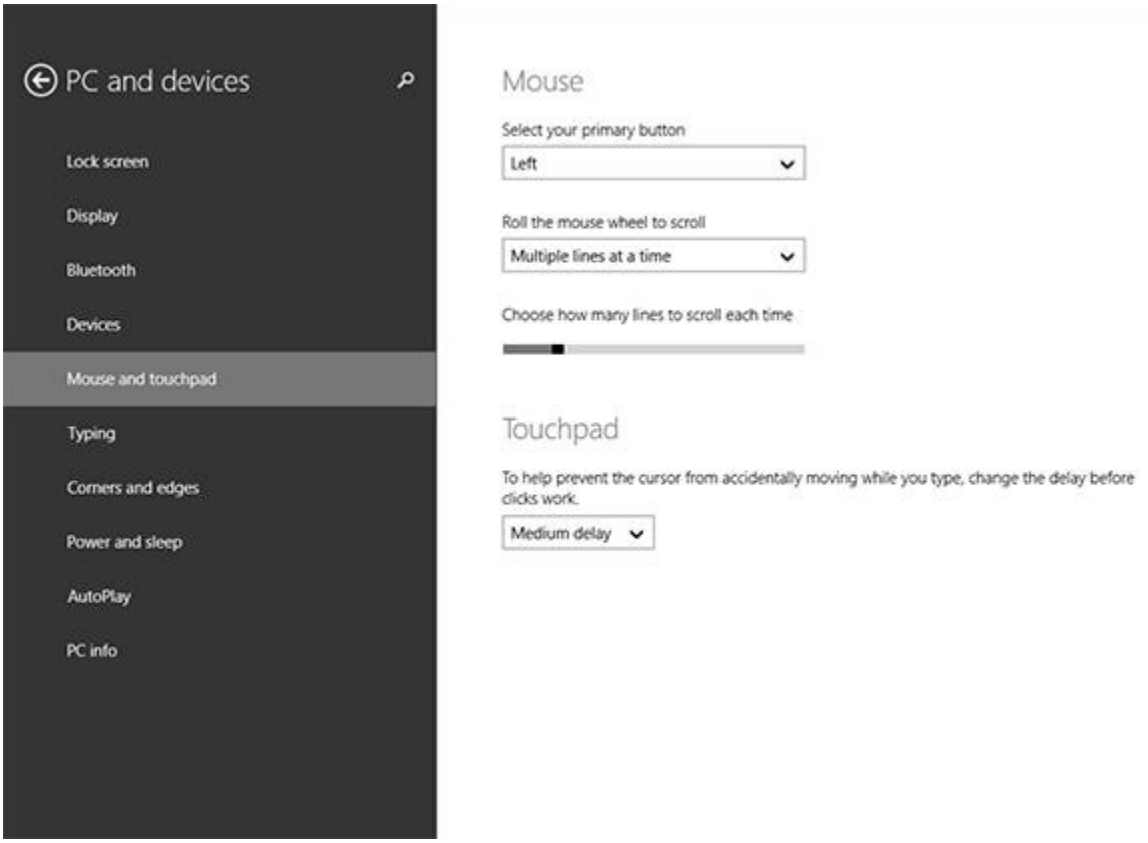
Windows Phone app: slider controls

Description

A slider control (also known as a range control) lets the user set a value in a given range by tapping—or scrubbing back and forth—on a track.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Examples



Is this the right control?

Use a slider when you want your users to be able to set defined, contiguous values (such as volume or brightness) or a range of discrete values (such as screen resolution settings).

A slider is a good choice when you know that users think of the value as a relative quantity, not a numeric value. For example, users think about setting their audio volume to low or medium—not about setting the value to 2 or 5.

Don't use a slider for binary settings. Use a toggle switch instead.

Here are some additional factors to consider when deciding whether to use a slider:

- **Does the setting seem like a relative quantity?** If not, use radio buttons or select controls.
- **Is the setting an exact, known numeric value?** If so, use a numeric text box.
- **Would a user benefit from instant feedback on the effect of setting changes?** If so, use a slider. For example, users can choose a color more easily by immediately seeing the effect of changes to hue, saturation, or luminosity values.
- **Does the setting have a range of four or more values?** If not, use radio buttons.

© 2014 Microsoft. All rights reserved. | May 13, 2014

- **Can the user change the value?** Sliders are for user interaction. If a user can't ever change the value, use read-only text instead.

If you are deciding between a slider and a numeric text box, use a numeric text box if:

- Screen space is tight.
- The user is likely to prefer using the keyboard.

Use a slider if:

- Users will benefit from instant feedback.

Dos and don'ts

- Size the control so that users can easily set the value they want. For settings with discrete values, make sure the user can easily select any value using the mouse. Make sure the endpoints of the slider always fit within the bounds of a view.
- Give immediate feedback while or after a user makes a selection (when practical). For example, the Windows volume control beeps to indicate the selected audio volume.
- Use labels to show the range of values. Exception: If the slider is vertically oriented and the top label is Maximum, High, More, or equivalent, you can omit the other labels because the meaning is clear.
- Disable all associated labels or feedback visuals when you disable the slider.
- Consider the direction of text when setting the flow direction and/or orientation of your slider. Script flows from left to right in some languages, and from right to left in others.
- Don't use a slider as a progress indicator.
- Don't change the size of the slider thumb from the default size.
- Don't create a continuous slider if the range of values is large and users will most likely select one of several representative values from the range. Instead, use those values as the only steps allowed. For example if time value might be up to 1 month but users only need to pick from 1 minute, 1 hour, 1 day or 1 month, then create a slider with only 4 step points.

Additional usage guidance

Choosing the right layout: horizontal or vertical

You can orient your slider horizontally or vertically. Use these guidelines to determine which layout to use.

- Use a natural orientation. For example, if the slider represents a real-world value that is normally shown vertically (such as temperature), use a vertical orientation.
- If the control is used to seek within media, like in a video app, use a horizontal orientation.
- When using a slider in page that can be panned in one direction (horizontally or vertically), use a different orientation for the slider than the panning direction. Otherwise, users might swipe the slider and change its value accidentally when they try to pan the page.
- If you're still not sure which orientation to use, use the one that best fits your page layout.

© 2014 Microsoft. All rights reserved.

Guidelines for the range direction

The range direction is the direction you move the slider when you slide it from its current value to its max value.

- For vertical slider, put the largest value at the top of the slider, regardless of reading direction. For example, for a volume slider, always put the maximum volume setting at the top of the slider. For other types of values (such as days of the week), follow the reading direction of the page.
- For horizontal styles, put the lower value on the left side of the slider for left-to-right page layout, and on the right for right-to-left page layout.
- The one exception to the previous guideline is for media seek bars: always put the lower value on the left side of the slider.

Guidelines for steps and tick marks

- Use step points if you don't want the slider to allow arbitrary values between min and max. For example, if you use a slider to specify the number of movie tickets to buy, don't allow floating point values. Give it a step value of 1.
- If you specify steps (also known as snap points), make sure that the final step aligns to the slider's max value.
- Use tick marks when you want to show users the location of major or significant values. For example, a slider that controls a zoom might have tick marks for 50%, 100%, and 200%.
- Show tick marks when users need to know the approximate value of the setting.
- Show tick marks and a value label when users need to know the exact value of the setting they choose, without interacting with the control. Otherwise, they can use the value tooltip to see the exact value.
- Always show tick marks when step points aren't obvious. For example, if the slider is 200 pixels wide and has 200 snap points, you can hide the tick marks because users won't notice the snapping behavior. But if there are only 10 snap points, show tick marks.

Guidelines for labels

- **Slider labels**

  The slider label indicates what the slider is used for.

  - Use a label with no ending punctuation.
  - Position labels above the slider when the slider is in a form that places its most of its labels above their controls.
  - Position labels to the sides when the slider is in a form that places most of its labels to the side of their controls.
  - Avoid placing labels below the slider because the user's finger might occlude the label when the user touches the slider.
- **Range labels**

  The range, or fill, labels describe the slider's minimum and maximum values.

© 2014 Microsoft. All rights reserved. | May 13, 2014

    - Label the two ends of the slider range, unless a vertical orientation makes this unnecessary.
    - Use only one word, if possible, for each label.
    - Don't use ending punctuation.
    - Make sure these labels are descriptive and parallel. Examples: Maximum/Minimum, More/Less, Low/High, Soft/Loud.
- **Value labels**

  A value label displays the current value of the slider.

    - If you need a value label, display it below the slider.
    - Center the text relative to the control and include the units (such as pixels).
    - Since the slider's thumb is covered during scrubbing, consider showing the current value some other way, with a label or other visual. A slider setting text size could render some sample text of the right size beside the slider.

Appearance and interaction

A slider is composed of a track and a thumb. The track is a bar (which can optionally show various styles of tick marks) representing the range of values that can be input. The thumb is a selector, which the user can position by either tapping the track or by scrubbing back and forth on it.

A slider has a large touch target. To maintain touch accessibility, a slider should be positioned far enough away from the edge of the display.

When you're designing a custom slider, consider ways to present all the necessary info to the user with as little clutter as possible. Use a value label if a user needs to know the units in order to understand the setting; find creative ways to represent these values graphically. A slider that controls volume, for example, could display a speaker graphic without sound waves at the minimum end of the slider, and a speaker graphic with sound waves at the maximum end.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for time pickers



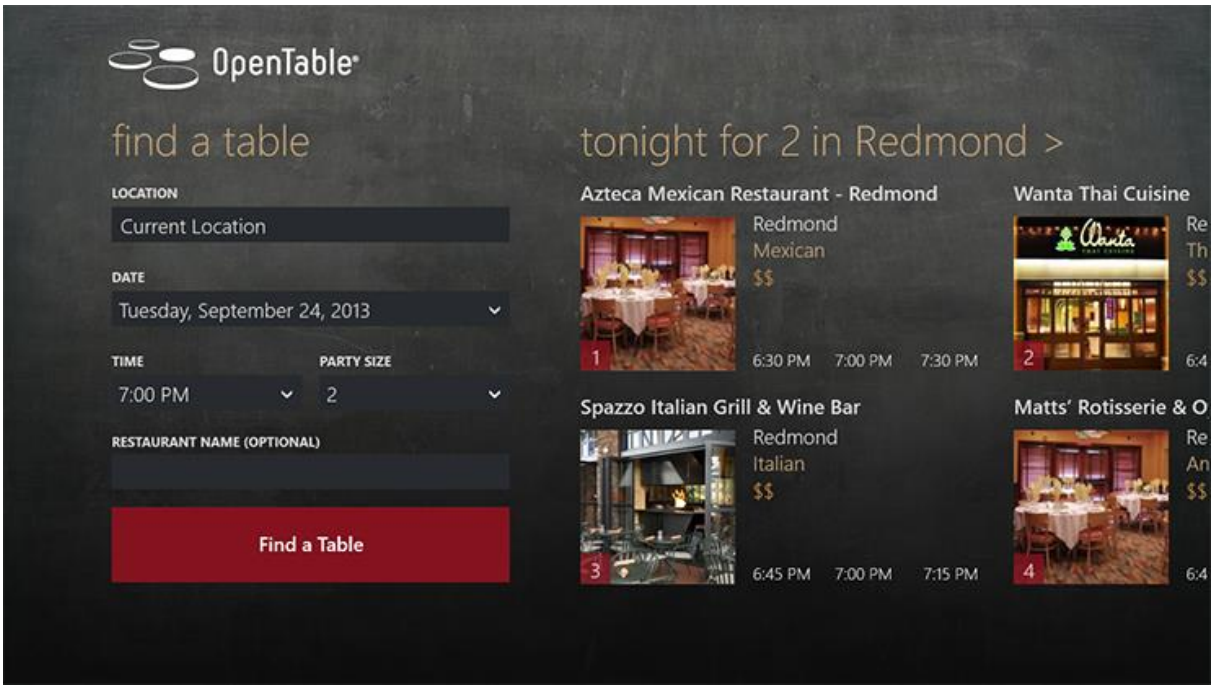


Description

The time picker provides a standardized way to let users pick a localized time by using touch, mouse, or keyboard input.

Use the time picker when a user needs to select a single time. The time picker is a good option for conserving real estate because the screen space used is fixed and independent of the number of choices.

Example



© 2014 Microsoft. All rights reserved. | May 13, 2014

Dos and don'ts

- Use the time picker when a user needs to select a single time.
- Use a time picker to enable the selection of times when you need to use space efficiently.
- Don't use a time picker to display the current time. It's a static display that's meant to be set by the user.
- Don't use the time picker for performing commands, displaying other windows such as dialog boxes, or dynamically displaying other controls.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for toggle switch controls

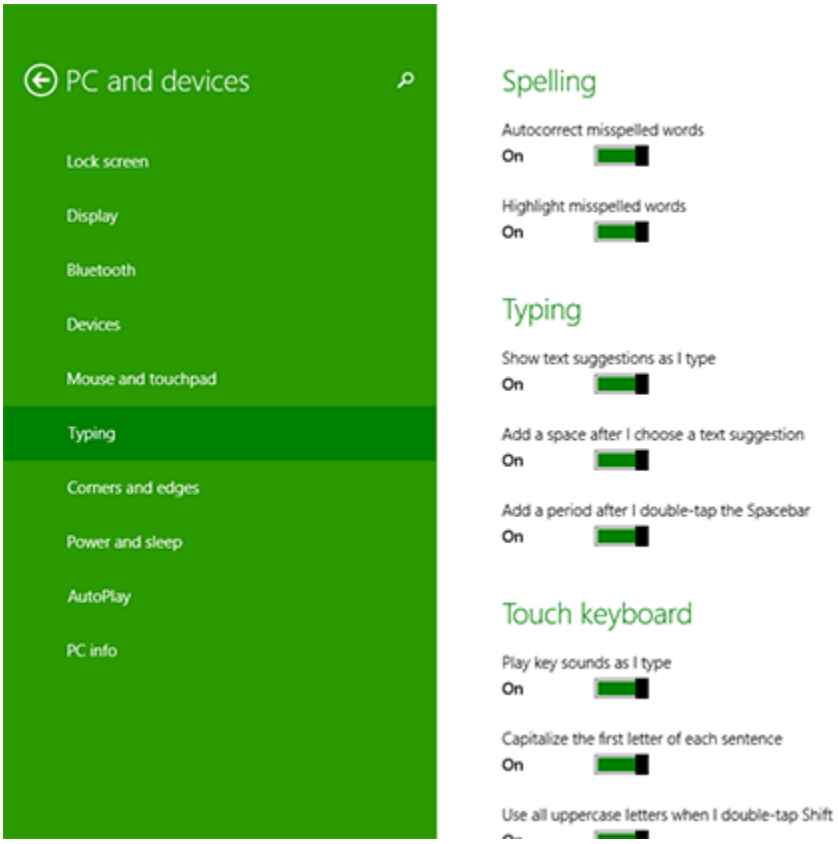




Description

The toggle switch mimics a physical switch that allows users to turn things on or off. The control has two states: on (checked is **true**) and off (checked is **false**). Use these guidelines when adding toggle switch controls to your Windows Store app.

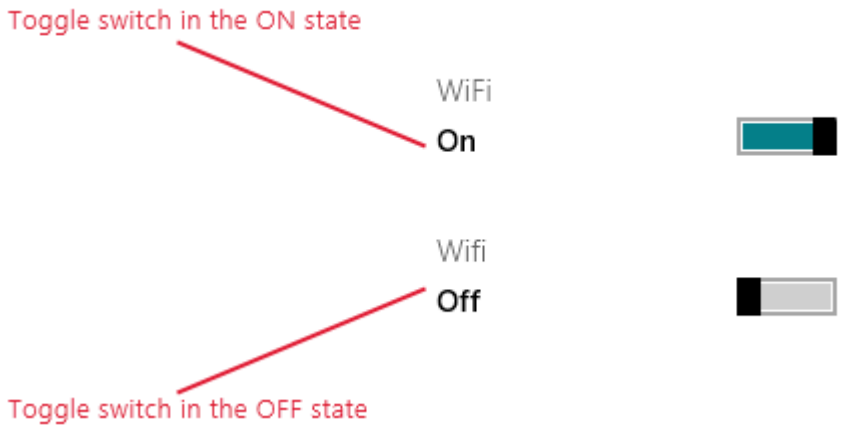
Example



© 2014 Microsoft. All rights reserved. | May 13, 2014

Is this the right control?

Use a toggle switch for binary operations that become effective immediately after the user changes it. For example, use a toggle switch to turn services or hardware components on or off.



A good way to test whether you should use toggle switch is to think about whether you would use a physical switch to perform the action in your context.

After the user toggles the switch on or off, you perform the corresponding action immediately.

Choosing between toggle switch and check box

Choosing between toggle switch and check box

In some cases, you could use either a toggle switch or check box. Follow these guidelines to choose between the two.

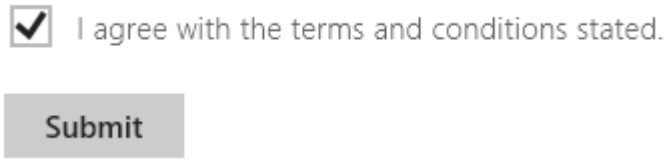
- Use a toggle switch for binary settings when changes become effective immediately after the user changes them.



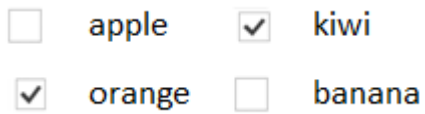
© 2014 Microsoft. All rights reserved. | May 13, 2014

It's clear in the toggle switch case that the wireless is set to on. But in the checkbox case, users need to think about whether the wireless is on now or whether they need to check the box to turn wireless on.

- Use a checkbox when the user has to perform extra steps for changes to be effective. For example, if the user must click a "submit" or "next" button to apply changes, use a check box.



- Use check boxes or a List View when the user can select multiple items:

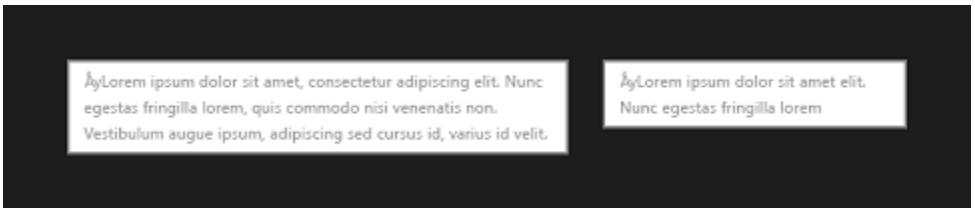


Dos and don'ts

- Replace the On and Off labels when there are more specific labels for the setting. If there are short (3-4 characters) labels that represent binary opposites that are more appropriate for a particular setting, use them. For example, you might use "Show/Hide" if the setting is "Show images." Using more specific labels can help when localizing the UI.
- Don't replace the On or Off label unless you must. You should use the default labels unless there are labels that are more specific for the setting.
- Don't use labels longer than 3 or 4 characters.

© 2014 Microsoft. All rights reserved. | May 13, 2014

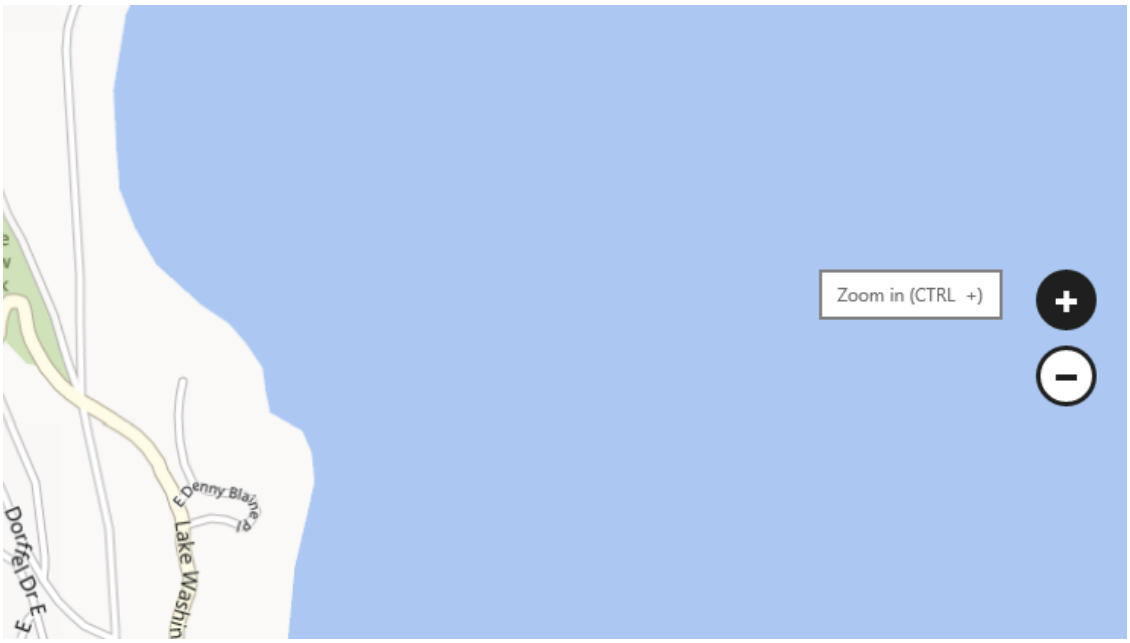
Guidelines for tooltips



Description

A tooltip is a short description that is linked to another control or object. Tooltips help users understand unfamiliar objects that aren't described directly in the UI. They display automatically when the user presses and holds or hovers the mouse pointer over a control. The tooltip disappears when the user moves the finger, the mouse pointer, or a pen pointer.

Example



Is this the right control?

A tooltip is a short description that is linked to another control or object. Tooltips help users understand unfamiliar objects that aren't described directly in the UI. They display automatically

© 2014 Microsoft. All rights reserved.

when the user presses and holds or hovers the mouse pointer over a control. The tooltip disappears when the user moves the finger, the mouse pointer, or a pen pointer.

Use a tooltip to reveal more info about a control before asking the user to perform an action. You can also use a tooltip to show the item under the finger during touchdown, so that users know where they are touching. (You should try to find other ways to disambiguate first, such as use a larger control, more spacing, or styling the control's active/hover state.)

When should you use a tooltip? To decide, consider these questions:

- **Is the info displayed based on pointer hover?**

  If not, use another control. Display tips only as the result of user interaction—never display them on their own.

- **Does a control have a text label?**

  If not, use a tooltip to provide the label. It is a good programming practice to label most controls and for these you don't need tooltips. Toolbar controls and command buttons with graphic labels need tooltips.

- **Does an object benefit from a more detailed description or further info?**

  If so, use a tooltip. But the text must be supplemental—that is, not essential to the primary tasks. If it is essential, put it directly in the UI so that users don't have to discover or hunt for it.

- **Is the supplemental info an error, warning, or status?**

  If so, use another UI element, such as a flyout.

- **Do users need to interact with the tip?**

  If so, use another control. Users can't interact with tips because moving the mouse makes them disappear.

- **Do users need to print the supplemental info?**

  If so, use another control.

- **Will users find the tips annoying or distracting?**

  If so, consider using another solution—including doing nothing at all. If you do use tips where they might be distracting, allow users to turn them off.

Here are some examples of good ways to use tooltips:

© 2014 Microsoft. All rights reserved. | May 13, 2014

- Showing the day of the week when users touch a date in a calendar.
- Showing a preview of the linked website when users touch a hyperlink.

Dos and don'ts

- Use tooltips sparingly (or not at all). Tooltips are an interruption. A tooltip can be as distracting as a pop-up, so don't use them unless they add significant value.
- Keep the tooltip text concise. Tooltips are perfect for short sentences and sentence fragments. Large blocks of text are difficult to read and overwhelming.
- Create helpful, supplemental tooltip text. Tooltip text must be informative. Don't make it obvious or just repeat what is already on the screen. Because tooltip text isn't always visible, it should be supplemental info that users don't have to read. Communicate important info using self-explanatory control labels or in-place supplemental text.
- Use images when appropriate. Sometimes it's better to use an image in a tooltip. For example, when the user touches a hyperlink, you can use a tooltip to show a preview of the linked page.
- Don't use a tooltip to display text already visible in the UI. For example, don't put a tooltip on a button that shows the same text of the button unless touching the button blocks its text.
- Don't put interactive controls inside the tooltip.
- Don't put images that look like they are interactive inside the tooltip.

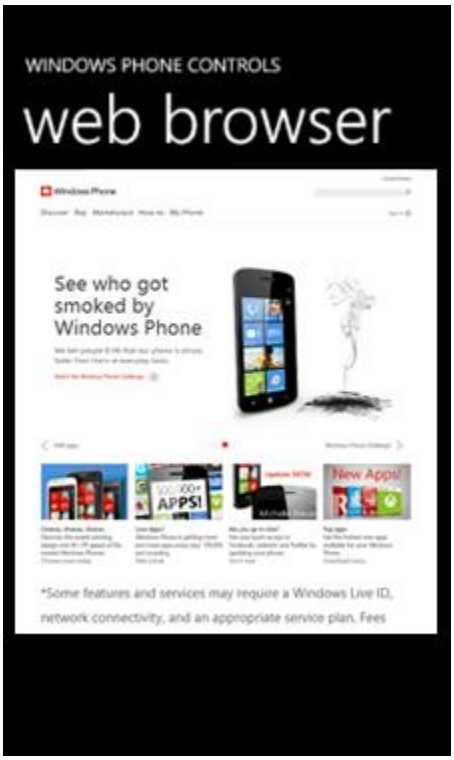
Additional usage guidance

Tooltips should be used sparingly, and only when they are adding distinct value for the user who is trying to complete a task. One rule of thumb is that if the information is available elsewhere in the same experience, you do not need a tooltip. A valuable tooltip will clarify an unclear action.

Use a tooltip to reveal more info about a control before asking the user to perform an action. You can also use a tooltip to show the item under the finger during touchdown, so that users know where they are touching.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for Web views



Windows Phone app: web view control

Description

A web view control embeds a view into your app that looks and behaves like Internet Explorer. Hyperlinks can also appear and function in a web view control.

Dos and don'ts

- Make the text size appropriate for the form factor. For example, for Windows Phone the minimum is 15 points.
- Make sure the website loaded is formatted correctly for the device, and uses colors, typography, and navigation that are consistent with the rest of your app. For more info, see Web development for Windows Phone.
- Input fields should be sized appropriately, since users may not realize that they can zoom in to enter text.

© 2014 Microsoft. All rights reserved. | May 13, 2014

- If a web view doesn’t look like the rest of your app consider alternative controls or ways to accomplish relevant tasks. If your web view matches the rest of your app users will see it all as one seamless experience.

Additional usage guidance

You can embed a web view control for a number of reasons, as described at WebBrowser control for Windows Phone.

Use a web view control to display richly formatted HTML content from a remote web server, dynamically generated code, or content files in your app package. Rich content can also contain script code and communicate between the script and your app's code.

When you’re developing an app that uses the web view control, consider the best practices and info regarding security at WebBrowser control security best practices for Windows Phone.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for files, data, and connectivity



This section includes guidance for accessing files and data from your app, and for connecting users to the Web and to their accounts.

You can allow users to access folders, files, and data from you app. To allow users to access data from an account or in cloud services like OneDrive, you can provide them with a signed-in experience. Include application data roaming in your app to give a seamless experience across devices.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for connection usage data



Consider the following recommendations when using Windows Runtime Network Information APIs in your connected app.

Modifying app behavior according to network cost type

Although Windows 8 automatically provides connectivity options when a device detects new networks, there is no guarantee of a seamless transition from existing connections to new networks for every operation. A Windows Store app that connects to destinations on the Web should use Network Information APIs to obtain cost information and status change events for the network on which they are sending and receiving data.

Use the **NetworkCostType** value indicated for each connection to modify its behavior appropriately:

Network Cost Type	Recommended App Behavior
Unrestricted	• Use the network connection freely.
Variable/Approaching Data Cap	• Delay or schedule lower priority operations until an unrestricted network is available. • When streaming content to a user, such as a movie or a video, use a lower bit-rate. For example, if your app is streaming HD-Quality video, stream Standard Definition when on a metered network. • Use less bandwidth. For example, switching to header-only mode when receiving emails. • Use the network less frequently. An example solution is to reduce the frequency of any polling operations for syndicating news feeds, refreshing content for a website, or getting web notifications. • Allow users to explicitly choose to stream HD-Quality video, retrieve full emails, download lower priority updates, etc., rather than doing so by default. • Explicitly ask for user permission prior to using the network.
Unknown	• If the network cost type is unknown, then treat it as an unrestricted network.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Maintaining a reliable connection to the Web

One of the most fundamental ways your app can demonstrate agility in the network space is by maintaining a consistent level of quality when interfacing with the Web. This can be done by leveraging the information provided by the connection profile and subsequent network status change notifications and by identifying available networks that meet current requirements.

All Windows Store apps should do the following to support Web connectivity:

1. Call **GetInternetConnectionProfile** to check the cost of connecting to the Internet.
2. Register for network status change notifications for the connection.
3. Initialize the network operation on the connection.
4. When a network status changed notification is received, the cost/connectivity options available may have changed. The app should:
    - Check the cost of connecting to the Internet. If the cost characteristic has changed from unrestricted to metered or from metered to unrestricted, retry the network operation. Windows will automatically use the lowest cost network available.
    - If the cost characteristic of connecting to the Internet has not changed, but a cost related notification is received (for example, >80% cap consumed, variable cost, roaming, etc.), adapt behavior as suggested in the **NetworkCostType** table above.
5. If an error indicates that the connection went away, the app should:
    - Check the cost of connecting to the Internet via another available network. Follow the guidelines provided in the **NetworkCostType** table above.
    - Retry the operation; if this fails, then wait for a **NetworkStatusChange** notification.

Debugging and troubleshooting your connected app

Network glitches can cause applications to hang, crash, or show non-actionable dialog boxes and confusing error messages to the users. Debugging these errors can be difficult because errors can occur anywhere within the networking stack.

All Windows Store apps that use the network either directly (using sockets) or indirectly (using an API that eventually uses the network) are impacted. The ideal situation is for the operating system to handle error conditions automatically on behalf of the developer and when that does not work applications should be prepared to deal with errors.

All connected Windows Store apps need to do the following:

- When a networking error happens, retry the operation if appropriate. For instance, do not retry the operation if authentication fails, but retry the operation if the network you were communicating on went away because another one may be available. Many errors can simply go away when retrying the operation. On retrying, follow the guidelines specified earlier in Reacting to network status changes.
- Ensure that you use asynchronous APIs so that there is no blocking call on the UI thread. In other words, if a networking operation takes a long time to complete or there is an error,

© 2014 Microsoft. All rights reserved.

your application should not hang. Do not emulate synchronous behavior on top of the asynchronous nature of the Windows Runtime.
- Test your application in various networking environments with activities like disconnecting or reconnecting to your network, suspending or resuming, or switching from one network to another.
- When you are testing your application and find errors that are not immediately obvious, enable ETW tracing.

Security Considerations

The following articles provide guidance for writing secure C++ code.

- Patterns & Practices Security Guidance for Applications

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for creating custom data formats



Users share a variety of information when they're online. Successful apps take the time to analyze the types of information that users are most likely to share with others, and package that information so that the receiving apps can process it correctly. In many cases, the information users want to share falls into one of the six standard formats that Windows supports. However, there are some instances in which having a more targeted data type can create a better user experience. For these situations, your app can support custom data formats.

Example

To illustrate how to think about creating a custom format, consider the following example.

At the fictional company, Fabrikam, a developer writes an app that shares files stored online. One option would be to use stream-backed StorageItems, but that could be time-consuming and inefficient because the target app would end up downloading the files locally to read them. Instead, the developer decides to create a custom format for use with these file types.

First, the developer considers the definition of the new format. In this case, it's a collection of any file type (document, image, and so on) that is stored online. Because the files are on the web instead of the local machine, the developer decides to name the format WebFileItems.

Next, the developer needs to decide on the specifics of the format, and decides on the following:

- The format should consist of an **IPropertyValue** that contains an InspectableArray that represents the Uniform Resource Identifiers (URIs).
- The format must contain at least 1 item, but there is no limit as to how many items it can contain.
- Any valid URI is allowed.
- URIs that are not accessible outside of the source application's boundary (such as authenticated URIs) are discouraged.

With this information mapped out, the developer now has enough information to create and use a custom format.

Should my app use custom formats?

The share feature in Windows 8 supports six standard data formats:

© 2014 Microsoft. All rights reserved. | May 13, 2014

- text
- HTML
- bitmap
- StorageItems
- URI
- RTF

These formats are very versatile, which makes it easy for your app to quickly support sharing and receiving shared content. The drawback to these formats is that they don't always provide a rich description of the data for the receiving app. To illustrate this, consider the following string, which represents a postal address:

```
1234 Main Street, New York, NY 98208
```

An app can share this string using **DataPackage.setText**. But because the app that receives the text string won't know exactly what the string represents, it is limited in what it can do with that data. By using a custom data format, the source app can define the data being shared as a "place" using the http://schema.org/Place format. This gives the receiving app some additional information that it can use to process the information in way the user expects. Using existing schema formats allows your app to hook into a larger database of defined formats.

Custom formats can also help you provide more efficient ways of sharing data. For example, a user has a collection of photos stored on Microsoft OneDrive, and decides to share some of them on a social network. This scenario is tricky to implement using the standard formats, for a few reasons:

- You can only use the URI format to share one item at a time.
- OneDrive shares collections as stream-backed StorageItems. Using StorageItems in this scenario would require that your app download each picture, and then share them.
- Text and HTML let you provide a list of links, but the meaning of those links is lost—the receiving app won't know that these links represent the pictures the user wants to share.

Using an existing schema format or a custom format to share these pictures provides two main benefits:

- Your app can share the pictures faster, because you could create a collection of URIs, instead of downloading all the images locally.
- The receiving app would understand that these URIs represent images, and could process them accordingly.

Dos and don'ts

Defining a custom format

If you decide that your app can benefit from defining a custom format, there are a few things you should consider:

© 2014 Microsoft. All rights reserved.

- Be sure you understand the standard data formats, so you don't create a custom format unnecessarily. You should check to see if there is an existing format on http://www.schema.org that would suit your scenario. It is more likely that another app would use an existing format over your custom format, which means that a greater audience could achieve your desired end-to-end scenarios.
- Consider the experiences you want to enable. It's important that you think about what actions your users want to take, and what data format best supports those actions.
- Make the definition of your custom format available to other app developers.
- When naming your format, make sure the name matches the contents of the format. For example, `UriCollection` indicates that any URI is valid, while `WebImageCollection` indicates that it contains only URIs that point to online images.
- Carefully consider the meaning of the format. Have a clear understanding of what the format represents and how it should be used.
- Review the structure of the format. Think through whether the format supports multiple items or serialization, and what limitations the format has.
- Don't change your custom format after you publish it. Consider it like an API: elements might get added or deprecated, but backward-compatibility and long-term support are important.
- Don't rely on format combinations. For example, don't expect an app to understand that if sees one format, it should also look for a second format. Each format must be self-contained.

Adding custom formats to your app

After you define a custom format, follow these recommendations when adding the format to your app:

- Test the format with other apps. Make sure that the data is processed correctly from the source app to the target app.
- Stick to the intended purpose of the format. Don't use it in unintended ways.
- If you're writing a source app, provide at least one standard format as well. This ensures that users can share data with apps that don't support the custom format. The experience may not be as ideal for the user, but it is better than not letting them share data with the apps they want. You should also document your format online so that other applications can know to adopt it.
- If you're writing a target app, consider supporting at least one standard format. This way, your app can receive data from source apps—even if they don't use the custom format you prefer.
- If you want to accept a specific custom format from another app, particularly one from a different company, you should double-check to see that it is documented publicly online. Undocumented formats are more likely to change without notice, potentially creating inconsistencies or breaking your app.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Additional usage guidance

Choosing a data type

One of the most important decisions you'll make when defining a custom format is the WinRT data type used for transferring it between source and target applications. The **DataPackage** class supports several data types for a custom format:

- Any scalar type (integer, string, **DateTime**, and so on) through **IPropertyValue**.
- **IRandomAccessStream**
- **IRandomAccessStreamReference**
- **IUri**
- **IStorageItem**
- A homogenous collection of any of the above items

When defining a format, select the type appropriate for that data. It is very important that all consumers and recipients of this format use the same data type—even if there are different options—otherwise it may lead to unexpected data type mismatch failures in target applications.

If you choose string as the data type for your format, you can retrieve it on the target side using the **GetTextAsync(formatId)** form of the **GetTextAsync** function. This function performs data-type checking for you. Otherwise, you have to use **GetDataAsync**, which means you must protect against potential data-type mismatches. For example, if a source application provides a single URI, and the target attempts to retrieve it as a collection of URIs, a mismatch occurs. To help prevent these collisions, you can add code similar to this:

Populating the DataPackage

JavaScript

```
var uris = new Array();
uris[0] = new Windows.Foundation.Uri("http://www.msn.com");
uris[1] = new Windows.Foundation.Uri("http://www.microsoft.com");
var dp = new Windows.ApplicationModel.DataTransfer.DataPackage();
dp.setData("UriCollection", uris);
```

C#

```
System.Uri[] uris = new System.Uri[2];
uris[0] = new System.Uri("http://www.msn.com");
uris[1] = new System.Uri("http://www.microsoft.com");
DataPackage dp = new DataPackage();
dp.SetData("UriCollection", uris);
```

Retrieving the data

JavaScript

© 2014 Microsoft. All rights reserved. | May 13, 2014

```
if (dpView.contains("UriCollection")) {
    dpView.getDataAsync("UriCollection").done(function(uris) {
        // Array.isArray doesn't work - uris is projected from InspectableArray
        if (uris.toString() === "[object ObjectArray]") {
            var validUriArray = true;
            for (var i = 0; (true === validUriArray) && (i < uris.length); i++) {
                validUriArray = (uris[i] instanceof Windows.Foundation.Uri);
            }
            if (validUriArray) {
                // Type validated data
            }
        }
    }
}
```

C#

```
if (dpView.Contains("UriCollection"))
{
    System.Uri[] retUris = await dpView.GetDataAsync("UriCollection") as
System.Uri[];
    if (retUris != null)
    {
       // Retrieved Uri collection from DataPackageView
    }
}
```

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for file types and URIs



Description

In Windows 8, the relationship between apps and the file types they support differs from previous versions of Windows. By understanding these differences, you can provide a more consistent and elegant experience for your users.

Dos and don'ts

- Position the flyout near its point of invocation.

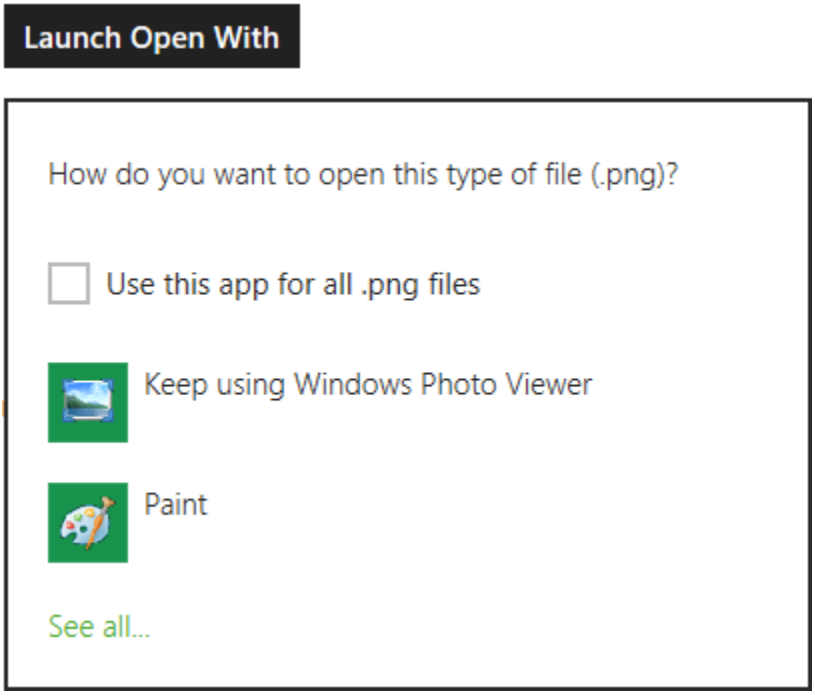
Additional usage guidance

Guidelines for Windows Store apps

When opening a file or URI, the user may need to use the **Open With** list to select which app to use as the default. Windows 8 implements this list as a flyout. Although you can't customize the contents of the **Open With** flyout, you can control its position in your app. Be sure to follow the guidelines and position the flyout near its point of invocation whenever possible.

Here's an example of an ideal way to use the flyout. Notice that it's located right next to the button that called it.

© 2014 Microsoft. All rights reserved. | May 13, 2014



You can present files and URIs however you see fit—typically as a thumbnail or a hyperlink. The primary action for these items should be **Open**. This option should invoke the default handler for the file or URI, which might result in showing the **Open With** flyout. (We recommend that you assume that the flyout appears in some cases and position it accordingly.)

If you choose to implement any secondary actions for files or URIs in your app, such as Save As or Download, consider letting the user choose an alternate app from an **Open With** flyout.

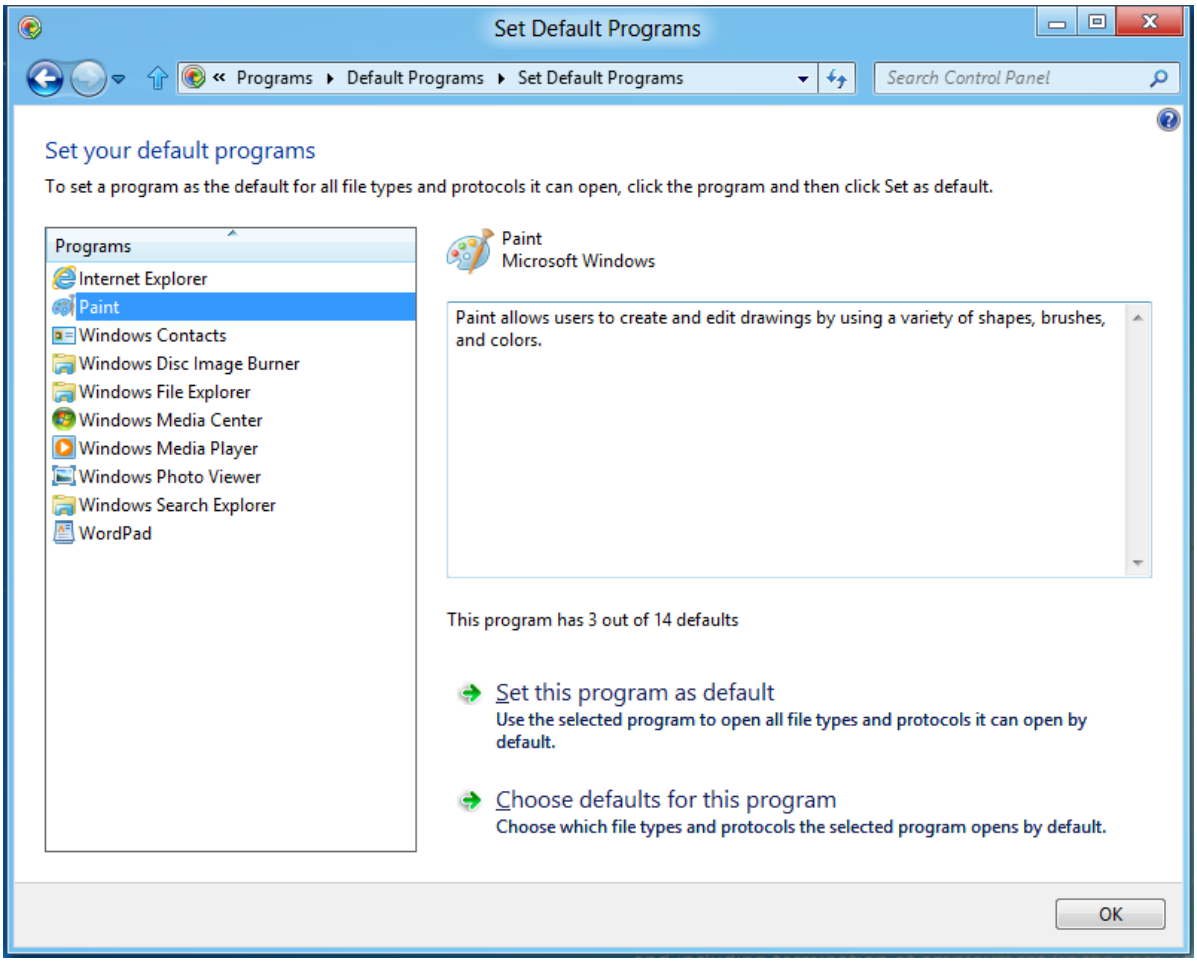
Remember, Windows Store apps can't set, change, or query default apps for file types and URIs, so you shouldn't try to add that functionality to your app.

The Association launching sample provides examples of how to implement the preceding scenarios in the recommended way.

Guidelines for desktop applications

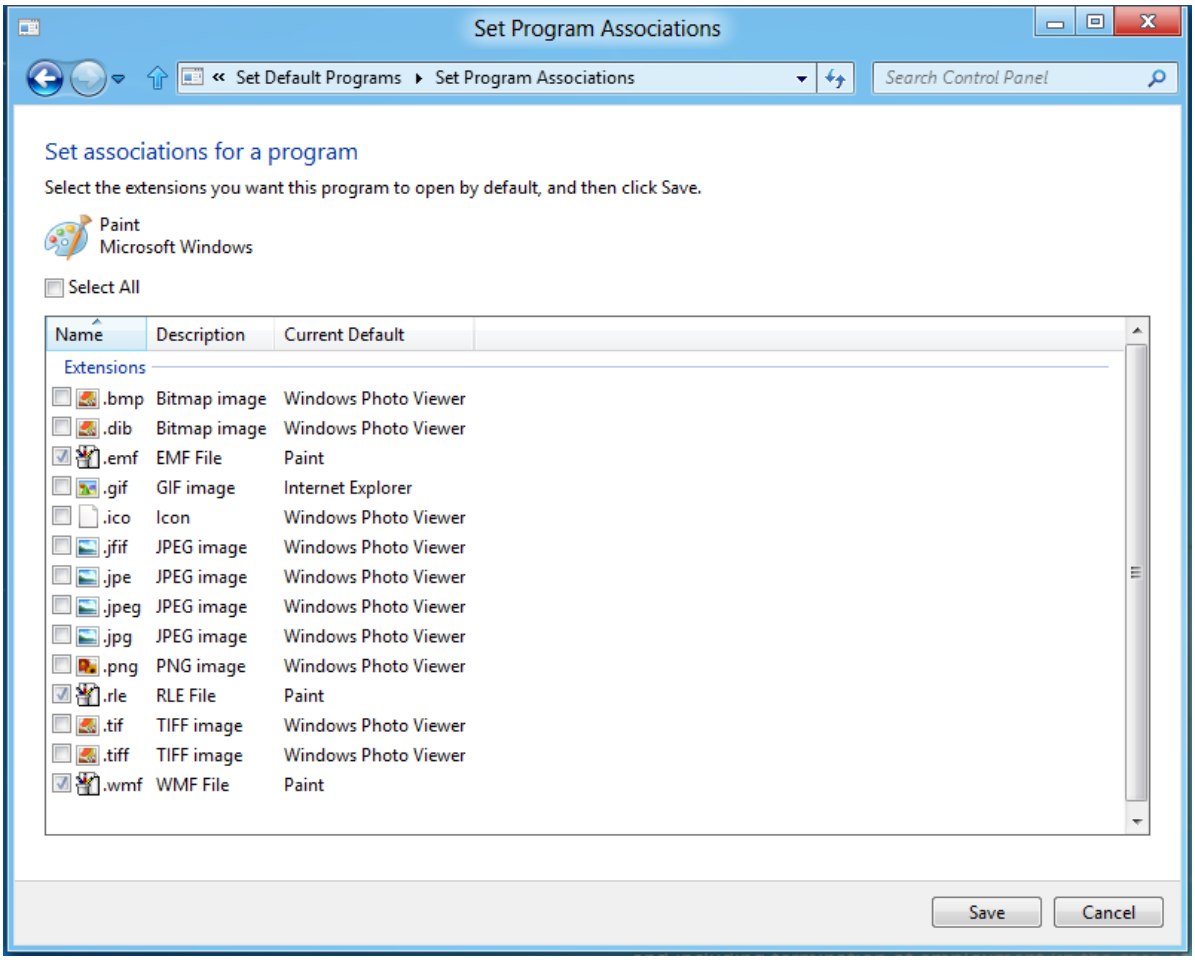
In Windows 8, apps no longer have the ability to set, change, or query the default handlers for file types and URI scheme names. When developing a Windows 8 desktop application version of your app, we recommend that you remove any user interface elements related to these functions. Instead, we recommend that you link to **Set Default Programs** in Control Panel. The UI for **Set Default Programs** is shown here.

© 2014 Microsoft. All rights reserved. | May 13, 2014



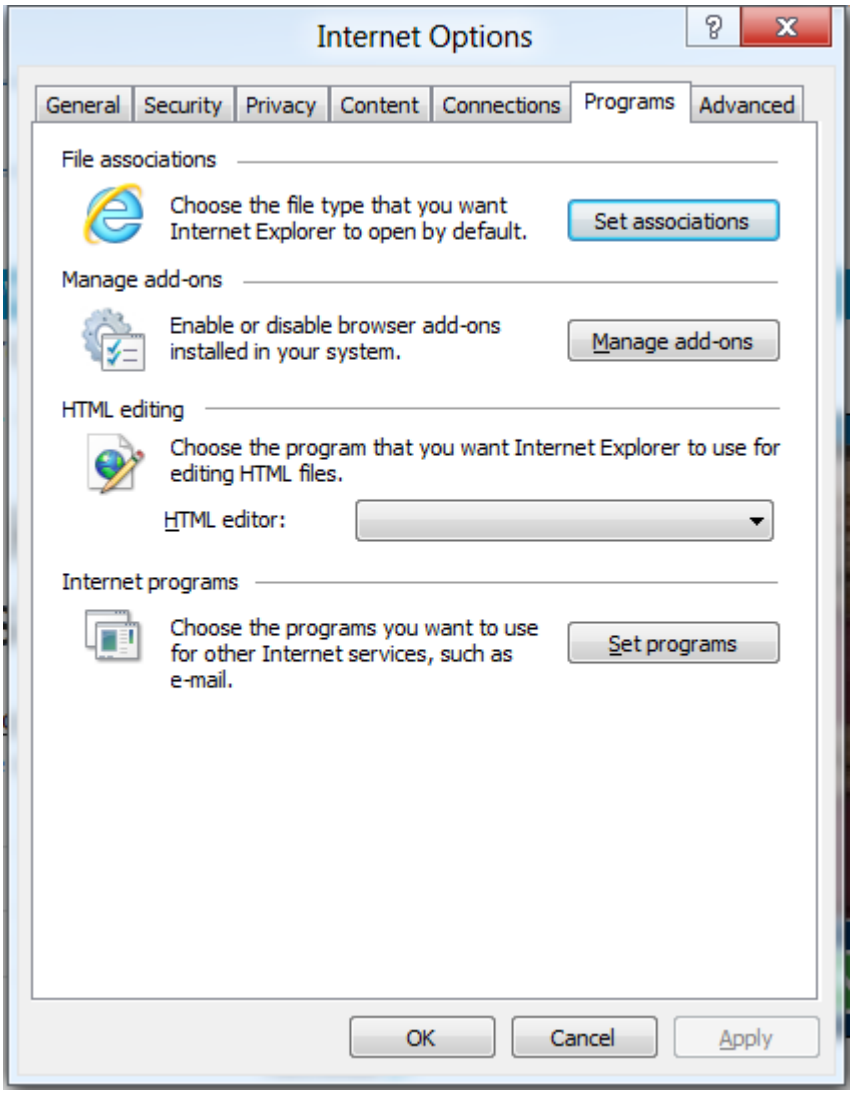
From here, users can select **Set this program as default**, which lets your app open all file types and URIs it can open by default. They can also select **Choose defaults for this program**, to select specific file types and URIs in the Set Program Associations UI, as shown here.

© 2014 Microsoft. All rights reserved. | May 13, 2014



The following is an example of how Microsoft Internet Explorer lets users access **Set Default Programs** in Control Panel.

© 2014 Microsoft. All rights reserved. | May 13, 2014



© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for login



Many Windows Store apps provide a personalized or premiere experience to users when they're logged in. Some apps require the user to log in to a registered account to get value from the app. Other apps provide a rich baseline experience for all users but enable enhanced features when the user is logged in.

Dos and don'ts

Login settings

- If your app offers a way for users to log in to the app, create an account, or manage account settings, you should enable users to swipe from the edge and change the login settings in the Settings flyout. This design ensures predictability and ease of access, regardless of where the user might be in the app's workflow. Also, this design frees room on the app's canvas that would otherwise be dedicated to login-related UI.

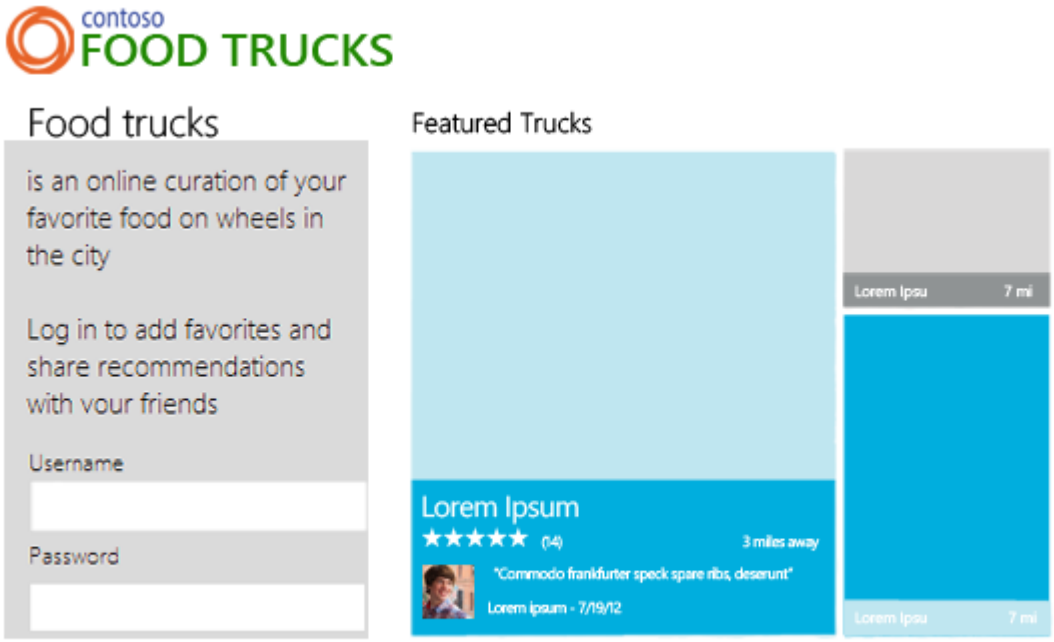
Required login

- If you app requires users to log in or create an account when the app first runs, design the first screen of the app to feature the login UI prominently. Once the user is logged in, there is no need for an onscreen login UI.
- Allow users to log out by using the Settings charm.

Recommended login

- If your app needs to elevate the login UI to the app's canvas, provide the controls inline in your content. This design ensures that users see the login option on the landing page when they first launch the app, but the login UI doesn’t get in the way of the overall experience.
- If your login UI is in a list view control, place it as the first section of the control in the app's landing page. As users browse the app’s content or views, the logon UI scrolls out of view, but still has a presence in the app.
- Make sure the user can always find the login UI in the Settings flyout.

© 2014 Microsoft. All rights reserved. | May 13, 2014

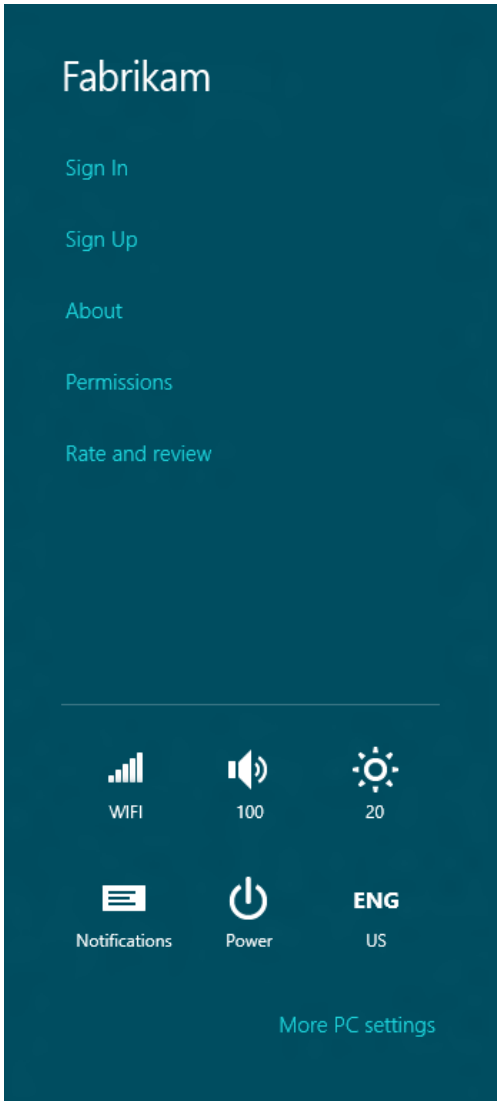


App with login UI hosted as the first section of the ListView control

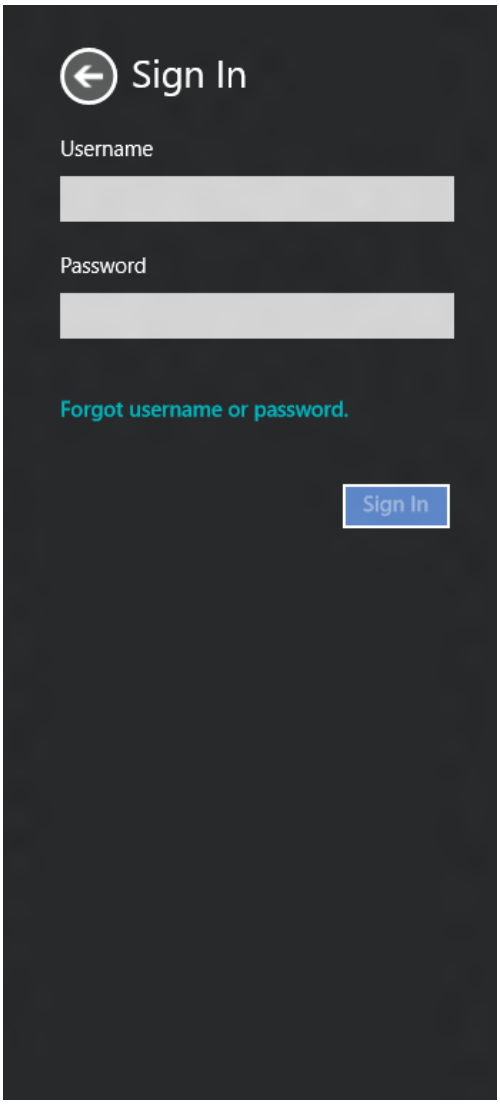
Optional login

- If logging in to your app is optional, place the login UI in the Settings flyout so that it doesn't distract from the content in the app or consume space on the canvas. Some apps provide great value without requiring that users are logged in. For example, a news app may offer an initial view of news articles that are interesting to many different readers. User can gain a lot of value from the app without logging in.
- If your app requires a login UI that's specific to some content in the app, indicate the need for a user login by putting a contextual login button on the page. The button launches the Settings flyout that hosts the login UI. For example, a news app may require login if the user wants to post a comment on a news article.

© 2014 Microsoft. All rights reserved. | May 13, 2014



Settings charm flyout login



Settings flyout login

Logout UI

- Place logout UI in the Settings flyout. Once the user has logged in to the app, logging out happens rarely, if the app is delivering meaningful personal content. Provide users with a familiar place to log out if necessary.

App personalization on login

- Design your app's logged in experience with content that makes your app personal and connected.

© 2014 Microsoft. All rights reserved. | May 13, 2014

- Once the user has logged in, update the app's content based on the user’s preferences, instead of showing generic content. Each app has a unique way of showing and delivering personal content, which enhances the user’s experience in a Windows Store app.
- Avoid putting a persistent UI on the app's canvas that shows identity. There may be times when the user who logs in to the app is different than the user who logged in to Windows. For example, a friend using your laptop or tablet may want to log in to their social network. Having different identities across the Start screen and the app is more confusing than rewarding for users.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for roaming app data



When you store app data using the roaming **ApplicationData** APIs, Windows replicates this data to the cloud and synchronizes the data to all other user devices on which the app is installed. Follow these guidelines when you design your Windows Store app to include roaming app data.

The application data sample shows how to use the **ApplicationData** APIs to store and retrieve data specific to each user and Windows Store app.

Should my app using roaming data?

Use roaming data to store a user's settings, preferences, and session info to create a cohesive app experience across multiple devices. Keep in mind that roaming data is associated with a user's Microsoft account. Roaming data will only sync if a user logs into her devices using the same Microsoft account and installs the app on several devices.

For example, if a user installs your app on a second device after installing it on another device, all preferences set on the first device are automatically applied to the app on the second device (assuming the user uses the same Microsoft account to log into both devices). Any future changes to these settings and preferences will also transition automatically, providing a uniform experience regardless of device. By storing session info as roaming data, you'll enable users to continue an app session that was closed or abandoned on one device when they transfer to another device.

Note: The **RoamingSettings**, **RoamingFolder**, and **RoamingStorageQuota** properties aren't implemented for Windows Phone.

Note: These kinds of files won't roam even if they are placed in the **RoamingFolder**:

- File types that behave like folders (for example, files with the .zip and .cab extensions)
- Files that have names with leading spaces
- Files that have names with these Unicode characters:

  e794, e795, e796, e7c7, e816, e817, e818, e81e, e826, e82b, e82c, e831, e832, e83b, e843, e854, e855, e864, e7e2, e7e3, and e7f3

- File paths (file name + extension) that are longer than 256 characters
- Empty folders
- Files with open handles

© 2014 Microsoft. All rights reserved. | May 13, 2014

Dos and don'ts

- Use roaming for user preferences and customizations, links, and small data files. For example, use roaming to preserve a user's background color preference across all devices.
- Use roaming to let users continue a task across devices. For example, roam app data like the contents of a drafted email or the most recently viewed page in a reader app.
- Handle the **DataChanged** event by updating app data. This event occurs when app data has just finished syncing from the cloud.
- Roam references to content rather than raw data. For example, roam a URL rather than the content of an online article.
- For important, time critical settings, use the *HighPriority* setting associated with **RoamingSettings**. For more info, see the Application data sample.
- Don't roam app data that is specific to a device. Some info is only pertinent locally, such as a path name to a local file resource. If you do decide to roam local information, make sure that the app can recover if the info isn't valid on the secondary device.
- Don't roam large sets of app data. There's a limit to the amount of app data an app may roam; use **RoamingStorageQuota** property to get this maximum. If an app hits this limit, no data can roam until the size of the app data store no longer exceeds the limit. When you design your app, consider how to put a bound on larger data so as to not exceed the limit. For example, if saving a game state requires 10KB each, the app might only allow the user store up to 10 games.
- Don't use roaming for data that relies on instant syncing. Windows doesn't guarantee an instant sync; roaming could be significantly delayed if a user is offline or on a high latency network. Ensure that your UI doesn't depend on instant syncing.
- Don't use roam frequently changing data. For example, if your app tracks frequently changing info, such as the position in a song by second, don't store this as roaming app data. Instead, pick a less frequent representation that still provides a good user experience, like the currently playing song.

Additional usage guidance

Roaming pre-requisites

Any user can benefit from roaming app data if they use a Microsoft account to log on to their device. However, users and group policy administrators can switch off roaming app data on a device at any time. If a user chooses not to use a Microsoft account or disables roaming data capabilities, she will still be able to use your app, but app data be local to each device.

Data stored in the **PasswordVault** will only transition if a user has made a device “trusted”. If a device isn't trusted, data secured in this vault will not roam.

Conflict resolution

Roaming app data is not intended for simultaneous use on more than one device at a time. If a conflict arises during synchronization because a particular data unit was changed on two devices, the system will always favor the value that was written last. This ensures that the app utilizes the

© 2014 Microsoft. All rights reserved. | May 13, 2014

most up-to-date information. If the data unit is a setting composite, conflict resolution will still occur on the level of the setting unit, which means that the composite with the latest change will be synchronized.

When to write data

Depending on the expected lifetime of the setting, data should be written at different times. Infrequently or slowly changing app data should be written immediately. However, app data that changes frequently should only be written periodically at regular intervals (such as once every 5 minutes), as well as when the app is suspended. For example, a music app might write the “current song” settings whenever a new song starts to play, however, the actual position in the song should only be written on suspend.

Excessive usage protection

The system has various protection mechanisms in place to avoid inappropriate use of resources. If app data does not transition as expected, it is likely that the device has been temporarily restricted. Waiting for some time will usually resolve this situation automatically and no action is required.

Versioning

App data can utilize versioning to upgrade from one data structure to another. The version number is different from the app version and can be set at will. Although not enforced, it is highly recommended that you use increasing version numbers, since undesirable complications (including data loss) could occur if you try to transition to a lower data version number that represents newer data.

App data only roams between installed apps with the same version number. For example, devices on version 2 will transition data between each other and devices on version 3 will do the same, but no roaming will occur between a device running version 2 and a device running version 3. If you install a new app that utilized various version numbers on other devices, the newly installed app will sync the app data associated with the highest version number.

Testing and tools

Developers can lock their device in order to trigger a synchronization of roaming app data. If it seems that the app data does not transition within a certain time frame, please check the following items and make sure that:

- Your roaming data does not exceed the maximum size.
- Your files are closed and released properly.
- There are at least two devices running the same version of the app.

Developers can use the Windows 8 Roaming Monitor to monitor and affect the status of their app's roaming state.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Roaming data between a Windows Store app and a Windows Phone Store app

If you publish two versions of your app - a version for Windows Store and a version for Windows Phone - you can roam app data across the apps running on the two different types of devices. To roam data across different versions of your app on different types of devices, assign the same Package Family Name (PFN) to each version of the app.

For more info, see How to roam data between a Windows Store app and a Windows Phone Store app.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for accessing OneDrive from an app



Follow these guidelines for designing a Windows Store app that interacts with a Microsoft OneDrive user's files, documents, pictures, videos, folders, albums, or comments.

Dos and don'ts

Users of OneDrive assume that Microsoft works to help protect the security and privacy of their data. They rely on OneDrive to help preserve their important documents, to save their photos, and to share their experiences with friends. Your app can enhance the value of OneDrive for users by providing considerate, well-designed access to their data.

To maintain the trust that users have in OneDrive, have your app follow these design principles.

Let the user opt in

Users want to choose how apps handle their data. They expect an app to ask their permission before connecting to their account. They want to be notified before their data is changed. To meet their expectations, follow these practices:

- Upload files to OneDrive only in response to an explicit user request or choice.

  An app that connects to OneDrive should include a button that allows users to intentionally upload their files to OneDrive. If your app syncs files to OneDrive as its default, make the user aware of this and provide the opportunity to opt in before any data is saved.

- Sign users into and out of their account using the Account charm.

  Your app must give users a way to actively sign into and out of their Microsoft account. (Keep in mind that, if the user has signed into Windows with their Microsoft account, your app can't explicitly sign them out.)

- Access only files that are owned by the signed-in user.

  Unless your app is meant for sharing files between OneDrive users, ensure it accesses only the signed-in user's files. Your app should access files and folders that have been shared with the user only if the user asks to do so. Conversely, your app should not save files to shared folders unless the user chooses to.

© 2014 Microsoft. All rights reserved. | May 13, 2014

- Give users a choice about where they want data stored in their OneDrive.

  Your app can use the Windows file picker by using the **Windows.Storage.Pickers** namespace for opening and saving files to the user's OneDrive. If your app syncs multiple files, consider creating a uniquely named subfolder in the user's folders.

Help protect the user's data and privacy

Your app must not undermine the user's trust in their OneDrive. Handle user data discretely. Users assume that their files are shared only with people whom they select. Their important info must be preserved so that they can return to it when they need it.

Note: Once set, your app cannot change the permissions set on a OneDrive object programmatically.

- Upload files to OneDrive with user-only access as the default.

  Share files with others only if the user specifically requests that the files be shared.

- Warn the user about sharing links to files with others.

  When users request to share a link to their files, have your app inform them about the consequences of sharing. In particular, if your app allows users to share *preauthenticated* links to their files, tell them that anyone who received the link can view the files. File permissions are not evaluated for these links and anyone who opens the links can view the content.

- Create links to OneDrive objects intentionally, based upon the use of the link.

  Whenever possible, share embedded, read-only, and read-write links. These links work only for users who have permission to view the file. Provide pre-authenticated links to files only when the user wants to share a folder or file with specific people. File permissions are not evaluated for these links and anyone who opens the links can view the content.

- Alert the user when overwriting an existing file.

  When uploading a file to OneDrive, the default behavior for upload is to overwrite any existing file with the same name. Let the user know that an existing file is going to be overwritten if a conflict occurs. You can add an **Overwrite** header set to 'false' to prevent a file from being overwritten.

Use OneDrive and Windows as intended

It's tempting to use the storage that's freely available through OneDrive as a catch-all cloud-data solution. Even though it does present a lot of options for a Windows Store app, OneDrive provides the most benefit to your app when used as intended. OneDrive is designed to provide users with access to their documents, photos, and other important info from any device.

© 2014 Microsoft. All rights reserved. | May 13, 2014

- Use OneDrive for storing, viewing, editing documents, or creating and sharing photo albums.

  OneDrive is not intended as an alternative for storing scalable databases, sharing configuration files, or hosting web applications (as a few examples). It is meant solely for the easy storage and sharing of a user's discrete files.

- Ensure that the user has space in their OneDrive before uploading a file.

  Each OneDrive user has a limited amount of storage available. If your app attempts to save a file that pushes the user's account over its allotted quota, the call returns an error. It is a best practice to check users' available storage before saving a file to their OneDrive.

- Use the built-in Windows features.

  Whenever possible, use the Windows features and UI to host or interact with OneDrive. For instance, use the file picker provided by the **Windows.Storage.Pickers** namespace for opening and saving files. As another example, have your app use the Windows application data APIs to save smaller pieces of data across a user's devices.

Additional usage guidance

OneDrive provides users with a trustworthy and accessible place to store their files in the cloud. Users can access their OneDrive files from any Windows device by signing in with their Microsoft account. OneDrive gives users 7 gigabytes (GB) of free storage that they can use to save and share their photos, documents, videos, and audio files.

Your Windows Store app can provide users with access to the files and folders in their OneDrive. With the connection to OneDrive, your app gives users the ability to open, read, save, and download files from their OneDrive without cluttering their hard drive. The OneDrive APIs are designed to be used from within a Windows Store app and integrate smoothly into your app's design.

App designs for using OneDrive

In a broad sense, OneDrive can play a role for any app that interacts with discrete files. If your app reads or displays files, saves files, or downloads or opens files, you can add OneDrive to the app's design. OneDrive integrates well into the architecture of your Windows Store app, making use of the built-in features of Windows without requiring you to write a lot of extra code.

Note: The OneDrive APIs are in the Live Connect SDK. Before you start developing a Windows Store app that connects to OneDrive, you need to install the Live Connect SDK and add a reference to the SDK in your project.

- To download the Live Connect SDK, go to the Live Connect SDK download page.
- To view the OneDrive API documentation, see OneDrive for Developers.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Signing users into and out of their Microsoft account

Of course, any app that interacts with OneDrive must provide the user with a way to sign in and sign out of the Microsoft account associated with OneDrive. While not an app design in itself, signing users into their account is a critical step in building an app that integrates with OneDrive.

The recommended technique for signing users in is to create an Accounts page and a Privacy statement page in your app's Settings charm. The Accounts page should provide a sign-in and sign-out button that signs users in and out of their account. The WindowsUI handles the rest of the sign-in process for your app.

Saving new files or updating existing files in OneDrive

For some users, OneDrive is their 'My Documents'. For users who prefer to use OneDrive for storing their files, your app can provide the option to save their data in OneDrive. For example, when they create new files in your app, you can offer OneDrive as a location to save. When they edit files in your app, they can save the edits back to their OneDrive.

Realistically, any app that lets users to create new files can benefit from giving users access to OneDrive.

- For guidelines on how to build an app that integrates with OneDrive, see the dos and don'ts section.

Downloading, opening, and viewing files from OneDrive

As we noted earlier, some users keep a lot of their data in the cloud. They expect to be able to view the data contained there. Your app can give users the option to open and read files from OneDrive. The app can download, open, and display the contents of the file for the user to view.

For example, if your app plays videos, you can give users the ability to open movies from their OneDrive folders. Or your app can let users open and view a specific file type, as a reader.

Note: We recommend that your Windows Store app do more than just allow users to view the files contained in their OneDrive. Windows comes with a OneDrive app. Users are more likely to download and install your app if it provides them with a unique experience.

To learn more, have a look at these resources:

- For guidelines on how to build an app that integrates with OneDrive, see the dos and don'ts.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for single sign-on and connected accounts



User authentication scenarios

Users can use their Microsoft account credentials to sign in to devices running Windows 8. When they do this, Windows 8 works with your Windows Store app to enable authenticated experiences for them. These experiences include the following:

- A user can associate his or her most commonly used operating-system settings with a Microsoft account. These settings are available whenever the user signs in with that account on any device that is running Windows 8 and connected to the cloud. After the user signs in, that device automatically attempts to get the user's settings from the cloud and apply them to the device.
- Windows Store apps can store user-specific settings so that these settings roam across any devices running Windows 8. As with operating-system settings, these user-specific app settings are available whenever the user signs in with the same Microsoft account on any device that is running Windows 8 and is connected to the cloud. After the user signs in, that device automatically downloads the settings from the cloud and applies them when the app is installed.
- On Windows 8, a user can associate a Microsoft account with his or her sign-in credentials for any apps or websites, so that these credentials roam across any devices running Windows 8. After the user signs in with that account to a device running Windows 8 and then runs an app or visits a website, if the corresponding stored sign-in credentials are available, Windows 8 attempts to sign the user in automatically.
- When a user signs in with a Microsoft account to a device running Windows 8, any apps and services running on that device that also use Microsoft accounts for authentication can sign in with that user's Microsoft account and get data that the user has consented to share.

When to use the Microsoft account authentication API

The more you answer "yes" to the following questions, the more you should consider integrating your apps —and your apps' companion websites—with the Microsoft account authentication API.

- Is your app a Windows Store app?
- Does your app offer a personalized user experience?
- Does your app access proprietary cloud services or Microsoft cloud services like Outlook.com and Microsoft OneDrive?
- Does you app need to provide a user-authentication system, but you don't have the time, knowledge, or infrastructure to create one yourself?

© 2014 Microsoft. All rights reserved. | May 13, 2014

Here are a few categories of apps and companion websites that can benefit from the Microsoft account authentication API:

- **Apps that access proprietary cloud services and that require user authentication**. If your app accesses a cloud service and needs to authenticate the user, your code can request an *authentication token* that allows your app or website to access the cloud service on the user's behalf. The Microsoft account authentication API creates JavaScript Object Notation (JSON)-formatted authentication tokens for the cloud service to use; each authentication token contains a user ID that is specific to your app. To use the Microsoft account authentication APIs to get a JSON-formatted authentication token for use with a proprietary cloud service, see **OnlineIdServiceTicketRequest**.
- **Apps and companion websites that access Microsoft cloud services like Outlook.com and OneDrive**. If your app or its companion website accesses user data in Outlook.com or OneDrive, the Live Connect APIs manage some of the complexities of authentication tokens and make it a bit easier to write code to work with these cloud services.

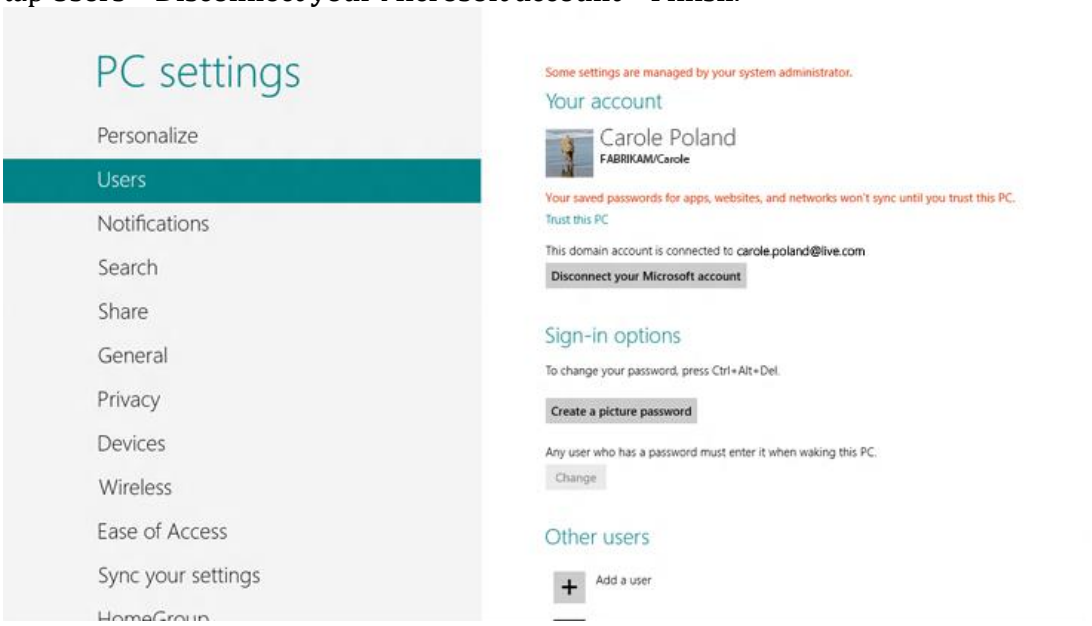
Adding user-authentication functionality to your Windows Store apps

Note: On a Windows 8 device, a user can sign in to that device with a local or domain Windows account that is associated with (or *connected* to) a Microsoft account. The user can then run a Windows Store app that relies on this particular Microsoft account for sign-in. If the user does so, and if that Windows Store app attempts to sign the user out of the app later by using the guidelines that are described in the topic just mentioned, the user will not be successfully signed out. To prevent this, the app must not show a sign-out command because it won't sign the user out. Instead, the user has a couple of possible ways to sign out of the app:

- The user can disassociate (or *disconnect*) his or her Microsoft account from the local or domain Windows account. To do this, in **PC settings** (shown in the following screen shot),

© 2014 Microsoft. All rights reserved. | May 13, 2014

tap **Users** > **Disconnect your Microsoft account** > **Finish**.



- The user can switch to using a different account. To do this, on the **Start** screen, the user taps the account picture, taps **Switch account**, and then signs in with different account credentials.

If the user follows either of these options, he or she is also automatically signed out of all Windows Store apps that rely on this particular Microsoft account for sign-in.

Putting the user in control

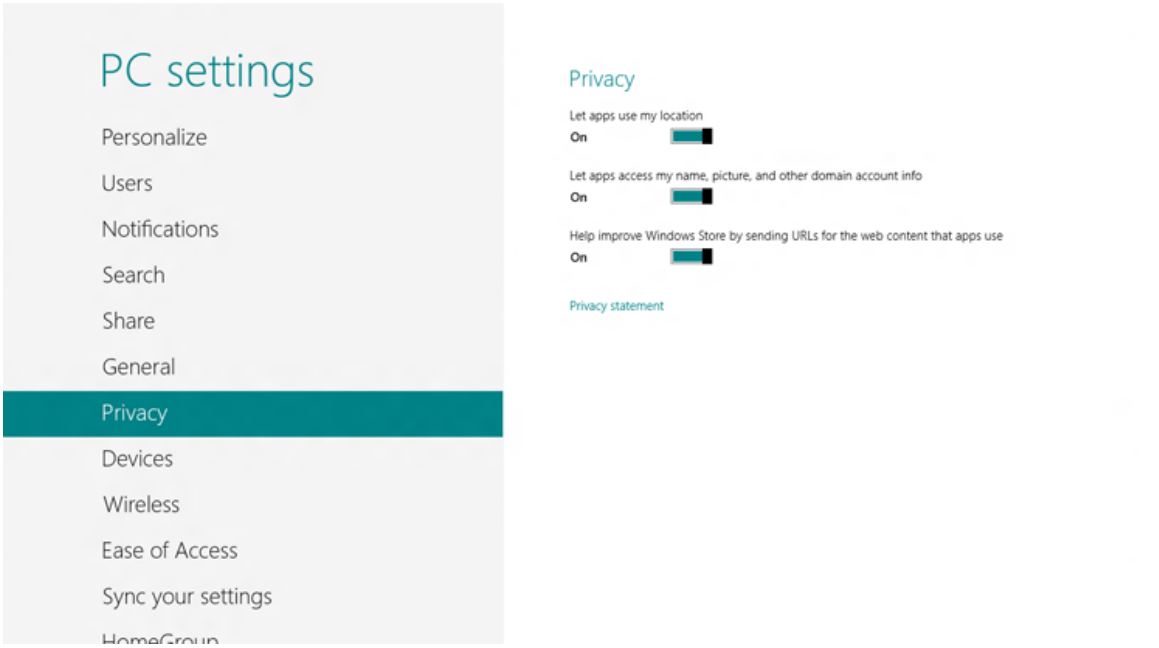
Whenever a user signs in with his or her Microsoft account (or with his or her local or domain Windows account that is connected to a Microsoft account) to a device running Windows 8, the user controls to what extent any app or companion website that uses the Microsoft account authentication APIs can act on his or her behalf. For example, when a user signs in to a participating app that accesses a Microsoft cloud service like Outlook.com or OneDrive, he or she is prompted to give an okay, or *consent*, to enable that app or its companion website to access the user's associated data or files from those particular cloud services.

However, if the app doesn't want to access data from a Microsoft cloud service like Outlook.com or OneDrive but instead wants to access a proprietary cloud service, typically the user isn't prompted to provide consent unless the user explicitly chose to override this in **PC settings** > **Privacy**. However, using the Microsoft account authentication APIs, participating apps and websites can get a user ID that's unique to that app or website. The app or website can then associate data with that user ID. The next time the user signs in, the app or website receives that same user ID and can use it to retrieve any data it has already associated with that user.

On Windows 8, a user can restrict all apps from accessing the name, picture, and other data associated with his or her account without consent. To do this, in **PC settings** (shown in the

© 2014 Microsoft. All rights reserved. | May 13, 2014

following screen shot), the user taps **Privacy** and then slides **Let apps access my name, picture, and other account info** to **Off**. (This option is set to **On** by default.)



Responding to user sign-in and sign-out status changes

Apps that use the Microsoft account authentication APIs must respond appropriately whenever a user connects or disconnects his or her Microsoft account from the local or domain Windows account. When a user does this, the **ConnectedStateChange** background task starts. Whenever this task starts, participating apps must check whether the user's ID for the app has been cleared.

- If the user's ID for the app has been cleared, the app must first clear any associated tile notifications for that user. If the app indicates whether users have signed in, the app's state should change to show that the user whose ID has been cleared is no longer signed in. However, the app should not reset to its default state. Instead, the app should assume that the user is still using it and is continuing where he or she left off, except in a signed-out state. The only difference is that the signed-out user may no longer have access to cloud services such as tile notifications. Note that this approach will result in different behaviors for different apps. Some apps may need to ask the user to sign in again to continue using them, and others may continue to work but in a signed-out state.

Note: If the app doesn't store app-specific data to the cloud for the user who has signed out, it's up to the app to decide what to do. We recommend that the app clear all app-specific data for that user from appearing. However, the app should also save this data locally in case the user reconnects his or her Microsoft account to the local or domain Windows account.

© 2014 Microsoft. All rights reserved.

- If the user's ID for the app has not been cleared, it means that the user connected the same account that he or she was already using for the app, and there's nothing further for the app to do here.

For more info about the **ConnectedStateChange** background task, see the **SystemTriggerType** enumeration for the **OnlineIdConnectedStateChange** system event.

For more info about how to check whether a user's ID for an app has been cleared, see the **authenticatedSafeCustomerId** property.

Code sample

For code that shows how to call the Microsoft account authentication APIs, see the Microsoft account authentication sample.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for thumbnails



These guidelines describe how to use thumbnail images to help users preview files as they browse in your Windows Store app.

Should my app include thumbnails?

If your app allows users to browse files, let your users quickly preview those files by displaying thumbnail images.

Use thumbnails when:

- Displaying previews for many items (like files and folders). For example, a photo gallery app should use thumbnails to give users a small view of each picture as they browse through their photo files.
- Displaying a preview for an individual item (like a specific file). For example, the user may want to see more information about a file, including a larger thumbnail for a better preview, before deciding whether to open the file.

Dos and don'ts

- Specify the thumbnail mode (**picturesView**, **videosView**, **documentsView**, **musicView**, **listView**, or **singleItem**) when you call a method to retrieve thumbnails. This ensures that thumbnail images are optimized for displaying the type of files your user wants to see.
- Use the **singleItem** mode to retrieve a thumbnail for a single item, regardless of file type. The other thumbnail modes (**picturesView**, **videosView**, **documentsView**, **musicView**, and **listView**) are meant to display previews of multiple files.
- Display generic placeholder images in place of thumbnails while the thumbnails load. Using placeholders in this way helps your app seem more responsive because users can interact with items before the preview images load.

  A placeholder images should be:

  - Specific to the kind of item that it stands in for. For example, folders, pictures, and videos should all have their own specialized placeholders that use different icons, text, and/or colors.
  - Of the same size and aspect ratio as the thumbnail image it stands in for.
  - Displayed until the thumbnail image is loaded. If a thumbnail can't be retrieved, display a placeholder image instead.
- Use placeholder images with text labels to represent folders and file groups. This differentiates system constructs like folders and file groups from individual files. A visual

© 2014 Microsoft. All rights reserved. | May 13, 2014

distinction between these types of items will help make it easier for users who are browsing with your app. Be sure to include a text label for the placeholder that is either the name of the folder or the criteria used to form the group of files.
- If you can't retrieve a thumbnail for an item (like a file, folder, or file group), display a placeholder image.
- Display file info in addition to thumbnail images when providing previews for document and music files. This lets users identify key information about a file that may not be readily available from a thumbnail image alone. For example, you might display the name of the artist for a music file along with a thumbnail showing the album art.
- Don't display additional file info alongside thumbnails for picture and video files. In most cases, a thumbnail image is sufficient for users browsing through pictures and videos.

Additional usage guidelines

Recommended thumbnail modes and their features

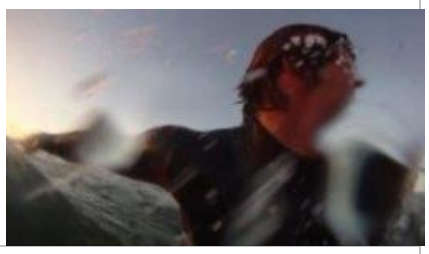
Display previews for	Thumbnail modes	Features of the retrieved thumbnail images
Pictures **Videos**	**picturesView** **videosView**	**Size:** Medium, preferably at least 190 (if the image size is 190 x 130) **Aspect ratio:** Uniform, wide aspect ratio of about .7 (190 x 130 if the size is 190) Cropped for previews Good for aligning images in a grid because of uniform aspect ratio
Documents **Music**	**documentsView** **musicView** **listView**	**Size:** Small, preferably at least 40 x 40 pixels **Aspect ratio:** Uniform, square aspect ratio Good for previewing album art because of the square aspect ratio Documents look the same as they look in a file picker window (it uses the same icons)
Any single item (regardless of file type)	**singleItem**	**Size:** Large, at least 256 pixels on the longest side **Aspect ratio:** Variable, uses the original aspect ratio of the file

Tip: The features of thumbnail images for different modes might become even more specific in the future. To account for this, we recommend that you specify the thumbnail mode that most closely describes the kinds of files you want to display previews for. For example, if you want to display video files you should use the **videosView** thumbnail mode (unless you're displaying a single video file, in which case, use the **singleItem** mode).

© 2014 Microsoft. All rights reserved. | May 13, 2014

Why use recommended thumbnail modes?

Here are examples showing how retrieved thumbnail images differ depending on file type and thumbnail mode.

Item type	When retrieved using: • **picturesView** • **videosView**	When retrieved using: • **documentsView** • **musicView** • **listView**	When retrieved using: • **singleItem**
Picture		The thumbnail is cropped to a square aspect ratio. 	The thumbnail image uses the original aspect ratio of the file. 
Video	The thumbnail has an icon that differentiates it from pictures. 	The thumbnail is cropped to a square aspect ratio. 	The thumbnail image uses the original aspect ratio of the file. 
Music	The thumbnail is an icon on a background of appropriate size. The background color is determined by the app associated with the file. (If the associated app is a Windows Store app, the app's tile background color is used.)	If the file has album art, the thumbnail is the album art. Otherwise, the thumbnail is an icon on a background of appropriate size. The background color is determined by the app that is associated with the file. (If the associated app is a Windows Store app, the app's tile	If the file has album art, the thumbnail is the album art and uses the original aspect ratio of the file. Otherwise, the thumbnail is just an icon. 

© 2014 Microsoft. All rights reserved. | May 13, 2014

		background color is used.) 	
Document	The thumbnail is an icon on a background of appropriate size. The background color is determined by the app that is associated with the file. (If the associated app is a Windows Store app, the app's tile background color is used.) 	The thumbnail is an icon on a background of appropriate size. The background color is determined by the app that is associated with the file. (If the associated app is a Windows Store app, the app's tile background color is used.) 	The document thumbnail, if one exists.  Otherwise, the thumbnail is an icon. 
Folder	If there is a picture file in the folder, the picture thumbnail is used.  If the folder doesn't contain a picture, no thumbnail is retrieved.	No thumbnail image is retrieved.	The thumbnail is an icon that represents a folder.
File group	If there is a picture file among the files in the group, the picture thumbnail is used.	If there is a file that has album art among the files in the group, the thumbnail is the album art.	If there is a file that has album art among the files in the group, the thumbnail is the album art and uses the original aspect ratio of the file.

© 2014 Microsoft. All rights reserved. | May 13, 2014

	 If the file group doesn't contain a picture, no thumbnail is retrieved.	 If no album art exists in the file group, no thumbnail is retrieved.	 Otherwise, the thumbnail is an icon that represents a group of files. 

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for user names and account pictures



Description

In Windows 8, apps can retrieve the current user's name and image (called the *account picture*) and use them to identify and create personalized experiences for the user. For example, a messaging app could use the name and account picture to identify the user as a participant in a conversation, a game app could use them to identify the user as a player in a game's leader board page, and so on.

Dos and don'ts

- Use appropriate image sizes for the application interface.
- Use a status bar to indicate status as a small portion of the account picture.
- Don't use an account status icon that may be lost in the account picture behind it.

Additional usage guidance

If your app can take pictures, consider declaring your app as an account picture provider. That way, your app will be listed in the **Account picture** page under **Personalize** in **PC settings**. From there, users can select it to create a new account picture.

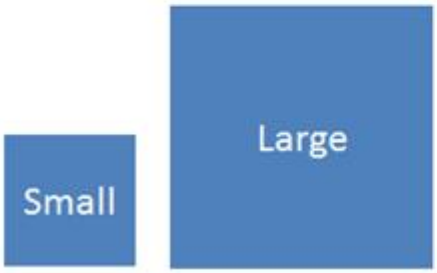
To learn more, see the Account picture name sample.

Account picture sizes

An app can retrieve the account picture as a small image, a large image, or a video (dynamic image). The images are sized to look great at the largest DPI plateau (1.8x):

© 2014 Microsoft. All rights reserved. | May 13, 2014

Small	Large
96x96	448x448



Plateau	Small	Large
1.0	48	224
1.4	67	314
1.8	86	403

An account picture or video has a 448x448 frame size that can be up to five seconds long and five megabytes in size.

Use a small image when the account picture is not a central part of a particular experience, such as displaying a long list of people. Use large images when you need to identify a small number of people in an app area, or when you need to clearly identify each user. For example, use large images for conversation participants in a messaging app or to display the caller's image for an incoming phone call.

User name

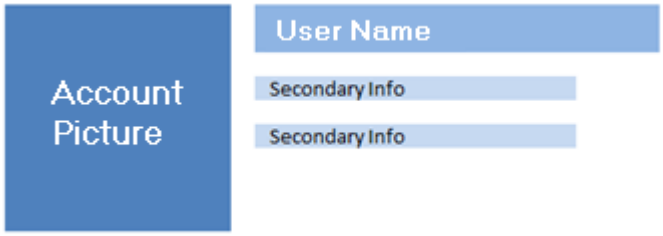
Apps can retrieve the user name by using members of the **UserInformation** class. The user's first and last name are available independently, as is the user's display name. Windows formats the display name so that it is in the correct order for the locale.

Displaying an account picture and user name together

We recommend one of the following formats for displaying an account picture together with the corresponding user name:

- **Side-by-side**—If sufficient space is available, display the account picture and user name side-by-side because this format is easiest for the user to read. The following example shows the preferred alignment and arrangement of the elements. Note that the user name is aligned with the top edge of the account picture, and any secondary information is displayed in a smaller font size smaller than the user name.

© 2014 Microsoft. All rights reserved. | May 13, 2014



- **Above-and-below**—When space in constrained, display the user name below the account picture. Be aware, however, that the entire user name might not fit on a single line. Note that the user name is aligned with the left edge of the account picture, and secondary information has a smaller font size than the user name.



Showing status

If your app needs to combine status information with an account picture, we recommend that you use a status bar or an icon overlay. We prefer a status bar because it doesn't obstruct the image and is easier to notice. A status bar should be wide enough to be easily distinguished from the account picture image. Here are some examples of status bars:



You can use icon overlays to communicate more complicated kinds of status. Icon overlays can get lost in the details of the image, so be sure to use icons that stand out clearly against a variety of backgrounds. Here are some examples of icon overlays:

© 2014 Microsoft. All rights reserved. | May 13, 2014



© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for files, data, and globalization



Consider how to design your app so that you can easily make it available in worldwide markets. Be sure that app resources, such as strings and images, are separated from their code, to help make localization easy.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for app resources



Separating app resources, like strings and images, from code is one way to ease the process of maintaining and localizing your app. This topic describes best practices for using app resources in a Windows Store app.

Dos and don'ts

Creating resources

- Don't put resources, such as UI strings and images, in code. Instead, put them into resource files, such as .resjson or .resw files.
- Use qualifiers to support file and string resources that are tailored for different display scales, UI languages, or high contrast settings.
- Set the default language in the app manifest (package.appxmanifest).
- String resources, even those in the default language, should have a file or folder named with the language tag.
- Add comments to your string resource for the localizer.

Referring to resources

- Add unique resource identifiers in the code and markup to refer to resources.
- Refer to images in markup, code, or manifest files without the qualifiers.
- Listen for events that fire when the system changes and it begins to use a different set of qualifiers. Reprocess the document so that the correct resources can be loaded.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for globalization and localization



Follow these best practices when globalizing your apps for a wider audience and when localizing your apps for a specific market.

Dos and don'ts

Globalization

Prepare your app to easily adapt to different markets by choosing globally appropriate terms and images for your UI, using **Globalization** APIs to format app data, and avoiding assumptions based on location or language.

Recommendation	Description
Use the correct formats for numbers, dates, times, addresses, and phone numbers.	The formatting used for numbers, dates, times, and other forms of data varies between cultures, regions, languages, and markets. If you're displaying numbers, dates, times, or other data, use **Globalization** APIs to get the format appropriate for a particular audience.
Support international paper sizes.	The most common paper sizes differ between countries, so if you include features that depend on paper size, like printing, be sure to support and test common international sizes.
Support international units of measurement and currencies.	Different units and scales are used in different countries, although the most popular are the metric system and the imperial system. If you deal with measurements, like length, temperature, or area, get the correct system measurement by using the **Globalization** namespace. If your app supports displaying currencies, make sure you use the correct formatting. You can also get the currency for the user’s geographic region by using the **CurrenciesInUse** property.
Display text and fonts correctly.	The ideal font, font size, and direction of text varies between different markets.
Use Unicode for character encoding.	By default, recent versions of Microsoft Visual Studio use Unicode character encoding for all documents. If you're using a different editor, be sure to save source files in the appropriate Unicode character encodings. All Windows Runtime APIs return UTF-16 encoded strings.
Record the language of input.	When your app asks users for text input, record the language of input. This ensures that when the input is displayed later it's presented to the user with the appropriate formatting. Use the

© 2014 Microsoft. All rights reserved. | May 13, 2014

	CurrentInputMethodLanguage property to get the current input language.
Don't use language to assume a user's location, and don't use location to assume a user's language.	In Windows, the user's language and location are separate concepts. A user can speak a particular regional variant of a language, like en-gb for English as spoken in Great Britain, but the user can be in an entirely different country or region. Consider whether your apps require knowledge about the user's language, like for UI text, or location, like for licensing issues.
Don't use colloquialisms and metaphors.	Language that's specific to a demographic group, such as culture and age, can be hard to understand or translate, because only people in that demographic group use that language. Similarly, metaphors might make sense to one person but mean nothing to someone else. For example, a "bluebird" means something specific to those who are part of skiing culture, but those who aren't part of that culture don't understand the reference. If you plan to localize your app and you use an informal voice or tone, be sure that you adequately explain to localizers the meaning and voice to be translated.
Don't use technical jargon, abbreviations, or acronyms.	Technical language is less likely to be understood by non-technical audiences or people from other cultures or regions, and it's difficult to translate. People don't use these kinds of words in everyday conversations. Technical language often appears in error messages to identify hardware and software issues. At times, this might be necessary, but you should rewrite strings to be non-technical.
Don't use images that might be offensive.	Images that might be appropriate in your own culture may be offensive or misinterpreted in other cultures. Avoid use of religious symbols, animals, or color combinations that are associated with national flags or political movements.
Avoid political offense in maps or when referring to regions.	Maps may include controversial regional or national boundaries, and they're a frequent source of political offense. Be careful that any UI used for selecting a nation refers to it as a "country/region". Putting a disputed territory in a list labeled "Countries", like in an address form, could get you in trouble.
Don't use string comparison by itself to compare language tags.	BCP-47 language tags are complex. There are a number of issues when comparing language tags, including issues with matching script information, legacy tags, and multiple regional variants. The resource management system in Windows takes care of matching for you. You can specify a set of resources in any languages, and the system chooses the appropriate one for the user and the app.
Don't assume that sorting is always alphabetic.	For languages that don't use Latin script, sorting is based on things like pronunciation, number of pen strokes, and other factors. Even languages that use Latin script don't always use alphabetic sorting. For example, in some cultures, a phone book might not be sorted alphabetically. The system can handle sorting for you, but if you create your own sorting algorithm, be sure to take into account the sorting methods used in your target markets.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Localization

Recommendation	Description
Separate resources such as UI strings and images from code.	Design your apps so that resources, like strings and images, are separated from your code. This enables them to be independently maintained, localized, and customized for different scaling factors, accessibility options, and a number of other user and machine contexts. Separate string resources from your app's code to create a single language-independent codebase. Always separate strings from app code and markup, and place them into a resource file, like a ResW or ResJSON file. Use the resource infrastructure in Windows to handle the selection of the most appropriate resources to match the user's runtime environment
Isolate other localizable resource files.	Take other files that require localization, like images that contain text to be translated or that need to be changed due to cultural sensitivity, and place them in folders tagged with language names.
Set your default language, and mark all of your resources, even the ones in your default language.	Always set the default language for your apps appropriately. The default language determines the language that's used when the user doesn't speak any of the supported languages of the app. Mark default language resources, for example en-us/Logo.png, with their language, so the system can tell which language the resource is in and how it's used in particular situations.
Determine the resources of your app that require localization.	What needs to change if your app is to be localized for other markets? Text strings require translation into other languages. Images may need to be adapted for other cultures. Consider how localization affects other resources that your app uses, like audio or video.
Use resource identifiers in the code and markup to refer to resources.	Instead of having string literals or specific file names for images in your markup, use references to the resources. Be sure to use unique identifiers for each resource.
Enable text size to increase.	Allocate text buffers dynamically, since text size may expand when translated. If you must use static buffers, make them extra-large (perhaps doubling the length of the English string) to accommodate potential expansion when strings are translated. There also may be limited space available for a user interface. To accommodate localized languages, ensure that your string length is approximately 40% longer than what you would need for the English language. For really short strings, such as single words, you may needs as much as 300% more space. In addition, enabling multiline support and text-

© 2014 Microsoft. All rights reserved. | May 13, 2014

	wrapping in a control will leave more space to display each string.
Support mirroring.	Text alignment and reading order can be left-to-right, as in English, or right-to-left (RTL), as in Arabic or Hebrew. If you are localizing your product into languages that use a different reading order than your own, be sure that the layout of your UI elements supports mirroring. Even items such as back buttons, UI transition effects, and images may need to be mirrored.
Comment strings.	Ensure that strings are properly commented, and only the strings that need to be translated are provided to localizers. Over-localization is a common source of problems.
Use short strings.	Shorter strings are easier to translate and enable translation recycling. Translation recycling saves money because the same string isn't sent to the localizer twice. Strings longer than 8192 characters may not be supported by some localization tools, so keep string length to 4000 or less.
Provide strings that contain an entire sentence.	Provide strings that contain an entire sentence, instead of breaking the sentence into individual words, because the translation of words may depend on their position in a sentence. Also, don't assume that a phrase with multiple parameters will keep those parameters in the same order for every language.
Optimize image and audio files for localization.	Reduce localization costs by avoiding use of text in images or speech in audio files. If you're localizing to a language with a different reading direction than your own, using symmetrical images and effects make it easier to support mirroring.
Don't re-use strings in different contexts.	Don't re-use strings in different contexts, because even simple words like "on" and "off" may be translated differently, depending on the context.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for help and instructions



Provide help or troubleshooting tips to your users, and teach them to effectively interact with your app. This section provides best practices for instructing users along the way as they use your app.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for app help



These guidelines describe how to design effective Help content for your app. Help content should be a single page and can include text, links, and images. If you need to provide dynamic Help content, link to a support website or embed an online page in your Help section.

Should my app include Help content?

Whether to include Help is entirely up to you. Keep in mind not every app needs a designated Help section. For example, if you app contains only one or two UI elements that might confuse a user, you could integrate instructional UI, create a simple in-app demo, or consider redesigning those elements instead of creating a separate help section.

Dos and don'ts

- Keep Help pages short and easy for users to skim.
- If your Help content doesn't fit on a single page or if you need to include dynamic content that'll need updating, link to a support website or embed an online page in your Help flyout. Keep in mind that linking to a web page takes your user out of the app experience. If possible, embed the online content to create a more cohesive user experience.
- Don't use technical terms and jargon.
- Don't use Help to document all your app features. If you want to provide detailed content about your app, consider providing a link to a support web page at the end of your Help content.
- Don't use Help to notify customers that a newer version of the app is available.

Recommendations for Windows Store apps:

  - Allow users to access Help from the Settings pane.
  - Use a Settings flyout to display your Help content. When you use a Settings flyout, you can easily add Help to the Settings pane provided by Windows.
  - Label the entry point in the Settings pane "Help" to clearly identify it.
  - Don't directly link to a website from the Help entry point in the Settings pane. Instead, provide a link to online content in the associated Help flyout.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for designing instructional UI



Design an instructional user interface (UI) that teaches users how to work with your Windows Store app.

Dos and don'ts

- Use instructional UI to introduce a new user to what your app can do.
- Use for tips on new features or specific details about how your app has changed after an update.
- Integrate instructional UI with specific tasks.
- Don't block interaction with the application UI.

Additional usage guidance

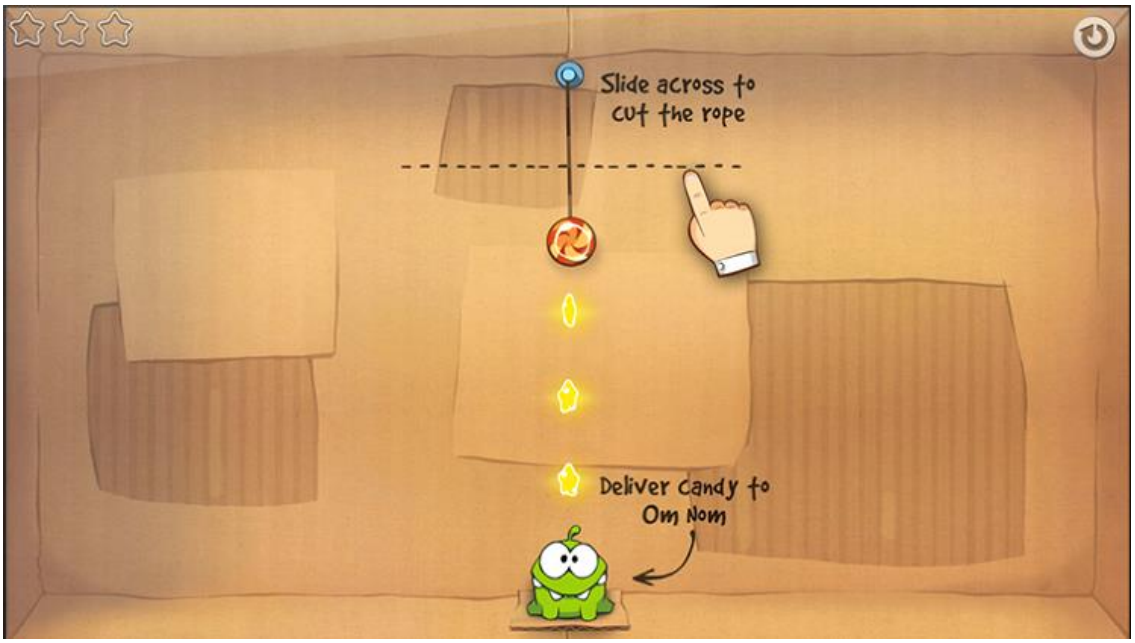
In some circumstances, the best way to help users interact with your Windows Store app is to teach them from within your app's UI. We use the term *instructional UI* to refer to this type of guidance. A great example is using a UI element, like inline text or a flyout, to tell a user when they need to use a touch interaction to complete a task.

Keep in mind that instructional UI is not a replacement for thoughtful design. If you use it too often or out of context, you can disrupt the flow of your app and diminish its effectiveness. Before adding instructional UI, explore other ways to introduce users to your app. See Related Topics for more info about layouts, navigation, and controls.

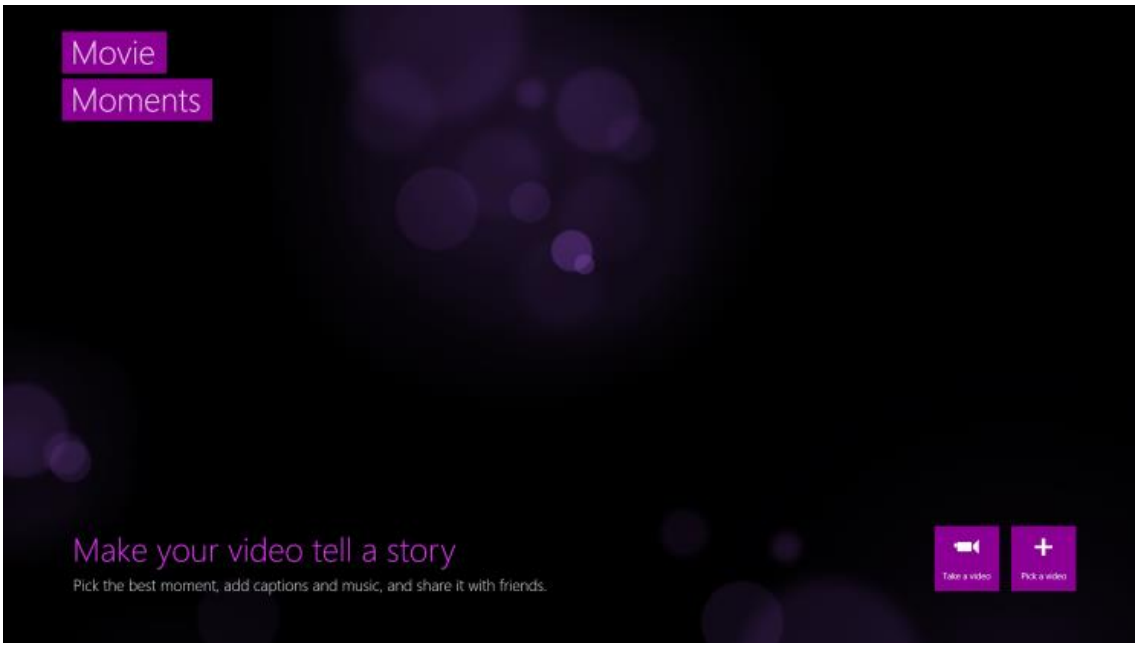
Here are a few instances in which instructional UI can help your users learn:

- **Helping users discover touch interactions.** The following screen shot shows instructional UI teaching a player how to use touch gestures in the game, Cut the Rope.

© 2014 Microsoft. All rights reserved. | May 13, 2014

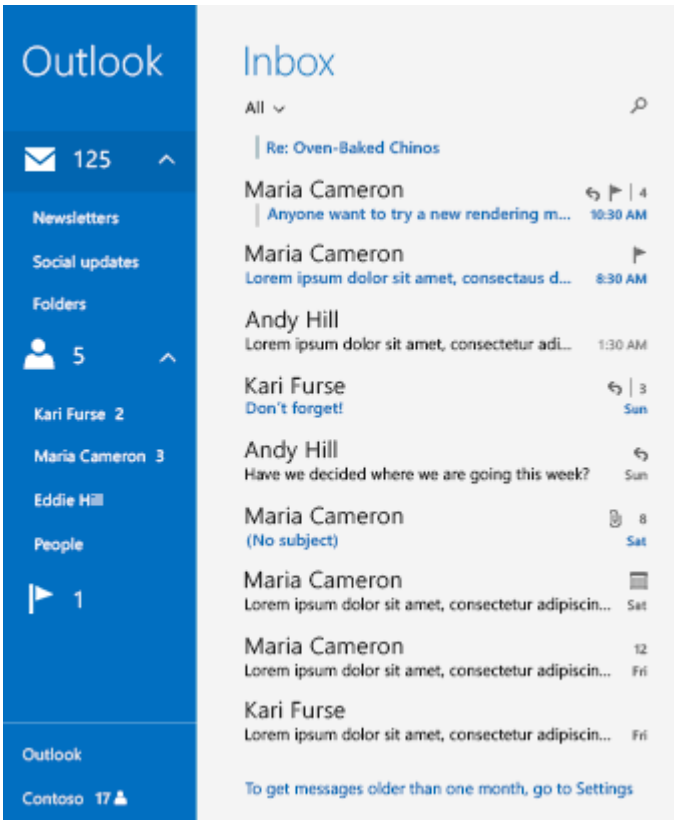


- **Making a great first impression.** Consider using instructional UI to introduce a new user to what your app can do. For example, when Movie Moments launches for the first time, instructional UI prompts the user to begin creating movies.

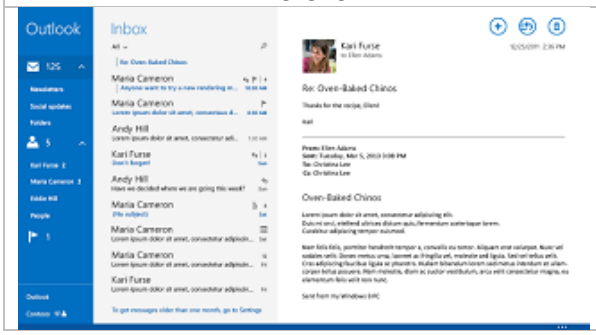
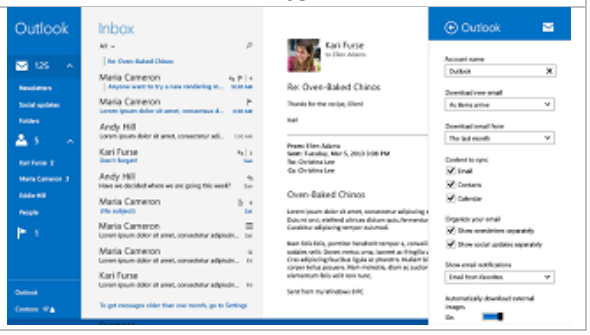


- **Guiding users to take the next step in a complicated task.** In the Windows Mail app, a hint at the bottom of the Inbox directs users to **Settings** to access older messages.

© 2014 Microsoft. All rights reserved. | May 13, 2014



When the user clicks the message, the app's **Settings** flyout appears on the right side of the screen, allowing the user to complete the task. These screen shots show the Mail app before and after a user clicks the instructional UI message.

Before	After
	

- **Introducing UI changes.** If you made significant UI changes in the latest version of your app, users might like tips on new features or specific details about how your app has changed.

Principles of instructional UI design

© 2014 Microsoft. All rights reserved. | May 13, 2014

- **Keep it simple.** Introduce one basic concept at a time and use images when possible. Consider adding a Help section to a **Settings** flyout in your Windows Store app to address complex features.
- **Teach in context.** Integrate instructional UI with the task it helps a user complete. A user is more likely to retain a concept when it's introduced when they need it most.
- **Don't block interaction.** Make sure users can still interact with your app while instructional UI is present. Instructional UI should help users, not annoy them by getting in the way.
- **Teach, then disappear.** Remove instructional UI as soon as it's no longer relevant or allow the user to dismiss it. Also, in most cases, users only need instructional UI displayed once. Avoid repeatedly displaying the same instructional UI.
- **Use sparingly.** Thoughtful design and a Help section can often teach users everything they need to know to enjoy your app. Consider the breadth of design options before adding instructional UI to your app.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for launch, suspend, and resume



This section provides guidelines for creating an inviting launch experience, and for designing your app to suspend when the user switches away from it and resume when the user switches back to it.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for app suspend and resume



Design your app to suspend when the user switches away from it and resume when the user switches back to it. Carefully consider the purpose and usage patterns of your app to ensure that your user has the best experience possible when your app is suspended and resumed. Follow these guidelines when you design the suspend and resume behavior of your Windows Runtime app.

For a summary of the lifecycle of a Windows Runtime app, see App lifecycle.

Note: To improve system responsiveness in Windows 8.1 and Windows Phone 8.1, apps are given low priority access to resources after they are suspended. To support this new priority, the suspend operation timeout is extended so that the app has the equivalent of the 5-second timeout for normal priority on Windows and a range of 1 to 10 seconds on Windows Phone. You cannot extend or alter this timeout window.

Dos and don'ts

- When resuming after a short period of time, return users to the state the app was in when the user left. For example, if a user navigates to another app before finishing an email, the user should be returned to the page with their unfinished email instead of the mail app's main landing page.
- When resuming after a long period of time, return users to your app's default landing page. For example, return a user to the main landing page of your news or weather app instead of returning them to an old, stale article or displaying outdated weather data.
- If appropriate, allow users to choose whether they want to restore their app to its previous state or start fresh. For example, when the user switches back to your game app, you could display a prompt so the user can decide whether to resume the game or start a new one.
- Save app data when the app is being suspended. Suspended apps don't receive notification when the system terminates them, so it is important to explicitly save app data to ensure that app state can be restored.

  If your app supports multiple launch points such as secondary tiles, and toast notifications, and File and URI associations, you should consider creating a separate navigation history for each launch point. When suspending, save the state associated with the primary launch point, and only save the state for secondary launch points in scenarios where it would frustrate the user to lose state. Saving too much state can cause the app to be slow to resume.

- Use saved app data to restore your app.
- Release exclusive resources and file handles when the app is being suspended. As stated earlier, suspended apps aren't notified when they're terminated, so make sure to release

© 2014 Microsoft. All rights reserved. | May 13, 2014

resources and handles (like webcams, I/O devices, external devices, and network resources) when your app is suspended to ensure that other apps can access them.
- Update the UI if content has changed since it was last visible to the user. Your resumed app should look as if it was running while the user was away.
- Don't terminate the app when it's moved off screen. The operating system ensures that there is a consistent way for the user to access and manage apps. Your app is suspended when it's moved off screen. By leaving the application lifecycle to the system, you ensure that your user can return to your app as efficiently as possible. Doing so also provides the best system performance and battery life from the device.
- Don't restore state for an app that was explicitly terminated by the user. The user may have closed the app because it got into an unrecoverable state. If your app was explicitly closed by the user, provide a fresh experience rather than a resume experience. When the app was closed by the user, the **PreviousExecutionState** property will have the value **ClosedByUser**.
- Don't restore state for an app that was terminated as the result of a crash. If your app was terminated unexpectedly, assume that stored app data is possibly corrupt. The app should not try to restore to its previous state using this stored data.
- Don't include Close buttons or offer users other ways to terminate your app in its UI. Users should feel confident that the system is managing their apps for them. The system can terminate apps automatically to ensure the best system performance and reliability, and users can choose to close apps using gestures on Windows or through the Task Switcher on Windows Phone.
- Don't strand users on deep-linked pages. When the user launches your app from a launch point other than the primary tile, landing on a deep-linked page, provide UI to allow the user to navigate to the app's top page. Or, allow the user to get to the top page by tapping the primary tile.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for splash screens



Follow these guidelines to customize the splash screen and create an extended splash screen, to help ensure a good launch experience for your users.

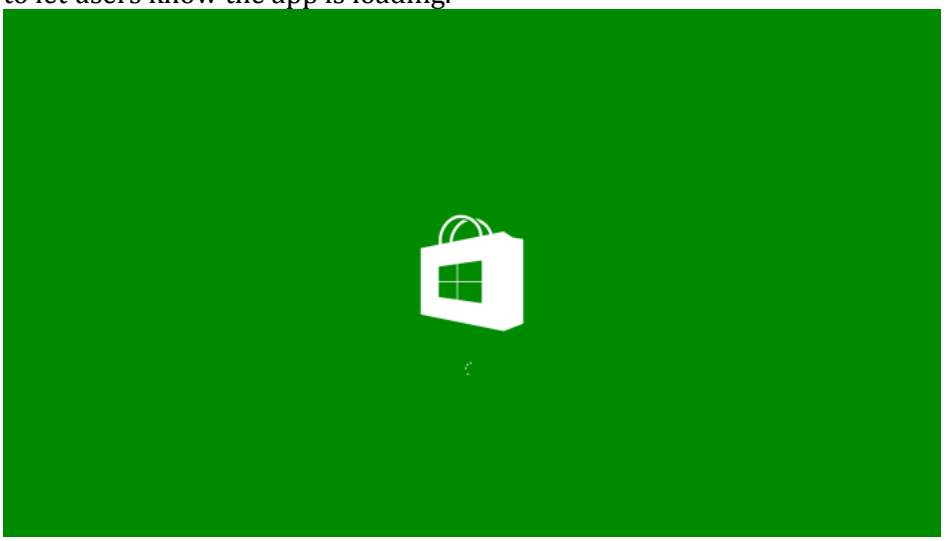
Dos and don'ts

- Customize the splash screen to differentiate your app. Your splash screen consists of an image and a background color, both of which you can customize. A well-designed splash screen can make your app more inviting.

  Putting an image and background color together to form the splash screen helps the splash screen look good regardless of the form factor of the device your app is installed on. When the splash screen is displayed, only the size of the background changes to compensate for a variety of screen sizes. Your image always remains intact.

- Create an extended splash screen so that you can complete additional tasks before showing your app's landing page. You can further control the loading experience of your app by creating an extended splash screen that imitates the splash screen displayed by Windows. By imitating the splash screen displayed by the system, you can construct a smooth and informative loading experience for your users. If your app needs more time to prepare its UI or load network data, you can use your extended splash screen to display a message for the user as your app completes those tasks.

  The extended splash screen for the Windows Store is shown below. Note that this screen is identical to the initial splash screen except it adds an "indeterminate ring" progress control to let users know the app is loading.



© 2014 Microsoft. All rights reserved. | May 13, 2014

Tip: If you use fragment loading to load your extended splash screen, you may notice a flicker between the time when the Windows splash screen is dismissed and when your extended splash screen is displayed. You see this flicker because fragment loading begins to load your extended splash screen asynchronously, before the **activated** event handler finishes executing. You can avoid this unsightly flicker entirely by using the design pattern demonstrated by the Splash screen sample. Instead of loading the extended splash screen as fragments, it is simply painted on top of the app's UI. When your additional loading tasks are complete you can then stop displaying your extended splash screen to reveal your app's landing page. Alternatively, if you wish to continue loading your extended splash screen as a fragment, you can also prevent the flicker by getting an activation deferral and responding to **activated** events asynchronously. Get a deferral for an activated event by calling the **activatedOperation.getDeferral** method.

- Don't use the splash screen or your extended splash screen to display advertisements. The purpose of the splash screen is to let users know, while the app is loading, that the app they wanted to start is starting. Introducing foreign elements into the splash screen reduces the user's confidence that they launched the correct app and makes the app harder to identify at a glance.
- Don't use your extended splash screen as a mechanism to display multiple, different splash screen images. The purpose of the splash screen and extended splash screen is to provide a smooth, polished loading experience for your users. Using your extended splash screen to display multiple, different splash screen images distracts from this purpose and could be jarring or confusing to your users. Instead, your extended splash screen should only continue the current loading experience while other tasks are completed.
- Don't use the splash screen or your extended splash screen to display an "about" page. The splash screen should not show version information or other app metadata. Display this information in your app's Windows Store description or within the app itself.

User experience

- Use an image that clearly identifies your app. Use an image and color scheme that clearly identify your app, so that users are confident that they launched the correct app. Making a unique screen also helps reinforce your brand.
- Use a transparent PNG as your splash screen image for best visual results. Using a transparent PNG lets the background color you chose show through your splash screen image. Otherwise, if the image has a different background color, your splash screen may look disjointed and unappealing.
- Provide a version of your splash screen image that is sized for all three scale factors. All apps must have a splash screen image that is 620 x 300 pixels, for when the device uses 1x scaling. We also recommend that you include additional splash screen images for 1.4x and 1.8x scaling. Providing images for all three scale factors helps you create a clean and consistent launch experience across different devices.

  Use the following table to determine the required size the splash screen image for each scale factor:

© 2014 Microsoft. All rights reserved. | May 13, 2014

Scale	Image size (pixels)
1x	620 x 300
1.4x	868 x 420
1.8x	1116 x 540

- Choose an image that uses the area allotted by the system for the splash screen image. When you choose a splash screen image, try to take advantage of the space allotted at each scale factor. Refer to the scale and image size table to determine the size of the splash screen image for each scale factor.

  This helps you produce a high-quality splash screen, by ensuring the quality of the image.

- Show system and event-related UI after the splash screen is dismissed. You can determine when it is safe to show system or event-related UI by listening for the splash screen **dismissed** event. Otherwise, the associated UI (like the search pane, a message dialog, or web authentication broker) might show while the splash screen is displayed. This might cause unwanted visual effects.
- Start entrance animations after the splash screen is dismissed. Many apps wish to show content entrance animations each time the app's landing page is loaded. You can determine when to start your animations by listening for the splash screen **dismissed** event.

Extended splash screens

- Make sure your extended splash screen looks like the splash screen that is displayed by Windows. Your extended splash screen should use the same background color and image as the Windows splash screen. Using a consistent image and background color helps ensure that the transition from the Windows splash screen to your app's extended splash screen looks professional and is not jarring for your users.
- Position your extended splash screen image at the coordinates where Windows displayed the splash screen image.
- Adjust the position of your extended splash screen to respond to resize events like rotation. Your extended splash screen should adjust the coordinates of its splash screen image, if the app is scaled or the device is rotated, by listening for the **onresize** event. This helps ensure that your app's loading experience looks smooth and professional, regardless of how users manipulate their devices or change the layout of apps on their screens.
- If you show your extended splash screen for more than a few seconds, add a progress ring so users know your app is still loading. Use the indeterminate progress ring control to let users know that your app hasn't crashed and will be ready soon. Consider displaying a single line of text alongside the progress ring to briefly explain what your app is doing to your users. For example, your extended splash screen could feature a progress ring and a "Loading" message.

  Making your app seem more responsive and keeping your users informed is a great way to create a positive loading experience for users.

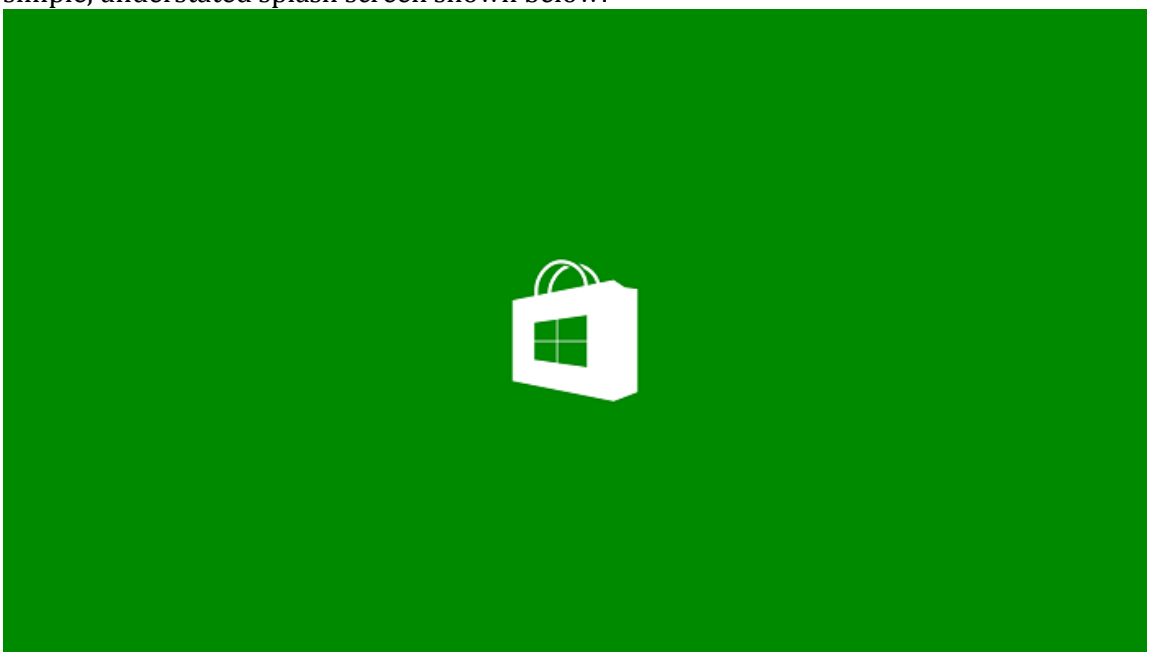
© 2014 Microsoft. All rights reserved.

Additional usage guidance

Every Windows Store app must have a splash screen, which consists of a splash screen image and a background color. You can customize both of these features.

Windows displays this splash screen immediately when the user launches an app. This provides immediate feedback to users while app resources are initialized. As soon as your app is ready for interaction, Windows dismisses the splash screen.

A well-designed splash screen can make your app more inviting. The Windows Store uses the simple, understated splash screen shown below:



This splash screen is created by combining a green background color with a transparent PNG.

You can use the **SplashScreen** class to customize your app's launch experience by extending the splash screen and triggering entrance animations.

Security considerations

The following articles provide guidance for writing secure C++ code.

- Security Best Practices for C++
- Patterns & Practices Security Guidance for Applications

Troubleshooting

JavaScript: Avoiding a flicker during the transition to your extended splash screen

© 2014 Microsoft. All rights reserved. | May 13, 2014

If you notice a flicker during the transition to your extended splash screen, add onload="" on your <img> tag like this: . This helps prevent flickering by making the system wait until your image has been rendered before it switches to your extended splash screen.

C#: Avoiding a flicker during the transition to your extended splash screen

This flicker occurs if you activate the current window (by calling Window.Current.Activate) before the content of the page finishes rendering. You can reduce the likelihood of seeing a flicker by making sure your extended splash screen image has been read before you activate the current window. Additionally, you should use a timer to try to avoid the flicker by making your application wait briefly, 50ms for example, before you activate the current window. Unfortunately, there is no guaranteed way to prevent the flicker because XAML renders content asynchronously and there is no guaranteed way to predict when rendering will be complete.

If you notice a flicker during the transition to your extended splash screen, follow these steps to make sure your extended splash screen image been read and that your app waits briefly before the current window is activated:

1. In ExtendedSplash.xaml, update the markup for your extended splash screen image to notify you when your extended splash screen image has been read.

XAML

```
<Image x:Name="extendedSplashImage" Source="Assets/SplashScreen.png"
ImageOpened="extendedSplashImage_ImageOpened"/>
```

2. The **ImageOpened** event fires after the image has been read. As shown in the example, you should register for the **ImageOpened** event by adding the **ImageOpened** attribute and specifying the name of the event handler (`extendedSplashImage_ImageOpened`).
3. In ExtendedSplash.xaml, add code in your ExtendedSplash class that activates the current window based on a timer and after your extended splash screen image has been read.

C#

```
        private DispatcherTimer showWindowTimer;
        private void OnShowWindowTimer(object sender, object e)
        {
            showWindowTimer.Stop();

            // Activate/show the window, now that the splash image has rendered
            Window.Current.Activate();
        }

        private void extendedSplashImage_ImageOpened(object sender, RoutedEventArgs
e)
        {
            // ImageOpened means the file has been read, but the image hasn't been
painted yet.
            // Start a short timer to give the image a chance to render, before
showing the window
            // and starting the animation.
            showWindowTimer = new DispatcherTimer();
            showWindowTimer.Interval = TimeSpan.FromMilliseconds(50);
```

© 2014 Microsoft. All rights reserved. | May 13, 2014

```
            showWindowTimer.Tick += OnShowWindowTimer;
            showWindowTimer.Start();
        }
```

4. This example shows both how to respond to an **ImageOpened** event and how to use a timer to make your application wait briefly before you activate the current window.
5. Revise your **OnLaunched** method like this:

C#

```
protected override void OnLaunched(LaunchActivatedEventArgs args)
{
    if (args.PreviousExecutionState != ApplicationExecutionState.Running)
    {
        bool loadState = (args.PreviousExecutionState ==
ApplicationExecutionState.Terminated);
        ExtendedSplash extendedSplash = new ExtendedSplash(args.SplashScreen,
loadState);
        Window.Current.Content = extendedSplash;
    }

    // ExtendedSplash will activate the window when its initial content has been
painted.
}
```

6. In the example, we have removed the call to **Activate** the current window. Instead that activation is performed by the `ExtendedSplash` object after the **ImageOpened** event fires to indicate that the extended splash screen image has been read.
7. Test your code on as many different devices and in as under as many difference circumstances as you can to make sure your code avoids the flicker effectively!

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for layout and scaling



This section provides guidelines for laying out the elements of your app on each app page and for scaling and your app when users interact with it on devices of varying sizes and in resized windows. It also includes guidelines for integrating advertising into the layout of your app.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for advertising

Advertising is a great way for app developers to make money, and with the unprecedented reach of Windows 8, it's also a compelling opportunity for advertisers. To learn more about advertising in your app, download Ads 101.

Examples

Here are two examples of ads designed to integrate with a specific app layout. The first image shows an app with a full screen grid layout. Note that the ad is the same size and shape as the content.



In this image, an app appears in a narrow window. The ad, as well as the content, adjusts to the narrow layout.

© 2014 Microsoft. All rights reserved. | May 13, 2014



Dos and don'ts

When incorporating ads into your app, follow these recommendations:

- Integrate ads into your design. Create a cohesive experience by integrating advertising into the original design and content layout throughout your app. Keep in mind how much space to allocate for ads at the top level versus at more granular levels.
- Pick complementary ad formats. Choose ad formats that make sense for your app. There are many ad formats available, and not all of them will engage your users or offer a seamless experience.
- Pick complementary ad size and placement. Select ad sizes and placement that complement the design of your app, adhere to industry standards, and are in high demand by potential advertisers. For example, if you are using the grid layout, consider using the 250x250 pixel ad size to keep the ads within the grid. If the ads don't fit, consider creating a separate cluster. For more examples of ad sizes, see the Microsoft Advertising SDK for Windows 8.
- Design for all window sizes. Consider how ad real estate changes based on the app's window size and orientation.
- Be thoughtful about local ads. Decide whether you want to offer location-targeted (local) ads.
- Include keywords. Include keywords that help categorize your app for potential advertisers that want to reach a specific audience.
- Use well-established metrics. Consider the two key metrics for monetization, CPM (cost per thousand impressions) and fill rate (how many impressions yield an ad) when you enable advertising. Many consider CPM * fill rate to be the optimal monetization solution.
- Don't only display ads in your app. If your app includes ads, it must provide additional functionality beyond the ads. Also, your app must allow users to complete primary tasks within the app. Apps must do more that open a website or mimic the behavior of a website.
- Don't include ads that conflict with Windows Store content policies. These policies are described in section 5 of the App certification requirements for the Windows Store.

© 2014 Microsoft. All rights reserved. | May 13, 2014

- Don't use your app's description, tiles, notifications, app bar, or the swipe-from-edge interaction to display ads.

The specific requirements for advertising are described in section 2 of the App certification requirements for the Windows Store. Complying with these requirements helps you serve ads in your app without damaging the user experience.

Additional usage guidance

Ask these basic questions when choosing an ad provider for your app:

- Does the ad provider adhere to the App certification requirements for the Windows Store?
- Does the ad provider offer ad quality that is commensurate with the quality of your app?

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for multiple windows



Support for multiple windows lets users to interact with different parts of your app at the same time. With multiple windows, users can compare content or view several specific pieces of content simultaneously. Follow these recommendations if you choose to support multiple windows in your Windows Store app.

Description

In an app that supports multiple windows, each window behaves as if it is its own app. Charms interact separately with each of the windows. When the user clicks the app's tile on the Start screen, the most recently used window of the app appears. The user can resize each window, dismiss each window from the screen independently, and view each window separately in the list of recently used apps.

Designing multiple windows

If it makes sense for your app to support multiple windows, you'll need to decide what content you want to show in each window. For example, you can choose to have one main window and other secondary windows that have a specific, limited set of functionality, or you can design each new window as a copy of the original app window. You can also specify the title of the secondary window, which is displayed when the user switches between apps.

You also designate where the new windows will open onscreen (relative to the original app window). A new window can be placed in one of the following locations:

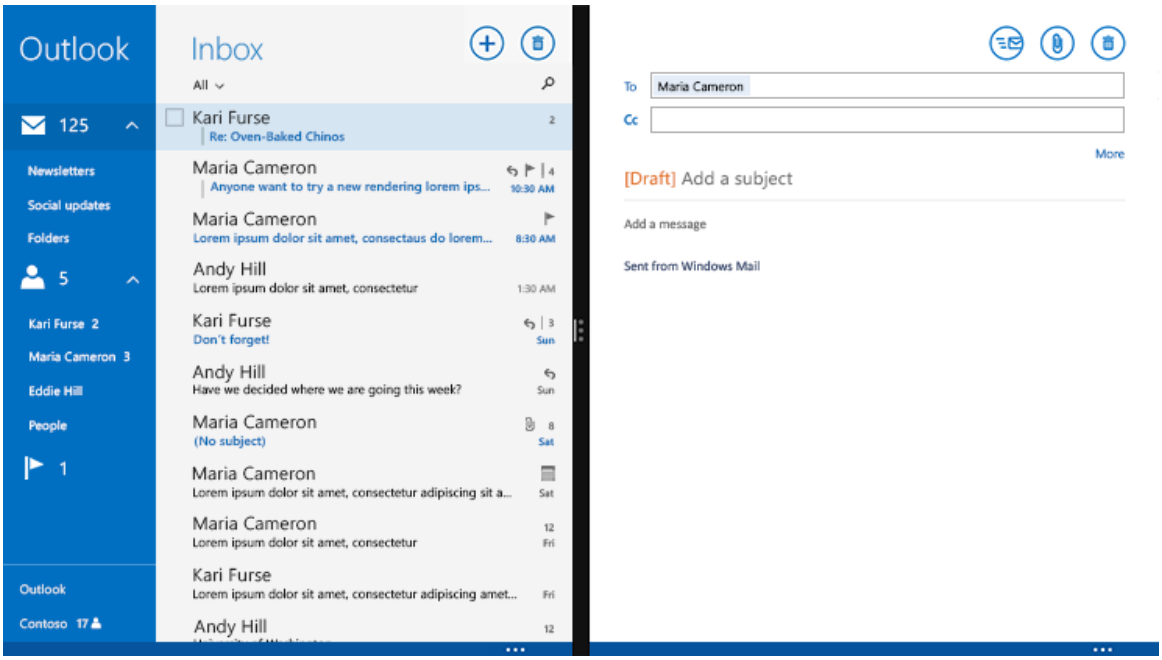
- Next to the original window, sharing the screen space.
- In place of the main window.
- Not on the screen at all.

Once the secondary window is initially displayed, the user controls the placement and size of the window.

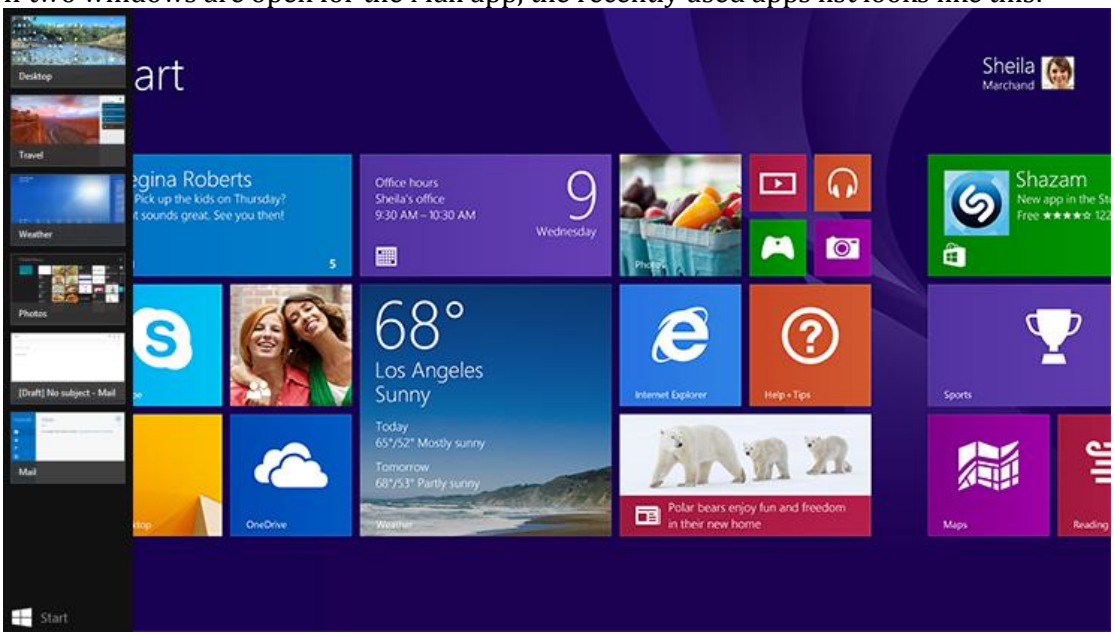
Examples

The Mail app uses multiple windows. A user can view messages in the main app window or open a new window. This is useful when, for example, a user wants to compose a new message, but use the main window to search for other messages at the same time.

© 2014 Microsoft. All rights reserved. | May 13, 2014



If two windows are open for the Mail app, the recently used apps list looks like this:

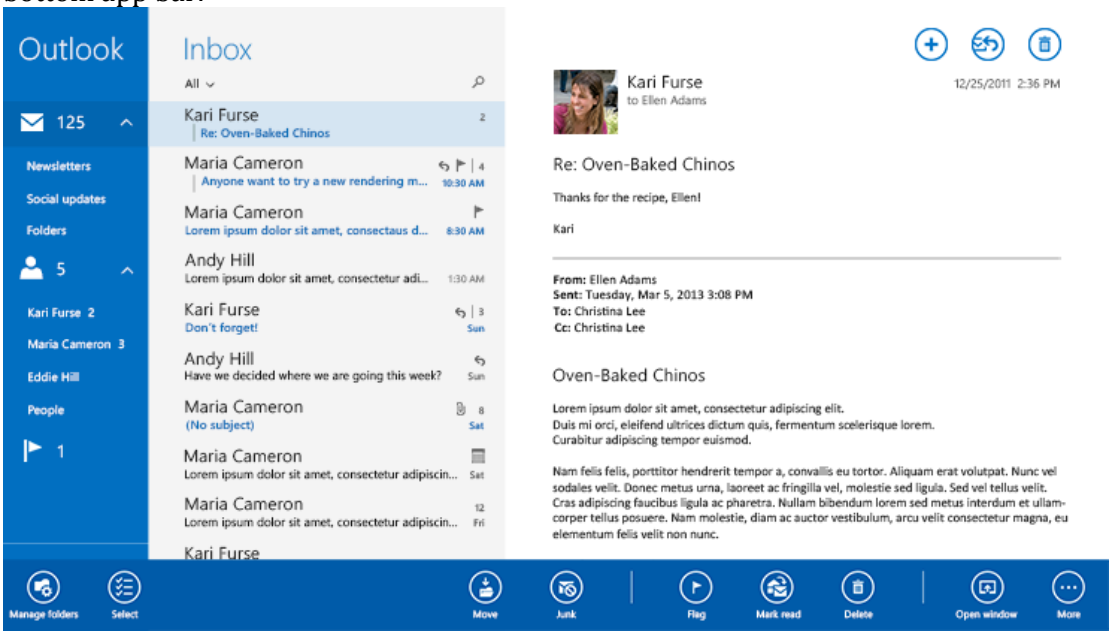


Dos and don'ts

- Provide a way for the user to navigate from a secondary window back to the main window.
- Provide a clear way for the user to open a new window. For example, add a button to the app bar for opening a new window. The Mail app has an **Open Window** button on its

© 2014 Microsoft. All rights reserved.

bottom app bar:



- Make sure the title of the new window reflects the contents of that window. The user should be able to differentiate between the windows of an app based on the title.
- Subscribe to the **consolidated event** and, when the event fires, close the window's contents. The consolidated event occurs when the window is removed from the list of recently used apps or if the user executes a close gesture on it.
- If the new window replaces the original app window, provide custom animation when the windows switch.
- Enable new windows in an app for scenarios that enhance productivity and enable multitasking.
- Design new windows that allow users to accomplish tasks entirely within the window.
- Don't automatically open a new window when a user navigates to a different part of the app. The user should always initiate the opening of a new window.
- Don't require the user to open a new window to complete the main purpose of the app.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for projection manager



Projection manager lets you project a separate window of your app on another screen. For example, a game app can display the main game-playing window on a larger monitor and display the game controls on the local screen. Or in a multi-player word game such as Scrabble, you can display the shared game board on a projected screen and display the user's game pieces on the local screen. In the case of a presentation app, you can display the presentation in a window on a projected screen and display notes for the presenter on the local screen.

By default, without using the projection manager, if a user connects to another display device, the app window is duplicated on the 2nd screen. When in duplicate mode, Windows automatically picks a resolution that works for both screens instead of using the screen info for the external screen, but this resolution might not be ideal for video playback or gaming. When you use the projection manager, Windows retrieves the resolution and the aspect ratio of the projection screen and optimizes the window display.

The projection manager is similar to using multiple windows for an app. This table describes when to use multiple windows and when to use projection manager.

Scenario	Use multiple windows?	Use projection manager?
The user interacts with both windows	Recommended	Not recommended unless the external display is a touch device, such as Perceptive Pixel (PPI) by Microsoft
The 2nd window is for display only, not for interaction	Not recommended	Recommended
You expect the user to display the 2nd window on a screen that has a significantly different aspect ratio or resolution	Not recommended	Recommended

Dos and don'ts

- Let the user control the projection from the local app window.

  The user must be able to:

  - Start a new projected window.
  - Resume a window's projection after it's been interrupted.

© 2014 Microsoft. All rights reserved. | May 13, 2014

    - Stop the projection after it has started.
    - Swap the local and projected windows by using a control on the projected window. If the automatic placement of the windows is wrong and the projected window is placed on the local screen, the user must be able to swap the windows.
- Use the following icons to start projecting, stop projecting, or swap the windows.

Code	Icon	Description
U+E2B4		Start or resume projecting
U+E2B3		Stop projecting
U+E13C		Swap projected view

- Don’t automatically start or stop the projection. Only user input should start or stop the projection.

Note: You can implement "resume" functionality to make it easy for a user to restart a projection after pausing, or after switching to other apps. If a projected window leaves the screen on which it is displayed, typically because of another app projecting, use **StartProjectingAsync** to resume display of the projected window. You can subscribe to the **VisibilityChanged** event to find out when a projected window leaves the screen on which it is displayed. You can subscribe to the **consolidated** event to find out when a projected window is removed from the list of recently used apps or is closed.

Additional usage guidance

Styling and layout

You can choose the color and label text for icons you use to start, stop, and swap a projection. You can choose where to place the icons, but it's a good idea to place them in the bottom app bar and follow the guidance for app bar buttons.

You cannot change the placement of the projected window, because it's determined automatically. A user who has a mouse, keyboard, or touchpad, can move the projected window after it's placed.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for resizing to narrow layouts



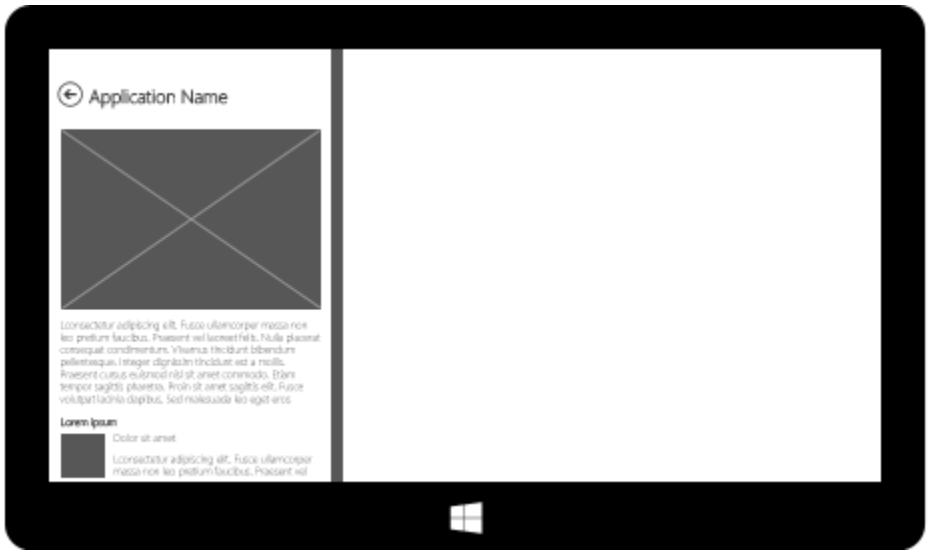
Design your app's UI to adjust when a user resizes it to a tall or narrow view. The guidelines in this topic apply if you plan to:

- Modify your app's minimum width to 320 pixels instead of the default 500 pixels (narrow layout).
- Design your app to switch to a vertical layout when the user resizes it so that the height is greater than the width (tall layout).

Examples

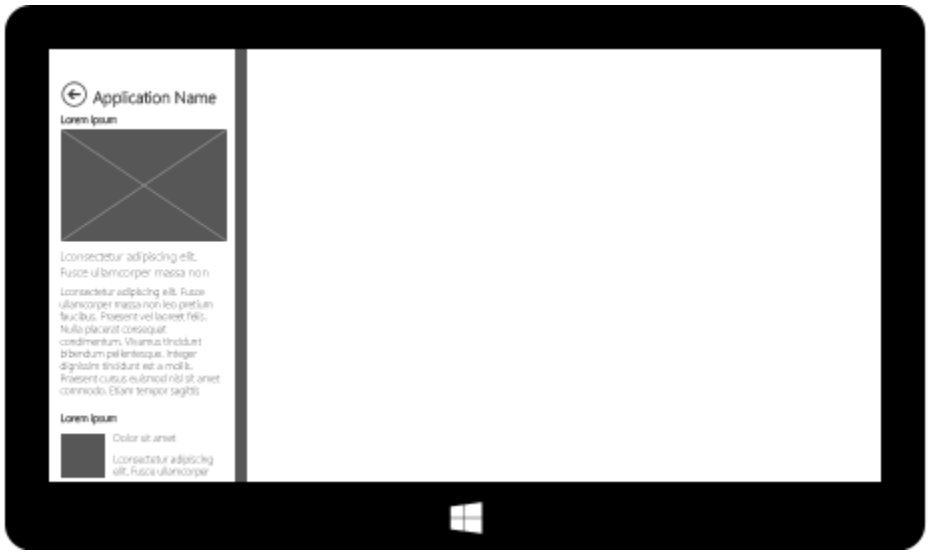
Narrow layouts

The default minimum width for a Windows Store app is 500 pixels. Here is an app at 500 pixels wide.



And here is the app at 320 pixels wide.

© 2014 Microsoft. All rights reserved. | May 13, 2014

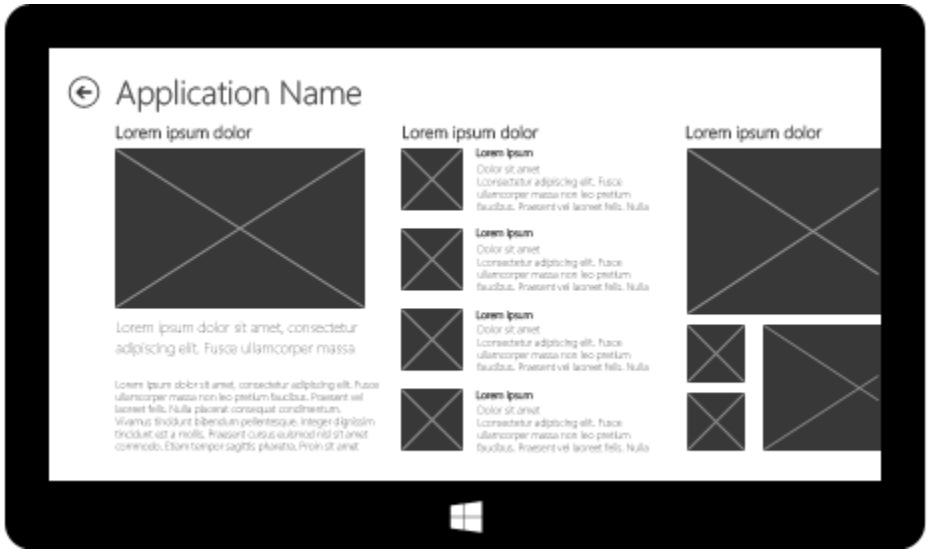


If you choose to support a minimum width narrower than 500 pixels, you should make some design changes so that the app is functional and usable at this size. Follow the *Dos and don'ts* to ensure that your app is effective at widths narrower than 500 pixels.

Tall layouts

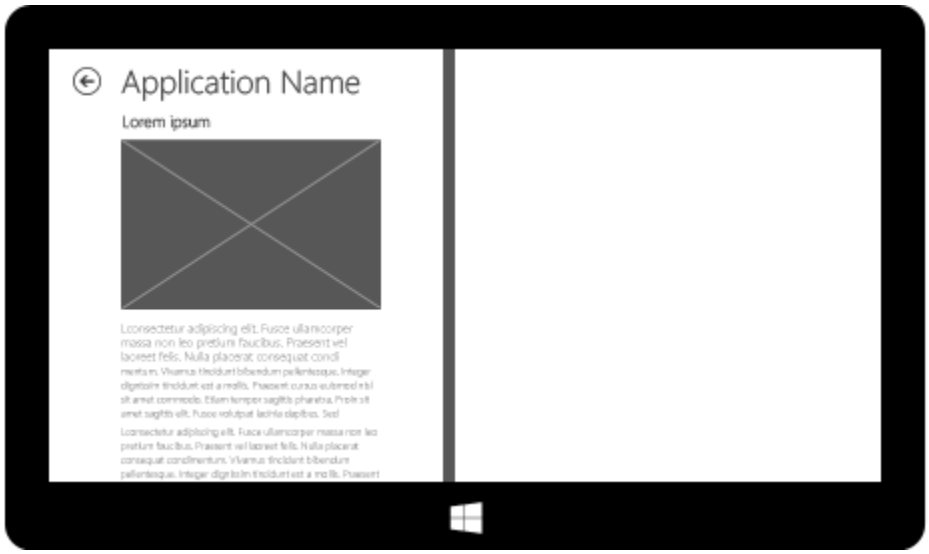
Additionally, you can choose to make design changes when your app is taller than it is wide. For example, you can design the app to pan vertically instead of horizontally when it is taller than it is wide.

Here is an app that pans horizontally when it is at full screen.



And here is the app when it is taller than it is wide. The app now pans vertically.

© 2014 Microsoft. All rights reserved. | May 13, 2014



Should my app support a minimum width of 320 pixels?

Whether you should support widths narrower than the default minimum depends on what you expect users to do with your app. Here are some common scenarios in which you should support narrow widths, down to 320 pixels:

- Multitasking is important for your app.
- You want users to keep your app on the screen.
- Your app works with another app in a companion scenario.
- Your app adapts well to narrow widths.

If you keep the default minimum width of 500 pixels, you don't have to make special considerations for your app at narrow widths.

Dos and don'ts

- If your app pans horizontally when it is full screen, switch to pan vertically when the app window is taller than it is wide.
- To accommodate the narrow size, make following design changes when your app width is less than 500 pixels:
    - Use the smaller back button style.
    - Use 20 pixels for the left margin.
    - Use 20 pt size for the app's header.
    - Use the smaller offset values for page transition animations and content transition animations.

    Here's the Calculator app at a normal width:

© 2014 Microsoft. All rights reserved. | May 13, 2014

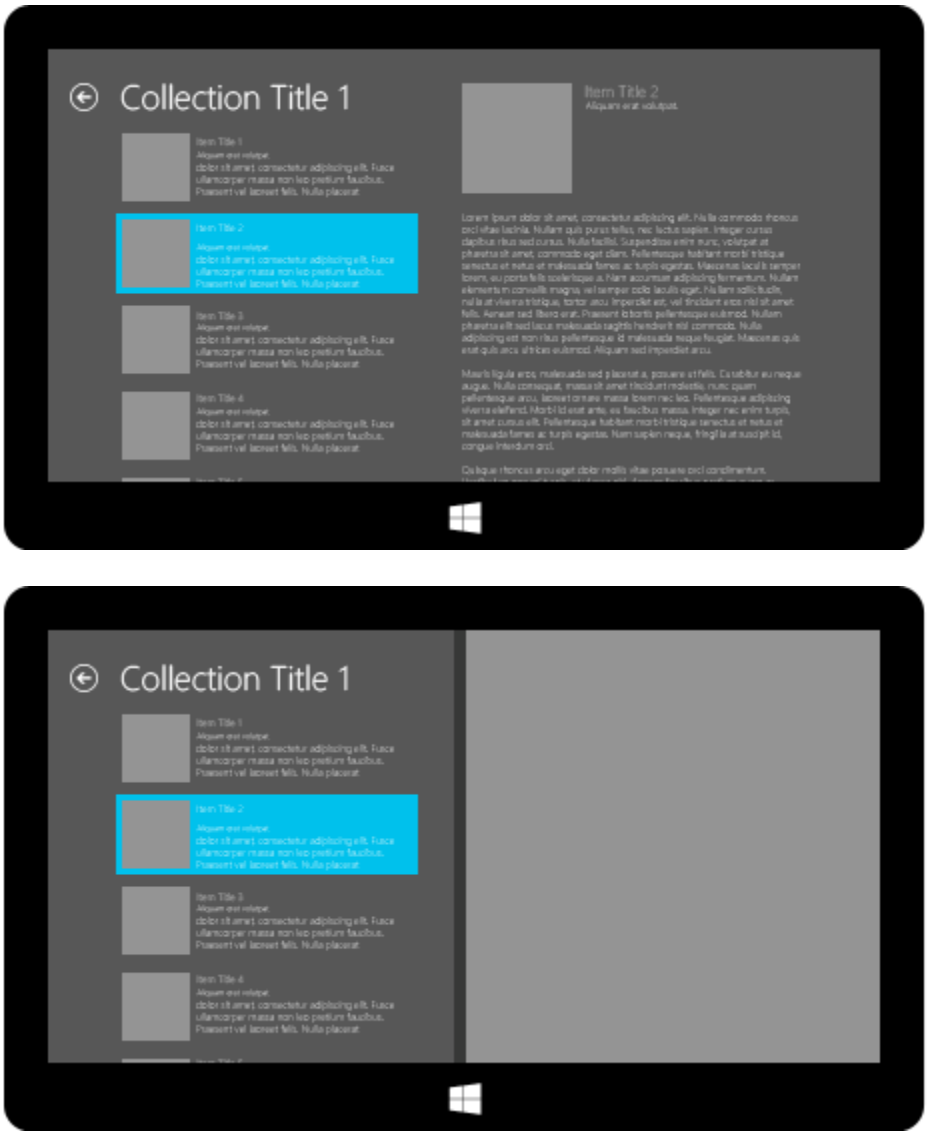


Here's the same app at 320 pixels wide. Its left margin is now 20 pixels and its header font was reduced to 20pt size, per the narrow layout recommendations.



- If your app pans vertically when it is full screen, scale down to display a single column or pane when the app is taller than it is wide. You decide the exact width at which the app switches to a single column or pane. In the single column or single pane view, make sure you include navigation to let users move from one pane to the other.

© 2014 Microsoft. All rights reserved. | May 13, 2014



- Design your app layout and all controls to scale down to the minimum size and to be usable in both tall and narrow app windows. Important controls to consider are:
    - Top and bottom app bars
    - Message dialogs
    - Flyouts
    - Settings pane
- Don't navigate the user to a different part of the app when the window size changes to a narrow width.
- Don't support widths narrower than the default minimum (500 pixels) if you can't preserve most of your app's functionality at narrow sizes.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for scaling to pixel density



Windows Runtime apps (that run on Windows, Windows Phone, or both) are automatically scaled by the system to ensure consistent readability and functionality regardless of a screen's pixel density. Follow these guidelines to preserve the quality of your app's UI when it is scaled for devices with different pixel densities.

Note: These guidelines don't apply to Windows Phone apps using Silverlight.



Description

Without scaling, the physical sizes of objects on screen get smaller as the pixel density of a device increases. When UI elements would otherwise be too small to touch or when text would become too small to read, Windows automatically scales your app based on the following scaling plateaus:

Windows Store apps:

- 1.0 (100%, no scaling is applied)
- 1.4 (140% scaling)
- 1.8 (180% scaling)

Windows Phone Store apps:

- 1.0 (100%, no scaling)
- 1.4 (140% scaling)
- 2.4 (240% scaling)

© 2014 Microsoft. All rights reserved. | May 13, 2014

Windows determines which scaling plateau to use based on the physical screen size, the screen resolution, the DPI of the screen, and form factor. If the screen specifications meet a specific threshold, Windows uses the next higher scaling plateau. You can use **ResolutionScale** (Windows) or **RawPixelsPerViewPixel** (Windows Phone) to determine scale factor.

Your app will be scaled automatically by the system, but in order to ensure that your UI is crisp and functional regardless of a device's pixel density, use the following guidance to prepare your app for scaling.

Dos and don'ts

- Use scalable vector graphics. Windows scales these formats for you automatically, without noticeable artifacts. For JavaScript apps, use SVG. You can use XAML-defined graphics in apps using C#, C++, or Visual Basic.
- Use resource loading for bitmap images in the app package and provide a separate image for each scaling factor. Include the scale factor in the name of your image file (for example, Assets\Square7070Logo.scale-100.png). Note that Windows will automatically load the right image for the current scale. The Scaling according to DPI sample shows how to use resource loading for images.
- When creating your assets for different scaling plateaus:
    - Don't design bitmap images at 100% and manually scale them up. Even if you use a high-quality image program, you'll likely get blurry results.
    - Keep in mind that scaling down a large, high-resolution image won't always produce clean, crisp results. However, if the original vector isn't available, manually scaling down a higher resolution file is better than scaling up from a low-resolution file.
- If your app is loading images at runtime using code (for example, if you're using DirectX, not XAML or HTML to create your UI), use **ResolutionScale** (Windows) or **RawPixelsPerViewPixel** (Windows Phone) to determine the scale and manually load images based upon that scale percentage.
- Use the **Thumbnail** APIs for file system images. The thumbnail APIs optimize performance by caching smaller versions of the image for thumbnail use.
- Specify width and height for your images instead of using auto sizing to prevent layouts from changing when larger images are loaded.
- Use typographic grid-units and sub-units. Use the typographic, grid-defined sizes of 20px for major grid-units and 5px for minor grid-units, to ensure that your layout doesn't experience pixel shifting due to pixel rounding. Any sized unit that is divisible by 5px will not experience pixel rounding.
- Use the resolution media query for remote web images. If your app uses JavaScript and you have a remote web image, use the CSS @media resolution media feature with the **background-image** property to replace images at runtime.
- Don't use images that aren't sized to multiples of 5px. Units that aren't multiples of 5px can experience pixel shifting when scaled to 140%, 180%, and 240%.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for supporting multiple screen sizes



Windows and Windows Phone Store apps can run on a variety of devices with different screen sizes and resolutions. Users can also modify the screen's orientation, or, if the app is running on Windows, continuously resize it down to a minimum width or display it beside other apps. Whether a user runs your app on a phone, tablet, laptop, desktop, or PPI device, ensure that its UI looks good and maintains its functionality. Follow these guidelines when designing a UI that adapts to screens of all sizes and orientations.

Description

Your app will likely run on a variety of screen sizes which could range from a small phone screen, to a medium laptop screen, and perhaps even an all-in-one or PPI device screen. Depending on screen size and resolution, the amount of viewable area for your app to take advantage of will vary.



© 2014 Microsoft. All rights reserved. | May 13, 2014



The following terms are important for understanding scaling to different screen sizes.

Term	Description
Screen size	The physical size of the screen, in inches. Usually measured on the diagonal.
Screen resolution	The number of pixels the screen supports, in horizontal and vertical dimensions. For example, 1366x768.
Aspect ratio	The shape of the screen as a proportion of width to height. For example, 16:9.

The platform, controls, and templates have all been designed to accommodate different screen sizes, resolutions, and aspect ratios. Although much of your app's layout will automatically adjust to display changes, you must consider your top-level layout, content regions, app navigation, and commands to ensure that they are placed predictably and intuitively on all screens.

The following tables show the most important screen sizes to consider when designing your app.

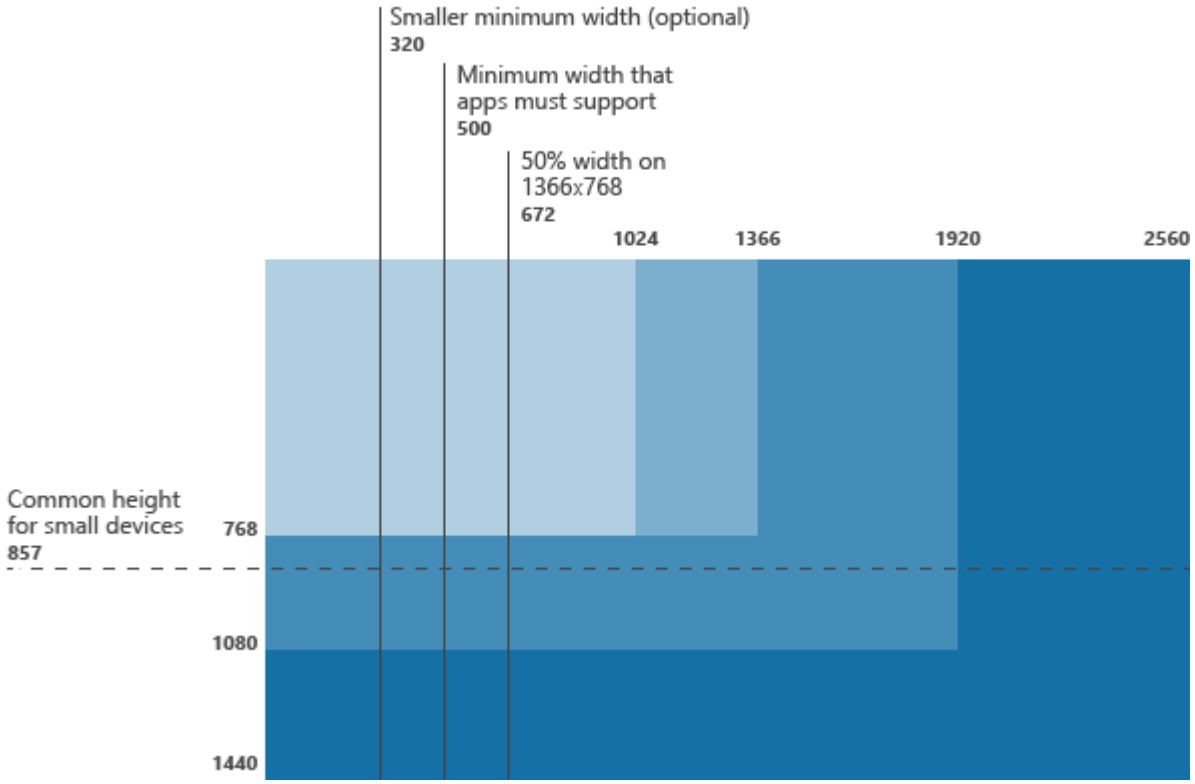
Full-screen screen size (effective pixel resolution)	Device description
1366x768	Tablets, convertibles, and many laptops (16:9 aspect ratio); baseline laptop/desktop resolution
1920x1080	Large laptops and devices (16:9 aspect ratio)

© 2014 Microsoft. All rights reserved. | May 13, 2014

2560x1440	Very large all-in-one device (16:9 aspect ratio)
1280x800 and 800x1280	Portrait-first small devices (16:10 aspect ratio)
1024x768 and 768x1024	Landscape-first small devices (4:3 aspect ratio)
1371x857 and 857x1371	Small devices (16:10 aspect ratio)
384x640	4.5" phone (15:9 aspect ratio)
400x711	4.7" phone (16:9 aspect ratio)
450x800	5.5" phone (16:9 aspect ratio)
491x873	6" phone (16:9 aspect ratio)

When designing an app to run on Windows, not Windows Phone, consider the screen size available when a user has two apps share the screen, or resizes your app to its minimum width.

Split screen size (effective pixel resolution)	Description
672x768	Screen split in half on a 1366x768 device
500x768	Default minimum size for app; Screen split in half on a 1024x768 device
320x768	Minimum size for apps that support 320 pixel minimum width



Dos and don'ts

- When possible, rely on flexible controls to support content that reflows automatically. Flexible controls include the **XAML Grid control**, CSS grid, CSS Multi-column layout, and **ScrollViewer control**. Grid controls, for example, flex specific sections of your UI to fill available space depending on the screen resolution of the display device and assign content to different cells based on available screen space.
- Design your app layout and controls to adjust and function on screens of the minimum size:
    - Default minimum width for Windows Store apps: 500px.
    - Non-default minimum width for Windows Store apps: 320px.
    - Minimum (not adjustable) for Windows Phone Store apps: 384px (portrait) and 640px (landscape).
- The UI and controls must be usable at all screen sizes, down to the minimums (listed above). Important controls to consider are:
    - Top and bottom app bars
    - Message dialogs (Windows Store apps)
    - Flyouts (Windows Store apps)
    - Settings pane (Windows Store apps)
    - Pivot control (Windows Phone)
    - Hub control (Windows Phone)
- Design your app to effectively use the space on a large screen and to have a layout that reflows automatically. Don't leave large empty spaces.
- Test that your app works well on the most important device sizes. In addition to testing your app on actual devices, you can use the Microsoft Visual Studio simulator for Windows Store apps to simulate running your app on different physical screen sizes, resolutions, and orientations.
- Specify a minimum size for all **input fields**. Minimum sizes ensure that the input fields don't disappear when the user resizes the window.
- Test that your app's input fields are not occluded by the soft keyboard.
- Be cautious when using absolute positioning; if used improperly it can prevent your UI from responding to changes in window size and orientation. Rather than hardcoding your layout, use positions computed at runtime to layout your UI.
- Design for different pixel densities.

Windows Store apps only

- Ensure that your app is functional down to the default minimum width of 500 pixels. See **Guidelines for narrow layouts** for specific recommendations.
- If your app works well at smaller sizes and you want to encourage users to keep your app on the screen, you can support a non-default minimum width of 320 pixels.
- Make sure that users can continue their current tasks when they resize apps. For example, maintain the current page of the app, state of scroll bars, selection, and focus.
- Support charms at all screen sizes. Make sure flyouts and panes, are scaled appropriately.

Note: In Windows 8, users could resize apps to only three view states: full screen, snapped, and fill. In Windows 8.1, users can resize apps to any width from full screen down to the minimum width.

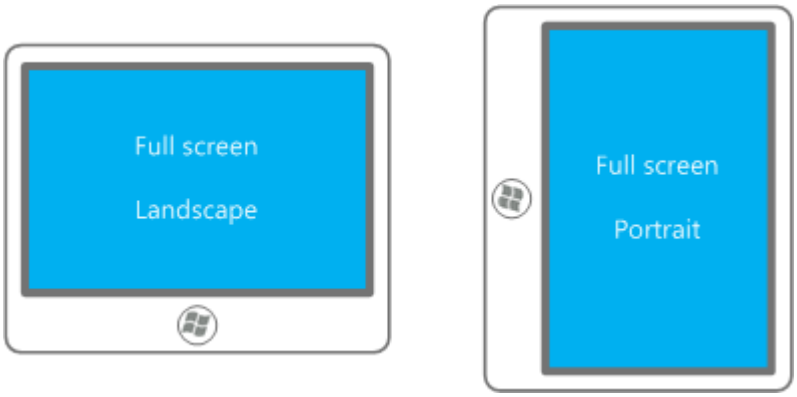
© 2014 Microsoft. All rights reserved. | May 13, 2014

Additional usage guidance

Automatic support for orientation changes

Users can rotate their phones, tablets, and monitors. Windows handles both landscape and portrait orientations automatically, as long as your app isn't using a fixed layout. You'll only need to consider how the app's width affect its layout.

See the fixed layout section below for more info on when it might be appropriate to forgo a flexible layout.



Resized app behavior (Windows Store apps only)

If a user has multiple apps on the screen, note these unique UI interactions:

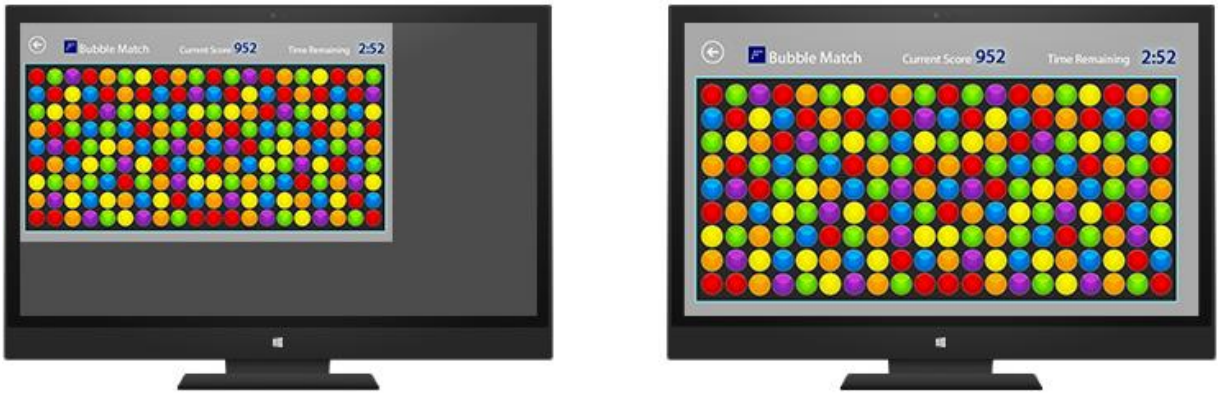
- If a user invokes the charms, the charms apply to the last app that the user used, regardless of the size of the app or the position of the app on the screen.
- Between each app on the screen is a handle. Users resize app windows by sliding the handle. The handle also shows which app has focus.
- If a user grabs the handle between apps and tries to resize an app to a width that is less than the app's minimum width, the app leaves the screen.
- If a user rotates a device or monitor while multiple apps are on the screen, the apps do not switch orientation.

Fixed layouts

Most apps can use a dynamic layout that adapts to changes in screen size and resolution, and reflows content automatically. But in some cases, your app may require a fixed layout. Apps that aren't content-focused or depend on the integrity of graphics, such as gaming apps, need to use fixed (absolute) layouts. Windows accommodates these apps with a "scale-to-fit" approach that is built into the platform.

© 2014 Microsoft. All rights reserved. | May 13, 2014

If you determine that your app requires a fixed layout that won't automatically adapt to different screen sizes, you can use a scale-to-fit approach to make your fixed layout fill the screen on different screen sizes, as shown in the following images.

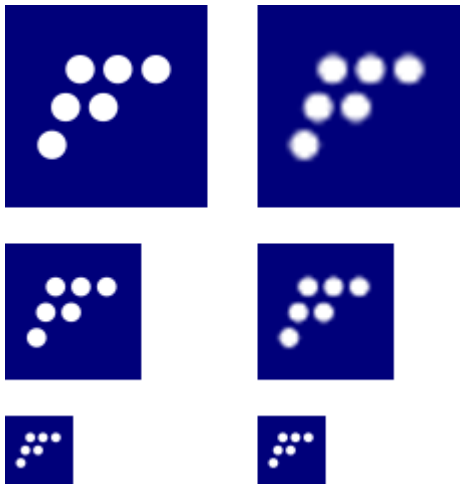


To implement the scale to fit approach, do the following:

- Design your layout for the baseline resolution, for example, 1366x768 pixels (PC/tablet) or 384x640 pixels (phone). This is the layout that will be scaled to the larger screens.
- Place your fixed content within a ViewBox control. The ViewBox control scales a fixed layout to fit the screen.
- Ensure that the ViewBox control is sized to 100% width and height.
- Define the fixed size properties of the ViewBox to the fixed pixel sizes of your layout (for example, 1366x768 or 384x640).
- Choose a letterboxing color. Fixed controls do not dynamically change in response to aspect ratio or screen size changes, so the scale-to-fit technique automatically centers and letterboxes (horizontally, or vertically) your app's content. Your top-level app layout's color determines the color of the letterbox bars. We recommend choosing a dark color like black that blends in with the hardware, a neutral color like grey that looks intentional, or a color that matches your app content color.
- Provide vector or high-resolution assets. The scale-to-fit technique scales your application to varying sizes up to 180% (for Windows) or 280% (for Windows Phone) of the design size for your app on a large desktop monitor. Vector assets like Scalable Vector Graphics (SVG), Extensible Application Markup Language (XAML), or design primitives scale without scaling artifacts or blurriness. If raster assets (such as bitmap images) are required, provide MRT assets.

  The following images demonstrate how scalar images (right) degrade when scaled up in size compared to vector images (left).

© 2014 Microsoft. All rights reserved. | May 13, 2014



- Don't place adaptive controls in a ViewBox control.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for text and input



This section includes info about fonts and typography, icons, text boxes, text forms, and copy and paste commands for text. It also includes guidelines for searching for specific text on a page, for example, in document viewer or reader apps.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for clipboard commands



Clipboard commands—copy, paste, and cut—provide users with a familiar way to transfer content from one location to another. With these commands you can help users transfer content:

- Within the same app
- Between Windows Store apps
- Between desktop applications
- Between Windows Store apps and desktop applications

Although Windows 8 and Windows 8.1 support other ways for apps to exchange information—such as through sharing content—copy and paste commands remain an expected part of the Windows experience. Your app should support them whenever possible.

Dos and don'ts

- Support copy and paste for any editable content that a user can explicitly select—such as a subset of a document or an image.
- Consider supporting copy and paste commands for content that users might want to use somewhere else. For example:
    - Images in a photo gallery application
    - Computation results in a calculator
    - Restaurant address in a restaurant-search application
- Be aware of rights management and other factors that might restrict the use of copy and paste commands. For example, if your app supports viewing rights-managed mail, a policy might restrict the user from copying all or parts of such content.
- Make sure it's clear what a user is copying, or where a user can paste content.
- Provide support for paste only on editable regions and canvases in your application.
- Consider implementing an undo command, as copy and paste can lead to content being deleted or replaced.
- If a control already supports copy and paste, use the control's implementation. If you need to build your own implementation of copy and paste, make sure the experience you create is consistent with these controls.
- Consider supporting sharing if you are also supporting copy.
- Determine if users should access the copy and paste commands by using a context menu or the app bar. Use a context menu:
    - For items that users can select only through tap-and-hold gestures—such as hyperlinks or embedded images. For example, let's say your app displays an address to the user, and you want the user to be able to copy that address. A great user experience would be to create a Copy Address command that users can access when they either right-click or tap-and-hold the address. This command would then copy

© 2014 Microsoft. All rights reserved. | May 13, 2014

the address to the clipboard, from which the user can paste it into the app of their choice.



  - For text selection (both editable and read-only).
  - For paste operations where the target is well defined, such as a cursor location or a table cell.

  Use the app bar if the preceding guidelines don't apply. Some examples include:

  - When your app supports the selection of multiple items.
  - When the user can select a portion of an image.
  - When the target of a paste command is clear—such as pasting a screen shot on a canvas.
- We strongly encourage you to always support keyboard shortcuts for clipboard commands.
- Don't provide support for copying content that can’t be selected—either explicitly, or through a context menu.
- Don't provide support for copying text that is not part of the core content of your application. Titles, headers, and button text do not need to be copied.
- Don't enable the paste command when the clipboard is empty or if it contains content that your app doesn't support.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for find-in-page



This topic describes best practices for implementing find-in-page functionality in your Windows Store apps.

Dos and don'ts

- Implement find-in-page to enable users to find matches in the current body of text.
- Use an app bar to let users find text with your app.
- Enable the keyboard shortcut (CTRL+F).
- Don't use the Search charm to find text in the current body of text.
- Avoid summary panes.

Additional usage guidance

Implementing find-in-page

Find-in-page enables users to find matches in the current body of text.

- Document viewers and readers are the most likely kind of apps to provide find-in-page.
- The primary purpose of these apps is to enable the user to have a full screen viewing/reading experience.
- Find-in-page functionality is secondary and should be located in an app bar, along other functionality provided for the user that they use when needed.

The Search charm enables searching through a set of items, like web pages, movies, and events. The user enters a query into the search box, and a related result set is returned. Often, these results are sorted by using a relevance algorithm.

User experience guidelines



The find-in-page user experience in Internet Explorer in the new Windows UI.



The find-in-page user experience in the reader app.

© 2014 Microsoft. All rights reserved. | May 13, 2014

This list presents the practices recommended for adding find-in-page to an app.

- Use an app bar to let users find text with your app. Find-in-page functionality is secondary and should be located in an app bar.
    - Apps that provide find-in-page should have all necessary controls in an app bar.
    - If your app includes a lot of functionality beyond find-in-page, you can provide a **Find** button in the top-level app bar as an entry point to another app bar that contains all of your find-in-page controls.
- The find-in-page app bar should remain visible when interacting with the touch keyboard. The touch keyboard appears when a user taps the input box. The find-in-page app bar should move up, so it's not obscured by the touch keyboard.
- Find-in-page should remain available while the user interacts with the view. Users need to interact with the in-view text while using find-in-page. For example, users may want to zoom in or out of a document or pan the view to read the text. Once the user starts using find-in-page, the app bar should remain available with a **Close** button to exit find-in-page.
- Enable the keyboard shortcut (CTRL+F). Implement the keyboard shortcut CTRL+F to enable the user to invoke the find-in-page app bar quickly.
- Include the basics of find-in-page functionality. These are the UI elements that you need in order to implement find-in-page:
    - Input box
    - Previous and Next buttons
    - A match count
    - Close
- The view should highlight matches and scroll to show the next match on screen. Users can move quickly through the document by using the **Previous** and **Next** buttons and by using scroll bars or direct manipulation with touch.
- Find-and-replace functionality should work alongside the basic find-in-page functionality. For apps that have find-and-replace, ensure that find-in-page doesn't interfere with find-and-replace functionality.
- Summary panes are redundant and should not be included in find-in-page functionality.
    - The easiest way for a user to identify the match they are looking for is to see it in the document, because seeing the match inline gives much more context than a summary-of-results pane.
    - Users can move quickly through the document by using the **Previous** and **Next** buttons and by using scroll bars or direct manipulation with touch.
    - If you choose to include a summary pane, follow these guidelines.
        - Users should be able to access the summary pane by using a toggle button.
        - When the summary pane is toggled on, it should be visible whenever text is in the input box.
        - If the user closes the find-in-page app bar, then the summary pane should not be visible until the next time that the user enters text in the input box.
        - If the summary pane is toggled off it should not show again until the user toggles it back to the on state.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for fonts



The proper use of font sizes, weights, colors, tracking, and spacing can help give your Windows Store app a clean, uncluttered look that makes it easier to use. Follow these guidelines when selecting fonts and specifying font sizes and colors.

There are style resources available that make it easier to adhere to these guidelines:

- For Windows Store apps using JavaScript, use the **WinJS style sheets**. These style sheets use the recommended colors, weights, and tracking values provided in this topic.
- If you're creating a Windows Store app using XAML, use the theme resources associated with styling a **TextBlock** or **RichTextBlock** to easily implement these guidelines.

Dos and don'ts

Achieving simplicity and clarity require careful attention to typographic detail. In developing your app's UI, we recommend that you apply these rules:

- Use the recommended fonts.
    - Use Segoe UI (the primary Windows typeface) for UI elements such as buttons and date pickers. Segoe UI supports Latin, Cyrillic, Greek, Arabic, Hebrew, Armenian, and Georgian alphabets.
    - Use Calibri for text that the user both reads and writes such as email and chat. Calibri supports Latin, Greek and Cyrillic alphabets. See *For reading and writing* in the *Additional usage guidance* section below.
    - Use Cambria for larger blocks of text such as for a magazine or RSS reader. Cambria supports Latin, Greek and Cyrillic alphabets. See *For reading* in the *Additional usage guidance* below.

Note: Other languages and writing systems use different fonts and may require different sizing and spacing. When localizing your app, if you are using in-box fonts, use the **Windows.Globalization.Fonts** APIs to identify the correct UI and document fonts for a given language.

- For your UI elements, use these font styles and sizes:
    - Use 11 pt Segoe UI Semilight for most text.
    - Use 9 pt Segoe UI, the smallest size, for short text elements, such as button captions.
    - Use 20 pt Segoe UI Light for text elements that need to be clearly visible and draw user attention but fit on a single line. If 20 pt is too big, use 16 pt Segoe UI Semilight instead. For example, 16pt font might be better if you have a dense list with many items. However, don't use both 20 pt and 16 pt Segoe UI fonts on the same page.

© 2014 Microsoft. All rights reserved. | May 13, 2014

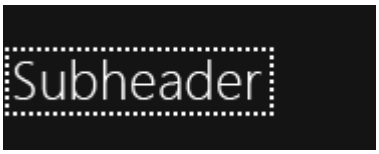
    - Use 42 pt Segoe UI Light for prominent UI elements that consist of one or two words on a single line.

  See *Additional usage guidance* for more detailed Segoe UI weight, opacity, letter spacing, and stylistic set recommendations.

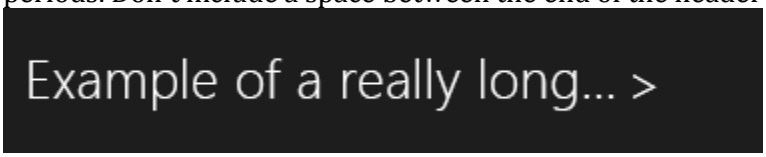
- Use contrast (of weight, size, and position) to distinguish page elements if necessary.
- Use the following opacities:
  - Primary text has an opacity of 100%.
  - Secondary text has an opacity of 60%.
  - Text in the hover state has an opacity of 80%.
  - Black and white text in the pressed state has an opacity of 40%. Text in other colors has an opacity of 60%.

  For examples of Segoe UI fonts at different sizes and opacities, see *Segoe UI type ramp* in *Additional usage guidance*.

- Indicate keyboard focus by alternating black and white pixel squares.



- If a page header or subheader is too long to display, use an ellipsis (U+2026), not three periods. Don't include a space between the end of the header text and the ellipsis.



- Follow the layout guidelines for Windows Store apps.
- Don't use italic fonts.

Additional usage guidance

Predefined weights for Segoe UI

Segoe UI is the most recognizable font in the system and is the font most closely associated with new Windows UI. Use Segoe UI for all user-interface elements.

The Windows Store app style sheets provide five different versions of Segoe UI, each with a different weight:

- Segoe UI Light

  Segoe UI with a weight of 200.

© 2014 Microsoft. All rights reserved. | May 13, 2014

- Segoe UI Semilight

  Segoe UI with a weight of 300.

- Segoe UI

  Segoe UI at the regular weight, 400.

- Segoe UI Semibold

  Segoe UI with a weight of 600.

- Segoe UI Bold

  Segoe UI with a weight of 700.

To use these predefined weights, set the font family to one of these Segoe UI font names. For apps using JavaScript and HTML, use the Cascading Style Sheets (CSS) **font-family** property. If you're using C#/VB/C++ or XAML, use **FontFamily**.

CSS

```
<p style="font-family: 'Segoe UI Semibold'">Semibold text</p>
```

C#

```
<TextBlock x:Name="semiBold" Text="Semibold font." FontFamily="Segoe UI Semibold"/>
```

Segoe UI type ramp

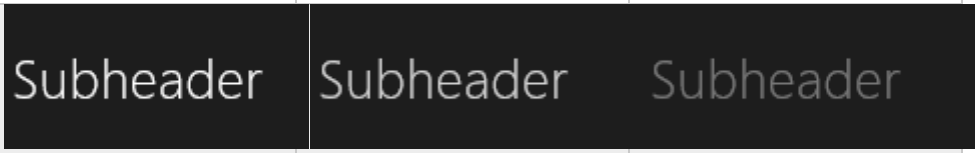
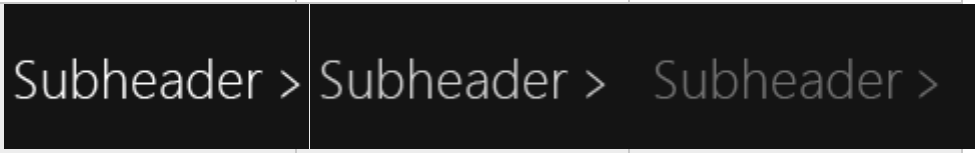
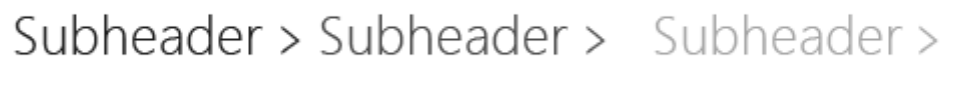
This type ramp shows which sizes, weights, and colors to use for different parts of the UI. For each portion of the UI, it shows which colors to use when using the dark style sheet and which colors to use when using the light style sheet.

- Primary text has an opacity of 100%.
- Secondary text has an opacity of 60%.
- Text in the hover state has an opacity of 80%.
- Black and white text in the pressed state has an opacity of 40%. Text in other colors has an opacity of 60%.

(The "Illustration key" in each entry identifies that text style in the image that follows this table.)

Default view	Rest state	Hover state (opacity: 80%)	Pressed state (opacity: 40%)

© 2014 Microsoft. All rights reserved. | May 13, 2014

Page header • **Do not use when the app width is 500 pixels or less** • **Font family: Segoe UI Light** • **Font size: 42pt** • **Illustration key: A**			
	rgba(255, 255, 255, 1.0)	rgba(255, 255, 255, 0.8)	rgba(255, 255, 255, 0.4)
			
	rgba(0, 0, 0, 1.0)	rgba(0, 0, 0, 0.8)	rgba(0, 0, 0, 0.4)
Page subheader • **Font family: Segoe UI Light** • **Font size: 20pt** • **Illustration key: B**			
	rgba(255, 255, 255, 1.0)	rgba(255, 255, 255, 0.8)	rgba(255, 255, 255, 0.4)
			
	rgba(0, 0, 0, 1.0)	rgba(0, 0, 0, 0.8)	rgba(0, 0, 0, 0.4)
Interactive subheader • **Font family: Segoe UI Light** • **Font size: 20 pt** • **Glyph: Segoe UI Symbol U+E26B** • **Spacing between text and glyph: 10 pixels**			
	rgba(255, 255, 255, 1.0)	rgba(255, 255, 255, 0.8)	rgba(255, 255, 255, 0.4)
			
	rgba(0, 0, 0, 1.0)	rgba(0, 0, 0, 0.8)	rgba(0, 0, 0, 0.4)

© 2014 Microsoft. All rights reserved. | May 13, 2014

Item title/header • **Small subheader** • **Font family: Segoe UI Semibold** • **Font size: 11pt** • **Illustration key: C**	Small subheader	Small subheader	Small subheader
	rgba(255, 255, 255, 1.0)	rgba(255, 255, 255, 0.8)	rgba(255, 255, 255, 0.4)
	Small subheader	Small subheader	Small subheader
	rgba(0, 0, 0, 1.0)	rgba(0, 0, 0, 0.8)	rgba(0, 0, 0, 0.4)
Item title/header, secondary • **Font family: Segoe UI Semibold** • **Font size: 11pt** • **Illustration key: D**	Small secondary	Small secondary	Small secondary
	rgba(255, 255, 255, 0.6)	rgba(255, 255, 255, 0.8)	rgba(255, 255, 255, 0.4)
			Small secondary
	rgba(0, 0, 0, 0.6)	rgba(0, 0, 0, 0.8)	rgba(0, 0, 0, 0.4)
Navigation • **Item title/header, Navigation** • **Font family: Segoe UI** • **Font size: 11pt**	Navigation	Navigation	Navigation
	rgba(255, 255, 255, 1.0)	rgba(255, 255, 255, 0.8)	rgba(255, 255, 255, 0.4)
	Navigation	Navigation	Navigation
	rgba(0, 0, 0, 1.0)	rgba(0, 0, 0, 0.8)	rgba(0, 0, 0, 0.4)
Body text • **Font family: Segoe UI Semilight**	Body text	N/A	N/A

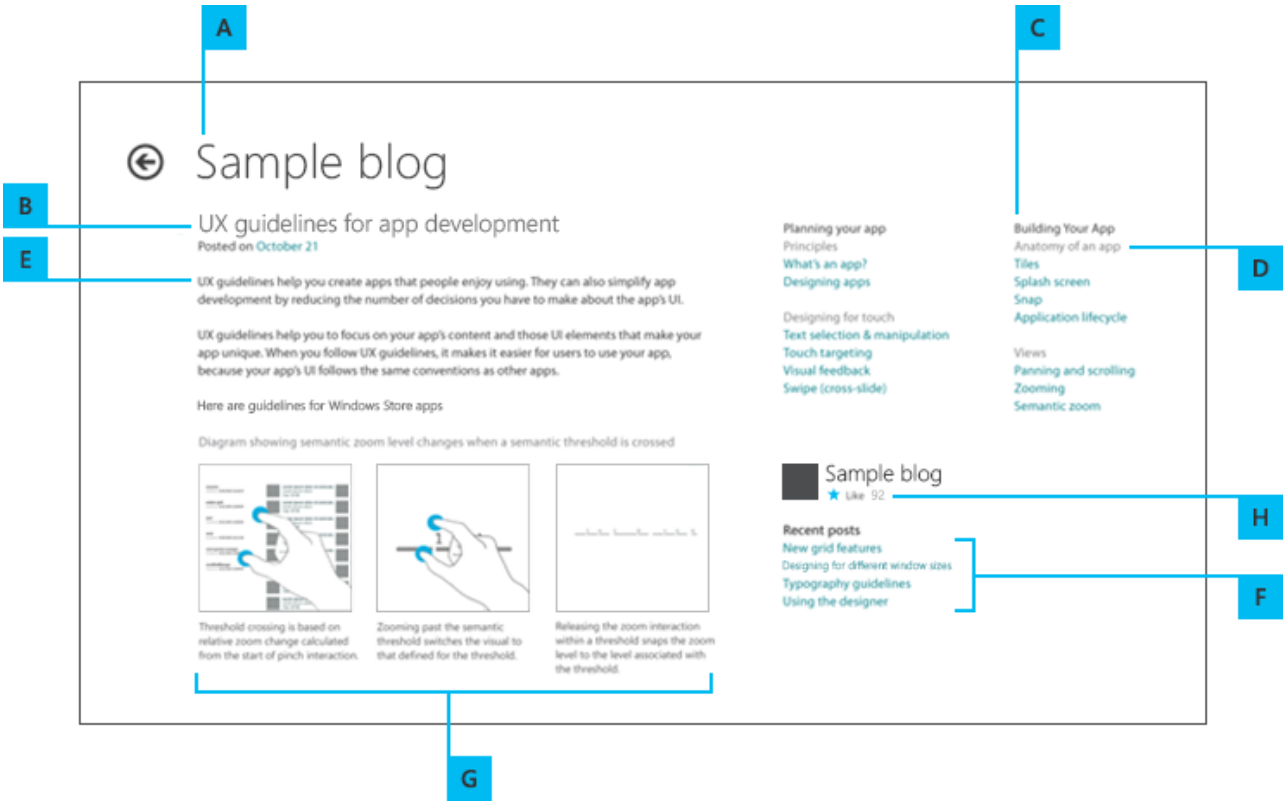
© 2014 Microsoft. All rights reserved. | May 13, 2014

• **Font size: 11pt** • **Illustration key: E**	rgba(255, 255, 255, 1.0)		
	Body text	N/A	N/A
	rgba(0, 0, 0, 1.0)		
Links • **Font family: Segoe UI Semilight** • **Font size: 11pt** • **Illustration key: F**	Links	Links	Links (60% opacity)
	rgba(148, 102, 255, 1.0)	rgba(148, 102, 255, 0.8)	rgba(148, 102, 255, 0.6)
	Links	Links	Links (60% opacity)
	rgba(79,26,203, 1.0)	rgba(79,26,203, 0.8)	rgba(79,26,203, 0.6)
Tertiary info • **Font family: Segoe UI** • **Font size: 9pt** • **Illustration key: G**	Tertiary info	Tertiary info	Tertiary info
	rgba(255, 255, 255, 1.0)	rgba(255, 255, 255, 0.8)	rgba(255, 255, 255, 0.4)
	Tertiary info	Tertiary info	Tertiary info
	rgba(0, 0, 0, 1.0)	rgba(0, 0, 0, 0.8)	rgba(0, 0, 0, 0.4)
Tertiary info, secondary • **Font family: Segoe UI** • **Font size: 9pt**	Tertiary info	Tertiary info	Tertiary info
	rgba(255, 255, 255, 0.6)	rgba(255, 255, 255, 0.8)	rgba(255, 255, 255, 0.4)

© 2014 Microsoft. All rights reserved. | May 13, 2014

• **Illustration key: H**	Tertiary info `rgba(0, 0, 0, 0.6)`	Tertiary info `rgba(0, 0, 0, 0.8)`	Tertiary info `rgba(0, 0, 0, 0.4)`

You can see how Windows Store app layout and typography applies to a basic blog article in this illustration:



Note: The Windows type ramp is optimized for light text on dark backgrounds, and uses lighter-weight fonts. If your app mostly displays dark text on a light background, it might not be suitable to follow this ramp because fonts such as Segoe UI Light or Semilight don't work well at small sizes (dark gray text on a light gray background). Heavier font weights, such as Regular or Semibold, are better for dark text on a light background.

If you decide to diverge from the recommended type ramp, keep the range of type sizes and styles to a minimum.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Segoe UI tracking (letter-spacing)

Tracking (global letter-spacing) in the UI is important to the overall readability of the text, particularly when it appears against dark or complex backgrounds.

Tracking adjustments are usually very tiny, and are almost imperceptible unless the letters are very large, but these tiny adjustments can make a huge difference in the readability of a paragraph or a page.

Tracking is measured in proportional units, rather than fixed units like pixels or millimeters. The proportional unit, an em, is equal to the type size in points. For example, the width of em for an 11pt type is 11 points. To set the tracking, set the **letter-spacing** CSS property.

We recommend these tracking values for Segoe UI, based on font size and weight:

Size	Weight	Tracking (letter-spacing) value
42 pt	Light	0.00 em
20 pt	Regular	0.01 em
All others	All	0.02 em

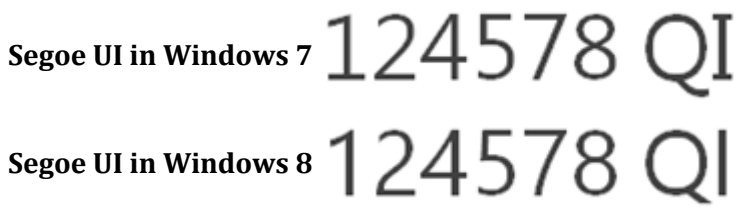
When you use the Windows Library for JavaScript style sheets, these tracking values are set for you automatically.

We recommend that you keep the default tracking of the Cambria or Calibri fonts.

Changes in Windows 8 to Segoe UI stylistic set

Windows 8 uses a version of Segoe UI that has a number of redesigned default characters, new weights, new Microsoft OpenType alternates, and expanded language support. Note that these redesigned characters don't change character widths in the regular and bold styles. That means existing dialogs, content, and web sites don't have to reflow when you use the new version of Segoe UI.

This illustration shows the notable characters that have been redesigned.



© 2014 Microsoft. All rights reserved. | May 13, 2014

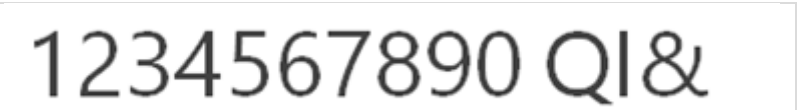
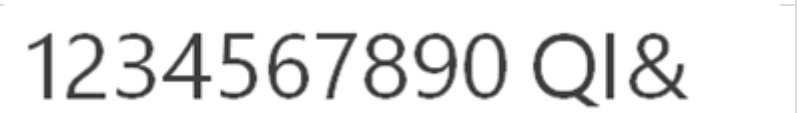
You can use the Windows 7 characters by using the SS01 OpenType stylistic set. To use this set in an app using CSS, set the **-ms-font-feature-settings** property to `'ss01' 1`, as shown in this example:

CSS

```
<p style="font-family: 'Segoe UI'; -ms-font-feature-settings: 'ss01' 1;">
    Windows 7 Segoe UI
</p>
```

If your app uses C#/VB/C++ and XAML, use the **Typography.SetStylisticSet1** method to set the value of the **StylisticSet1 attached property** to true for the desired **dependency object**.

You can also use the SS20 stylistic set, which provides preferred design and spacing. This illustration highlights the differences between the default Windows 8 Segoe UI and the SS01 and SS20 versions.

Windows 8 default	
SS01 (compatible with Windows 7 Segoe UI)	
SS20 (preferred design and spacing)	

If you aren't concerned with backwards compatibility, use the SS20 stylistic set.

For reading: Cambria

Most print media use serif fonts, so users expect a serif font when reading a book or magazine. Cambria is a serif font designed for extended onscreen reading. Use Cambria when displaying large amounts of text, such as book or magazine content.

Note Although Cambria is the default font for unformatted eBook content, reading apps may allow the reader to change the font. We also expect more eBook content to be formatted in a designer-specified, embedded font.

For reading apps, follow these guidelines:

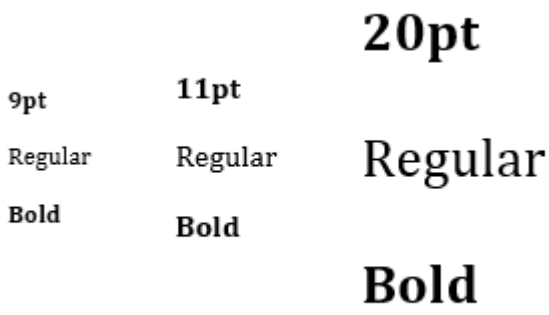
- Keep the text columns narrow enough that the lines of text don't get too long to read comfortably. The typographic grid has specs for this.
- Break up a long text into chapters or sections.

© 2014 Microsoft. All rights reserved. | May 13, 2014

- Support continuing from the point where a reader left off.

Use these font sizes when using the Cambria font: 9pt, 11pt, and 20pt. You can use the normal and bold font-weights. We recommend that you keep the default tracking.

Unlike the Segoe UI font, the WinJS doesn't provide different versions of Cambria for different font weights. Instead, set the font weight by setting the font weight (**font-weight** CSS property or **FontWeight** for apps using XAML) to normal or bold. Here's an example of the recommended Cambria sizes and weights.

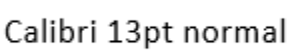


For reading and writing: Calibri

Calibri is a sans-serif font family designed for reading long texts onscreen. Use Calibri for text that the user writes or edits, such as text in an email or chat client. It is the default font in Microsoft Outlook, Microsoft Word, and Microsoft PowerPoint.

When using Calibri, set its font-size to 13 and its weight to normal. We recommend that you keep the default tracking in Calibri.

Here's an example of the Calibri font:



Note: 13pt Calibri has the same x-height as 11pt Segoe UI, so their sizes appear consistent when used together in the same line.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for form layouts



Design forms that provide a great touch experience, optimize the use of screen real estate, and minimize panning/scrolling in your Windows Store app.

Dos and don'ts

- Use a form layout that is appropriate for the content and app.
- Use the same style of label placement for all controls in a form.
- Use in-line place holder text when form content is simple or easily understood.
- Don't use multiple columns when there is extensive vertical panning.
- Don't place labels to the left when there is a lot of variance in the length of the labels.
- Don't automatically launch the touch keyboard without touch input.

Additional usage guidance

When you design a form and the control layout, think about how you want the user to fill out the form and the effects panning/scrolling might have on the experience. Make sure you also consider the impact of the touch keyboard (it can use up to 50% of the screen in landscape orientation) and inline error notifications, if used.

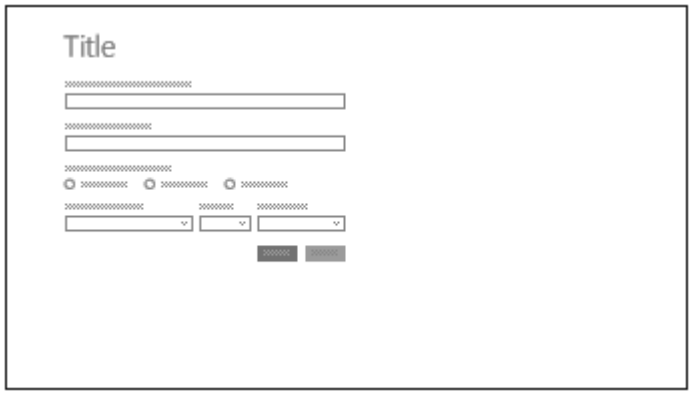
Single-column layouts

Use a single-column layout for your form when:

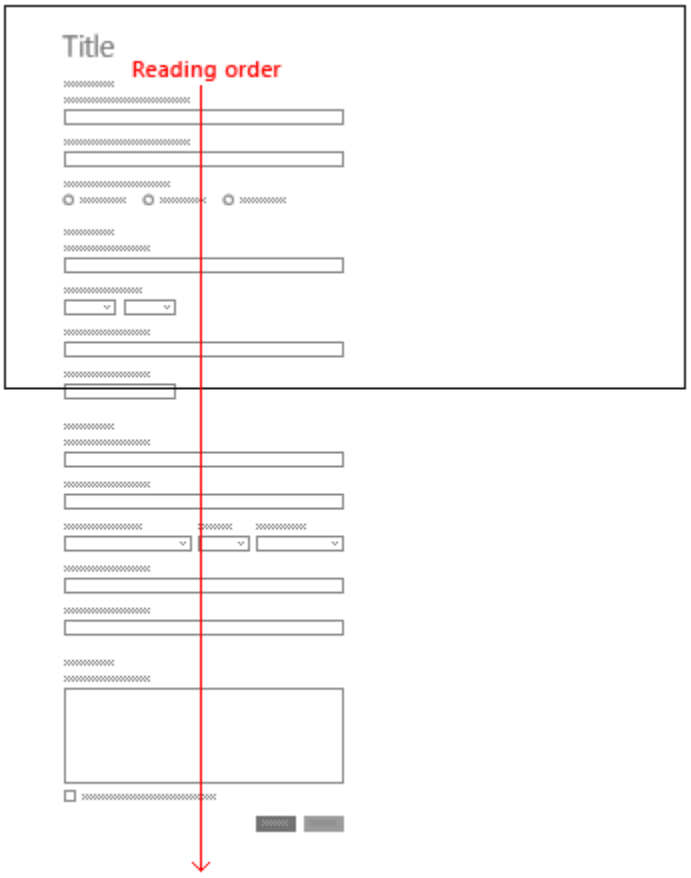
- You want to encourage users to fill out the form in a specific order.
- The form spans multiple pages.
- The app is resized to a tall, narrow layout.
- The app displays additional notes and info, instructions, or branding and advertising.

Here's an example of a short form that uses a single-column layout:

© 2014 Microsoft. All rights reserved. | May 13, 2014

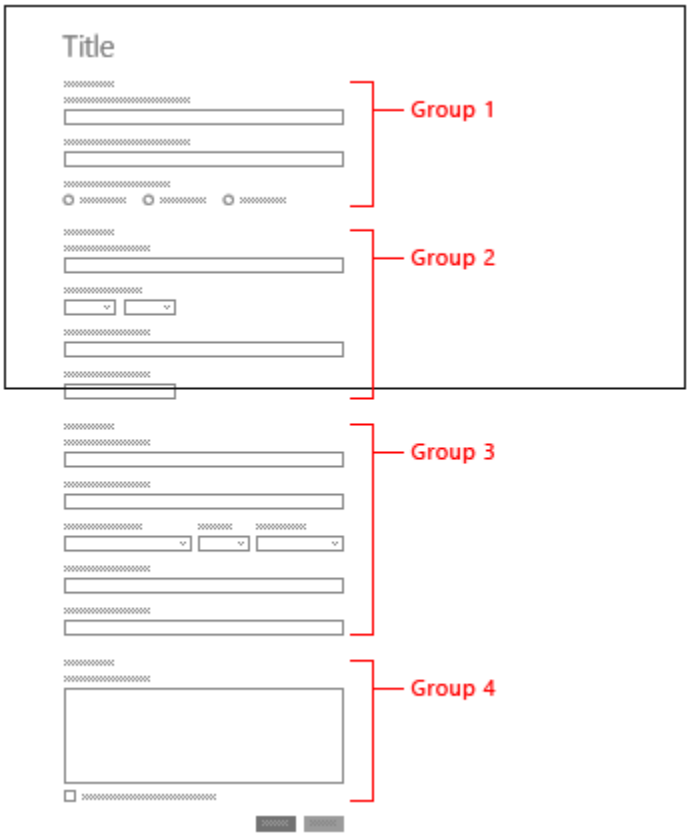


Here's an example of a longer form that uses a single-column layout:



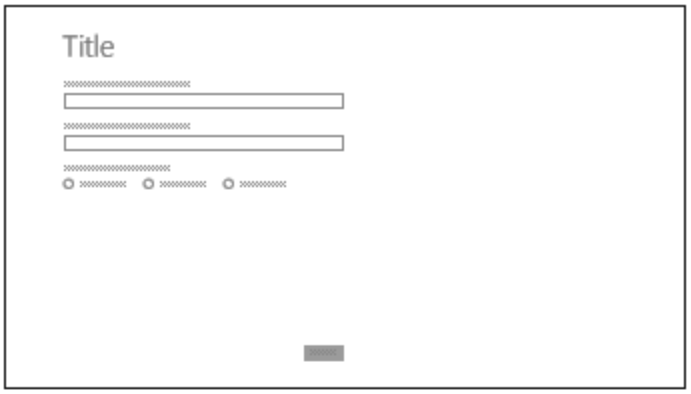
Instead of trying to fit a lot of controls into one very long form, consider breaking up the task into groups or into a sequence of several forms. Here's an example of a longer form that is broken up into groups:

© 2014 Microsoft. All rights reserved. | May 13, 2014

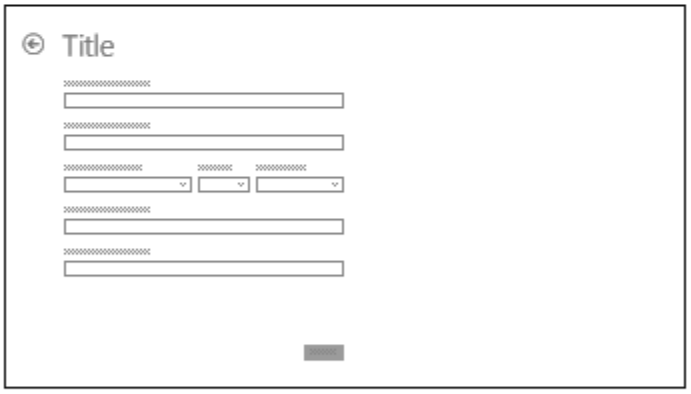


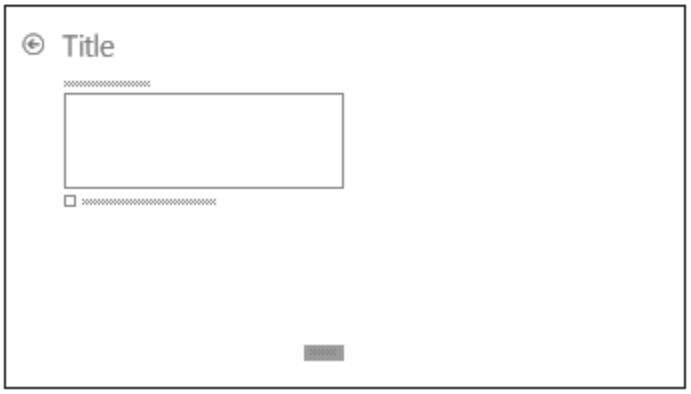
Here's an example of a longer form that's broken up into multiple pages:

© 2014 Microsoft. All rights reserved. | May 13, 2014



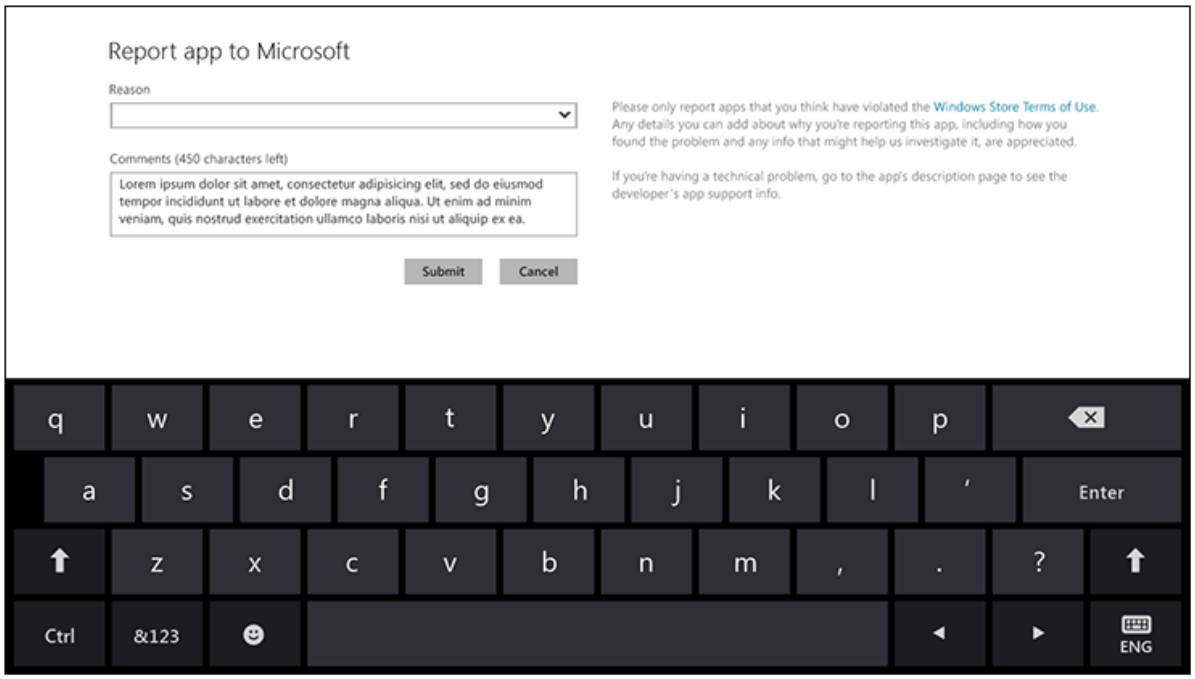






© 2014 Microsoft. All rights reserved. | May 13, 2014

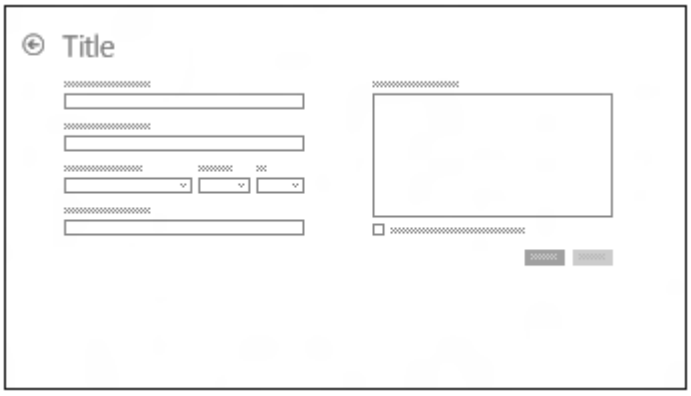
Here's an example of a form that uses a single-column layout because the UI contains additional notes and info:



When to use two-column layouts

Use a two-column layout for short forms with limited vertical panning. The two-column layout makes the best use of screen real estate in landscape orientation. Remember to reserve ample gutter space between the two columns.

Here's an example of a form that uses two columns:



Don't use multiple columns when there is extensive vertical panning. To fill out the form, the user has to reach the bottom of the first column, move to the top of the second column, and then back down again. This experience becomes even more cumbersome if the touch keyboard is displayed.

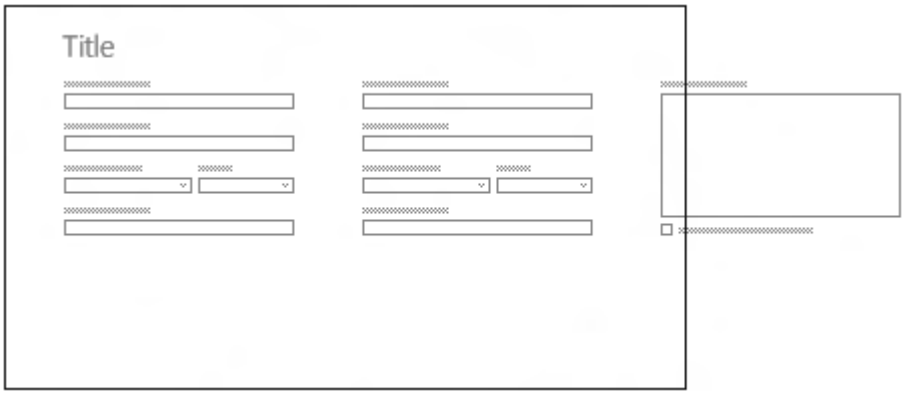
© 2014 Microsoft. All rights reserved. | May 13, 2014

Here's an example of a form that uses two columns improperly:



When to use three or more columns

Use three or more columns to make the best use of visible screen real estate in landscape orientation. This layout works well in fixed or horizontally panning screens, but is awkward in vertically panning screens. Use this layout only when the input order is not important.



Label placement

Most controls need a label.

- Place labels above controls or to the left of a control.
- Use the same style of label placement for all controls in a form.

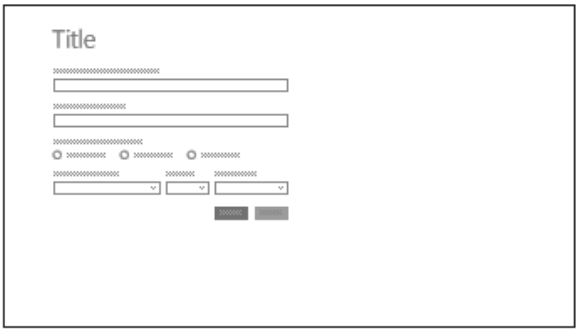
© 2014 Microsoft. All rights reserved. | May 13, 2014

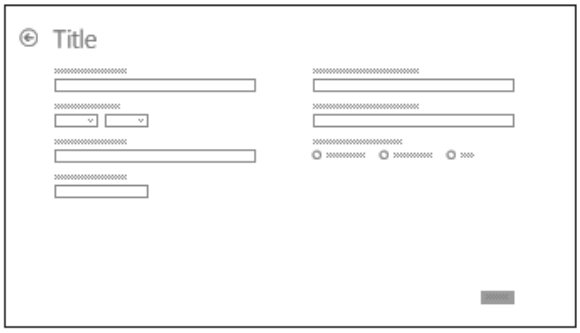
When to use upper label placement

In general, place labels above controls. When using multi-column form layouts, always place labels above controls.

Placing labels above controls helps establish the relationship between a label and its control and helps create a clean layout, as all labels and controls can be left-aligned. Placing labels above controls makes it easier to present long strings and handle localization and accessibility issues.

Here are a two examples of upper label placement.



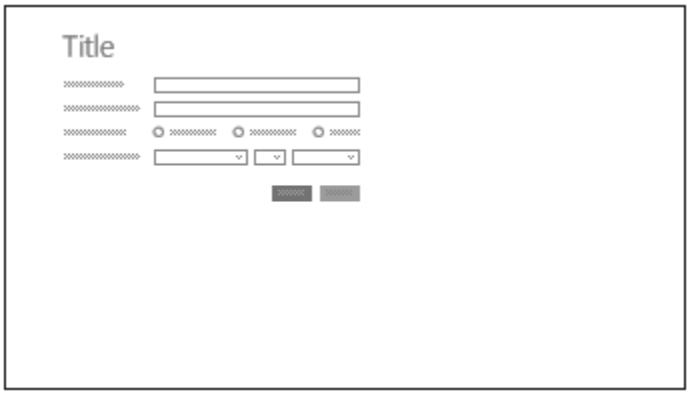


When to use left label placement

In single-column forms where vertical real estate must be conserved, place labels to the left of controls when:

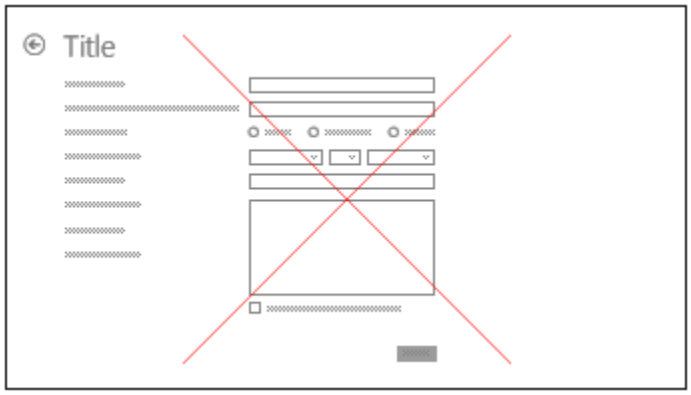
- The label strings are short and approximately the same length.
- There is enough horizontal space for the longest label string (in any locale).

Here's an example of a form that uses left label placement.

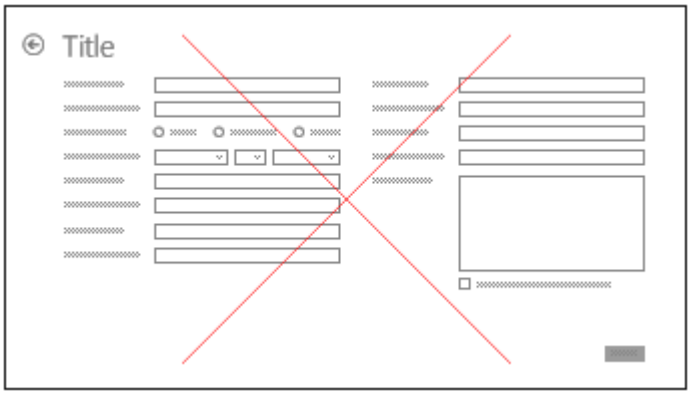


Don't place labels to the left when there is a lot of variance in the length of the labels as some labels will be too far away from their controls.

© 2014 Microsoft. All rights reserved. | May 13, 2014



Don't use left label placement in multi-column form layouts. The labels themselves might look like a separate column.



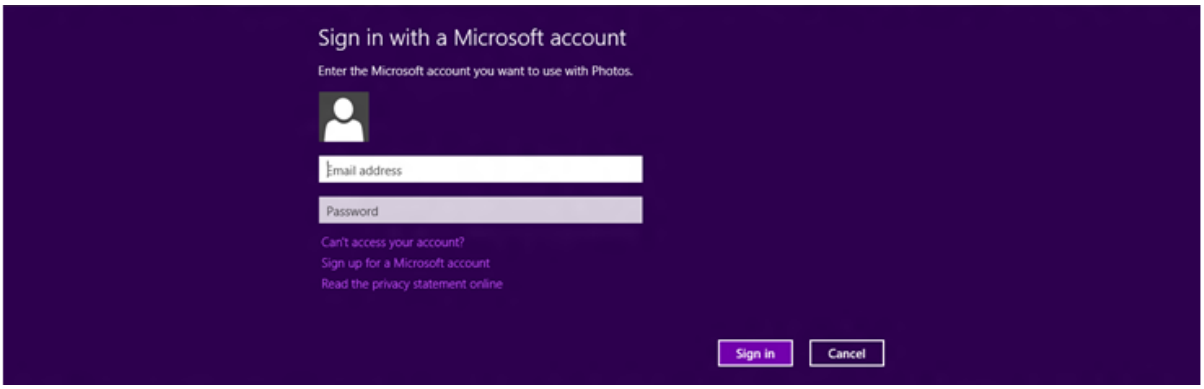
Inline placeholder text

Sometimes you can simplify your layout by using inline placeholder text instead of labels. Use inline placeholder text when:

- The form pattern is commonly understood by a wide audience (for example, a user logon control or a password entry field).
- The label can easily be implied or construed after the data has been entered in the input field (since hint text disappears after data is entered).
- There are a limited number of input fields.

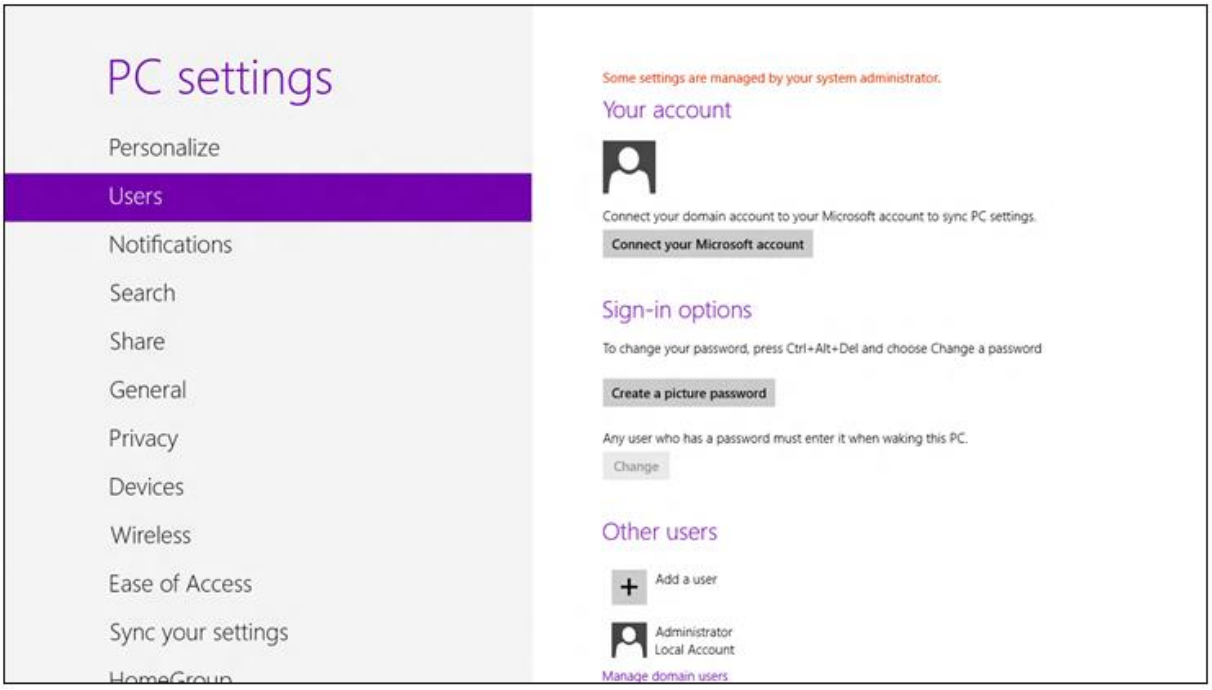
Here's an example of a form that uses inline placeholder text:

© 2014 Microsoft. All rights reserved. | May 13, 2014



Button placement

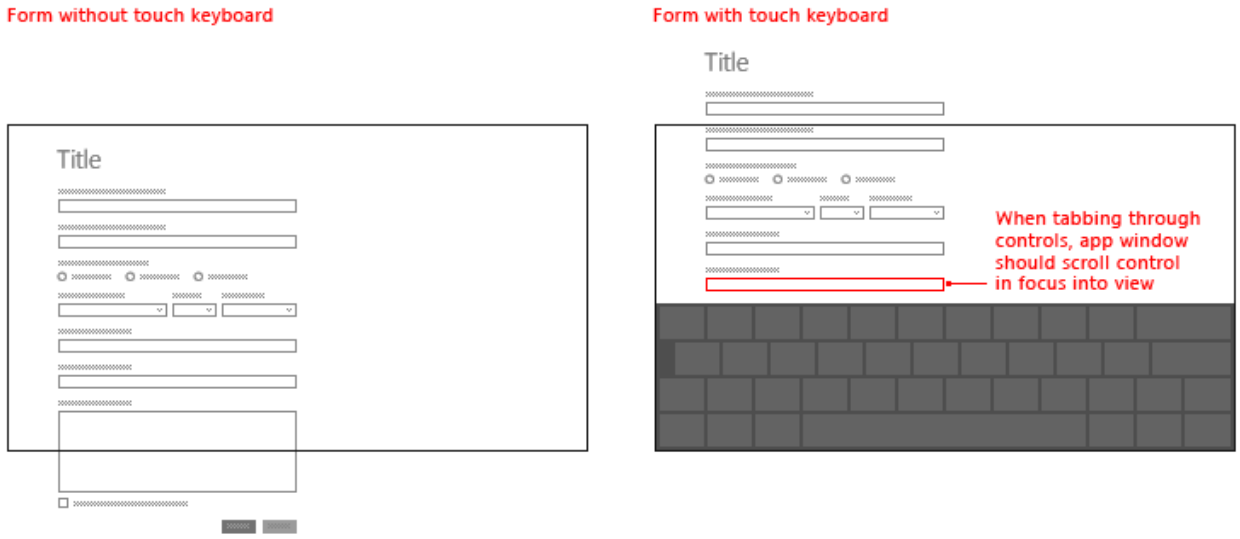
Align form buttons to the right, except when multiple buttons are embedded with other controls. To help a form appear more cohesive, buttons should match the alignment of those other controls. For example, in the **PC Settings** screen, the embedded buttons are left-aligned:



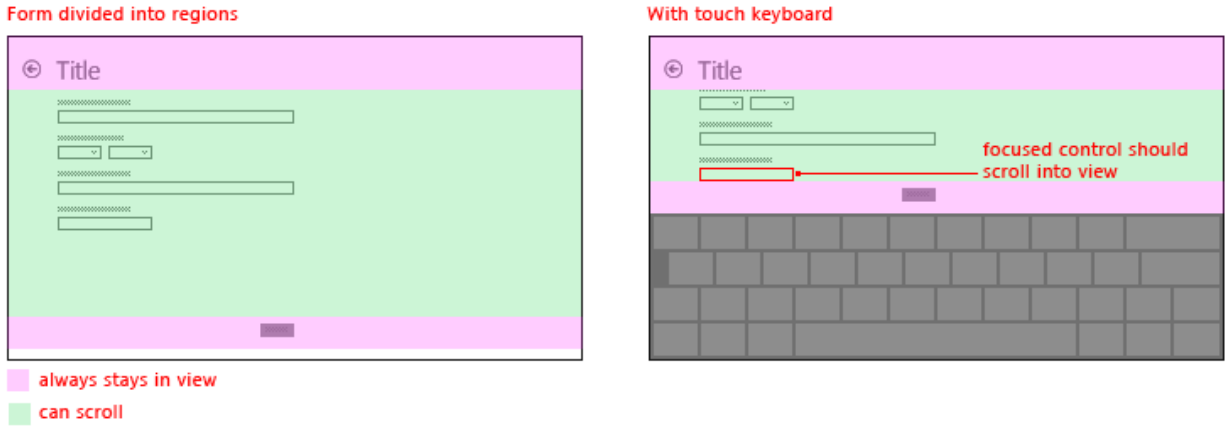
Focus and navigation

The control that a user is interacting with on the form has focus. A user may use the Tab key on the keyboard to navigate through the controls on the form, including controls that are not currently in view. Ensure the panning region that contains the focused control auto-pans so that entire control is in view. There should be a minimum of 30 pixels between the control in focus and the edge of the screen or the top of the touch keyboard (if visible) to account for various edge gestures/UI and the text selection gripper.

© 2014 Microsoft. All rights reserved. | May 13, 2014



In some cases, there are UI elements that should stay on the screen the entire time. Design the UI so that the form controls are contained in a panning region and the important UI elements are static. For example:



Invoking and dismissing the touch keyboard

When no physical keyboard is detected, the touch keyboard is displayed when the user taps an input text field. Don't make your app automatically launch the touch keyboard without this interaction.

The keyboard is dismissed in one of two ways:

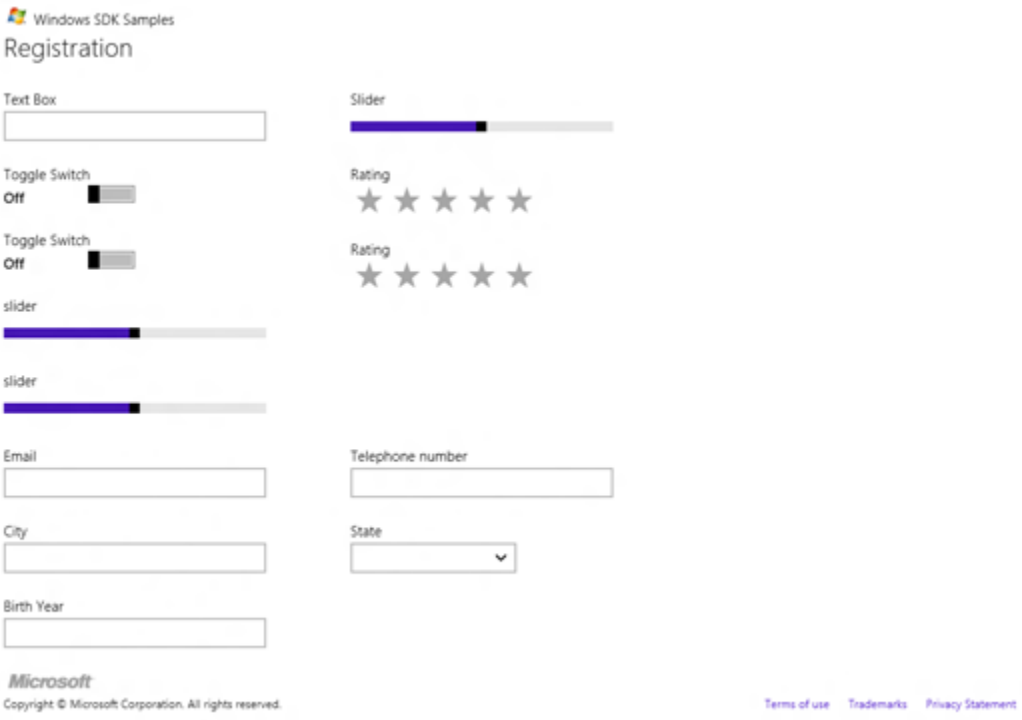
- The user completes and submits the form, leaving the view.
- The user dismisses the keyboard via the dismiss key.

The touch keyboard does not automatically dismiss while the user navigates between controls in a form.

© 2014 Microsoft. All rights reserved. | May 13, 2014

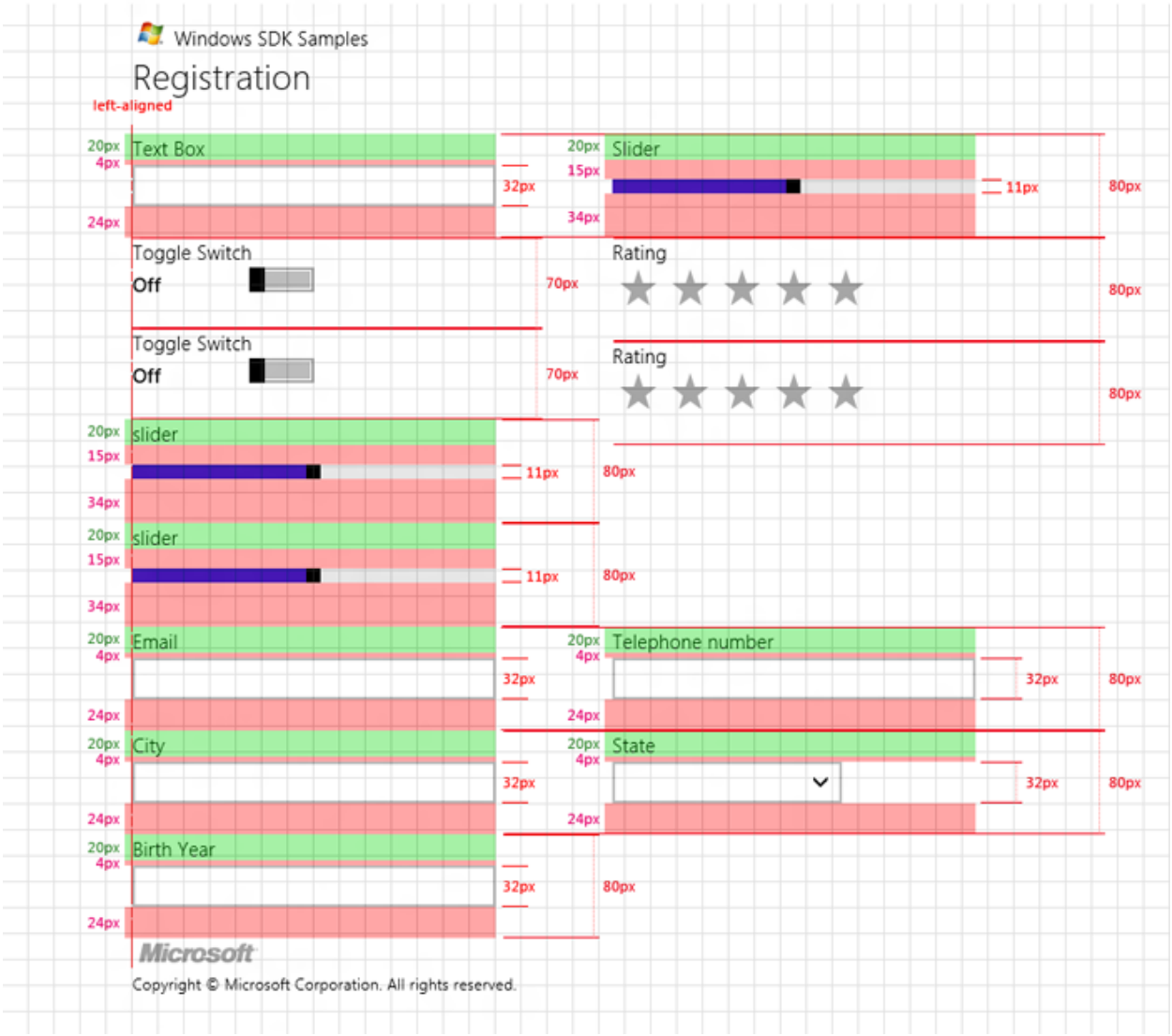
Layout examples

Here's an example of a registration form that follows the guidelines in this document:



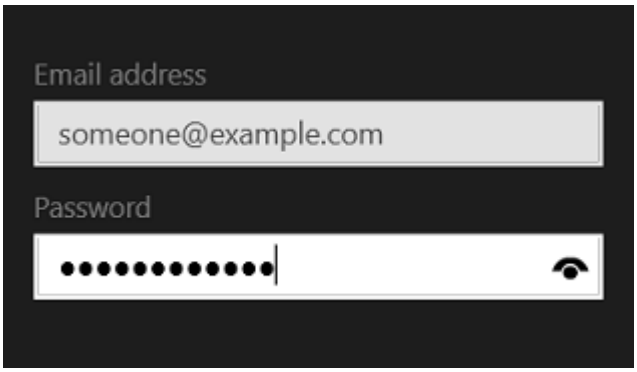
Here are recommended sizes for various elements, with spacing guidelines:

© 2014 Microsoft. All rights reserved. | May 13, 2014

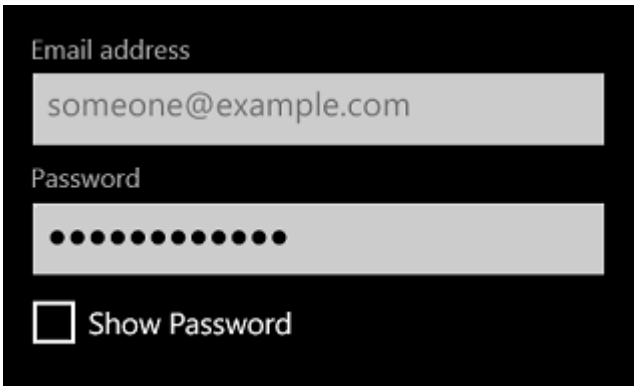

Windows SDK Samples
Registration
left-aligned
Text Box
Slider
Toggle Switch
Off
Rating
Toggle Switch
Off
Rating
slider
slider
Email
Telephone number
City
State
Birth Year
Microsoft
Copyright © Microsoft Corporation. All rights reserved.

Guidelines for password boxes





Windows app: a password box, with password reveal



Windows Phone app: a password box, with password reveal

Description

A password box is a text input box that conceals the characters typed into it for the purpose of privacy.

A password box looks like a text input box, except that it renders bullets in place of the text that has been entered. The bullet character can be configured.

After the user operates the control, or changes are saved, the bullets continue to display. However, the number of bullets that remain won't necessarily equal the number of characters entered.

© 2014 Microsoft. All rights reserved. | May 13, 2014

When a user taps a password box that has already been filled in, the existing bullets become highlighted, allowing typing to replace the existing value.

 On Windows Phone, each character entered by the user is shown only for an instant and is then normalized to a bullet.

Is this the right control?

Use a password box to let the user enter sensitive data that it's very important no one can see over their shoulder. A password box is a more private version of a text input box.

A password box can collect a password or other private data such as a Social Security number. The reduced feedback of what's being typed makes typos harder to spot, though. For more than a word or two of input, consider turning on the password reveal button so that the user can briefly peek at what's been typed.

Dos and don'ts

- When a user is creating an account, and entering a username and password, consider presenting two password boxes, one for the password, and a second one to confirm the password.
- When a user is logging on, present only a single password box.
- When the user is entering a PIN, consider respond the instant the final, correct character has been entered, rather than needing a confirmation button to be pressed. That might mean dismissing the logon dialog, or navigating to the next page in the workflow.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for Segoe UI Symbol icons



This document lists the useful glyphs provided by the Segoe UI Symbol font that you can use as icons.

Dos and don'ts

- Use these glyphs only when you can specify the Segoe UI Symbol font.

Additional usage guidance

Most of the UI controls added to Segoe UI Symbol are mapped to the Private Use Area of Unicode (PUA). The PUA allows font developers to assign private Unicode values to glyphs that don’t map to existing code points. This can be useful when creating a symbol font, but it creates an interoperability problem. If the font is not available, the glyphs won’t show up. Only use these glyphs when you can specify the Segoe UI Symbol font.

If you are working with tiles, you can't use these glyphs because you can't specify the tile font and PUA glyphs are not available via font-fallback.

Many of these glyphs are available with zero width. You can layer zero-width glyphs on top of each other. For example, if you insert a red solid heart with zero width (U+E00B), the cursor does not advance to the end of the heart since it has zero width. Next, if you insert a black outlined heart (U+E006), the outlined heart is layered on top of the solid heart.



Many of the icons also have mirror forms available for use in languages that use bi-directional text.

If you are developing an app in C#/VB/C++ and XAML, you can choose to use the **Symbol enumeration** instead of the Unicode ID to specify glyphs from the Segoe UI Symbol font. If you are developing an app in JavaScript and HTML, you can choose to use the **AppBarIcon enumeration** instead of the Unicode ID to specify glyphs from the Segoe UI Symbol font.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Hearts

U+E006	♡	Outlined heart
U+E0A5	♥	Solid heart
U+E007	💔	Broken heart
U+E00B	♥	Solid heart (zero width)
U+E00C	💔	Broken heart (zero width)

Rating stars

U+E224	☆	Outlined star
U+E0B4	★	Solid star
U+E00A	★	Solid star (zero width)
U+E082	★	Solid star (reduced padding for use in HTML)
U+E0B5	★	Small star

Checkbox components

U+E001	✓	Check mark
U+E002	■	Fill
U+E003	□	Box
U+E004	■	Indeterminate state
U+E005	☑	Reversed
U+E008	✓	Check mark (zero width)
U+E009	■	Fill (zero width)
U+E0A2	☑	Composite

© 2014 Microsoft. All rights reserved. | May 13, 2014

Radio button components

U+E070		Outer circle (zero width)
U+E072		Inner circle (zero width)
U+E080		Fill (zero width)
U+E0A3		Composite

Miscellaneous

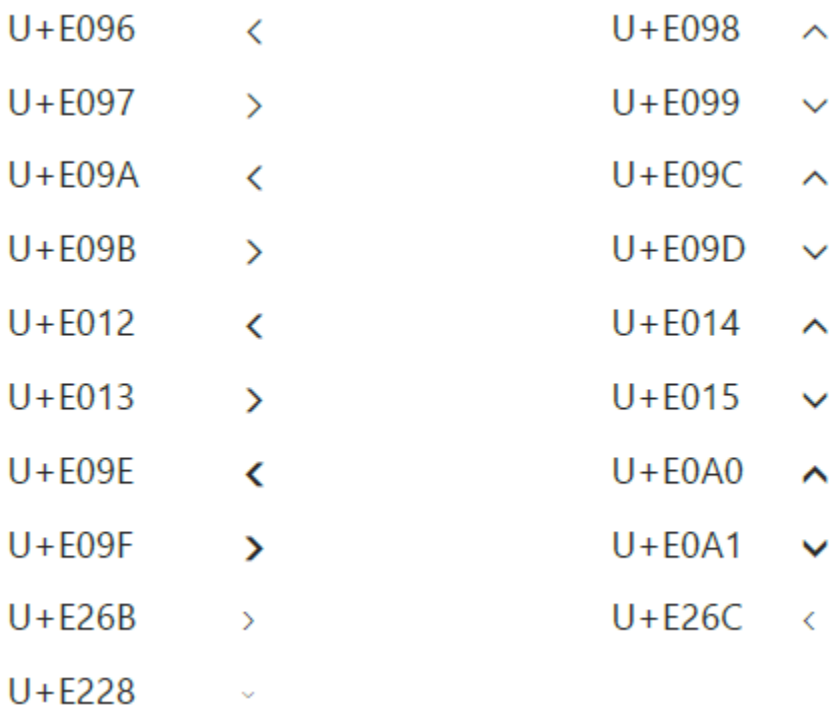
U+E206		Comment
U+E208		Favorite
U+E209		Like
U+E20A		Video
U+E20B		Mail message
U+E248		Reply
U+E249		Favorite
U+E24A		Unfavorite
U+E25A		Mobile contact
U+E25B		Blocked contact
U+E25C		Typing indicator
U+E25D		Video presence chicklet
U+E25E		Presense chicklet

Scroll bar arrows

U+E00E		U+E010	
U+E00F		U+E011	
U+E016		U+E018	
U+E017		U+E019	

© 2014 Microsoft. All rights reserved. | May 13, 2014

Progressive disclosure arrows



Back buttons

Glyphs for back buttons are available in 3 different sizes so that the weight of the outer ring is consistent at 16pt, 20pt, and 42pt. These glyphs are designed to support layering.

16pt code	20pt code	42pt code	Symbol	Description
U+E2A4	U+E0BA	U+E071		Back button
U+E2A5	U+E0BB	U+E0A6		Back button outline
U+E2A6	U+E0C4	U+E0A7		Back button arrow
U+E2A7	U+E0D4	U+E0A8		Back button fill
U+E2A8	U+E0B3	U+E08E		Back button reversed
U+E2A9	U+E0AC	U+E0AA		Mirrored back button
U+E2AA	U+E0AD	U+E0AB		Mirrored back button arrow
U+E2AB	U+E0AF	U+E257		Mirrored back button reversed

Back arrows for HTML

Add additional code to create circles around these glyphs.

© 2014 Microsoft. All rights reserved. | May 13, 2014

U+E0D5		Arrow intended for use in HTML
U+E0AE		Mirrored version of U+E0D5

AppBar glyphs

Use these glyphs in an **AppBar**. They're designed to be displayed at 14pt. Use additional code to create a circle around them.

U+E100		Previous	U+E16E		Filter
U+E101		Next	U+E16F		Copy
U+E102		Play	U+E170		Emoji2
U+E103		Pause	U+E171		High priority
U+E104		Edit (mirrored at U+E1C2)	U+E172		Reply (mirrored at U+E1AF)
U+E105		Save	U+E173		Slideshow
U+E106		Clear	U+E174		Sort
U+E107		Trash	U+E179		All applications (mirrored at U+E1EC)
U+E108		Remove	U+E17A		Disconnect network drive
U+E109		Add	U+E17B		Map network drive
U+E10A		Cancel	U+E17C		Open new windows
U+E10B		Accept	U+E17D		Open with (mirrored at U+E1ED)
U+E10C		More	U+E17E		Circle for Direct UI
U+E10D		Redo	U+E17F		Fill for Direct UI
U+E10E		Undo	U+E181		Status/Presence
U+E10F		Home	U+E182		Options
U+E110		Up	U+E183		OneDrive
U+E111		Forward	U+E184		Calendar
U+E112		Back	U+E185		Font

© 2014 Microsoft. All rights reserved. | May 13, 2014

U+E113		Favorite	U+E186		Font color
U+E114		Camera	U+E187		Status
U+E115		Settings	U+E188		Browse by album (mirrored at U+E1C1)
U+E116		Video	U+E189		Alternate audio track
U+E117		Sync	U+E18A		Placeholder
U+E118		Download	U+E18B		View
U+E119		Mail	U+E18C		Set as lock screen image
U+E11A		Find	U+E18D		Set as tile image
U+E11B		Help (mirrored at U+E1F3)	U+E18E		Closed caption, Japan
U+E11C		Upload	U+E18F		Closed caption, Euro
U+E11D		Emoticon	U+E190		Closed caption
U+E11E		Layout	U+E191		Autoplay stop
U+E11F		Leave multiple party conversation	U+E192		Permissions
U+E120		Forward (mirrored at U+E1A8)	U+E193		Highlight
U+E121		Timer	U+E194		Disable updates
U+E122		Send/Send calendar event (mirrored at U+E1A9)	U+E195		Unfavorite
U+E123		Crop	U+E196		Unpin
U+E124		Rotate camera	U+E197		Open file location
U+E125		People	U+E198		Volume mute
U+E126		Close metadata (mirrored at U+E1BF)	U+E199		Italic
U+E127		Open metadata (mirrored at U+E1C0)	U+E19A		Underline
U+E128		Open in web	U+E19B		Bold

© 2014 Microsoft. All rights reserved. | May 13, 2014

U+E129		View notifications	U+E19C		Move to folder
U+E12A		Preview link	U+E19D		Choose like or dislike
U+E12B		Category not in A-Z	U+E19E		Dislike
U+E12C		Trim video	U+E19F		Like
U+E12D		USB camera	U+E1A0		Align right
U+E12E		Zoom	U+E1A1		Align center
U+E12F		Bookmarks (mirrored at U+E1EE)	U+E1A2		Align left
U+E130		PDF	U+E1A3		Zoom neutral
U+E131		Password protected PDF	U+E1A4		Zoom out
U+E132		Page	U+E1A5		Open file
U+E133		More options (mirrored at U+E1EF)	U+E1A6		Run as other user
U+E134		Post	U+E1A7		Run as admin
U+E135		Mail2	U+E1B0	C	Italic (Italian)
U+E136		Contact Info	U+E1B1	G	Bold (Italian, French)
U+E137		Hang up	U+E1B2	S	Underline (Italian, Spanish, French, Portugese, Brazillian Portugese)
U+E138		View all albums	U+E1B3	F	Bold (Danish, German, Swedish)
U+E139		Map address	U+E1B4	K	Italic (Danish, German, Swedish, Russian, Spanish)
U+E13A		Phone	U+E1B5	U	Underline (Danish, German, Swedish)
U+E13B		Video chat	U+E1B6	I	Italic (French, Portugese, Brazil-lian Portugese)
U+E13C		Switch	U+E1B7	N	Bold (Spanish, Portugese, Brazil-lian Portugese)
U+E13D		Presence	U+E1B8	Ч	Underline (Russian)
U+E13E		Rename	U+E1B9	К	Bold (Russian)

© 2014 Microsoft. All rights reserved. | May 13, 2014

U+E13F		Expand Tile (mirrored at U+E176)	U+E1BA		Font Style (Korean)
U+E140		Reduce Tile (mirrored at U+E177)	U+E1BB		Underline (Korean)
U+E141		Pin	U+E1BC		Italic (Korean)
U+E142		View all info	U+E1BD		Bold (Korean)
U+E143		Go (mirrored at U+E1AA)	U+E1BE		Font Color (Korean)
U+E144		Keyboard	U+E1C3		Street
U+E145		Dock (mirrored at U+E1AB)	U+E1C4		Map
U+E146		Charms bar (mirrored at U+E1AC	U+E1C5		Clear selection (mirrored at U+E1F4)
U+E147		App bar	U+E1C6		Font size decrease
U+E148		Remote desktop home	U+E1C7		Font size increase
U+E149		Refresh	U+E1C8		Font size
U+E14A		Rotate	U+E1C9		Cell phone
U+E14B		Shuffle	U+E1CA		Retweet (full size)
U+E14C		Details (mirrored at U+E175)	U+E1CB		Tag
U+E14D		Shop	U+E1CC		Repeat once
U+E14E		Select all	U+E1CD		Repeat/Loop
U+E14F		Rotate	U+E1CE		Favorite star empty
U+E150		Import (mirrored at U+E1AD)	U+E1CF		Add to favorite
U+E151		Import all files (mirrored at U+E1AE)	U+E1D0		Calculator
U+E153			U+E1D1		Direction
U+E154		Show all files	U+E1D2		Current location/GPS
U+E155		Browse photos	U+E1D3		Library
U+E156		Take webcam picture	U+E1D4		Address book

© 2014 Microsoft. All rights reserved. | May 13, 2014

U+E158		Picture library	U+E1D5		Voicemail/Memo
U+E159		Save local	U+E1D6		Microphone
U+E15A		Caption	U+E1D7		Post update
U+E15B		Stop	U+E1D8		Back to window
U+E15C		Results (find) (mirrored at U+E1F1)	U+E1D9		Full screen
U+E15D		Volume	U+E1DA		Add new folder
U+E15E		Tools	U+E1DB		Calendar reply
U+E15F		Start chat	U+E1DD		Unsync folder
U+E160		Document	U+E1DE		Report hacked
U+E161		Day	U+E1DF		Sync folder
U+E162		Week	U+E1E0		Block contact
U+E163		Month (mirrored at U+E1DC)	U+E1E1		Remote desktop switch apps
U+E164		Match options	U+E1E2		Add friend/view invites
U+E165		Reply all (mirrored at U+E1F2)	U+E1E3		Remote desktop touch pointer
U+E166		Read	U+E1E4		Remote desktop go to start
U+E167		Link	U+E1E5		Remote desktop zero bars
U+E168		Accounts	U+E1E6		Remote desktop one bar
U+E169		Show bcc	U+E1E7		Remote desktop two bars
U+E16A		Bcc	U+E1E8		Remote desktop three bars
U+E16B		Cut	U+E1E9		Remote desktop four bars
U+E16C		Attach	U+E1EA	*K*	Italic (Russian)

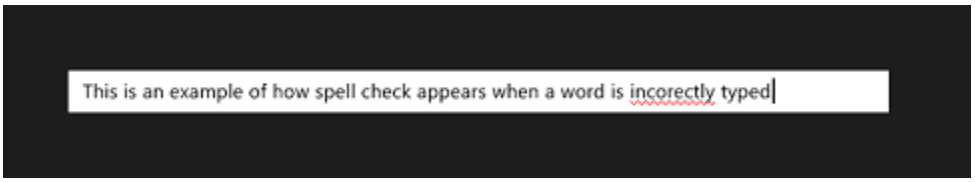
© 2014 Microsoft. All rights reserved. | May 13, 2014

U+E1F5		Record	U+E2AC		Resolution
U+E1F6		Lock	U+E2AD		Length
U+E1F7		Unlock	U+E2AE		Layout
U+E1FD		Arrow down	U+E2AF		People
U+E28F		Save as	U+E2B0		Type
U+E290		Decrease indent (mirrored at U+E297)	U+E2B1		Color
U+E291		Increase indent (mirrored at U+E298)	U+E2B2		Size
U+E292		Bulleted list (mirrored at U+E299)	U+E2B3		Return to window (used in projection manager)
U+E29B		Play on			
U+E294		Scan	U+E2B4		Open content in new window (used in projection manager)
U+E295		Preview			

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for spell checking

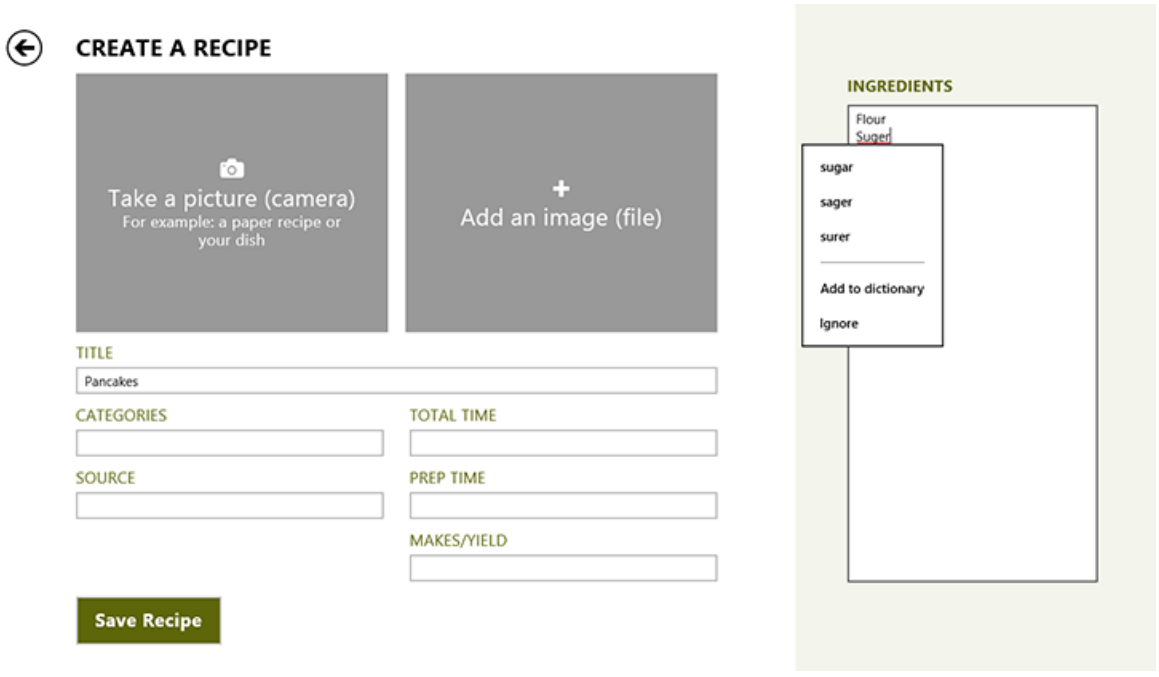




Description

During text entry and editing, spell checking informs the user that a word is misspelled by highlighting it with a red squiggle and providing a way for the user to correct the misspelling.

Example



Dos and don'ts

- Use spell checking to help users as they enter words or sentences into text input controls. Spell checking works with touch, mouse, and keyboard input.
- Don’t use spell checking where a word is not likely to be in the dictionary or where users wouldn’t value spell checking. For example, don’t turn it on for input boxes of passwords,

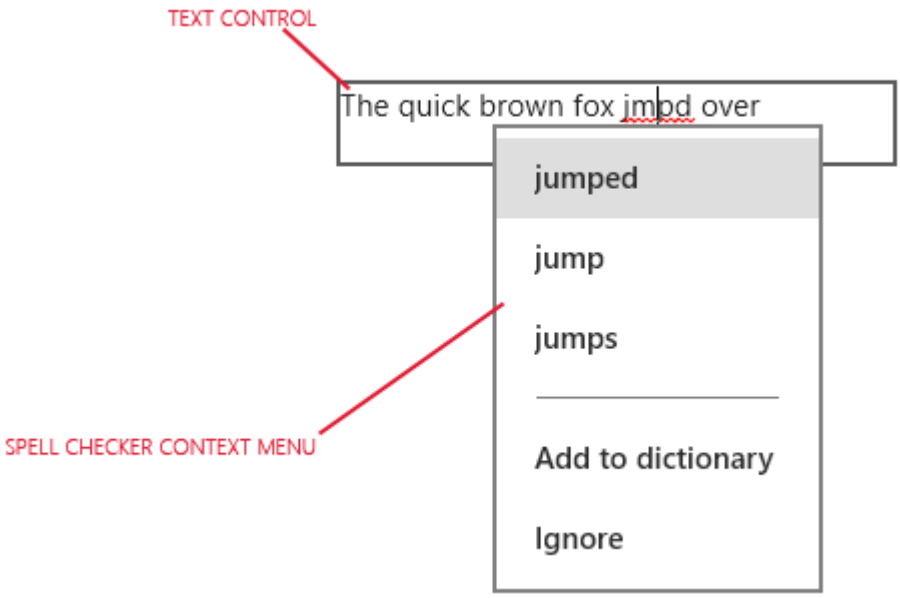
© 2014 Microsoft. All rights reserved. | May 13, 2014

telephone numbers, or names. (Spell checking is disabled by default for these controls.) Telephone numbers, passwords, and names are rarely in the dictionary, so spell checking them doesn't do any good and might be distracting.

- Don’t disable spell checking just because the current spell checking engine doesn't support your app language. When the spell checker doesn't support a language, it doesn't do anything, so there's no harm in leaving the option on. Also, some users might use an Input Method Editor (IME) to enter another language into your app, and that language might be supported. For example, when building a Chinese app, although the spell checking engine doesn’t recognize Chinese now, don’t turn spell checking off. The user may switch to an English IME and type English into the app; if spell checking is enabled, the English will get spell checked.

Additional usage guidance

Windows Store apps provide a built-in spell checker for multiline and single text input boxes, and elements that have their **contentEditable** property set to **true**. Here's an example of the built-in spell checker:



Use spell checking with text input controls for these two purposes:

- **To auto-correct misspellings**

  The spell checking engine automatically corrects misspelled words when it’s confident about the correction. For example, the engine automatically changes "teh" to "the."

- **To show alternate spellings**

© 2014 Microsoft. All rights reserved. | May 13, 2014

When the spell checking engine is not confident about the corrections, it adds a red line under the misspelled word and displays the alternates in a context menu when you tap or right-click the word.

For JavaScript controls, spell checking is turned on by default for multiline text input controls and turned off for single-line controls. You can manually turn it on for single-line controls by setting the control's **spellcheck** property to **true**. You can disable spell checking for a control by setting its **spellcheck** property to **false**.

For XAML TextBox controls, spell checking is turned off by default. You can turn it on by setting the **IsSpellCheckEnabled** property to **true**.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for text input

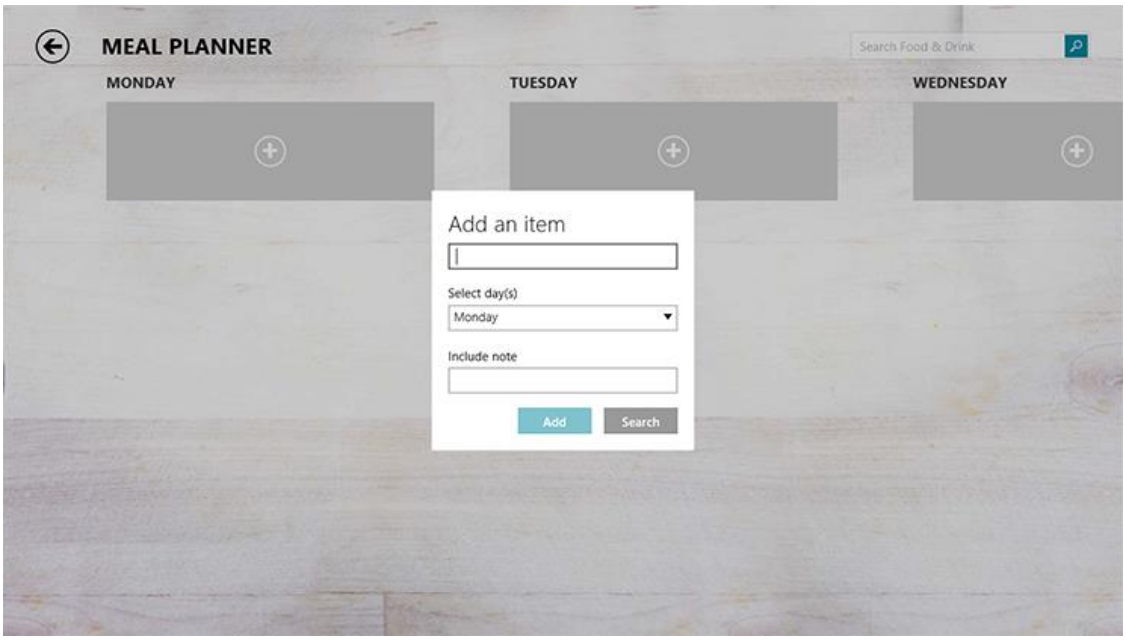


Description

A text input box lets the user enter and edit a text or numeric value via the physical or on-screen keyboard. Using text wrapping, a text input box can be configured to accept a single line or multiple lines of text.

A text input box is composed of a rectangular border with an editable text area within it. On a touch-enabled device, the on-screen keyboard appears when the text input box receives focus.

Example



© 2014 Microsoft. All rights reserved. | May 13, 2014

Is this the right control?

Text input controls let users enter and edit a text or numeric value. Consider these questions when deciding whether to use a text input control.

- **Is it practical to enumerate all the valid values efficiently?** If so, consider using one of the selection controls instead, such as a check box, drop-down list, list box, radio button, slider, toggle switch, date picker, or time picker.
- **Is there a fairly small set of valid values?** If so, consider a drop-down list, or a list box, especially if the values are more than a few characters long.
- **Is the valid data completely unconstrained? Or is the valid data constrained only by format (constrained length or character types)?** If so, use a text input control. You can limit the number of characters that can be entered, and you can choose from a list of input scopes, which limit the input to a particular character set or format—for example, a number, a Uniform Resource Identifier (URI), or a currency.
- **Does the value represent a data type that has a specialized common control?** If so, use the appropriate control instead of a text input control. For example, use a **DatePicker** instead of a text input control to accept a date entry.
- If the data is numeric:
    - **Is the value being entered is approximate, and/or relative to another quantity on the same page?** If so, use a slider.
    - **Would the user benefit from instant feedback on the effect of setting changes?** If so, use a slider, possibly along with an accompanying control.
    - **Is the value entered likely to be adjusted after the result is observed—for example, volume or brightness is being set?** If so, use a slider.

Dos and don'ts

- If it's not obvious what the user should type into a text input box, or if there are any constraints on input, then set a label, or placeholder text. A label is visible whether or not the text input box has a value. Placeholder text is displayed inside the text input box and disappears once a value has been entered.
- Consider giving the control a width that's appropriate for the range of values that can be entered. Bear in mind that word length can vary between languages, so take localization into account.
- A text input box is typically single-line (text wrapping off). When users need to enter or edit a long string, set the text input box to multi-line (text wrapping on).
- Generally, you use a text input box for editable text. But you can make a text input box read-only so that its content can be read, selected, and copied, but not edited.
- If you need to reduce clutter in a view, consider making a set of text input boxes appear only when a controlling checkbox is checked. You can also bind the enabled state of a text input box to a control such as a checkbox.
- Consider how you want a text input box to behave when it contains a value and the user taps it. The default behavior is appropriate for editing the value rather than replacing it—the insertion point is placed between words and nothing is selected. If replacing is the most common use case for a given text input box, you can select all the text in the field whenever the control receives focus; typing replaces the selection.

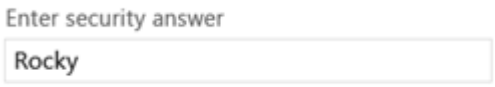
© 2014 Microsoft. All rights reserved. | May 13, 2014

- If you want to limit the input to a particular character set or format—for example, a number, a URI, or a currency—set the input scope appropriately. This determines which on-screen keyboard is used.

Single-line input boxes

- Use several single-line text boxes to capture many small pieces of text information. If the text boxes are related in nature, you should group them together.
- Make the size of your single-line text boxes slightly wider than the longest anticipated input. If doing so makes the control too wide, separate it into two controls; for example, you could split a single address input into "Address line 1" and "Address line 2".
- Set a maximum length. If the backing data source doesn't allow a long input string, limit the input and use a validation popup to let users know when they reach the limit.
- Use single-line text input controls to gather small pieces of text from users.

  The following example shows a single-line text box to capture an answer to a security question. The answer is expected to be short, and so a single-line text box is appropriate here. Because the information collected does not match any of the specialized input types that Windows recognizes, the generic "Text" type is appropriate.



- Use a set of short, fixed-sized, single-line text input controls to enter data with a specific format.



- Use a single-line, unconstrained text input control to enter or edit strings, combined with a command button that helps users select valid values.

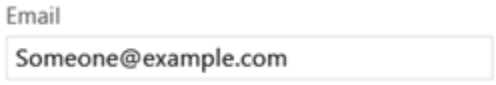


- Use a single-line, number input control to enter or edit numbers.
- Use a single-line password input control to enter passwords and PINs securely.



© 2014 Microsoft. All rights reserved. | May 13, 2014

- Use the single-line email input control to enter an email address.



When you use an email input control, you get the following for free:

  - When users navigate to the text box, the touch keyboard appears with an email-specific key layout.
  - When users enter an invalid email format, a dialog appears to let them know.

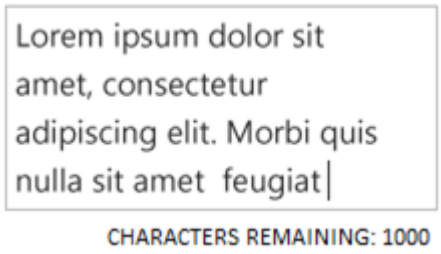


- Use the URL input control for entering web addresses.
- Use the telephone number input control for entering telephone numbers.
- Don't use a text area with a row height of 1 to create a single-line text box. Instead, use the text box.
- Don't use placeholder text to pre-populate the text control. Text boxes clear placeholder text when users use the control. Use the "value" attribute instead.
- Don't use a text box as a search box. It's common practice in web pages to use an input element to create a search box. However, you create a much better and more consistent experience when you use the Search charm instead. The Search charm provides a consistent searching experience that your app can plug into.
- Don't put another control right next to a password input box. The password input box has a password reveal button for users to verify the passwords they have typed. Having another control right next to it might make users accidentally reveal their passwords when they try to interact with the other control. To prevent this from happening, put some spacing between the password in put box and the other control, or put the other control on the next line.

Multi-line text input controls

- When you create a rich text box, provide styling buttons and implement their actions. (Windows Store apps using JavaScript don't automatically provide these controls for you.)
- Use a font that represents the feel of your app.
- Make the height of the text control tall enough to accommodate typical entries.
- When capturing long spans of text where users are expected to keep their word count or character count below some maximum, use a plain text box and provide a live-running counter to show the user how many characters or words they have left before they reach the limit. You will need to create the counter yourself; place it near the text box and update it dynamically as the user enters each character or word.

© 2014 Microsoft. All rights reserved. | May 13, 2014



- Don't let your text input controls grow in height while users type.
- Don't use a multi-line text box when users only need a single line.
- Don't use a rich text control if plain text is adequate.

Additional usage guidance

Use a text input box to let the user enter a text value, or to edit a value already entered. If the input needs to be constrained, that's possible, too. You can limit the number of characters that can be entered, and you can choose from a list of input scopes, which limit the input to a particular character set or format—for example, a number, a URI, or a currency.

Choosing the right multi-line text input control

When users need to enter or edit long strings, use a multi-line text control. There are two types of multi-line text input control: the plain text input control and the rich text control.

- If the primary purpose of the multi-line text box is for creating documents (such as blog entries or the contents of an email message), and those documents require rich text, use a rich text box.
- If you want users to be able to format their text, use a rich text box.
- When capturing text that will only be consumed, and not redisplayed at a later time to users, use a plain text input control. For example, suppose you have a survey; the user completes the survey and the data is sent to some server, but the user doesn't ever see it again. It is generally unnecessary to allow users to style this text.
- For all other scenarios, use a plain text input control.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for typography



Description

Typography is at the heart of the Microsoft design language. Each of the Microsoft design principles reinforces the importance of typography. For the first time, app developers have a set of frameworks that support advanced typographic features.

Dos and don'ts

- Use a typographic grid to lay out your text.
- Use Unicode to improve your text.
- Apply Microsoft OpenType features globally.
- Use sentence case.
- Use an en-dash instead of a hyphen in numerical ranges.
- Don’t apply discretionary ligatures if not using recommended fonts.
- Don't use justified text.

Additional usage guidance

Segoe UI

The signature user interface font of Windows, Segoe UI, was introduced in the Windows Vista/Office 2007 timeframe. It’s a sans serif design drawn in the tradition of signage and way-finding typefaces. Since its introduction, Segoe UI has defined the personality of all Microsoft offerings, influencing the feel of not only operating systems, apps, and devices, but also logos, branding, advertising, and packaging. This typeface represents the typographic voice of Microsoft.

In Windows 8 we made significant upgrades to the Segoe UI family. These are the major changes:

- Segoe UI Light and Segoe UI Semibold were *rehinted* (tuned) to improve the quality of their on-screen rendering.
- We adjusted the design of the default numerals and the "Q" and "I" to match the Windows Phone version of Segoe. You can still use the original Windows Vista forms by using an OpenType "stylistic set" (SS01) .
- In Segoe UI Light, Semilight, and Semibold, we made the numerals space proportionally by default. You can access the more traditional fixed pitch numerals in Windows Store apps using C#/VB/C++ and XAML by setting the **Typography.NumeralAlignment attached property** to "Tabular". If you use JavaScript and HTML, set the **font-feature-settings property** to “TNUM” to apply the tabular (fixed pitch) numeral spacing OpenType feature.

© 2014 Microsoft. All rights reserved. | May 13, 2014

- Because designers often use Segoe UI Light at sizes smaller than 20pt, which is smaller than it was designed for, we added Segoe UI Semilight, a light-weight version of Segoe UI that renders beautifully at smaller sizes, even down to 11pt. We recommend this font when you want to give smaller-sized text a light feel.
- In Windows Vista and Windows 7, Segoe UI supported Latin, Greek, Cyrillic and Arabic (only in regular and bold). For Windows 8, we expanded Segoe UI to include Hebrew, Armenian and Georgian, and we updated the Arabic. Segoe UI now includes character set coverage for all major European and Middle-Eastern languages.
- Although the Windows UI doesn't use italics, we added italic variants for all five Segoe UI weights for apps to use.
- We added advanced OpenType variants to the fonts, including small caps, ligatures and numeral styles across all weights.

In Windows 8.1, we added Segoe UI Black and Segoe UI Black Italic, with pan-European language coverage, further extending the palette of font choices available.

Fonts for other languages

Windows 8 also includes Segoe UI aligned fonts for most other languages. These are available in two weights: regular and bold. Although we don't include a Segoe UI-aligned font for Japanese, our Japanese UI font, Meiryo UI, includes Segoe UI-style Latin characters and numerals as an OpenType stylistic set (ss20).

Here's a list of the recommended fonts for the following common languages and scripts:

Script	Font
Latin, Greek, Cyrillic	Segoe UI
Arabic	Segoe UI
Hebrew	Segoe UI
Armenian & Georgian	Segoe UI
Simplified Chinese	Microsoft YaHei UI
Traditional Chinese	Microsoft JhengHei UI
Korean	Malgun Gothic
Thai	Leelawadee UI
Khmer	Leelawadee UI
Lao	Leelawadee UI
American indigenous languages, including Cherokee	Gadugi
African languages	Ebrima
Indian languages	Nirmala UI
Japanese	Meiryo UI

Please note that Windows 8.1 also offers the following fonts:

- Microsoft YaHei UI Light (available for use at 20pt and above)
- Microsoft JhengHei UI Light (available for use at 20pt and above)

© 2014 Microsoft. All rights reserved. | May 13, 2014

- Leelawadee UI Semilight (available for use at 11pt and above)
- Nirmala UI Semilight (available for use at 11pt and above)

If you are using in-box fonts when localizing your app, see the **Windows.Globalization.Fonts** APIs to identify the recommended UI and document fonts for a given writing system.

Using alternate fonts

Segoe UI's association with Windows 8 and Microsoft has both benefits and drawbacks. On the one hand, through proper use you add instant credibility to your app. But using the font can make it challenging to infuse your own unique personality or brand into your app.

Fortunately, Windows 8 ships with a variety of high quality alternatives to Segoe UI. A serif font like Cambria has good language coverage and provides a traditional feel. Calibri, our default authoring font, will be familiar to your users. In Windows 8, a light weight was added to Calibri, making it a good sans-serif alternative to Segoe UI. Like Segoe UI, Cambria and Calibri include advanced OpenType features. In Windows 8.1, we also added a new typeface family called Sitka, which is available in a range of sizes.

When you use an in-box font, test it on a machine that has a clean install of Windows 8 so that you can verify that the font is available and doesn't require the installation of Microsoft Office or another app. Windows 8 and Windows 8.1 come with the exact same set of fonts regardless of localization or edition.

If the core set of Windows fonts don't provide what you need, you can embed alternate fonts within your app. However, be aware of potential rendering and licensing issues, such as these:

- Most third party fonts don't have the same level of hinting as many of the Windows inbox fonts, so be sure to test them thoroughly at the sizes you care about.
- Although most font licenses allow for some form of document font embedding, most don't allow the font to be redistributed or embedded within an app or game. Before embedding a font, be sure to carefully read the font license and, if in doubt about whether you can embed it, contact the font's creator.

Choosing the right sizes and weights

With the dawn of desktop publishing came easy access to a plethora of type sizes and weights. Early desktop publishers frequently used a wide variety of type sizes and weights in the same document—a hallmark of amateur typography. Too many type sizes and weights make it hard to establish an effective information hierarchy. For that reason, the Windows 8 UI uses only a handful of font sizes and weights:

- Segoe UI Light 42 point for titles
- Segoe UI Light 20 point for headings
- Segoe UI Semilight 11 point for body copy
- Segoe UI Regular 9 point for captions

© 2014 Microsoft. All rights reserved. | May 13, 2014

Note: The Windows type ramp is optimized for light text on dark backgrounds, and deliberately uses lighter-weight fonts. If your app mostly displays dark text on a light background, it might not be suitable to follow this ramp because fonts such as Segoe UI Light or Semilight don't work well at small sizes (dark gray text on a light gray background). Heavier font weights, such as Regular or Semibold, are better for dark text on a light background.

If you decide to diverge from the Windows 8 type ramp, keep the range of type sizes and styles to a minimum.

Laying out your app page

The amount of space around text is just as important as the font size, and for this reason we recommend using a typographic grid to lay out your text. In addition to using the grid, pay particular attention to margins and space around pictures and illustrations. A good layout can be ruined if the text flows around an image poorly or if page elements have inconsistent margins.

8 tips for good typography

There's more to good typography than use of the right font at the right sizes with good spacing. While there are many manuals on typography, here are a few tips to help you take your typography to the next level.

1. Apply OpenType features globally

If you use one of the recommended UI fonts, apply these OpenType features to all text: kerning (kern), discretionary ligatures (dlig) and stylistic set 20 (ss20).

- Kerning improves the inter-letter spacing of your text, and is most evident at larger sizes (for an example, see the "To" in the before and after illustration that follows).
- Ligatures are key elements of high quality typography, but to prevent dialog reflow in legacy content, we encoded the standard ligatures as discretionary ligatures in Segoe UI, so you must turn them on to use them.
- OpenType stylistic set 20 lets you access preferred letter shapes and spacing. (This stylistic set is particularly helpful for numbers.) As an added benefit, when you apply this stylistic set, it ensures consistent text rendering across all the Segoe UI weights and styles.

If you aren't using the recommended UI fonts, it is best to only apply kerning (kern) and standard ligatures (liga). Don't apply discretionary ligatures if you decided not to use the recommended fonts.

CSS

```
<p class='note'>
    6/16/2012<br />
    To: Simon Daniels<br />
    Please find enclosed five flashing baffles.
```

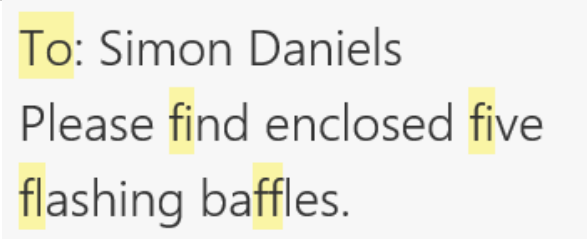
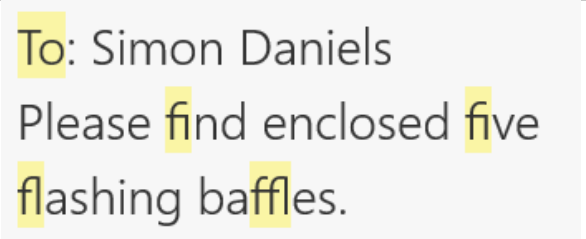
© 2014 Microsoft. All rights reserved. | May 13, 2014

```
</p>
```

CSS

```
.note {
    font-family: 'Segoe UI';
    -ms-font-feature-settings: 'kern' 1, 'dlig' 1, 'ss20' 1, 'lig' 1
}
```

These illustrations show the text before and after applying OpenType features.

Before	After
	

2. Exploit the power of Unicode

Avoiding plain ASCII text through the use of proper characters is the number one way to make your text look more professional:

- Avoid straight quotes and apostrophes
- Use the multiplication symbol instead of the lower case "x"
- Consider using the ratio symbol rather than colon as a time delimiter
- Employ proper dashes instead of hyphens or the minus sign

The most common improper use of the hyphen is in numerical ranges; use an en-dash instead. As an additional typographic refinement, use a hair space before and after the en-dash. Finally, with lining figures or text in all-caps, raise the en-dash slightly so it lies in the center of the optical letter.

If your app displays content that you don't control, such as a news feed, you can autocorrect this content on the fly using basic string search and replace techniques.

Here are a few examples of how you can use Unicode to improve your text.

Without Unicode symbols	With Unicode symbols	Unicode code-points to use
		**U+201C, U+201D**

© 2014 Microsoft. All rights reserved. | May 13, 2014

grocer's special	grocer's special	**U+2019**
12:01	12:01	**U+2236**
2x4		**U+00D7**
1/2 price	½ price	**U+00BD**
And the rest is history...	And the rest is history…	**U+2026**
A-C	A–C	**U+2013**, **U+200A** (hair space)

Unicode also includes thousands of symbols, including 722 emoji symbols that were added to the standard. You can use many of these symbols in place of text and bitmaps in your app.

3. Use sentence case

When communicating with text, the globally applicable approach is to use conventional sentence casing. Styling text to be all-caps or all-lowercase won't translate into international writing systems that use only one case. Using standard sentence case makes your app more accessible to a broader audience.

4. Use OpenType features selectively

If you follow the first three rules, your text will be in pretty good shape. If you want to take things to the next level, though, you can apply OpenType features to specific portions of text. For example, use the small caps OpenType feature to wrap acronyms. This feature applies true small caps (not scaled caps). If your content contains numbers, apply the old style figures feature to the text. This feature uses old style numbers (also known by some as lower case numbers) that might look better with your content.

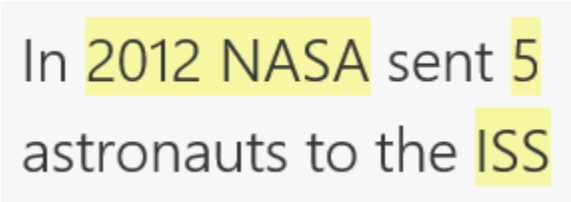
Here's an example:

HTML

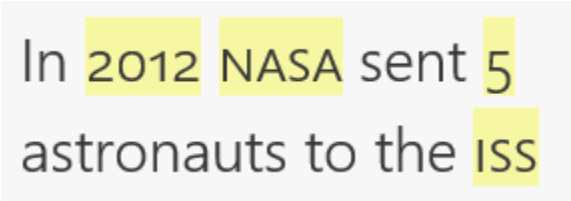
```
<p style="font-family: 'Segoe UI';" >
   In <span style="font-feature-settings: 'onum' 1;">2012</span> <span style="font-
feature-
   settings: 'c2sc' 1;">NASA</span> sent <span style="font-feature-settings: 'onum'
1;">5</span>
   astronauts to the <span style="font-feature-settings: 'c2sc' 1;">ISS</span>
</p>
```

Before applying the old style figures feature:

© 2014 Microsoft. All rights reserved. | May 13, 2014



After:



5. Edit

Editing is an aspect of typography that is often overlooked. Many designers don't consider the text itself to be something they should alter. But if the text isn't well-written and easy to understand, it doesn't matter how good it looks.

6. Use kerning, tracking, hyphenation, and alignment effectively

As any type designer will tell you, the space between letters is just as important as the letter shapes. Kerning (mentioned in the first rule), applies spacing adjustments between specific pairs of letters to improve the letter spacing. We recommend that you apply it to all of your content.

Tracking, on the other hand, adds (or removes) space between all the letters in a string equally. As a font's default letter spacing is often optimized for readability at a particular size, there are times when it might be appropriate to slightly widen the spacing. Tracking values of .01em or .02em may be helpful.

Using fully justified text makes it difficult to have good spacing. When text is forced to fit a container through justification, its spacing always suffers. Justified text usually has "rivers" of white space (large gaps between words) running through the content. Avoid fully-justified text and stick with left alignment whenever possible, and apply hyphenation to keep the right edge of the text from appearing too ragged.

7. Use font-based controls

One of the best kept secrets of Windows typography is that many of the buttons, icons, and controls used in the UI are actually font-based. These are not bitmaps or Scalable Vector Graphics (SVG), but glyphs mapped to the Unicode Private Use Area of the Segoe UI Symbol font. Various showcase apps use the same approach to deliver their icons and controls.

© 2014 Microsoft. All rights reserved. | May 13, 2014

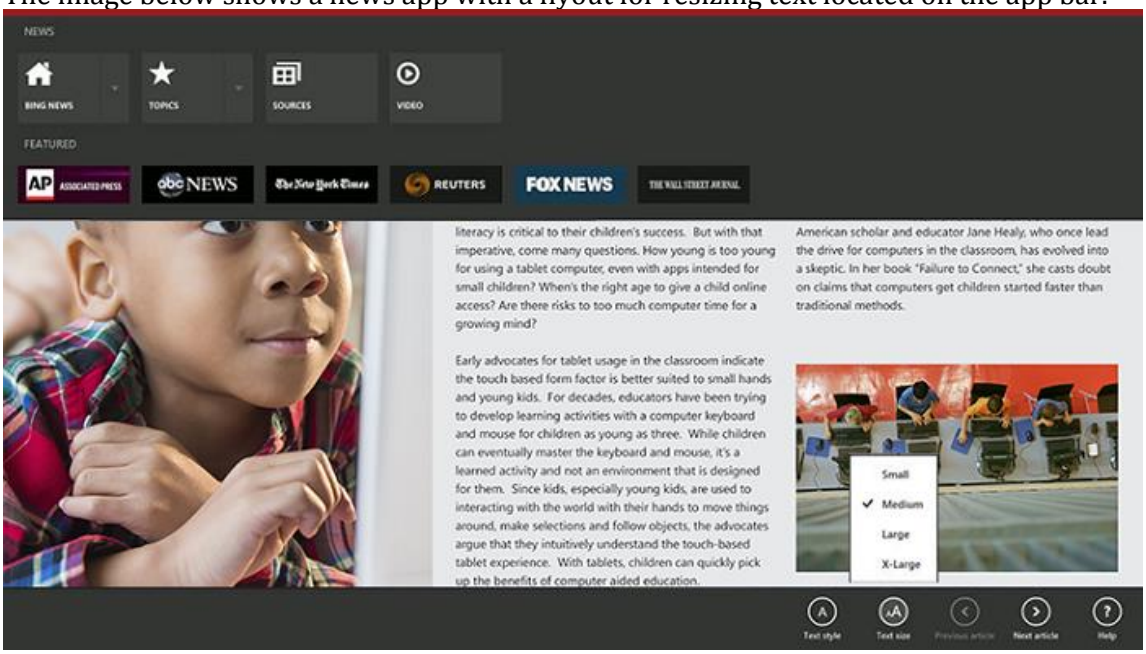
Font-based icons and controls scale just like regular font characters, can be placed in-line with other text elements and, through layering, can appear in color. Windows 8.1 supports full color fonts, further extending the possibilities for font based controls.

8. Let the reader customize the reading experience

Provide settings that allow users to change the body text size, without impacting system-wide settings. We recommend these values: small (11pt), medium (14pt) and large (18pt). In addition, the app could also allow readers to choose between dark text on a light background, which is best for readability, and dark text on a tinted background or light text on a dark background, which might help reduce eye fatigue in certain environments. Giving users a choice of different body font options is welcomed by readers used to this functionality on dedicated e-reading devices. Keep the list of options limited and language-appropriate.

When you create a reading app, consider adding a flyout in the bottom app bar that allows users to switch between text sizes. You can use the Segoe UI Symbol font size icon to create a consistent, predictable experience for users.

The image below shows a news app with a flyout for resizing text located on the app bar.



Suggested reading

Following the basic advice outlined here will give your app a typographic edge. But it barely scratches the surface of what's possible using modern typefaces and the power of OpenType. If your interest has been piqued, we recommend checking out these books:

- *The Elements of Typographic Style* by Robert Bringhurst
- *Inside Paragraphs: Typographic Fundamentals* by Cyrus Highsmith

© 2014 Microsoft. All rights reserved. | May 13, 2014

- *Detail in Typography* by Jost Hochuli
- *Thinking with Type* by Ellen Lupton
- *Stop Stealing Sheep* by Erik Spiekermann and E.M. Ginger

The *IEBlog* article Using the whole font provides a good primer on how to apply Cascading Style Sheets (CSS) OpenType features, and the linked demos by Monotype, FontShop and Font Bureau demonstrate the extent of what's possible using the technology.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for tiles and notifications



Use the guidelines in this section to learn to create a personal, engaging experience for users through tiles and notifications.

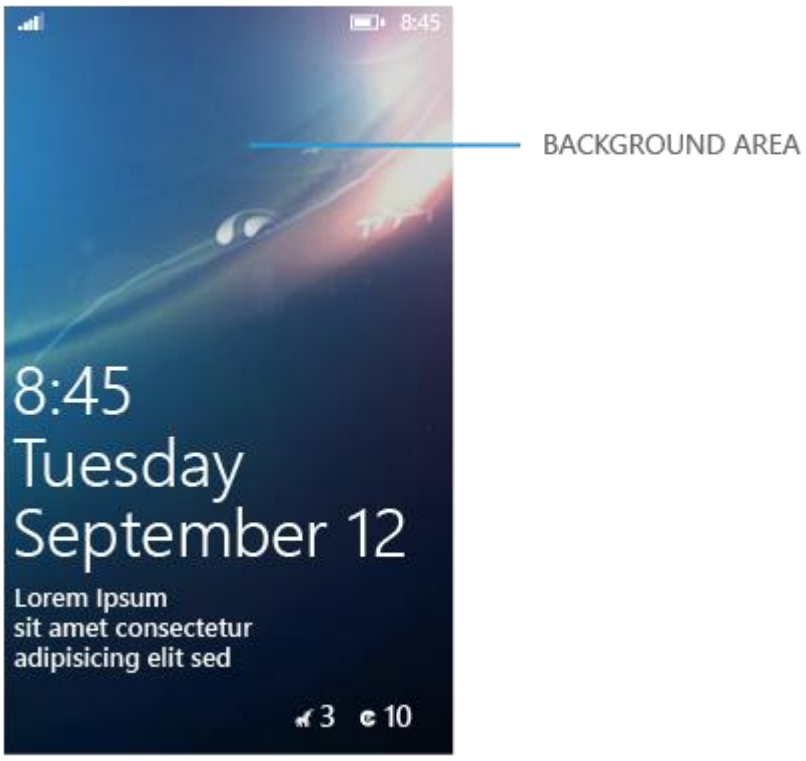
© 2014 Microsoft. All rights reserved. | May 13, 2014

Lock screen design guidelines (Windows Phone Store apps)

Users can customize the lock screen to use your app as the background provider when the Windows Phone is locked. They can also change the basic notifications found in the lock screen (email, text, etc.) to reflect notifications provided by your app. In this topic, we'll cover the best practices for implementing lock screen backgrounds and notifications.

Lock screen background

The lock screen background is a static image that's displayed when a phone is locked, as seen here:



Typically, users select visually pleasing or familiar images as their lock screen background. If it's not eye-catching and friendly, users probably won't select it.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Keep these basics in mind if you choose to implement a background image:

- Reduce the amount of text in your image so it doesn't clash with the on screen text.
- Keep it simple. "Busy" photographs or art make screen text and notifications difficult to read.

The following images show sizing info for using text or logos as part of the lock screen background image:

WVGA (480px by 800px)



WXGA (768px by 1280px)

© 2014 Microsoft. All rights reserved. | May 13, 2014



HD720p (720px by 1280px)

© 2014 Microsoft. All rights reserved. | May 13, 2014



Other design considerations

The following image shows three examples of good lock screen design:

© 2014 Microsoft. All rights reserved. | May 13, 2014



Lock screens: Logo and text, text only, and logo only

- Keep the logo and text small on the screen so it doesn't compete with the date, time or notifications.
- If you're including a logo, consider making it slightly transparent.
- If you choose to include text, it should relate directly to your image.
- The visual focus of the lock screen image should be the image, not the logo or text.

Note: To ensure that a lock screen visual appears in the event of an error, be sure to provide a default background image.

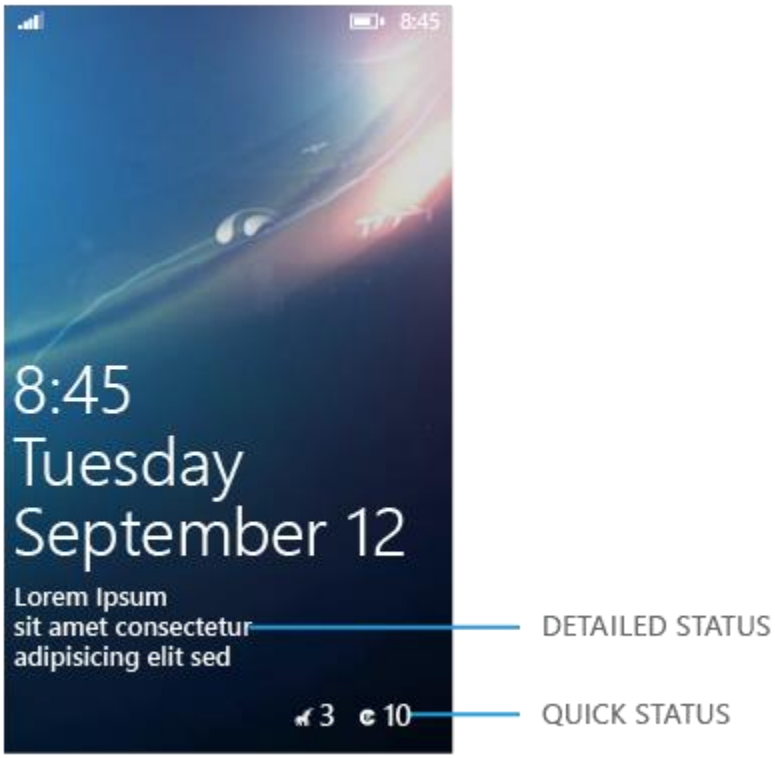
Updating the lock screen background

For more info about how your app can update the lock screen background, see Lock screen background for Windows Phone 8.

Lock screen notification

The following image shows two different lock screen notification types—detailed status and quick status. The text, icon and count for each app are pulled directly from their primary Tiles.

© 2014 Microsoft. All rights reserved. | May 13, 2014



The following image shows two different lock screen notification types—detailed status and quick status. The text, icon and count for each app are pulled directly from their primary Tiles.

- **Detailed status:** Notification text will be displayed in white pixels. The description provided comes from your primary Tile. If your default Tile is a flip Tile, the info from your wide Tile is displayed in the lock screen notifications. However, if there is no wide Tile info, the info provided in your medium Tile displays.
- **Quick status:** The icon you provide will identify your app in the notification area of the lock screen. The icon image should contain only white pixels plus some level of transparency known as watermarking. If implemented, the count from your primary Tile will also appear beside the icon.

Including your app in the lock screen notifications area

For more info about how your app can be included in the lock screen notifications area, see Lock screen notifications for Windows Phone 8.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for periodic notifications



Periodic notifications update tiles and badges at a fixed time interval by polling a cloud service for new content. At the start of each polling interval, Windows sends a request to the service, downloads the content supplied by the service, and displays the fresh content on your app's tile. This topic provides guidelines for using periodic (or polled) notifications in your Windows Store app.

Should my app include periodic notifications?

Use periodic notifications if your app provides content that needs to be updated at regular, fixed intervals. For example, this notification type would be well-suited for:

- A weather app that updates its live tile every 30 minutes to show the current forecast.
- An app that shares a new daily deal with users every morning.

Keep in mind that periodic notifications can't be used with toast notifications. If you want to share pressing, time-sensitive alerts (like breaking news updates) or scheduled reminders with toasts, use push or scheduled notification options. For a comparison of the four available notification options (local, scheduled, push, and periodic).

Dos and don'ts

General

- Expire a periodic notification when it is no longer relevant. For examples, a special online offer that ends at midnight shouldn't be displayed after it has expired.
- Request updates from the server no more than once every 30 minutes. This interval keeps your tile feeling up-to-date without overwhelming your user.
- Feature notification content in a prominent place within your app, like the home or landing page. That way, when a user launches your app in response to a tile notification, he or she can easily find the content that initially attracted them.
- Don't use periodic updates for content that user's expect to receive immediately, like breaking news reports. Use push notifications to deliver more time-sensitive updates.
- Don't use periodic notifications to show ads on your live tile. Tiles should never display ads.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Coding

- Call the **StartPeriodicUpdate** or **StartPeriodicUpdateBatch** method each time your app is launched or brought into focus. This ensures that the tile content is updated each time the user launches or switches to your app.
- Update the tile and badge XML content on your web service to match the polling frequency of your client. For instance, if your app's tile is set to poll at half-hour intervals, update the content on your web service every half an hour.
- If your cloud service become unreachable or the user disconnects from the network for an extended period of time, remove outdated or irrelevant content from your tile. For example, a shopping deal that expires at midnight should set its expiration time to midnight. For more information on setting the expiration time, see the Periodic notification overview.
- Use the *startTime* parameter in **StartPeriodicUpdate** or **StartPeriodicUpdateBatch** to cause the update to occur at a specific time of day. The *startTime* specifies the time of only the first poll, with subsequent polling being timed from that occurrence. Setting the *startTime* to 2:00 PM with a recurrence interval of 24 hours would ensure that updates will always happen at or soon after 2:00 each day.

Note: Tiles can cycle through up to five notifications at a given time. If there are five notifications in the queue, the next new notification replaces the oldest notification in the queue by default. However, if you use **StartPeriodicUpdateBatch**, your service can tag notifications with X-WNS-Tag HTTP response headers and modify the queue's replacement policy. If a new notification arrives with a tag that matches the tag on any of the five existing notifications in the queue, the new notification replaces the older notification with the matching tag (instead of automatically replacing the oldest notification).

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for push notifications



Push notifications are sent from a cloud server to update your app's live tile or send toast notifications. This topic provides general and coding guidelines for using push notifications in your Windows Store app.

Should my app use push notifications?

The push delivery method allows users to receive notifications from your app at any time, even when the app isn't running.

Push notifications are an excellent option if you want your app to share:

- real-time updates (like sports scores during the game)
- content that's generated at unpredictable times (such as breaking news, incoming emails, or social media updates)

Dos and don'ts

- Follow the general tile and toast notification guidelines. Whether a tile or toast notification is generated locally or through the cloud, it should respect the same user guidelines.
- Respect your user's battery life. Users can receive notifications at any time, even when their device is in a low power state. The more notifications that you send, the more resources it will require and the more frequently you will wake up the device. Keep this in mind when you determine the frequency of your notifications.
- Choose the lowest frequency of notifications that still delivers a great user experience. Increasing the frequency of notifications doesn't necessarily increase the value of your app. For example, if your tile content is updated too frequently, some of your updates will never be seen by the user.
- Don't send confidential or sensitive data through push notifications. For example, a bank account number or password should never be sent in a notification.
- Don't use Windows Push Notification Services (WNS) to send critical notifications. Although WNS is reliable, the delivery of notifications isn't guaranteed.
- Don't use push notifications for ads or spam. WNS reserves the right to protect its users and, if an app's use of notifications is deemed inappropriate, the service can block the app from using push notifications. If users report that an app is exhibiting malicious intent, that app may be subjected to Windows Store removal policies.

© 2014 Microsoft. All rights reserved. | May 13, 2014

For developers

- Register your app in the Dashboard to use WNS. Your app server has to use the specific credentials provided by the Dashboard to authenticate and send notifications.
- Request a channel each time the app launches. Channel URLs can expire and are not guaranteed to remain the same each time you request one. If the returned channel URL is different than the URL that you had been using, update your reference in your app server.
- Validate that the channel URL is from WNS. Never attempt to push a notification to a service that isn't WNS. Ensure that your channel URLs use the domain "notify.windows.com" (Windows or Windows Phone) or "s.notify.live.net" (Windows Phone-only).
- Always secure your channel registration callback to your app server. When your app receives its channel URL and sends it to your app server, it should send that information securely. Authenticate and encrypt the mechanism used to receive and send channel URLs.
- Send both the channel URL and device ID to your app server, so that the app server can track to which devices the URLs are assigned. If a URL changes, then the app server can replace the old URL associated with that device ID.
- Reuse your access token. Because your access token can be used to send multiple notifications, your server should cache the access token so that it doesn't have to reauthenticate each time it wants to send a notification. If the token has expired, your app server will receive an error and you should authenticate your app server and retry the notification.
- Don't share your Package Security Identifier (PKSID) and secret key with anyone. Store these credentials on your app server in a secure manner. If you believe that your secret key has been compromised, generate a new key. Routinely generate a new secret key to present villains with a moving target.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for raw notifications



A raw notification is a type of push notification without any associated UI, unlike the other three kinds of push notifications: toasts, tiles, and badges. As with other push notifications, Windows Push Notification Services (WNS) delivers raw notifications from your cloud service to your app. These guidelines describe how to create effective raw push notifications.

Should my app use raw notifications?

You can use raw notifications for a variety of purposes, including to trigger your app to run a background task if the user has given the app permission to do so. Raw notifications are also an excellent way to be informed when there's data available for your app to download from its cloud service. Your app could:

- Use the raw notification as a trigger to initiate a file download. For example, if a user purchases an e-book online, send a raw notification to the user's reading app to trigger the download of the new book.
- Use the raw notification to notify a communication app that there is an incoming instant message or phone call. The communication app can then establish the connection and use a local toast notification to get the user's attention.
- Use the raw notification to coordinate synchronization actions between the client and the cloud service, such as triggering the sync of the most recently read page of a book in a reader app.

By using Windows Push Notification Services (WNS) to communicate with your app, you can avoid the processing overhead of creating persistent socket connections, sending HTTP GET messages, and other service-to-app connections.

Dos and don'ts

- Transmit the smallest amount of information in the raw notification that you can. Note that WNS doesn't allow raw notifications to send more than 5 KB of data.
- Use the notification to indicate that more information is available for the app to download from its cloud service, rather than including that information in the notification.
- Encode any binary data in the notification as base64 before it is included in a raw notification. This guarantees that the content will not be encoded incorrectly in transit and can be retrieved successfully by the client.
- Choose the lowest frequency of notifications that still delivers a great user experience.
- Request a channel each time the app launches. Channel URLs can expire and are not guaranteed to remain the same each time you request one. If the returned channel URL is different than the URL that you had been using, update your reference in your app server.

© 2014 Microsoft. All rights reserved. | May 13, 2014

- Validate that the channel URL is from WNS. Never attempt to push a notification to a service that isn't WNS. Ensure that your channel URLs use the "windows.com" domain.
- Always secure your channel registration callback to your app server. When your app receives its channel URL and sends it to your app server, it should send that information securely. Authenticate and encrypt the mechanism used to receive and send channel URLs.
- Reuse your access token. Because your access token can be used to send multiple notifications, your server should cache the access token so that it doesn't have to reauthenticate each time it wants to send a notification. If the token has expired, your app server will receive an error. Authenticate your app server and retry the notification.
- Don't use raw notifications to stream information to an app by including small amounts of info in serial notifications. Raw notifications should be sent only in response to events that are triggered on the cloud service.
- Don't send raw notifications from a cloud service just to keep a background task running. This is an abuse of the user's battery life. A raw notification must communicate useful information to the app.
- Don't send raw notifications at a rate that causes the associated background task to exceed its resource quota.
- Don't use WNS to send critical notifications. Although WNS is reliable, the delivery of notifications isn't guaranteed.
- Don't use notifications for ads or spam. WNS reserves the right to protect its users and, if an app's use of notifications is deemed inappropriate, the service can block the app from using notifications. If users report that an app is exhibiting malicious intent, that app may be subjected to Windows Store removal policies.
- Don't include zero-sized payload content in a raw notification. Raw notifications without a payload are dropped by WNS and won't be delivered to your app.
- Don't send confidential or sensitive data through raw notifications.
- Don't share your Package Security Identifier (PKSID) and secret key with anyone. Store these credentials on your app server in a secure manner. Routinely generate a new secret key. If your secret key has been compromised, immediately generate a new one.

Additional usage guidance

Before using background tasks triggered by raw notifications in your app, consider the other available methods of communication. A user must explicitly give your app permission to run background tasks and only seven apps can have this permission at once. By using other communication mechanisms in your app, such as standard push notifications or toast updates, your app won't rely on a user allowing it to run background tasks.

Alternatives to background tasks include:

- To get the user's attention, send a toast push notification.
- To update a tile, use a tile push notification.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for scheduled notifications



You can use scheduled notifications to regularly update your app's tile or send toast notifications to a user. Follow these guidelines when adding scheduled tile and toast notifications to your Windows Store app.

Should my app use scheduled notifications?

Use scheduled notifications if you want your app's tile, badge, or toast to be regularly updated with content from within your app. Scheduled notifications are identical to local notifications, except that they specify the time when a tile or badge should be updated or when a toast should appear.

If your notification content is time-sensitive (like breaking news), appears at unpredictable times (like incoming emails), depends on data from outside of your app, or needs to be updated when your app isn't running, you'll need to use another form of notification delivery.

Dos and don'ts

- Follow the recommendations in *Guidelines for tiles* and *Guidelines for toast notifications* when planning the content for your tile or toast and determining how frequently each should be updated.
- Consider using **background tasks** to update the schedule periodically using the **MaintenanceTrigger** class. For example, your app can initially schedule notifications a week in advance and use the **MaintenanceTrigger** class to continue to schedule successive weeks on an ongoing basis, even without the user launching your app during any given week.
- Consider using a **timeZoneChange** system trigger to respond to changes to the system clock, such as Daylight Savings Time. By default, scheduled notifications are triggered in Coordinated Universal Time (UTC) and are not updated automatically in response to system clock changes. For example, a reminder app would need to change the scheduled time of the reminders when the system time changes. To do so, your app can use a background task that responds to the **timeZoneChange** trigger, adjusting its timing appropriately.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for secondary tiles



A secondary tile provides a consistent, efficient way for users to directly access specific areas within a Windows Store app from the Start screen. Although a user chooses whether or not to "pin" a secondary tile to the Start screen, the pinnable areas in an app are determined by the developer. Consider these guidelines when you enable secondary tiles and design the associated UI in your Windows Store app.

Note: Only users can pin a secondary tile to the Start screen; apps can't programmatically pin secondary tiles. Users also control tile removal, and can remove a secondary tile from the Start screen or from within the parent app.

Dos and don'ts

Consider the following recommendations when enabling secondary tiles in your app:

- When the content in focus is pinnable, the app bar should contain a "Pin to Start" button to create a secondary tile for the user.
- Create a flyout that appears when a user clicks the "Pin to Start" button. This flyout should confirm that the user wants to add a secondary tile to the Start screen. For example, here is a confirmation flyout from the ESPN app.



- If the content in focus is already pinned, replace the "Pin to Start" button on the app bar with an "Unpin from Start" button. The "Unpin from Start" button should remove the existing secondary tile (after receiving user confirmation).

© 2014 Microsoft. All rights reserved. | May 13, 2014

- When the content in focus is not pinnable, don't show a "Pin to Start" button (or show a disabled "Pin to Start" button).
- Use the system-provided glyphs for your "Pin to Start" and "Unpin from Start" buttons (see the pin and unpin members in **Windows.UI.Xaml.Controls.Symbol** or **WinJS.UI.AppBarIcon**).
- Use the standard button text: "Pin to Start" and "Unpin from Start". You'll have to override the default text when using the system-provided pin and unpin glyphs.
- Don't use a secondary tile as a virtual command button to interact with the parent app, such as a "skip to next track" tile.

Additional usage guidance

For developers

- When an app launches, it should always enumerate its secondary tiles, in case there were any additions or deletions of which it was unaware. When a secondary tile is deleted through the Start screen app bar, Windows simply removes the tile. The app itself is responsible for releasing any resources that were used by the secondary tile. When secondary tiles are copied through the cloud, current tile or badge notifications on the secondary tile, scheduled notifications, push notification channels, and Uniform Resource Identifiers (URIs) used with periodic notifications are not copied with the secondary tile and must be set up again.
- Use the **RequestCreateForSelectionAsync** class when creating a secondary tile. This allows you to place your confirmation flyout predictably, leading to a better user experience.
- Don't change the name of your tile's default image asset once your app is published. Before the secondary tile receives its first notification or when it has no notification to display, it displays its default image. If Windows cannot find the expected image, it will display a blank tile.
- An app should use meaningful, re-creatable, unique IDs for secondary tiles. This is important for the following reasons:
    - Secondary tiles can be reacquired by users when the app is installed on a second computer. Using predictable secondary tile IDs that are meaningful to an app helps the app understand what to do with these tiles when they are seen in a fresh installation on a new computer.
    - At runtime, the app can query whether a specific tile exists.
    - The secondary tile platform can be asked to return the set of all secondary tiles belonging to a specific app. Using meaningful, unique IDs for these tiles can help the app examine the set of secondary tiles and perform appropriate actions. For instance, for a social media app, IDs could identify individual contacts for whom tiles were created.
- Secondary tiles, like all tiles on the Start screen, are dynamic outlets that can be frequently updated with new content. Secondary tiles can surface notifications and updates by using the same mechanisms as any other tile. To update the tile when the app isn't running, the secondary tile must request and open a channel URI with the Windows Push Notification Services (WNS).

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for tiles and badges



Description

This topic describes best practices and globalization/localization recommendations for use when creating and updating your app's tile, both on the Start screen and on the lock screen, and lists any special tile-related requirements your app needs to meet to be accepted in the Windows Store.

Dos and don'ts

General guidelines

- Use only a small and medium tile if your app will not use tile notifications to send updates to the user. Wide and large tile content should always be fresh and regularly updated. If you aren't using a live tile, do not provide a wide or large logo in the manifest.
- Use only a small or medium tile with a badge if your app supports only scenarios with short summary notifications—that is, notifications that can be expressed through only a **badge image** or a single number. For instance, an SMS app that plans to use notifications to communicate only the number of new texts received would fit this scenario. Do not provide a wide logo in the manifest.
- Use only the small and medium tile if your app sends updates that should not be shown in detail on the Start screen. For instance, a pay stub app could simply say that a new pay stub is available instead of mentioning specifics such as the amount. Do not provide a wide or large logo in the manifest.
- Use the wide or large size tile only if your app has new and interesting content to display to the user and those notifications are updated frequently (at least weekly).
- Use the large tile to show multiple stories from a single notification simultaneously on a single tile, to display longer lists of items, or to show images that the user would appreciate seeing in a larger size on Start.
- Use the default tile image to reflect your app's brand, essentially as a canvas for your app's logo.
- Don't use live tiles if you don't have interesting, new, personalized content for the user. A calculator app, for instance, just isn't going to have that.
- Don't use live tiles if the only interesting thing to communicate is the user's last state. Utility apps, developer tools like Microsoft Visual Studio, and browsers that would only show thumbnails of the user's last session should not use live tiles.
- Don't use live tiles to spam the user or show advertisements. That will get you kicked out of the Windows Store.
- Don't use branding as one of the items in the notification queue or as one of the frames in a peek template. Both of these scenarios involve animated changes to the tile, which catches the user's eye. Calling the user's attention through an animation simply to display your brand instead of interesting new content will only annoy that user.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Default tiles

- If you are including a wide logo or both wide and large logos, consider the design relationship between the medium, wide, and large tile images that you will provide. Always remember that the user has the option to display your tile as any of the supported sizes and can change that size at any time. Here are some general rules:
    - Center the logo horizontally in the tile.
    - Keep the same vertical placement of the logo in both the square and wide tiles, which are of equal height. Keep the same proportional vertical placement of the logo in the large tile.
    - Include the app name at the bottom of the tile if your logo image itself does not include it. Remember, however, that the small tile does not have the option of displaying the app name. The following examples show both situations.

      Tiles using the app name element defined in the manifest:

      Tiles that include the app's name in the logo image:

    - For apps with longer names, and because the name can wrap over two lines, make sure that your logo image and the name do not overlap. For example, a safe approach is to restrict your logo to about 80x80 pixels in the 100% image resource for the medium and wide tile sizes.
    - If you make the space around the logo itself transparent in your image, your app's brand color (declared in the manifest) will show through with a gradient pre-applied to it as part of the Windows 8 look. This tactic would be used with a logo such as the mail app tile shown earlier.
- Don't design the default tile to include an explicit text call to launch the app, such as a tile that says "Click Me!"
- If your logo contains your app's name, don't repeat that name in the name field. Use one or the other, as shown here:

© 2014 Microsoft. All rights reserved. | May 13, 2014





Peek templates

- Use peek templates if your scenario includes image and text content that can each stand alone. For example, you might show a photo of a travel destination in the top of the template and the name of the location in the lower portion.
- A peek template grabs the user's attention when it animates, so be sure that it provides desirable content. Otherwise, you will just annoy your user.
- When you use a peek template, its display can start at either end (frame) of its cycle—text fully lowered or text fully raised—and animate up or down to the other frame. Therefore, make sure that the contents of each of your frames can stand alone.
- Don't use peek templates to display information about things the user already knows about. For example, a paused video notification shown on a tile shouldn't use a peek template.
- Don't use peek templates for notifications that are not conceptually grouped. For example, a peek template shouldn't be used if the photo has nothing to do with the text.
- Don't use peek templates if the most important part of your notification could be off-screen due to the peek animation. For example, for a weather app that displays the temperature and an accompanying image (a smiling sun or a cloud), using a peek template would mean that the temperature (the point of the notification) isn't always visible. A static template that shows the image and temperature at the same time would be more useful to the user.
- Don't use peek templates when the text is needed to provide context for the image, such as in a news story.

Badges

- Support just the medium tile size with a badge if your app supports only scenarios with short summary notifications. For instance, a short message service (SMS) app that plans to show only the number of new texts received. Remember that badges are seen even if the user resizes the tile to the small size.
- Display a number on your badge when the number is small enough to be meaningful in your scenario. If your badge is likely to always display a number of 50 or higher, then consider using a system glyph. Strategies to make the badge number less overwhelming include showing the count since the user last launched the app rather than the absolute count. For instance, showing the number of missed calls since the user last launched the app is more useful than showing the total number of missed calls since the app was installed.
- Use one of the provided system glyphs to indicate a change in cases where a number would be unhelpful or overwhelming. For instance, the number of new unread articles on a high volume RSS feed can be overwhelming. Instead, use the newMessage system glyph.
- Use a glyph if a number is not meaningful. For instance, if the tile shows a "paused" notification for a playlist, it should use the paused glyph because a number doesn't make any sense for this scenario.

© 2014 Microsoft. All rights reserved. | May 13, 2014

- Use the newMessage glyph in cases where a number is ambiguous. For instance, "10" in a social media tile badge could mean 10 new requests, 10 new messages, 10 new notifications, or some combination of them all.
- Use the newMessage glyph in high-volume scenarios, such as mail or some social media, where the tile's badge could be continually displaying the maximum value of "99+". It can be overwhelming for the user to always see the maximum value and it conveys no useful information by remaining constant.
- Don't repeat badge numbers elsewhere in a wide tile's body content, because the two instances could be out of sync at times.
- Don't use a glyph if what the glyph tells the user never changes. Glyphs represent notifications and transient state, not any sort of permanent branding or state.

Tile notifications

- Use what you know about the user to send personalized notifications to them through the tile. Tile notifications should be relevant to the user. The information you have to work with will be largely internal to your particular app and could be limited by a user's privacy choices. For example, a television streaming service can show the user updates about their most-watched show, or a traffic condition app can use the user's current location (if the user allows that to be known) to show the most relevant map.
- Send frequent updates to the tile so the user feels that the app is connected and receiving fresh, live content. The cadence of tile notifications will depend on your specific app scenario. For example, a busy social media app might update every 15 minutes, a weather app every two hours, a news app a few times a day, a daily offers app once a day, and a magazine app monthly. If your app will update less than once a week, consider using a simple medium tile with a badge to avoid the appearance of stale content.
- Provide engaging and informative tile notifications so that users can make an informed decision about whether they need to launch your app. In general, a notification is an invitation to the user to launch the app for more details or to perform an action. For example, a notification might cause the user to want to respond to a social media post, read a full news story, or get the details about a sale.
- Send notifications about content hosted on the home or landing page of your app. That way, when the user launches your app in response to your notification, they can easily find the content that the notification was about.

Additional usage guidance

A tile is an app's representation on the Start screen. A tile allows you to present rich and engaging content to your user on Start when your app is not running. Tapping or clicking the tile launches the app. Tiles come in three square sizes (small, medium, and large) and one wide size. Several template variations are provided for the medium, wide, and large sizes, with text, images, or a combination of text and images. Some templates, called peek templates, consist of two stacked frames that scroll back and forth within the tile space. Peek templates are available for the medium and wide tile sizes.

Tiles can be live (updated through notifications) or you can leave them static. Tiles begin as a default tile, defined in the app's manifest. A static tile will always display the default content, which

© 2014 Microsoft. All rights reserved.

is generally a full-tile logo image. A live tile can update the default tile to show new content, but can return to the default if the update expires or is removed. A tile can also display a status badge, which can be a number or a glyph.

A medium, wide, or large tile can optionally show branding in one lower corner, either using the app's name (on a default or live tile) or a small icon (on live tiles only).

Two very important points to always remember:

- The user can resize the tile to any size that the tile supports. There is no way for you to know which size is currently displayed on a user's Start screen. All tiles must support the small and medium tile sizes, but they can optionally also support the wide and large tile sizes. Note that large tile support requires wide tile support as well, so to support the large tile size, you must support all four tile sizes. Large and wide tiles should be used only when your tile supports live updates.
- If your tile supports live tiles, the user can turn tile notifications off and on at any time. When tile notifications are off, the tile is static.

Tile design philosophy

Your goal is to create an appealing tile for your app. If you use a live tile, your goal is to present engaging new content that the user will find valuable to see in their Start screen and that invites them to launch the app. To that end, avoid the overuse of loud colors. Simple, clean, elegantly designed tiles will be more successful than those that scream for attention like a petulant child.

When designing your app, you might ask yourself "Why should I invest in a live tile?" There are several reasons:

- Tiles are the "front door" to your app. A compelling live tile can draw users into your app when your app is not running. A user increasingly values an app that they use frequently.
- A live tile is a selling point that differentiates your app, both from other apps in the Windows Store (users are likely to prefer the app with the great live tile to a similar app with a static tile) and from apps on operating systems that only allow static tiles and icons on their home screens.
- If users like your live tile, a prominent placement of that tile in Start will drive re-engagement with your app. Serendipitous discovery of cool app content through the tile will make users happy.
- The use of a live tile will make the user more likely to pin your app from the Apps view to the Start screen so that they can see the live updates.
- If users don't like your tile, they might place it at the end of Start or unpin it altogether, turn off updates, or even uninstall your app.

Some characteristics that make a live tile compelling are:

- Fresh, frequently updated content that makes users feel that your app is active even when it's not running.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Example: Showing the latest headlines or a count of new emails.

- Personalized or tailored updates that use what you know about the user, such as interests that you allow the user to specify through app settings.

  Example: Deals of the day tailored to the user's hobbies.

- Content relevant to the user's current context.

  Example: A traffic condition app that uses the user's current location to display a relevant traffic map.

Choosing between different tile sizes

Your app will always have a small and medium tile. You must provide at least a medium tile image asset in your app's manifest. You can also provide an asset for the small tile, but if you don't, a scaled-down version of the medium tile asset will be used.

You must decide whether you want to allow for a wide or a large tile as well.

- To support a wide tile, include a wide (wide310x150) logo image as part of your default tile in your app's manifest. If you don't include that default wide logo image, your tile will only support the small (square70x70) and medium (square150x150) sizes; it cannot be resized to the wide size by the user and it cannot accept wide notifications.
- To support a large (square310x310) tile, include both a wide logo image and a large logo image as part of your default tile in your app's manifest. If you don't include that default large logo image, your tile cannot be resized to the large size by the user and it cannot accept notifications that use the large templates. Because large tile support requires wide tile support, including a default large logo image without including a default wide logo image has the same result as leaving them both out.

To support more tile sizes than your app currently supports, you must release a new version of your app with an updated manifest that includes the additional default logo images.

- Medium tiles show less content than wide and large tiles, so prioritize your content. Don't try to fit everything that you can show in a wide tile into a medium tile. Smaller still, the only live content supported by the small tile is badge notifications.





© 2014 Microsoft. All rights reserved. | May 13, 2014

If you have wide tile content that consists of an image plus text, you can use a square peek template to break the content into two frames. However, do not use a peek template if the image by itself is not sufficient to convey the gist of the story.

Notifications should supply template content for all supported tile sizes except the small tile, because it cannot know the current size of the tile. If a notification is defined using only a wide template and the tile is displayed as medium, or if the notification is defined using only a medium template and the tile is displayed as wide, the notification will not be shown.

Using default tiles

An app's default tile is defined in its manifest. It is static and generally simple in design. For some apps, the default tile is all that you'll ever need. If a user pins a tile from the Apps view to Start after the app is installed, the default tile is shown on the Start screen until that tile receives a notification. If you provided a wide logo image, you can specify whether the tile initially pins to Start as a medium or wide tile. By default, an app's tile is pinned as a wide tile, if the wide tile size is supported by the app through a wide logo image specified in the manifest; otherwise, the tile is pinned in the medium size. Once pinned, the user can resize the tile to any supported size. A live tile can revert to its default if it has no current, unexpired notifications to show.

Using peek templates

Peek templates supply tile content which cycles between two frames of information within the tile space. The upper frame is an image or image collection, and the lower frame is text or text plus an image.

Other design considerations

- When determining how to convey an app's brand information in a tile, choose either the app's name, as shown here:



or logo image, as shown here:

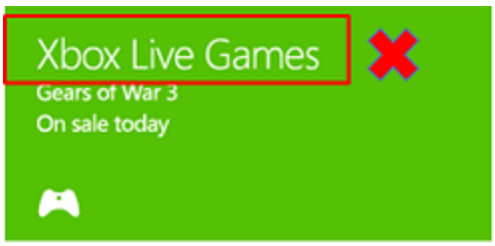
© 2014 Microsoft. All rights reserved. | May 13, 2014

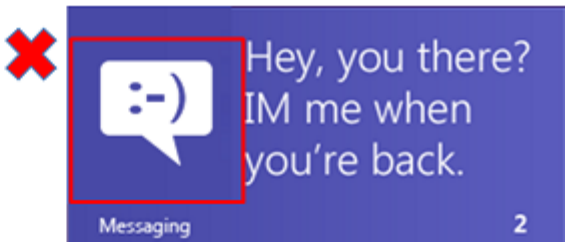


These items are originally defined in the app manifest and the developer can choose which of the two to display in each subsequent notification. However, after you make the choice of name or logo, you should stick with it for the sake of consistency. Note that, due to space constraints, some templates don't let you show the name—your only option is to show or hide the logo.

- Don't use the image or text elements to display app branding information in a tile notification. To reinforce your app's brand and provide consistency to the user, branding should be provided through the template's elements provided for that purpose: the app name (short name) or logo image. A live tile can change its appearance considerably from notification to notification, but the location of the name/logo is consistent. This ensures that users can find their favorite apps through a quick scan, seeing that information in the same place on each tile. If your app doesn't leverage the provided branding elements (name and logo), then it can be harder for users to quickly identify your app's tile.

  The following images show tiles that use the template's text and image elements to inappropriately convey branding. Note that in both cases, the tiles are also using the name or logo as designed, so the additional branding is redundant information.





- If your app's name does not fit in the space provided by the optional "short name", use a shorter version or a meaningful acronym. For example, you could use "Contoso Game" for the very addictive "Contoso Fun Game Version 3". Names that exceed the maximum number of pixels are truncated with an ellipsis. The maximum name length is approximately 40 English characters over two lines, but that varies with the specific letters involved. We encourage shorter app names from a design standpoint. Note that you can also specify a longer name for your app (the "display name") in your manifest. This name is used in the Apps view and in the tooltip, though not on the tile.

© 2014 Microsoft. All rights reserved. | May 13, 2014

- Don't use tiles for advertisements.
- Avoid the overuse of loud colors in tiles. Simple, clean, elegantly designed tiles will be more successful than those that scream for attention like a petulant child.
- Don't use images with text on them; use a template with text fields for any text content. Text in an image will not look as sharp as rendered tile text. If an image asset isn't provided that is appropriate to the current display, the image might be scaled, which can further degrade its legibility.
- Don't rely on tiles to send urgent, real-time information to the user. For instance, a tile is not the right surface for a communication app to inform the user of an incoming call. Toast notifications are a better medium for messages of a real-time nature.
- Avoid image content that looks like a hyperlink, button, or other control. Tiles do not support those elements and the entire tile is a single click target.
- Don't use relative time stamps or dates (for instance, "two hours ago") on tile notifications because those are static while time moves on, making the message inaccurate. Use an absolute date and time such as "11:00 A.M.".
- Because an app's tile can only launch the app into its home screen, tile updates should concern elements of the app that are easily accessible from that home screen. For example, a news app's tile should only show articles that the user can easily find on the app's home page once they click on the tile.

Using tile notifications

Choosing the right notification method to update your tile

There are several mechanisms which can be used to update a live tile:

- Local API calls
- One-time scheduled notifications, using local content
- Push notifications, sent from a cloud server
- Periodic notifications, which pull information from a cloud server at a fixed time interval

The choice of which mechanism to use largely depends on the content you want to show and how frequently that content should be updated. The majority of apps will probably use a local API call to update the tile when the app is launched or the state changes within the app. This makes sure that the tile is up-to-date when it launches and exits. The choice of using local, push, scheduled, or polling notifications, alone or in some combination, completely depends upon the app. For example, a game can use local API calls to update the tile when a new high score is reached by the player. At the same time, that same game app could use push notifications to send that same user new high scores achieved by their friends.

How often should your tile update?

If you choose to use a live tile, consider how often the tile should be updated.

- For personalized content, such as message counts or whose turn it is in a game, we recommend that you update the tile as the information becomes available, particularly if the user would notice that the tile content was lagging, incorrect, or missing.

© 2014 Microsoft. All rights reserved. | May 13, 2014

- For nonpersonalized content, such as weather updates, we recommend that the tile be updated no more than once every 30 minutes. This allows your tile to feel up-to-date without overwhelming your user.

Expiration of tile and badge notifications

Your tile's content should not persist longer than it is relevant. Set an expiration time on all tile and badge notifications that makes sense for your app. By default, local and scheduled tiles and badges never expire and tile and badge content sent through a push or periodic notification expires three days after it is sent. When a notification expires, the content is removed from the tile or queue and is no longer shown to the user.

You can set a specific date and time for the notification content to expire. An explicit expiration time is particularly useful for content with a defined lifespan. Also, if your cloud service stops sending notifications, if your app is not run for a long time, or if the user disconnects from the network for an extended period of time, explicit expiration assures the removal of stale content despite the system's connectivity state.

For example, during a stock market's active trading day, you can set the expiration for a stock price update to twice that of your sending interval (such as one hour after you send the notification if you are sending notifications every half-hour). As another example, a news app might determine that one day is an appropriate expiration time for a daily news tile update.

How you set the expiration depends on the delivery method. For push and periodic notifications, it is set in the HTTP headers used to communicate with the cloud service that delivers the notifications. For local and scheduled notifications, the expiration can be set as part of the API call.

Tiles and badges on the lock screen

To determine whether your app is a good candidate for a lock screen presence, you must understand the operation and limitations of the lock screen. A summary of the lock screen is given here.

- A maximum of seven app badges can appear on the lock screen. The badge information reflects the badge information on the app's Start screen tile. The badge (either a glyph or a number) is accompanied by a monochrome icon (logo image) to identify the app the badge is associated with.
- Only one of those seven apps can occupy a detailed status slot, which allows it to display the text content of the app's most recent tile update.
- The lock screen's detailed status tile does not show images included in that tile update.
- The user is in charge of which apps can display information on the lock screen, and which one of those apps can display detailed status.
- All apps that have a lock screen presence can also run background tasks. All apps that can run background tasks have a lock screen presence. An app cannot use background tasks without also claiming a slot on the lock screen.
- The notification queue is not supported by the lock screen's detailed status tile. Only the latest update is shown.

© 2014 Microsoft. All rights reserved. | May 13, 2014

- An app with a lock screen presence, as long as it has set the **Toast Capable** option to "Yes" in its manifest, displays its received toast notifications on the lock screen when the lock screen is showing. Toast shown on the lock screen is identical to toast shown elsewhere.
- Tile updates, badge updates, and toast notifications are not specifically designed for or sent to the lock screen. You, as the sender, don't know if the device is currently locked. For an app with a lock screen presence, any notification is reflected both on the Start screen and on the lock screen.

Characteristics of a good lock screen presence

The only way that your app can have a lock screen presence is if the user gives their explicit permission. They can do this either in response to a request from your app (and you can ask only once), or manually through the **Settings**. By giving that permission, the user declares that the information coming from your app is important to them, which your app must then live up to. Therefore, your first consideration should be whether your app is a good candidate to have a lock screen presence at all.

A good candidate for a lock screen presence will have these attributes:

- The information is quickly digestible
- The information is always up-to-date
- The information is understandable without additional context
- The information should be personal and useful to the user
- Showing nothing is better than showing status that never changes
- Only toast notifications should play a sound on arrival

The information is quickly digestible

If the lock screen is displayed, the user isn't currently interacting with the device. Therefore, any update information that your app displays on the lock screen should be something that the user can take in and understand at a glance. As an analogy, think of an incoming call on a cell phone. You glance at the phone to see who is calling and either answer or let it go to voice mail. Information displayed on the lock screen should be as easy to take in and deal with as the cell phone display. All of the other characteristics support this one.

The information is always up-to-date

Good badge updates, tile updates, and toast notifications, whether they're shown on the Start screen or the lock screen, are all potentially actionable. Based on the information those notifications provide, the user can decide whether they want to launch the app in response, such as to read a new email or comment on a social media post. From the lock screen, that also means unlocking the device. Therefore, the information needs to be up-to-date so that the user is making an informed decision. If users begin to see that your app's information on the lock screen is not up-to-date, then you've lost their trust and they'll probably find a more reliably informative app to occupy that lock screen slot.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Good examples: up-to-date information

- A messaging app sends a notification when a new message arrives. If that notification is ignored, the app updates its badge with a count of missed messages. If the user is present, they can turn on the screen to assess the importance of the message, and choose to respond promptly or let it wait. If the user isn't present, they will see an accurate count of missed messages when they return.
- A mail app uses its badge to display a count of its unread mail. It updates the badge immediately when a new mail arrives. A user can quickly turn on their screen to check how many unread emails they have, and they can be assured that the count is accurate. They have the information to decide if they want to unlock their device and read mail.

Bad examples: out-of-date information

- A messaging app updates its badge with its count of missed messages only once every half hour. The user can't rely on the badge count in deciding whether they want to unlock the device.
- A weather app that uses the detailed status slot continues to show a severe weather alert after the alert has expired. This not only gives the user incorrect information, but is particularly egregious if the text specifies when the alert ends, making it obvious to the user that this is old information. The user loses confidence that the app is capable of keeping them properly informed. The app should have cleared this information when it expired.
- A calendar app continues to display an appointment that has passed. Again, the app should have cleared this information when it expired.

The information is understandable without additional context

This contextual information is not present on the lock screen:

- The tile that goes with the badge, when the app is not allowed to display detailed status. Even when detailed status is shown, the badge is physically separate from the tile. The logo image next to a badge is the only identification of the app it represents.
- Images in tile updates. Only the text portion of the update is shown in the detailed status slot.
- The notification queue. Only the most recent update is shown in the detailed status slot.

Therefore, your updates must be understandable to the user without the additional context available to you on the Start screen. Again, keep in mind that notifications cannot be specifically targeted at the lock screen. Therefore, all of your app's update communications must fall under the "understandable without additional context" rule.

Note: Unlike the detailed tile, toast includes both image (if present) and text—toast displayed on the lock screen is identical to toast displayed elsewhere, so it does not lose context.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Good examples: understandable without additional context

- A mail app uses its badge to display the count of its unread mail. While its Start screen tile might display more information, such as text snippets from the most recent mails or pictures of the senders, what the badge is communicating is understandable without that extra information.
- A social networking app uses the detailed status slot to inform the user of their friends' recent activity. When a friend sends them a message, that friend's name is included in the notification text (for instance "Kyle sent you a new message!"). On the Start screen, the user can see a rich experience with their friend's picture in the notification, while on the lock screen, even though there is no image, the text still makes it clear who sent the message.

Bad examples: not understandable without additional context

- A messaging app updates its tile with the latest received message, and shows only the sender's picture and the message text. In the Start screen, it is clear to the user who the message is from. In the lock screen, without the sender's picture, the user has no idea who sent the message.
- A social networking app updates its tile with a collage of photos, with no text. In the Start screen this is a pleasant, lively tile. On the lock screen, because there is no text in the tile update, nothing is displayed at all.

The information should be personal and useful to the user

Two of the main purposes of the lock screen are to provide a personalized surface for the user and to display app updates. Consider both of these purposes when you judge whether your app is a good candidate for a lock screen presence.

Apps with a lock screen presence are very special—only seven can ever be on the lock screen at a time. By giving an app one of those precious lock screen slots, the user is stating that information coming from that app is important enough to be seen even when the user isn't actively using their device. Therefore, the app should provide information that is both personal and useful to the user.

Note: By definition, the lock screen is displayed when the device is locked. No login or other security hurdle is required to see the contents of the lock screen. Therefore, while the information displayed there is ideally personalized, keep in mind that anyone can see it.

Good examples: information personalized to the user

- A mail app displays the number of unread emails in the user's account.
- A messaging app displays the number of missed messages sent to the user.
- A news app displays the number of new articles in categories that a user has flagged as favorites.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Bad examples: impersonal information

- A news app displays the total number of new articles coming from its service, not taking into account the user's stated preferences.
- A shopping app sends a notification about a sale, but not based on any item or category preference that the user has given.

Showing nothing is better than showing status that never changes

The information should display only when a change has occurred

As we said earlier, the goal is that information on the lock screen can be taken in at a glance. To that end, if an app is not currently displaying a badge, a gap is left on the lock screen where that badge would otherwise appear. This increases the ability of a user to notice something that needs their attention—the appearance of a badge and logo following an event is more noticeable than if it has been there all along, communicating nothing new.

Do not show status simply for the sake of showing status. Long-running or never-changing status just clutters the lock screen, obscuring more important information. A badge should display only when something has happened that the user should be aware of. The same is true for a tile update. Remove stale notification content from your tile, which causes the tile to revert to its default image in the Start screen and displays nothing on the lock screen.

Good examples: information displayed only when it's useful

- A mail app displays a badge only when there is unread mail. When new mail arrives, its badge is updated and shown.
- A messaging app displays its connection status only when the user is unable to receive messages. A "connected" status is the assumed default state of the app, so there is no point in conveying that information. "Everything is fine" is not an actionable notification. However, informing the user when they cannot receive messages is useful, actionable information.

Bad examples: long-running status

- A mail or messaging app has no count of unread mail to display and so shows a connection status until new mail or messages arrive. This decreases the user's ability to see at a glance whether they have a new message, because the badge is always present.
- A calendar app displays a message stating that the user has no appointments. Again, this decreases the at-a-glance usability of the detailed status slot, since something would always be displayed there.

Only toast notifications should play a sound on arrival

Do not include code in your app that plays a sound when your badge or tile updates. However, an arriving toast can play a sound as it is designed to do.

© 2014 Microsoft. All rights reserved.

By following the guidance described in this article, you will be able to create apps that display the right information in the right way on the lock screen, thereby increasing user satisfaction and confidence in your app.

When to use the lock screen request API

Only call the lock screen request API (**RequestAccessAsync**) if your app truly needs background privileges to function properly. Because there are only seven background slots available, users must distinguish which apps truly need background privileges to function properly and which work fine without them (even if they might gain additional functionality with them).

If an app absolutely requires background privileges to meet user expectations, we recommend that it uses the request API to prompt the user to place the app on the lock screen.

However, if an app will meet user expectations without having background privileges, we recommend that you do not explicitly prompt the user to place the app on the lock screen. Instead, let the user place their app on the lock screen through the **Personalize** page of **PC Settings**.

Examples of apps that should call the request API:

- A messaging app that requires background privileges to receive messages when the app is not in the foreground
- A mail app that requires background privileges to sync the user's inbox when the app is not in the foreground

Examples of apps that should not call the request API:

- A weather app that uses periodic notifications rather than background activity to update its forecast
- A news app that refreshes its badge count of new articles at a specific time of day

Note: Your app should not implement its own dialog to prompt users to add the app to the lock screen. If your app requires lock screen access to run properly, you should rely on the dialog presented by the lock screen request API. If a user previously denied lock screen rights to your app through this dialog, the dialog may not be shown again. In this case, you can use inline text in your app to direct users to the **Personalize** page of **PC Settings** to manually add your app to the lock screen.

Checklist for Windows Store requirements

To be accepted in the Windows Store, you cannot use your tiles or notifications to display advertisements.

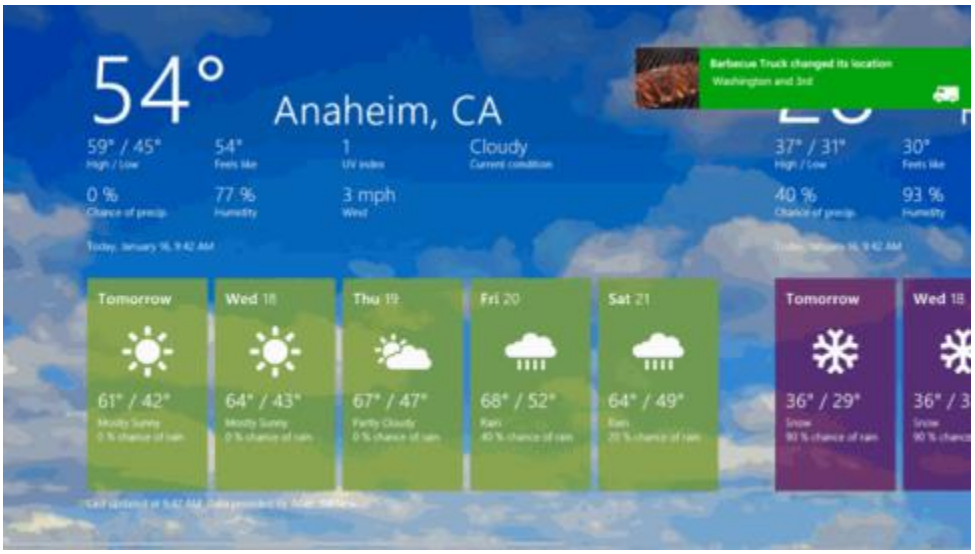
© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for toast notifications

This topic describes when to use toast notifications and provides recommendations for how to create and send toasts.

Example

A toast notification alerts a user that his or her favorite food truck changed its location.



Should my app include toast notifications?

Toasts allow your app to provide time-sensitive or personally relevant notifications to users regardless of whether they are in another app or on the Start screen, lock screen, or desktop. For example, you could use a toast to inform a user of:

- an incoming VOIP call
- a new instant message
- a new text message
- a calendar appointment or other reminder
- other personally valuable notifications that a user requests.

Keep in mind that users must opt-in to receive toast notifications and can disable them at any time.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Dos and don'ts

Consider the following recommendations when adding toast notifications to your app:

- Navigate to an appropriate destination in your app when the user clicks a toast. Consider that notifications are an invitation to switch context rather than a strictly informational update.
- Provide alternate ways for users to get the info provided in a toast if it's important. For example, you may want to display related information on your app's live tile or within your app.
- Combine multiple related updates that occur within a short period of time into a single toast notification. For instance, if you have three new updates that arrive at the same time, the app or app server should raise a single notification that states that there are three new updates rather than display three separate notifications.
- Present information in the simplest possible form. If your content doesn't require a headline, omit it. A message such as "Your download is complete." is entirely complete and needs no additional presentation.
- Use images when they add clear value to the message, such as a photo of the sender of a message.
- Hide notifications if they are no longer valid. For example, hide a toast about an incoming call if the other party has hung up or the user has already answered on another device. Note that you can only hide notifications when your app is running.
- Don't use toast notifications to notify the user of critical information. Instead, to ensure that critical alerts are seen, notify users within your app using a flyout, dialog, app bar, or other inline element.
- Don't include text telling the user to "click here to..." It is assumed that all toast notifications have a click or tap action that will take the user to the associated app.
- Don't use toast notifications to notify the user of transient failures or network events, such as a dropped connection.
- Don't use toast notifications for anything with a high volume of notifications, such as stock price information.
- Don't use toast notifications to notify the user of routine maintenance events, such as the completion of an anti-virus scan.
- Don't raise a toast notification when your app is in the foreground and a more contextual surface such as an inline element, flyout, dialog, or app bar is available. For example, additional instant messages that are related to an ongoing conversation that is in view should update the conversation inline rather than continue to raise a toast with each new message. Listen for the **PushNotificationReceived** event to intercept push notifications when your application is running.
- Don't add generic images such as icons or your app logo in the image field of a notification.
- Don't place your app's name in the text of the notification. Users will identify your app by its logo, which is automatically included in the toast notification.
- Don't use your app to ask users to enable toast notifications if they have chosen to disable them. Your app is expected to work without toast notifications.
- Don't automatically migrate your balloon notification scenarios to toast—consider that it may be more appropriate to notify the user when they aren't immersed in a full-screen experience (desktop style apps only).

© 2014 Microsoft. All rights reserved. | May 13, 2014

- Don't use toast notifications for non-real-time information, such as a picture of the day.
- Don't hide toast notifications unless absolutely necessary.
- Don't notify the user of something they didn't ask to be notified about. For instance, don't assume that all users want to be notified each time one of their contacts appears online.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for interactions



Use the guidelines in this section to learn about different ways users can interact with your app, and the recommendations for designing an app that is easy to use.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for designing accessible apps



As you design your app, always keep in mind that your users have a wide range of abilities, disabilities, and preferences. Follow these principles of accessible design to ensure that your app is accessible to the widest possible audience, and helps attract more customers to your app in the Windows Store.

Why plan for accessibility?

Designing your app with accessibility in mind helps ensure that it works well in the following scenarios.

- **Screen reading:** Blind or visually impaired users rely on screen readers to help them create and maintain a mental model of your app's UI. Hearing information about the UI and including the names of UI elements, helps users understand the UI content and interact with your app.

  To support screen reading, your app needs to provide sufficient and correct information about its UI elements, including name, role, description, state, and value.

  You must also provide additional accessibility information for UI elements that contain dynamic content, such as a live region in a Windows Runtime app using JavaScript with HTML. The additional accessibility information lets screen readers announce changes that occur to the content. To provide accessibility information for a live region in HTML, set the **aria-live** attribute on elements that contain dynamic content. To provide accessibility information for a live region using the ARIA metaphors for live content in XAML, use the attached property **AutomationProperties.LiveSetting**.

- **Keyboard accessibility:** The keyboard is integral to using a screen reader, and it is also important for users who prefer the keyboard as a more efficient way to interact with an app. An accessible app lets users access all interactive UI elements by keyboard only, enabling users to:
  - Navigate the app by using the Tab and arrow keys.
  - Activate UI elements by using the Spacebar and Enter keys.
  - Access commands and controls by using keyboard shortcuts.

  The On-Screen Keyboard is available for systems that don't include a physical keyboard, or for users whose mobility impairments prevent them from using traditional physical input devices.

- **Accessible visual experience:** Visually impaired users need text to be displayed with a high contrast ratio. They also need a UI that looks good in high-contrast mode and scales

© 2014 Microsoft. All rights reserved. | May 13, 2014

properly after changing settings in the **Ease of Access** control panel. Where color is used to convey information, users with color blindness need color alternatives like text, shapes, and icons.

Dos and don'ts

- Support screen reading by providing information about your app's UI elements, including name, role, description, state, and value.
- Let users navigate your app using the Tab and arrow keys.
- Activate UI elements by using the Spacebar and Enter keys.
- Access commands and controls by using keyboard shortcuts.
- Design text and UI to support high contrast themes.
- Ensure that text and UI to make appropriate scaling adjustments when **Ease of Access** settings are changed.
- Don’t use color as the only way to convey information. Users who are color blind cannot receive information that is conveyed only through color, such as in a color status indicator. Include other visual cues, preferably text, to ensure that information is accessible.
- Don’t use UI elements that flash more than three times per second. Flashing elements can cause some people to have seizures. It is best to avoid using UI elements that flash.
- Don’t change user context or activate functionality automatically. Context or activation changes should occur only when the user takes a direct action on a UI element that has focus. Changes in user context include changing focus, displaying new content, and navigating to a different page. Making context changes without involving the user can be disorienting for users who have disabilities. The exceptions to this requirement include displaying submenus, validating forms, displaying help text in another control, and changing context in response to an asynchronous event.
- Avoid building custom UI elements if you can use the default controls included with the Windows Runtime or controls that have already implemented Microsoft UI Automation support. Standard Windows Runtime controls are accessible by default and usually require adding only a few accessibility attributes that are app-specific. In contrast, implementing the **AutomationPeer** support for a true custom control is somewhat more involved.
- Don't put static text or other non-interactive elements into the tab order (for example, by setting the **TabIndex** property for an element that is not interactive). If non-interactive elements are in the tab order, that is against keyboard accessibility guidelines because it decreases efficiency of keyboard navigation for users. Many assistive technologies use tab order and ability to focus an element as part of their logic for how to present an app's interface to the assistive technology user. Text-only elements in the tab order can confuse users who expect only interactive elements in the tab order (buttons, check boxes, text input fields, combo boxes, lists, and so on).
- Avoid using absolute positioning of UI elements (such as in a **Canvas** element) because the presentation order often differs from the child element declaration order (which is the de facto logical order). Whenever possible, arrange UI elements in document or logical order to ensure that screen readers can read those elements in the correct order. If the visible order of UI elements can diverge from the document or logical order, use explicit tab index values (set **TabIndex**) to define the correct reading order.
- Don’t automatically refresh an entire app canvas unless it is really necessary for app functionality. If you need to automatically refresh page content, update only certain areas of

© 2014 Microsoft. All rights reserved. | May 13, 2014

the page. Assistive technologies generally must assume that a refreshed app canvas is a totally new structure, even if the effective changes were minimal. The cost of this to the assistive technology user is that any document view or description of the refreshed app now must be recreated and presented to the user again.

Note: If you do refresh content within a region, consider setting the **AccessibilityProperties.LiveSetting** accessibility property on that element to one of the non-default settings **Polite** or **Assertive**. Some assistive technologies can map this setting to the Accessible Rich Internet Applications (ARIA) concept of live regions and can thus inform the user that a region of content has changed.

Note: A deliberate page navigation that is initiated by the user is a legitimate case for refreshing the app's structure. But make sure that the UI item that initiates the navigation is correctly identified or named to give some indication that invoking it will result in a context change and page reload.

Additional usage guidance

Make your HTML custom controls accessible

If you use an HTML custom control, you need to provide all of the basic accessibility information for the control, including the accessible name, role, state, value, and so on. You also need to ensure that the control is fully accessible by keyboard, and that the UI meets requirements for visual accessibility.

For example, suppose you include a **div** element that represents a custom interactive element; that is, it handles the **onclick** event. You must:

- Set an accessible name for the **div** element.
- Set the role attribute to the corresponding interactive ARIA role, such as "button”.
- Set the tabindex attribute to include the element in the tab order.
- Add keyboard event handlers to support keyboard activation; that is, the keyboard equivalent of an **onclick** event handler.

Note: The HTML5 **canvas** element doesn’t support accessibility. Because it doesn’t provide any way to expose accessibility information for its content, avoid using **canvas** unless it is absolutely necessary. If you do use **canvas**, treat it as a custom UI element.

Make your XAML custom controls accessible

If you use a XAML custom control, you might need to adjust some of the basic accessibility information for the control, including the accessible name, role, state, value, and so on. You also need to verify that the control is fully accessible by keyboard, and that the UI meets requirements for visual accessibility. When you create custom controls for XAML, you inherit the UI Automation support that was available for whichever control you used as the base class for your custom control. Sometimes that's good enough. But depending on the degree that you're customizing your control,

© 2014 Microsoft. All rights reserved. | May 13, 2014

you might also want to create a custom UI Automation peer class that modifies or augments the default UI Automation support. This is enabled by the APIs in **Windows.UI.Xaml.Automation** and **Windows.UI.Xaml.Automation.Peers** namespaces.

Accessibility support in the development platform

The Windows Runtime development platform supports accessibility in all stages of the development cycle:

- **Creating:** The code generated from the Microsoft Visual Studio app templates includes accessibility information.
- **Coding:** The development platform includes the following accessibility support during the coding stage.
    - Microsoft IntelliSense in Visual Studio includes accessibility information.
    - The controls included in the Windows 8 platform have built-in support for accessibility. By using the standard HTML and platform controls, you can get most of your accessibility support as a default platform behavior. For example, the ratings control is fully accessible without any additional work, while the **ListView** controls require only an accessible name for the main list element—all other accessibility support is built in.
    - The Windows Dev Center documentation includes accessibility guidelines and sample apps.
- **Testing:** The Windows Software Development Kit (SDK) includes accessibility testing tools.
- **Selling:** You can mark your app as accessible when you publish it in the Windows Store, enabling users to discover your app by using the **Accessibility** filter when browsing the store.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for cross-slide



Use cross-slide to support selection with the swipe gesture and drag (move) interactions with the slide gesture.

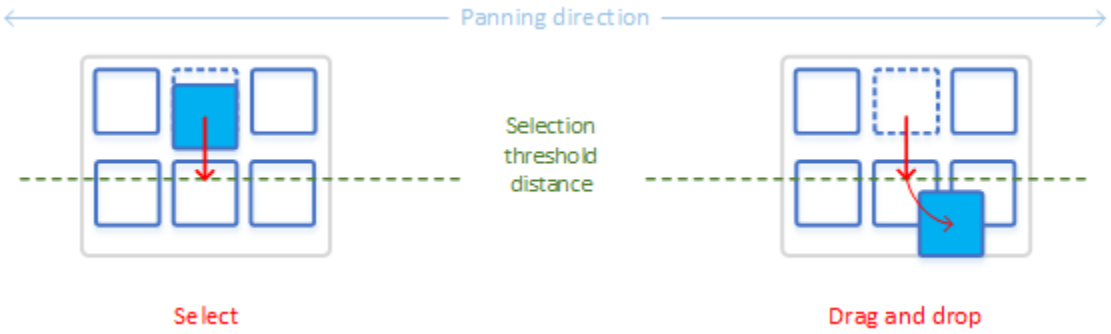
Dos and don'ts

- Use cross-slide for lists or collections that scroll in a single direction.
- Use cross-slide for item selection when the tap interaction is used for another purpose.
- Don't use cross-slide for adding items to a queue.

Additional usage guidance

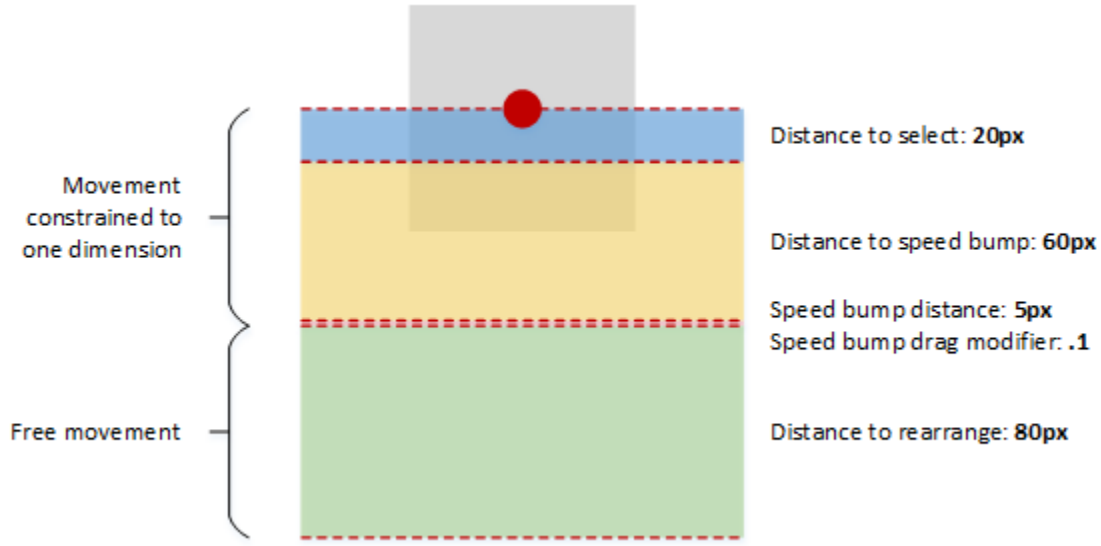
Selection and drag are possible only within a content area that is pannable in one direction (vertical or horizontal). For either interaction to work, one panning direction must be locked and the gesture must be performed in the direction perpendicular to the panning direction.

Here we demonstrate selecting and dragging an object using a cross-slide. The image on the left shows how an item is selected if a swipe gesture doesn't cross a distance threshold before the contact is lifted and the object released. The image on the right shows a sliding gesture that crosses a distance threshold and results in the object being dragged.



The threshold distances used by the cross-slide interaction are shown in the following diagram.

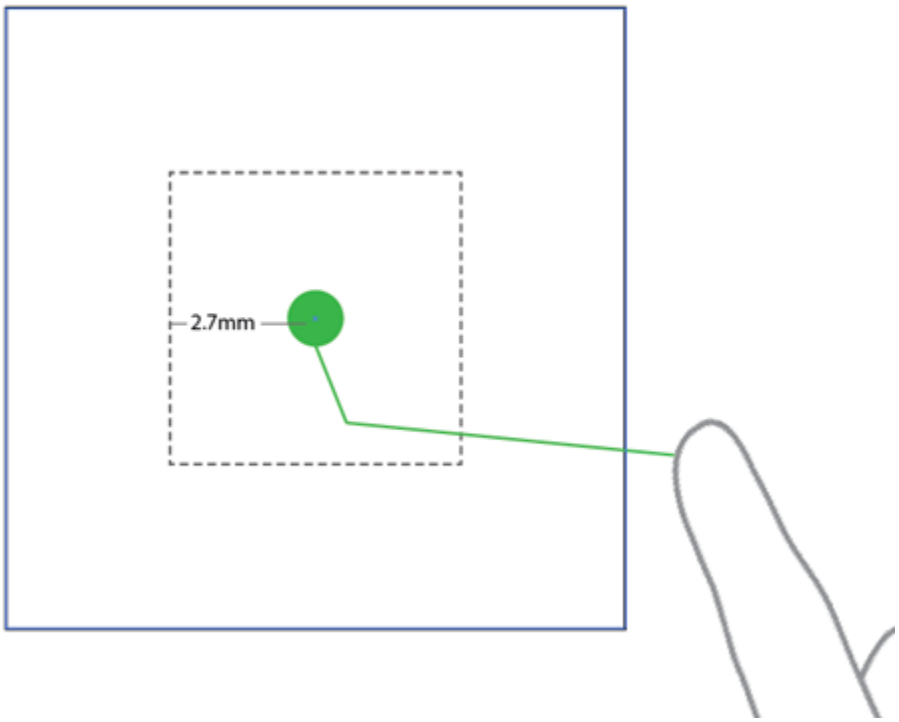
© 2014 Microsoft. All rights reserved. | May 13, 2014



To preserve panning functionality, a small threshold of 2.7mm (approximately 10 pixels at target resolution) must be crossed before either a select or drag interaction is activated. This small threshold helps the system to differentiate cross-sliding from panning, and also helps ensure that a tap gesture is distinguished from both cross-sliding and panning.

This image shows how a user touches an element in the UI, but moves their finger down slightly at contact. With no threshold, the interaction would be interpreted as a cross-slide because of the initial vertical movement. With the threshold, the movement is interpreted correctly as horizontal panning.

© 2014 Microsoft. All rights reserved. | May 13, 2014

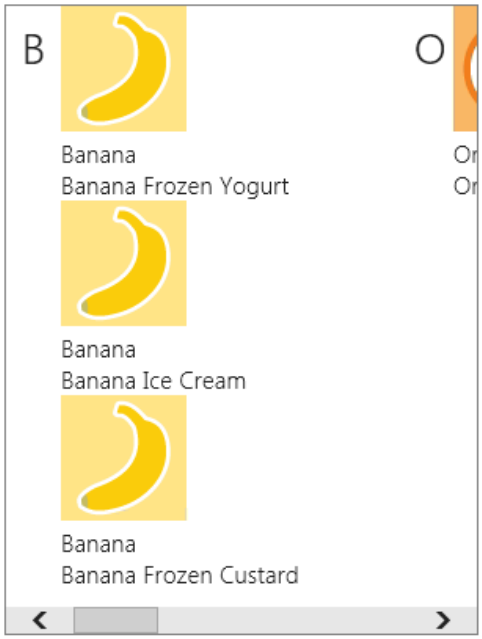


Here are some guidelines to consider when including cross-slide functionality in your app.

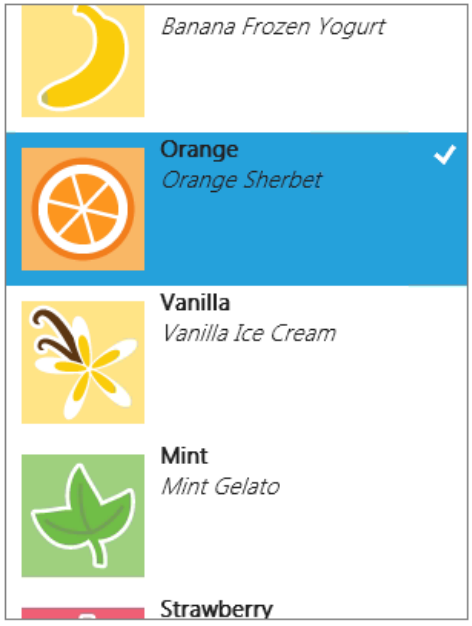
Use cross-slide for lists or collections that scroll in a single direction.

Note: In cases where the content area can be panned in two directions, such as web browsers or e-readers, the press-and-hold timed interaction should be used to invoke the context menu for objects such as images and hyperlinks.

© 2014 Microsoft. All rights reserved. | May 13, 2014



A horizontally panning two-dimensional list. Drag vertically to select or move an item.



A vertically panning one-dimensional list. Drag horizontally to select or move an item.

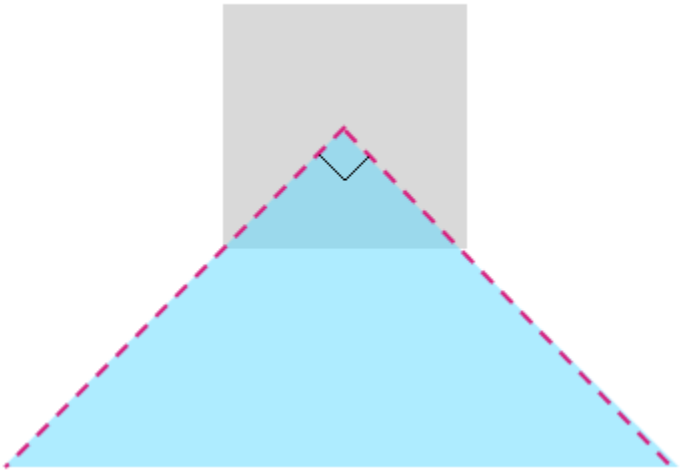
Selecting

Selection is the marking, without launching or activating, of one or more objects. This action is analogous to a single mouse click, or Shift key and mouse click, on one or more objects.

Cross-slide selection is achieved by touching an element and releasing it after a short dragging interaction. This method of selection dispenses with both the dedicated selection mode and the press-and-hold timed interaction required by other touch interfaces and does not conflict with the tap interaction for activation.

In addition to the distance threshold, cross-slide selection is constrained to a 90° threshold area, as shown in the following diagram. If the object is dragged outside of this area, it is not selected.

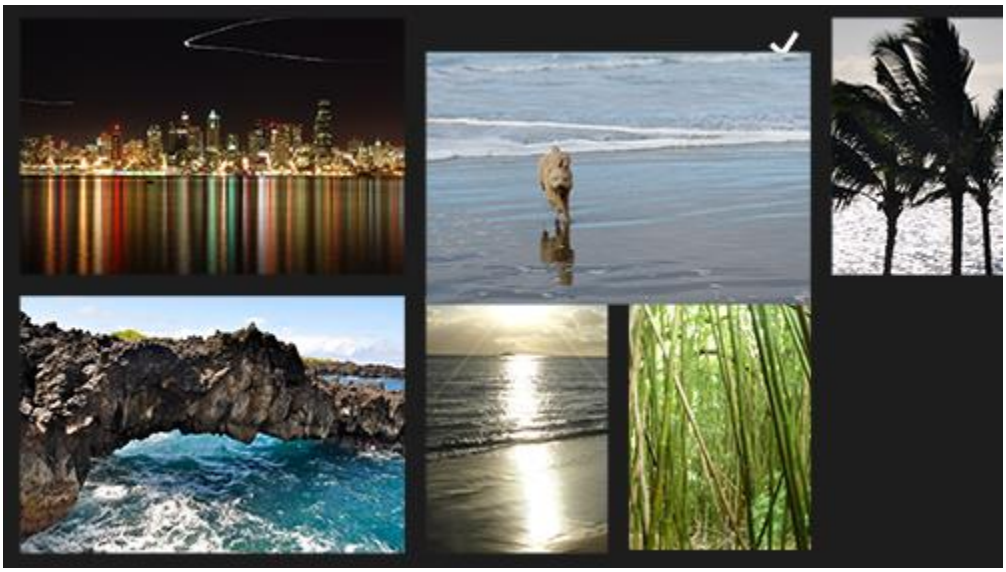
© 2014 Microsoft. All rights reserved. | May 13, 2014



The cross-slide interaction is supplemented by a press-and-hold timed interaction, also referred to as a "self-revealing" interaction. This supplemental interaction activates an animation that indicates what action can be performed on the object.

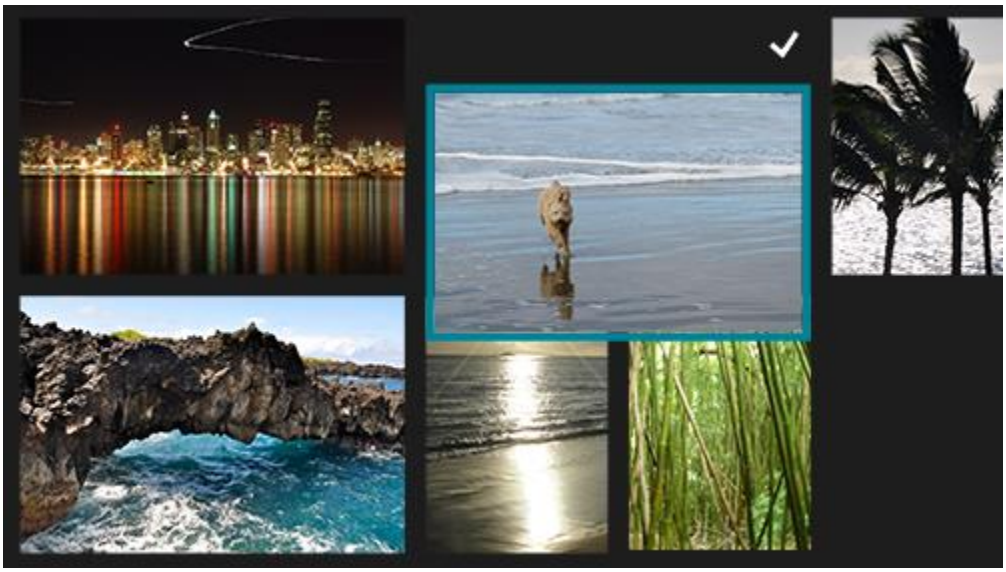
The following screen shots demonstrate how the self-revealing animation works.

1. Press and hold to initiate the animation for the self-revealing interaction. The selected state of the item affects what is revealed by the animation: a check mark if unselected and no check mark if selected.



2. Select the item using the swipe gesture (up or down).

© 2014 Microsoft. All rights reserved. | May 13, 2014



3. The item is now selected. Override the selection behavior using the slide gesture to move the item.



Use a single tap for selection in applications where it is the only primary action. The cross-slide self-revealing animation is displayed to disambiguate this functionality from the standard tap interaction for activation and navigation.

Selection basket

The selection basket is a visually distinct and dynamic representation of items that have been selected from the primary list or collection in the application. This feature is useful for tracking selected items and should be used by applications where:

© 2014 Microsoft. All rights reserved.

- Items can be selected from multiple locations.
- Many items can be selected.
- An action or command relies upon the selection list.

The content of the selection basket persists across actions and commands. For example, if you select a series of photographs from a gallery, apply a color correction to each photograph, and share the photographs in some fashion, the items remain selected.

If no selection basket is used in an application, the current selection should be cleared after an action or command. For example, if you select a song from a play list and rate it, the selection should be cleared.

The current selection should also be cleared when no selection basket is used and another item in the list or collection is activated. For example, if you select an inbox message, the preview pane is updated. Then, if you select a second inbox message, the selection of the previous message is canceled and the preview pane is updated.

Queues

A queue is not equivalent to the selection basket list and should not be treated as such. The primary distinctions include:

- The list of items in the selection basket is only a visual representation; the items in a queue are assembled with a specific action in mind.
- Items can be represented only once in the selection basket but multiple times in a queue.
- The order of items in the selection basket represents the order of selection. The order of items in a queue is directly related to functionality.

For these reasons, the cross-slide selection interaction should not be used to add items to a queue. Instead, items should be added to a queue through a drag action.

Drag

Use drag to move one or more objects from one location to another.

If more than one object needs to be moved, let users select multiple items and then drag all at one time.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for optical zoom and resizing



This topic describes Windows zooming and resizing elements and provides user experience guidelines for using these interaction mechanisms in your apps.

Description

Optical zoom lets users magnify their view of the content within a content area (it is performed on the content area itself), whereas resizing enables users to change the relative size of one or more objects without changing the view of the content area (it is performed on the objects within the content area).

Both optical zoom and resizing interactions are performed through the pinch and stretch gestures (moving fingers farther apart zooms in and moving them closer together zooms out), or by holding the Ctrl key down while scrolling the mouse scroll wheel, or by holding the Ctrl key down (with the Shift key, if no numeric keypad is available) and pressing the plus (+) or minus (-) key.

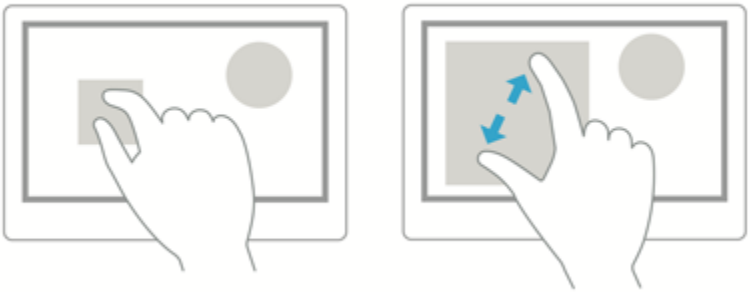
The following diagrams demonstrate the differences between resizing and optical zooming.

Optical zoom: User selects an area, and then zooms into the entire area.



Resize: User selects an object within an area, and resizes that object.

© 2014 Microsoft. All rights reserved. | May 13, 2014



Note

Optical zoom shouldn't be confused with Semantic Zoom. Although the same gestures are used for both interactions, semantic zoom refers to the presentation and navigation of content organized within a single view (such as the folder structure of a computer, a library of documents, or a photo album).

Dos and don'ts

Use the following guidelines for apps that support either resizing or optical zooming:

- If maximum and minimum size constraints or boundaries are defined, use visual feedback to demonstrate when the user reaches or exceeds those boundaries.
- Use snap points to influence zooming and resizing behavior by providing logical points at which to stop the manipulation and ensure a specific subset of content is displayed in the viewport. Provide snap points for common zoom levels or logical views to make it easier for a user to select those levels. For example, photo apps might provide a resizing snap point at 100% or, in the case of mapping apps, snap points might be useful at city, state, and country views.

  Snap points enable users to be imprecise and still achieve their goals. If you're using XAML, see the snap points properties of **ScrollViewer**. For JavaScript and HTML, use **-ms-content-zoom-snap-points**.

  There are two types of snap-points:

  - Proximity - After the contact is lifted, a snap point is selected if inertia stops within a distance threshold of the snap point. Proximity snap points still allow a zoom or resize to end between snap points.
  - Mandatory - The snap point selected is the one that immediately precedes or succeeds the last snap point crossed before the contact was lifted (depending on the direction and velocity of the gesture). A manipulation must end on a mandatory snap point.

© 2014 Microsoft. All rights reserved. | May 13, 2014

- Use inertia physics. These include the following:
    - Deceleration: Occurs when the user stops pinching or stretching. This is similar to sliding to a stop on a slippery surface.
    - Bounce: A slight bounce-back effect occurs when a size constraint or boundary is passed.
- Space controls according to the *Guidelines for targeting*.
- Provide scaling handles for constrained resizing. Isometric, or proportional, resizing is the default if the handles are not specified.
- Don't use zooming to navigate the UI or expose additional controls within your app, use a panning region instead. For more info on panning, see *Guidelines for panning*.
- Don't put resizable objects within a resizable content area. Exceptions to this include:
    - Drawing applications where resizable items can appear on a resizable canvas or art board.
    - Webpages with an embedded object such as a map.

Note: In all cases, the content area is resized unless all touch points are within the resizable object.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for panning



Panning or scrolling lets users navigate within a single view, to display the content of the view that does not fit within the viewport. Examples of views include the folder structure of a computer, a library of documents, or a photo album.

Dos and don'ts

Panning indicators and scroll bars

- Ensure panning/scrolling is possible before loading content into your app.
- Display panning indicators and scroll bars to provide location and size cues. Hide them if you provide a custom navigation feature.

Note: Unlike standard scroll bars, panning indicators are purely informative. They are not exposed to input devices and cannot be manipulated in any way.

Single-axis panning (one-dimensional overflow)

- Use one-axis panning for content regions that extend beyond one viewport boundary (vertical or horizontal).
    - Vertical panning for a one-dimensional list of items.
    - Horizontal panning for a grid of items.
- Don’t use mandatory snap-points with single-axis panning if a user must be able to pan and stop between snap-points. Mandatory snap-points guarantee that the user will stop on a snap-point. Use proximity snap-points instead.

Freeform panning (two-dimensional overflow)

- Use two-axis panning for content regions that extend beyond both viewport boundaries (vertical and horizontal).
    - Override the default rails behavior and use freeform panning for unstructured content where the user is likely to move in multiple directions.
- Freeform panning is typically suited to navigating within images or maps.

Paged view

- Use mandatory snap-points when the content is composed of discrete elements or you want to display an entire element. This can include pages of a book or magazine, a column of items, or individual images.
    - A snap-point should be placed at each logical boundary.
    - Each element should be sized or scaled to fit the view.

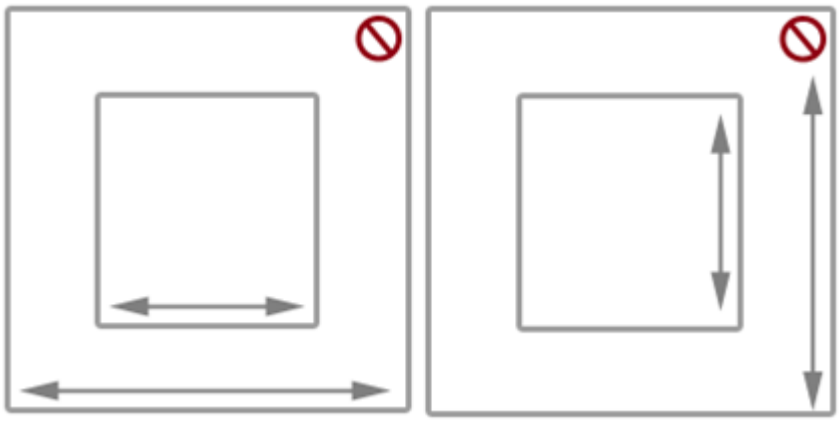
© 2014 Microsoft. All rights reserved. | May 13, 2014

Logical and key points

- Use proximity snap-points if there are key points or logical places in the content that a user will likely stop. For example, a section header.
- If maximum and minimum size constraints or boundaries are defined, use visual feedback to demonstrate when the user reaches or exceeds those boundaries.

Chaining embedded or nested content

- Use single-axis panning (typically horizontal) and column layouts for text and grid-based content. In these cases, content typically wraps and flows naturally from column to column and keeps the user experience consistent and discoverable across Windows Store apps.
- Don't use embedded pannable regions to display text or item lists. Because the panning indicators and scroll bars are displayed only when the input contact is detected within the region, it is not an intuitive or discoverable user experience.
- Don't chain or place one pannable region within another pannable region if they both pan in the same direction, as shown in this diagram:



This can result in the parent area being panned unintentionally when a boundary for the child area is reached. Consider making the panning axis perpendicular.

Additional usage guidance

Panning with touch, by using a swipe or slide gesture with one or more fingers, is like scrolling with the mouse. The panning interaction is most similar to rotating the mouse wheel or sliding the scroll box, rather than clicking the scroll bar. Unless a distinction is made in an API or required by some device-specific Windows UI, we simply refer to both interactions as panning.

Depending on the input device, the user pans within a pannable region by using one of these:

- A mouse, touchpad, or active pen/stylus to click the scroll arrows, drag the scroll box, or click within the scroll bar.
- The wheel button of the mouse to emulate dragging the scroll box.
- The extended buttons (XBUTTON1 and XBUTTON2), if supported by the mouse.

© 2014 Microsoft. All rights reserved. | May 13, 2014

- The keyboard arrow keys to emulate dragging the scroll box or the page keys to emulate clicking within the scroll bar.
- Touch, touchpad, or passive pen/stylus to slide or swipe the fingers in the desired direction.

Sliding involves moving the fingers slowly in the panning direction. This results in a one-to-one relationship, where the content pans at the same speed and distance as the fingers. Swiping, which involves rapidly sliding and lifting the fingers, results in the following physics being applied to the panning animation:

- Deceleration (inertia): Lifting the fingers causes panning to start decelerating. This is similar to sliding to a stop on a slippery surface.
- Absorption: Panning momentum during deceleration causes a slight bounce-back effect if either a snap point or a content area boundary is reached.

Types of panning

Windows 8 supports three types of panning:

- Single axis - panning is supported in one direction only (horizontal or vertical).
- Rails - panning is supported in all directions. However, once the user crosses a distance threshold in a specific direction, then panning is restricted to that axis.
- Freeform - panning is supported in all directions.

Panning UI

The interaction experience for panning is unique to the input device while still providing similar functionality.

Pannable regions

Pannable region behaviors are exposed to Windows Store app using JavaScript developers at design time through Cascading Style Sheets (CSS).

There are two panning display modes based on the input device detected:

- Panning indicators for touch.
- Scroll bars for other input devices, including mouse, touchpad, keyboard, and stylus.

Note: Panning indicators are only visible when the touch contact is within the pannable region. Similarly, the scroll bar is only visible when the mouse cursor, pen/stylus cursor, or keyboard focus is within the scrollable region.

Panning indicators

Panning indicators are similar to the scroll box in a scroll bar. They indicate the proportion of displayed content to total pannable area and the relative position of the displayed content in the pannable area.

© 2014 Microsoft. All rights reserved. | May 13, 2014

The following diagram shows two pannable areas of different lengths and their panning indicators.



Panning behaviors
Snap points

Panning with the swipe gesture introduces inertia behavior into the interaction when the touch contact is lifted. With inertia, the content continues to pan until some distance threshold is reached without direct input from the user. Use snap points to modify this inertia behavior.

Snap points specify logical stops in your app content. Cognitively, snap points act as a paging mechanism for the user and minimize fatigue from excessive sliding or swiping in large pannable regions. With them, you can handle imprecise user input and ensure a specific subset of content or key information is displayed in the viewport.

There are two types of snap-points:

- Proximity - After the contact is lifted, a snap point is selected if inertia stops within a distance threshold of the snap point. Panning can still stop between proximity snap points.
- Mandatory - The snap point selected is the one that immediately precedes or succeeds the last snap point crossed before the contact was lifted (depending on the direction and velocity of the gesture). Panning must stop on a mandatory snap point.

Panning snap-points are useful for applications such as web browsers and photo albums that emulate paginated content or have logical groupings of items that can be dynamically regrouped to fit within a viewport or display.

The following diagrams show how panning to a certain point and releasing causes the content to automatically pan to a logical location.

© 2014 Microsoft. All rights reserved. | May 13, 2014

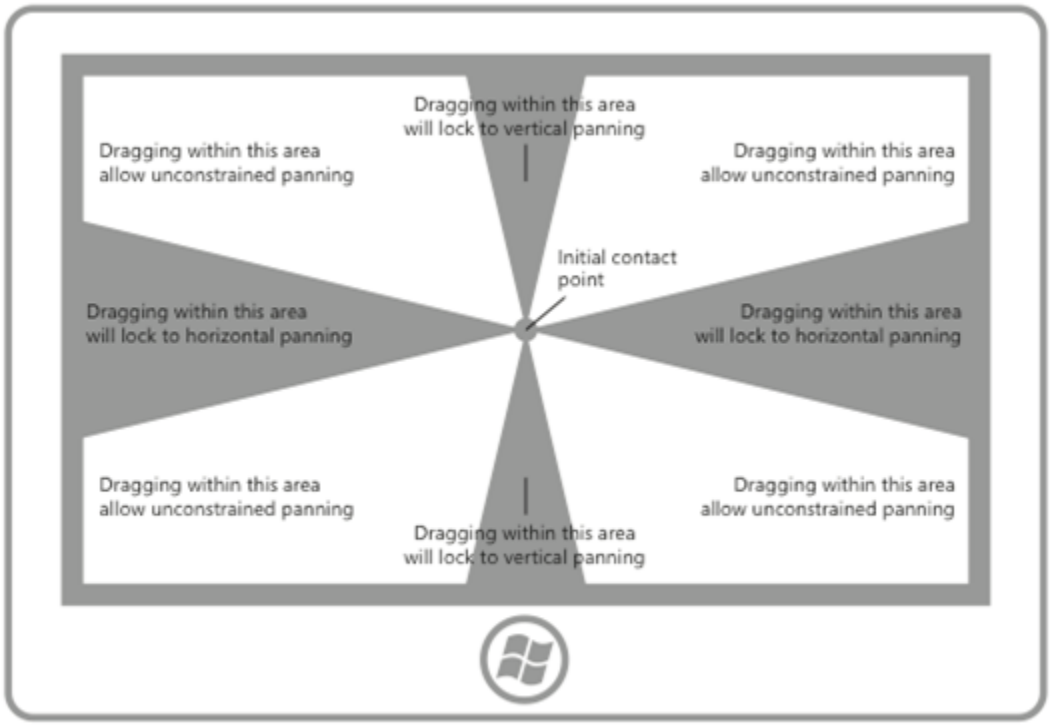
		
Swipe to pan.	Lift touch contact.	Pannable region stops at the snap point, not where the touch contact was lifted.

Rails

Content can be wider and taller than the dimensions and resolution of a display device. For this reason, two-dimensional panning (horizontal and vertical) is often necessary. Rails improve the user experience in these cases by emphasizing panning along the axis of motion (vertical or horizontal).

The following diagram demonstrates the concept of rails.

© 2014 Microsoft. All rights reserved. | May 13, 2014



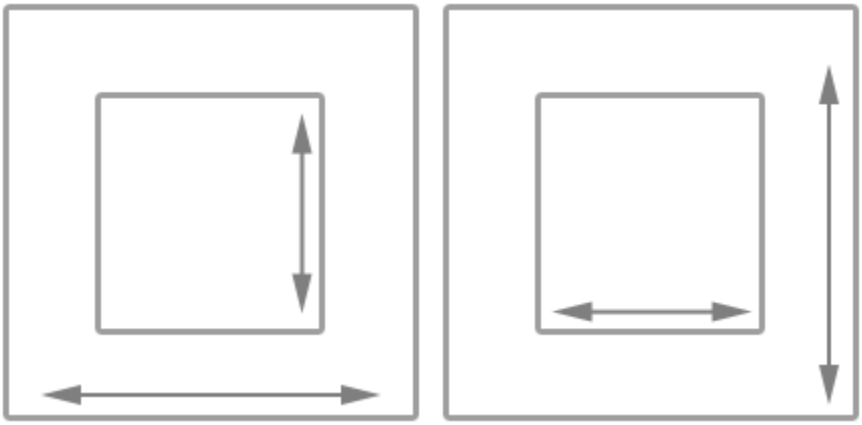
Chaining embedded or nested content

After a user hits a zoom or scroll limit on an element that has been nested within another zoomable or scrollable element, you can specify whether that parent element should continue the zooming or scrolling operation begun in its child element. This is called zoom or scroll chaining.

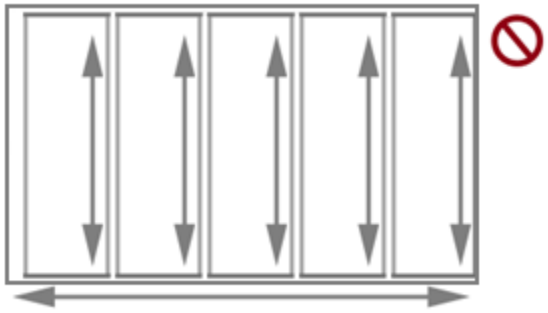
Chaining is used for panning within a single-axis content area that contains one or more single-axis or freeform panning regions (when the touch contact is within one of these child regions). When the panning boundary of the child region is reached in a specific direction, panning is then activated on the parent region in the same direction.

When a pannable region is nested inside another pannable region it's important to specify enough space between the container and the embedded content. In the following diagrams, one pannable region is placed inside another pannable region, each going in perpendicular directions. There is plenty of space for users to pan in each region.

© 2014 Microsoft. All rights reserved. | May 13, 2014



Without enough space, as shown in the following diagram, the embedded pannable region can interfere with panning in the container and result in unintentional panning in one or more of the pannable regions.



This guidance is also useful for apps such as photo albums or mapping apps that support unconstrained panning within an individual image or map while also supporting single-axis panning within the album (to the previous or next images) or details area. In apps that provide a detail or options area corresponding to a freeform panning image or map, we recommend that the page layout start with the details and options area as the unconstrained panning area of the image or map might interfere with panning to the details area.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for rotation



This topic describes the new Windows UI for rotation and provides user experience guidelines that should be considered when using this new interaction mechanism in your Windows Store app.

Dos and don'ts

- Use rotation to help users directly rotate UI elements.

Additional usage guidance

Overview of rotation

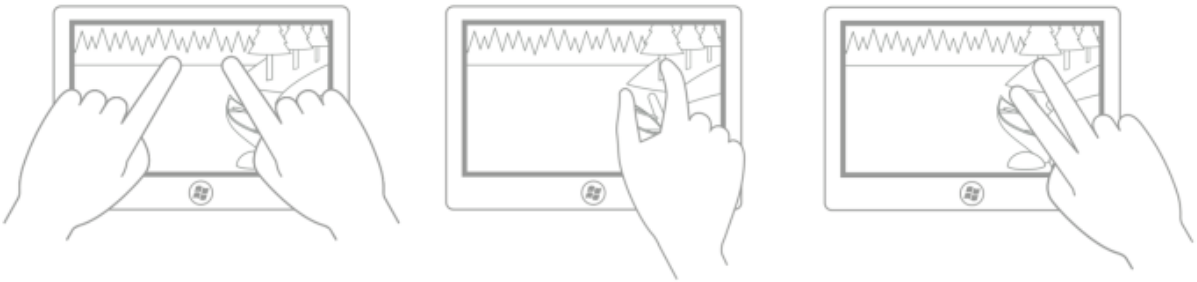
Rotation is the touch-optimized technique used by Windows Store apps in Windows 8 to enable users to turn an object in a circular direction (clockwise or counter-clockwise).

Depending on the input device, the rotation interaction is performed using:

- A mouse or active pen/stylus to move the rotation gripper of a selected object.
- Touch or passive pen/stylus to turn the object in the desired direction using the rotate gesture.

When to use rotation

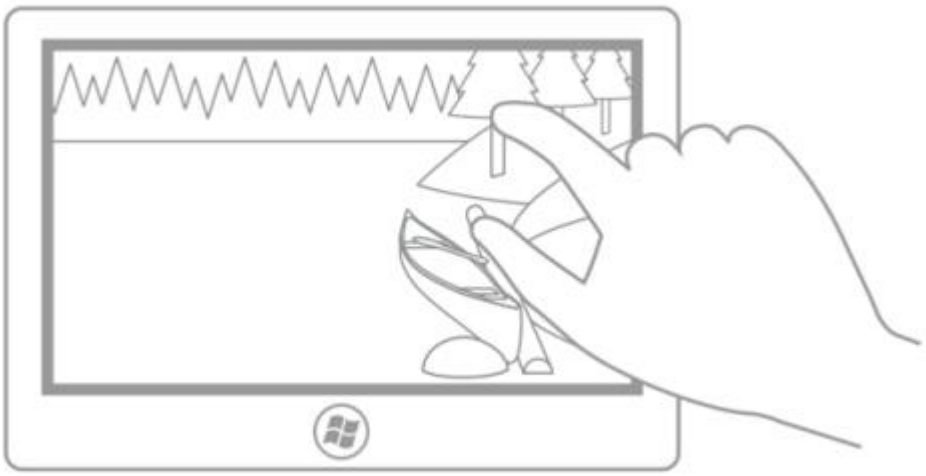
Use rotation to help users directly rotate UI elements. The following diagrams show some of the supported finger positions for the rotation interaction.



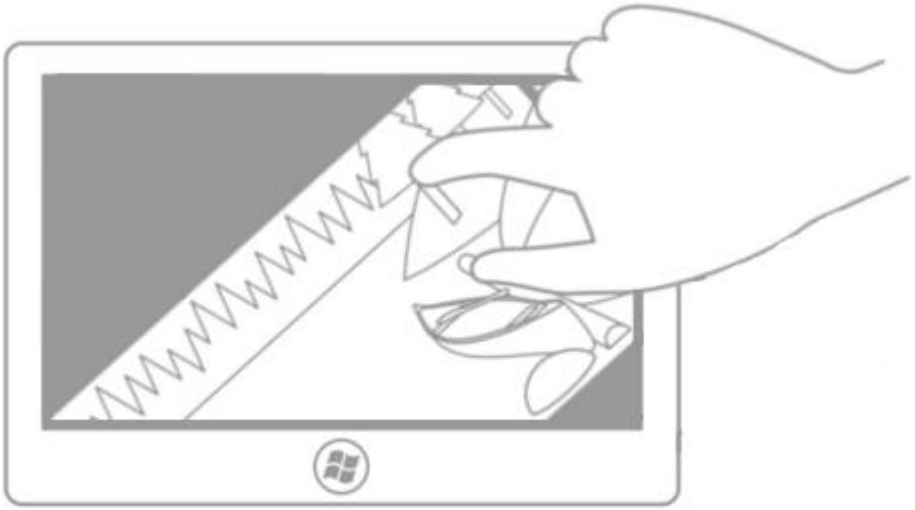
Note: Intuitively, and in most cases, the rotation point is one of the two touch points unless the user can specify a rotation point unrelated to the contact points (for example, in a drawing or layout application). The following images demonstrate how the user experience can be degraded if the rotation point is not constrained in this way.

© 2014 Microsoft. All rights reserved. | May 13, 2014

This first picture shows the initial (thumb) and secondary (index finger) touch points: the index finger is touching a tree and the thumb is touching a log.

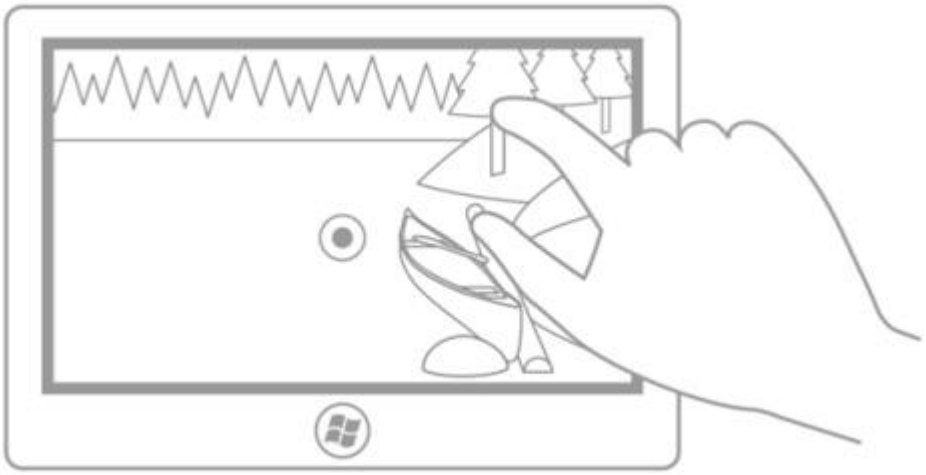


In this second picture, rotation is performed around the initial (thumb) touch point. After the rotation, the index finger is still touching the tree trunk and the thumb is still touching the log (the rotation point).

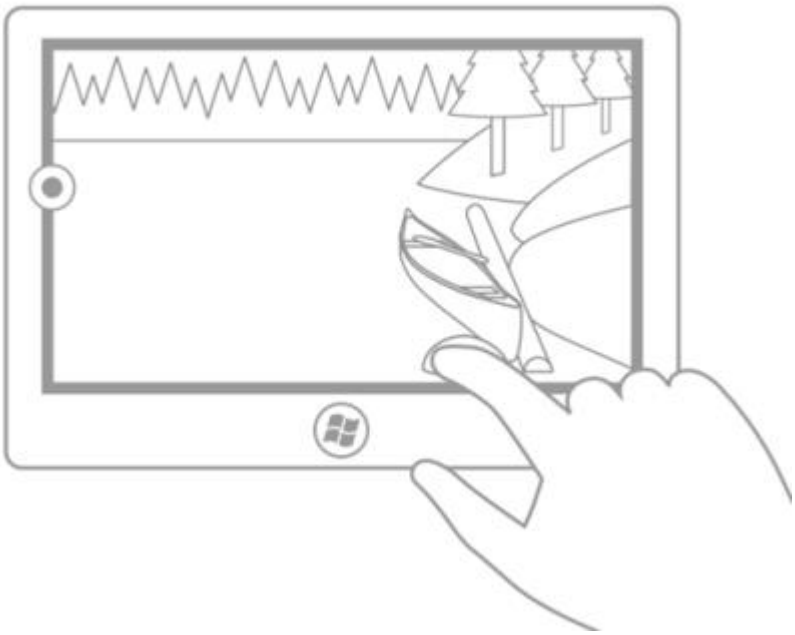


In this third picture, the center of rotation has been defined by the application (or set by the user) to be the center point of the picture. After the rotation, because the picture did not rotate around one of the fingers, the illusion of direct manipulation is broken (unless the user has chosen this setting).

© 2014 Microsoft. All rights reserved. | May 13, 2014



In this last picture, the center of rotation has been defined by the application (or set by the user) to be a point in the middle of the left edge of the picture. Again, unless the user has chosen this setting, the illusion of direct manipulation is broken in this case.



Windows 8 supports three types of rotation: free, constrained, and combined.

Type	Description
Free rotation	Free rotation enables a user to rotate content freely anywhere in a 360 degree arc. When the user releases the object, the object remains in the chosen position. Free rotation is useful for drawing and layout applications such as Microsoft PowerPoint, Word, Visio, and Paint; and Adobe Photoshop, Illustrator, and Flash.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Constrained rotation	Constrained rotation supports free rotation during the manipulation but enforces snap points at 90 degree increments (0, 90, 180, and 270) upon release. When the user releases the object, the object automatically rotates to the nearest snap point. Constrained rotation is the most common method of rotation, and it functions in a similar way to scrolling content. Snap points let a user be imprecise and still achieve their goal. Constrained rotation is useful for applications such as web browsers and photo albums.
Combined rotation	Combined rotation supports free rotation with zones (similar to rails in panning) at each of the 90 degree snap points enforced by constrained rotation. If the user releases the object outside of one of 90 degree zones, the object remains in that position; otherwise, the object automatically rotates to a snap point. **Note:** A user interface rail is a feature in which an area around a target constrains movement towards some specific value or location to influence its selection.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for selecting text and images (Windows Runtime apps)

This topic describes the new Windows UI for selecting and manipulating text, images, and controls and provides user experience guidelines that should be considered when using these new selection and manipulation mechanisms in your Windows Store app.

Dos and don'ts

- Use font glyphs when implementing your own gripper UI. The gripper is a combination of two Segoe UI fonts that are available system-wide. Using font resources simplifies rendering issues at different dpi and works well with the various UI scaling plateaus. When implementing your own grippers, they should share the following UI traits:
    - Circular shape
    - Visible against any background
    - Consistent size
- Provide a margin around the selectable content to accommodate the gripper UI. If your app enables text selection in a region that doesn't pan/scroll, allow a 1/2 gripper margin on the left and right sides of the text area and 1 gripper height on the top and bottom sides of the text area (as shown in the following images). This ensures that the entire gripper UI is exposed to the user and minimizes unintended interactions with other edge-based UI.



- Hide grippers UI during interaction. Eliminates occlusion by the grippers during the interaction. This is useful when a gripper isn't completely obscured by the finger or there are multiple text selection grippers. This eliminates visual artifacts when displaying child windows.
- Don't allow selection of UI elements such as controls, labels, images, proprietary content, and so on. Typically, Windows applications allow selection only within specific controls. Controls such as buttons, labels, and logos are not selectable. Windows Store apps using

© 2014 Microsoft. All rights reserved. | May 13, 2014

JavaScript require you to disable selection. Assess whether selection is an issue for your app and, if so, identify the areas of the UI where selection should be prohibited.

Additional usage guidance

Text selection and manipulation is particularly susceptible to user experience challenges introduced by touch interactions. Mouse, pen/stylus, and keyboard input are highly granular: a mouse click or pen/stylus contact is typically mapped to a single pixel, and a key is pressed or not pressed. Touch input is not granular; it's difficult to map the entire surface of a fingertip to a specific x-y location on the screen to place a text caret accurately.

Considerations and recommendations

Use the built-in controls exposed through the language frameworks in Windows 8 to build apps that provide the full platform user interaction experience, including selection and manipulation behaviors. You'll find the interaction functionality of the built-in controls sufficient for the majority of Windows Store apps.

When using standard Windows 8 text controls, the selection behaviors and visuals described in this topic cannot be customized.

Text selection

If your app requires a custom UI that supports text selection, we recommend that you follow the Windows 8 selection behaviors described here.

Editable and non-editable content

To accommodate the less precise targeting behavior of touch interactions, the selection and manipulation user experience visuals (such as "grippers" that both indicate a selection can be adjusted and identify the interaction target) have been completely redesigned for Windows 8.

With touch, selection interactions are performed primarily through gestures such as a tap to set an insertion cursor or select a word, and a slide to modify a selection. As with other Windows 8 touch interactions, timed interactions are limited to the press and hold gesture to display informational UI.

There are no changes to selection behavior when using mouse, pen/stylus, and keyboard in Windows 8.

Windows 8 recognizes two possible states for selection interactions, editable and non-editable, and adjusts selection UI, feedback, and functionality accordingly.

Editable content

© 2014 Microsoft. All rights reserved. | May 13, 2014

Tapping within the left half of a word places the cursor to the immediate left of the word, while tapping within the right half places the cursor to the immediate right of the word.

The following image demonstrates how to place an initial insertion cursor with gripper by tapping near the beginning or ending of a word.



The following image demonstrates how to adjust a selection by dragging the gripper.



© 2014 Microsoft. All rights reserved. | May 13, 2014

The following images demonstrate how to invoke the context menu by tapping within the selection or on a gripper (press and hold can also be used).

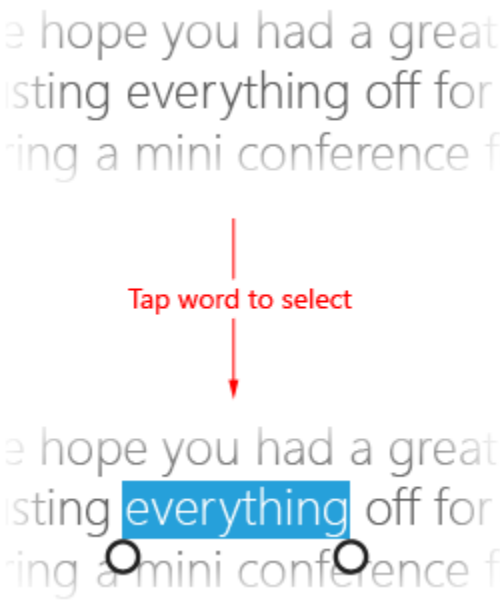


Note: These interactions vary somewhat in the case of a misspelled word. Tapping a word that is marked as misspelled will both highlight the entire word and invoke the suggested spelling context menu.

Non-editable content

The following image demonstrates how to select a word by tapping within the word (no spaces are included in the initial selection).

© 2014 Microsoft. All rights reserved. | May 13, 2014



Follow the same procedures as for editable text to adjust the selection and display the context menu.

Object manipulation

Wherever possible, use the same (or similar) gripper resources as text selection when implementing custom object manipulation in a Windows Store app. This helps provide a consistent interaction experience across the platform.

For example, grippers can also be used in image processing apps that support resizing and cropping or media player apps that provide adjustable progress bars, as shown in the following images.



Media player with adjustable progress bar.

© 2014 Microsoft. All rights reserved. | May 13, 2014



Image editor with cropping grippers.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Speech design guidelines (Windows Phone Store apps)



In Windows Phone users can interact with your app using speech. There are three speech components that you can integrate with your app: voice commands, speech recognition, and text-to-speech (also known as TTS, or speech synthesis).

Designed thoughtfully and implemented effectively, speech can be a robust and enjoyable way for people to interact with your Windows Phone app, complementing or even replacing interaction by touch, tap, and gestures.

Speech interaction design

Before you start coding a speech-enabled app, it's a good idea to envision and define the user experience and flow.

Using speech to enter your app

You can integrate voice commands into your app so users can deep link into your app, from outside of your app. For example, you could add voice commands that access the most frequently used sections of your app or perform important tasks.

Speech interaction inside your app

From inside your app, users can speak to give input or accomplish tasks by using speech recognition. Also while inside your app, you can use text-to-speech (TTS), also known as speech synthesis, to speak text to the user through the microphone.

Consider the following questions to help define speech interaction after users have opened your app (perhaps by using a voice command):

- What actions or app behavior can a user initiate using her voice, for example: navigate among pages, initiate commands, or enter data such as notes or messages?
- What phrases will users expect to say to initiate each action or behavior?
- How will users know when they can speak to the app and what they can say?
- What processes or tasks may be quicker using speech rather than touch? For example, browsing large lists of options or navigating several menu levels or pages.
- Will your app be used for speech recognition in the absence of network connectivity?

© 2014 Microsoft. All rights reserved. | May 13, 2014

- Does you app target specific user groups that may have custom vocabulary requirements, for example, specific business disciplines such as medicine or science, or gamers, or specific geographic regions?

When you have made some decisions about the speech interaction experience either through voice commands and/or with in-app speech, you'll be able to:

- List the actions a user can take with your app.
- Map each action to a command.
- Assign one or more phrases that a user can speak to activate each command.
- Write out the dialog that the user and your app will engage in, if your app will also speak to users.

Implementing speech design

Voice commands

To enable voice commands, you'll need to define a list of recognizable phrases and map them to commands in a Voice Command Definition (VCD) file. For text-to-speech readout that accompanies voice commands, you specify the string in the VCD file that the speech synthesizer will speak to confirm the action being taken. Here are some tips to keep in mind when creating a VCD file for your app:

- Prompts a user for input.
- Displays a readout of the audio level of speech input.
- Confirms the phrase that was matched to a user's speech.
- Informs a user that recognition was not successful and lets the user try again (repeatedly if necessary).
- Helps the user choose from among multiple recognition possibilities (if they exist).

Because the built-in recognition experience in Windows Phone leverages the same interactive model used in global speech contexts on the phone, users are more likely to know when to start speaking, to have familiarity with the built-in sounds, to know when processing is complete, and to receive feedback on errors and help with disambiguation when there are multiple match possibilities. See Presenting prompts, confirmations, and disambiguation choices for Windows Phone 8 for more info.

Prompts and confirmations

Provide your users with UI to initiate recognition. A good way to do this is with an app bar button whose icon property is set to "Microphone".

Let users know what they can say to your app based on the current app context, and give users an example of an expected input phrase. Unless you want a user to be able to say anything at all, for example, when inputting short-message dictation, strive to make your prompt elicit as specific a response as possible. For example, if you prompt the user with "What do you want to do today?", the range of responses could vary widely and it could take quite a large grammar to match the

© 2014 Microsoft. All rights reserved.

possible responses. However, if the prompt says "Would you like to play a game or listen to music?", then the prompt specifically requests one of two responses "play a game" or "listen to music". The grammar needed to match only the two responses would be more simple to author and would likely perform recognition more accurately than a much larger grammar.

Request confirmation from the user when speech recognition confidence is low. If the user's intent is unclear, it's usually better to prompt the user for clarification than for your app to initiate an action that the user didn't intend.

The built-in recognition experience includes screens that you can customize with prompt text and an example of expected speech input, as well as screens that confirm speech input.

Plan for recognition failures

Plan for what to do if recognition fails. For example, if there is no input, the recognition quality is poor, or only part of a phrase is recognized, your app logic should handle those cases. Consider informing the user that your app didn't understand her, and that she can try again. Give the user another example of an expected input phrase and restart recognition when necessary to allow additional input. If there are multiple successive failed recognition attempts, consider letting the user either key in text or exit the recognition operation. The built-in UI recognition experience includes screens that let a user know that recognition was not successful and allow the user to speak again to make another recognition attempt.

Listen for and try to correct issues in the audio input. The speech recognizer raises an event when it detects an issue in the audio input that may adversely affect speech recognition accuracy. You can use the information in the event arguments to inform the user about the issue, so she can take corrective action if possible. For example, if speech input is too quiet, you can prompt the user to speak louder. The speech recognition engine generates this event continuously whether or not you use the built-in speech recognition experience.

Constraints

A grammar is a kind of constraint that defines the set of phrases that a speech recognition engine can use to match speech input. You can either provide the speech recognition engine with the predefined grammars that are included with Windows Phone, or with custom grammars that you create. This section gives an overview of the types of grammars you can use and provides tips for authoring SRGS grammars. Also see Grammars for Windows Phone 8 for more info about different grammar types and when to use them.

Predefined grammars

Windows Phone supports two predefined grammars. To match a large number of phrases that a user might speak in a given language, consider using the pre-defined short-message dictation grammar. To match input that is in the context of a web query, consider using the predefined web search grammar. These online predefined grammars can be used as-is to recognize up to 10 seconds of speech audio and require no authoring effort on your part, but they do require connection to a network at run time because they are online.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Creating custom grammars

If you author your own grammars, a list constraint works well for recognizing short distinct phrases. These phrases can be programmatically updated and used for speech recognition when there is no app connectivity to the network.

For the greatest control over the speech recognition experience, author your own SRGS grammar, which is a particularly powerful method if you want to capture multiple semantic meanings in a single recognition. SRGS grammars can also be used for offline speech recognition.

Tips for authoring SRGS grammars

Keep your grammar small. A grammars that contains fewer phrases to be matched tends to provide better recognition accuracy than a larger grammar containing many phrases. It's generally preferable to have several smaller grammars for specific scenarios in your app than to have one grammar for your entire app. Prepare users for what to say in each app context and enable and disable grammars as needed so the speech recognition engine needs to search only a small body of phrases to match speech input for each recognition scenario.

Design your grammars to allow users to speak a command in a variety of ways, and to account for variations in the way people think and speak. For example in SRGS grammars, you can use the GARBAGE rule as follows:

- Match speech input that your grammar does not define. This will allow users to speak additional words that have no meaning for your app, for example "give me", "and", "uh", "maybe", and so forth, yet still be successfully recognized for the words that matter to your app, which you have explicitly defined in your grammars.
- Add the GARBAGE rule as an item in a list of alternatives to reduce the likelihood that words not defined in your grammar are recognized by mistake. Also, if the speech recognition engine matches unexpected speech input to a GARBAGE rule in a list of alternatives, you can detect the ellipsis (…) returned by the match to GARBAGE in the recognition result and prompt the user to speak again. However, using the GARBAGE rule in a list of alternatives may also increase the likelihood that speech input which matches phrases defined in your grammar is falsely rejected.

Use the GARBAGE rule with care and test to make sure that your grammar performs as you intend. See ruleref Element for more info.

Try using the sapi:subset element to help match speech input. The sapi:subset element is a Microsoft extension to the SRGS specification that can help match a user's speech input to enabled grammars. Phrases that you define using sapi:subset can be matched by the speech recognition engine even if only a part of the phrase is given in the speech input. You can define the subset of the phrase that can be used for matching in one of four ways.

Try to avoid defining phrases in your grammar that contain only one syllable. Recognition tends to be more accurate for phrases containing two or more syllables, although you should avoid defining phrases that are longer than they need to be.

© 2014 Microsoft. All rights reserved. | May 13, 2014

When you define a list of alternative phrases, avoid using phrases that sound similar, which the speech recognition engine may confuse. For example, including similar sounding phrases such as "hello", "bellow", and "fellow" in a list of alternatives may result in poor recognition accuracy.

Compiling a grammar is a necessary step in getting ready for speech recognition.

Custom pronunciations

Consider providing custom pronunciations for specialized vocabulary. If your app contains unusual or fictional words or words with uncommon pronunciations, you may be able to improve recognition performance for those words by defining custom pronunciations. Although the speech recognition engine is designed to generate pronunciations on the fly for words that are not defined in its dictionary, you may be able to improve the accuracy of both speech recognition and text-to-speech (TTS) by defining custom pronunciations. For fewer or infrequently used words, you can create custom pronunciations inline in SRGS grammars. See token Element for more info. For more or frequently used words, you may want to create separate pronunciation lexicon documents. See About Lexicons and Phonetic Alphabets for more info.

Test speech recognition accuracy

Test speech recognition accuracy and the effectiveness of any custom UI you provide to support speech recognition with your app, preferably with a target user group for your app. Testing the speech recognition accuracy of your app with target users is the best way to determine the effectiveness of the design and implementation of speech in your app. For example, if users are getting poor recognition results, are they trying to say something that your implementation isn't listening for? One solution would be to modify the grammar to support what users expect to say, but another solution would be to change your app to let users know what they can say prior to the speech interaction. Test results can help you to discover ways to improve your grammars or the speech recognition flow of your app to enhance its effectiveness.

Text-to-speech

Also known as speech synthesis, text-to-speech (TTS) generates speech output from text or Speech Synthesis Markup Language (SSML) XML markup that you provide. The following are some suggestions for implementing text-to-speech (TTS) in your app.

- Design prompts to be polite and encouraging.
- Think about whether you want to use text-to-speech (TTS) to read back to the user large bodies of text. For example, users may be inclined to wait while TTS reads back a text message, but may get impatient or disoriented listening to a long list of search results that will be difficult to memorize.
- Give users the option to stop text-to-speech (TTS) readout, particularly for longer readouts.
- Consider giving users a choice for a male and female voice for text-to-speech (TTS). All languages on Windows Phone have both a male and a female voice for each supported locale.
- Listen to the text-to-speech (TTS) readout before you ship an app. The speech synthesizer attempts to read out phrases in intelligible and natural ways, however, sometimes there may be an issue with either intelligibility or naturalness.

© 2014 Microsoft. All rights reserved.

- Intelligibility is most important and reflects whether a native speaker can understand the word or phrase being spoken by text-to-speech (TTS). Sometimes an intelligibility issue can arise with an infrequent pattern of language being strung together or where part numbers or punctuation are a factor.
- Naturalness is desired and can result when the prosody or cadence of a readout is different from how a native speaker would say the phrase. Both kinds of issues can be addressed by using SSML instead of plain text as input to the synthesizer. For more info about SSML, see Use SSML to Control Synthesized Speech and Speech Synthesis Markup Language Reference.

Related topics

Speech for Windows Phone 8

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for targeting



Touch targeting in Windows uses the full contact area of each finger that is detected by a touch digitizer. The larger, more complex set of input data reported by the digitizer is used to increase precision when determining the user's intended (or most likely) target.

This topic describes the use of contact geometry for touch targeting and provides best practices for targeting in Windows Runtime apps.

Measurements and scaling

To remain consistent across different screen sizes and pixel densities, all target sizes are represented in physical units (millimeters). Physical units can be converted to pixels by using the following equation:

Pixels = Pixel Density × Measurement

The following example uses this formula to calculate the pixel size of a 9 mm target on a 135 pixel per inch (PPI) display at a 1x scaling plateau:

Pixels = 135 PPI × 9 mm

Pixels = 135 PPI × (0.03937 inches per mm × 9 mm)

Pixels = 135 PPI × 0.35433 inches

Pixels = 48 pixels

This result must be adjusted according to each scaling plateau defined by the system.

Thresholds

Distance and time thresholds may be used to determine the outcome of an interaction.

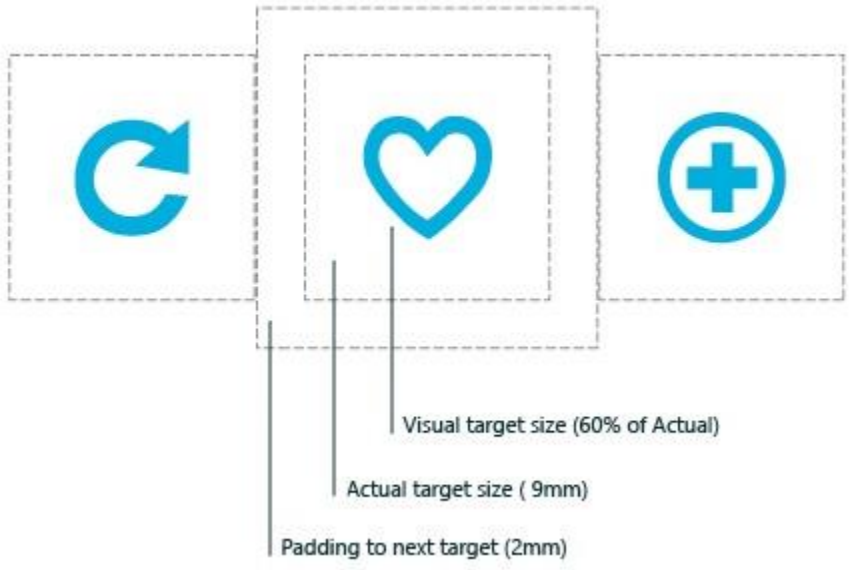
For example, when a touch-down is detected, a tap is registered if the object is dragged less than 2.7 mm from the touch-down point and the touch is lifted within 0.1 second or less of the touch-down. Moving the finger beyond this 2.7 mm threshold results in the object being dragged and either selected or moved. Depending on your app, holding the finger down for longer than 0.1 second may cause the system to perform a self-revealing interaction.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Target sizes

In general, set your touch target size to 9 mm square or greater (48x48 pixels on a 135 PPI display at a 1.0x scaling plateau). Avoid using touch targets that are less than 7 mm square.

The following diagram shows how target size is typically a combination of a visual target, actual target size, and any padding between the actual target and other potential targets.



The following table lists the minimum and recommended sizes for the components of a touch target.

Target component	Minimum size	Recommended size
Padding	2 mm	Not applicable.
Visual target size	< 60% of actual size	90-100% of actual size Most users won't realize a visual target is touchable if it's less than 4.2 mm square (60% of the recommended minimum target size of 7 mm).
Actual target size	7 mm square	Greater than or equal to 9 mm square (48 x 48 px @ 1x)
Total target size	11 x 11 mm (approximately 60 px: three 20-px grid units @ 1x)	13.5 x 13.5 mm (72 x 72 px @ 1x) This implies that the size of the actual target and padding combined should be larger than their respective minimums.

© 2014 Microsoft. All rights reserved. | May 13, 2014

These target size recommendations can be adjusted as required by your particular scenario. Some of the considerations that went into these recommendations include:

- Frequency of Touches: Consider making targets that are repeatedly or frequently pressed larger than the minimum size.
- Error Consequence: Targets that have severe consequences if touched in error should have greater padding and be placed further from the edge of the content area. This is especially true for targets that are touched frequently.
- Position in the content area
- Form factor and screen size
- Finger posture
- Touch visualizations
- Hardware and touch digitizers

Targeting assistance

Windows provides targeting assistance to support scenarios where the minimum size or padding recommendations presented here are not applicable; for example, hyperlinks on a webpage, calendar controls, drop down lists and combo boxes, or text selection.

These targeting platform improvements and user interface behaviors work together with visual feedback (disambiguation UI) to improve user accuracy and confidence.

If a touchable element must be smaller than the recommended minimum target size, the following techniques can be used to minimize the targeting issues that result.

Tethering

Tethering is a visual cue (a connector from a contact point to the bounding rectangle of an object) used to indicate to a user that they are connected to, and interacting with, an object even though the input contact isn't directly in contact with the object. This can occur when:

- A touch contact was first detected within some proximity threshold to an object and this object was identified as the most likely target of the contact.
- A touch contact was moved off an object but the contact is still within a proximity threshold.

This feature is not exposed to Windows Store app using JavaScript developers.

Scrubbing

Scrubbing means to touch anywhere within a field of targets and slide to select the desired target without lifting the finger until it is over the desired target. This is also referred to as "take-off activation", where the object that is activated is the one that was last touched when the finger was lifted from the screen.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Use the following guidelines when you design scrubbing interactions:

- Scrubbing is used in conjunction with disambiguation UI.
- The recommended minimum size for a scrubbing touch target is 20 px (3.75 mm @ 1x size).
- Scrubbing takes precedence when performed on a pannable surface, such as a webpage.
- Scrubbing targets should be close together.
- An action is canceled when the user drags a finger off a scrubbing target.
- Tethering to a scrubbing target is specified if the actions performed by the target are non-destructive, such as switching between dates on a calendar.
- Tethering is specified in a single direction, horizontally or vertically.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for touch interactions



Description

Create Windows Store apps with intuitive and distinctive user interaction experiences that are optimized for touch but functionally consistent across input devices.

Take advantage of the compelling interaction capabilities of Windows. These guidelines will help you.

Dos and don'ts

- Design applications with touch interaction as the primary expected input method.
- Provide visual feedback for interactions of all types (touch, pen, stylus, mouse, etc.)
- Optimize targeting by adjusting touch target size, contact geometry, scrubbing and rocking.
- Optimize accuracy through the use of snap points and directional "rails".
- Provide tooltips and handles to help improve touch accuracy for tightly packed UI items.
- Don't use timed interactions whenever possible (example of appropriate use: touch and hold).
- Don't use the number of fingers used to distinguish the manipulation whenever possible.

Additional usage guidance

First and foremost, design your app with the expectation that touch will be the primary input method of your users. If you use the platform controls, support for touchpad, mouse, and pen/stylus requires no additional programming, because Windows 8 provides this for free.

However, keep in mind that a UI optimized for touch is not always superior to a traditional UI. Both provide advantages and disadvantages that are unique to a technology and application. In the move to a touch-first UI, it is important to understand the core differences between touch (including touchpad), pen/stylus, mouse, and keyboard input. Do not take familiar input device properties and behaviors for granted, as touch in Windows 8 does more than simply emulate that functionality.

You will find throughout these guidelines that touch input requires a different approach to UI design.

Compare touch interaction requirements

The following table shows some of the differences between input devices that you should consider when you design touch-optimized Windows Store apps.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Factor	Touch interactions	Mouse, keyboard, pen/stylus interactions	Touchpad
Precision	The contact area of a fingertip is greater than a single x-y coordinate, which increases the chances of unintended command activations.	The mouse and pen/stylus supply a precise x-y coordinate.	Same as mouse.
	The shape of the contact area changes throughout the movement.	Mouse movements and pen/stylus strokes supply precise x-y coordinates. Keyboard focus is explicit.	Same as mouse.
	There is no mouse cursor to assist with targeting.	The mouse cursor, pen/stylus cursor, and keyboard focus all assist with targeting.	Same as mouse.
Human anatomy	Fingertip movements are imprecise, because a straight-line motion with one or more fingers is difficult. This is due to the curvature of hand joints and the number of joints involved in the motion.	It's easier to perform a straight-line motion with the mouse or pen/stylus because the hand that controls them travels a shorter physical distance than the cursor on the screen.	Same as mouse.
	Some areas on the touch surface of a display device can be difficult to reach due to finger posture and the user's grip on the device.	The mouse and pen/stylus can reach any part of the screen while any control should be accessible by the keyboard through tab order.	Finger posture and grip can be an issue.
	Objects might be obscured by one or more fingertips or the user's hand. This is known as occlusion.	Indirect input devices do not cause occlusion.	Same as mouse.
Object state	Touch uses a two-state model: the touch surface of a display device is either touched (on) or not (off). There is no hover state that can trigger additional visual feedback.	A mouse, pen/stylus, and keyboard all expose a three-state model: up (off), down (on), and hover (focus). Hover lets users explore and learn through tooltips associated with UI elements. Hover and focus effects can relay which objects are interactive and also help with targeting.	Same as mouse.
Rich interaction	Supports multi-touch: multiple input points (fingertips) on a touch surface.	Supports a single input point.	Same as touch.
	Supports direct manipulation of objects through gestures such as	No support for direct manipulation as mouse,	Same as mouse.

© 2014 Microsoft. All rights reserved. | May 13, 2014

	tapping, dragging, sliding, pinching, and rotating.	pen/stylus, and keyboard are indirect input devices.	

Note: Indirect input has had the benefit of more than 25 years of refinement. Features such as hover-triggered tooltips have been designed to solve UI exploration specifically for touchpad, mouse, pen/stylus, and keyboard input. UI features like this have been re-designed for the rich experience provided by touch input, without compromising the user experience for these other devices.

Use touch feedback

Appropriate visual feedback during interactions with your app helps users recognize, learn, and adapt to how their interactions are interpreted by both the app and Windows 8. Visual feedback can indicate successful interactions, relay system status, improve the sense of control, reduce errors, help users understand the system and input device, and encourage interaction.

Visual feedback is critical when the user relies on touch input for activities that require accuracy and precision based on location. Display feedback whenever and wherever touch input is detected, to help the user understand any custom targeting rules that are defined by your app and its controls.

Create an immersive interaction experience

The following techniques enhance the immersive experience of Windows Store apps.

Targeting

Targeting is optimized through:

- Touch target sizes

  Clear size guidelines ensure that applications provide a comfortable UI that contains objects and controls that are easy and safe to target.

- Contact geometry

  The entire contact area of the finger determines the most likely target object.

- Scrubbing

  Items within a group are easily re-targeted by dragging the finger between them (for example, radio buttons). The current item is activated when the touch is released.

- Rocking

© 2014 Microsoft. All rights reserved. | May 13, 2014

Densely packed items (for example, hyperlinks) are easily re-targeted by pressing the finger down and, without sliding, rocking it back and forth over the items. Due to occlusion, the current item is identified through a tooltip or the status bar and is activated when the touch is released.

Accuracy

Design for sloppy interactions by using:

- Snap-points that can make it easier to stop at desired locations when users interact with content.
- Directional "rails" that can assist with vertical or horizontal panning, even when the hand moves in a slight arc.

Occlusion

Finger and hand occlusion is avoided through:

- Size and positioning of UI

  Make UI elements big enough so that they cannot be completely covered by a fingertip contact area.

  Position menus and pop-ups above the contact area whenever possible.

- Tooltips

  Show tooltips when a user maintains finger contact on an object. This is useful for describing object functionality. The user can drag the fingertip off the object to avoid invoking the tooltip.

  For small objects, offset tooltips so they are not covered by the fingertip contact area. This is helpful for targeting.

- Handles for precision

  Where precision is required (for example, text selection), provide selection handles that are offset to improve accuracy.

Timing

Avoid timed mode changes in favor of direct manipulation. Direct manipulation simulates the direct, real-time physical handling of an object. The object responds as the fingers are moved.

A timed interaction, on the other hand, occurs after a touch interaction. Timed interactions typically depend on invisible thresholds like time, distance, or speed to determine what command to perform. Timed interactions have no visual feedback until the system performs the action.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Direct manipulation provides a number of benefits over timed interactions:

- Instant visual feedback during interactions make users feel more engaged, confident, and in control.
- Direct manipulations make it safer to explore a system because they are reversible—users can easily step back through their actions in a logical and intuitive manner.
- Interactions that directly affect objects and mimic real world interactions are more intuitive, discoverable, and memorable. They don't rely on obscure or abstract interactions.
- Timed interactions can be difficult to perform, as users must reach arbitrary and invisible thresholds.

In addition, the following are strongly recommended:

- Manipulations should not be distinguished by the number of fingers used.
- Interactions should support compound manipulations. For example, pinch to zoom while dragging the fingers to pan.
- Interactions should not be distinguished by time. The same interaction should have the same outcome regardless of the time taken to perform it. Time-based activations introduce mandatory delays for users and detract from both the immersive nature of direct manipulation and the perception of system responsiveness.

Note: An exception to this is where you use specific timed interactions to assist in learning and exploration (for example, press and hold).

- Appropriate descriptions and visual cues have a great effect on the use of advanced interactions.

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for touch keyboard

The touch keyboard enables text entry for Windows Store apps on devices that support touch and don’t have a built-in or peripheral keyboard. The touch keyboard is invoked when a user taps on an editable input field, and is dismissed when the input field loses focus. The touch keyboard is used for text entry only and doesn't contain command keys, such as the alt or function keys.



Dos and don'ts

- Display the touch keyboard for text input only. Note that the touch keyboard doesn't provide many of the accelerators or command keys found on a hardware keyboard, such as alt, the function keys, or the Windows Logo key.
- Don't redefine input from the touch keyboard into commands or shortcuts.
- Don't make users navigate the app using the touch keyboard.

Touch keyboard and a custom UI

These recommendations are only relevant if your app uses custom controls. Text input controls provided by Windows properly interact with the touch keyboard by default. If you're creating a custom UI using C#/VB/C++ and XAML, use the **AutomationPeer class** to access UI Automation controls. The Input: Touch keyboard sample shows how to launch the touch keyboard using custom controls in apps using XAML.

If you're using JavaScript and HTML, you'll have to access UI Automation by setting the Accessible Rich Internet Applications (ARIA) properties for your custom controls.

- Display the keyboard throughout the entire interaction with your form.

  Ensure that your custom controls have the proper UI Automation ControlType to ensure keyboard persistence when focus moves from a text input field while in the context of text entry. For example, if you have a menu that's opened in the middle of a text-entry scenario, and you want the keyboard to persist, the menu must have the ControlType Menu. For apps using C#/VB/C++ and XAML, see the **AutomationControlType enumeration**. For JavaScript and HTML, set the ARIA role to the appropriate control type.

© 2014 Microsoft. All rights reserved. | May 13, 2014

- Ensure that users can always see the input field that they're typing into.

  The touch keyboard occludes half of the screen. Windows Store apps provide a default experience for managing UI when the touch keyboard appears, by ensuring that the input field with focus scrolls into view. When customizing your UI, handle the **Showing** and **Hiding** events exposed by the **InputPane** object to customize your app's reaction to the keyboard's appearance.

- Don't manipulate UI Automation properties to control the touch keyboard. Other accessibility tools rely on the accuracy of UI Automation properties.
- Implement UI Automation properties for custom controls that have text input. For the keyboard to persist contextually as focus changes to different controls, a custom control must have one of the following properties:
  - Button
  - Check box
  - Combo box
  - Radio button
  - Scroll bar
  - Tree item
  - Menu
  - Menu item

  For apps using C#/VB/C++ and XAML, see the **AutomationControlType enumeration**. For JavaScript and HTML, set the ARIA role to the appropriate control type.

Additional usage guidance

Handling the Showing and Hiding events

Here's an example of attaching event handlers for the **showing** and **hiding** events of the touch keyboard.

JavaScript

```
<SCRIPT type="text/javascript">
    Windows.UI.Immersive.InputPane.getForCurrentView().addEventListener("showing",
onInputPaneShowing);
    Windows.UI.Immersive.InputPane.getForCurrentView().addEventListener("hiding",
onInputPaneHiding);

// Handle the showing event.
function onInputPaneShowing(e)
{
    var occludedRect = e.occludedRect;

    // For this hypothetical application, the developer decided that 400 pixels is
    // the minimum height that will work for the current layout. When the
    // app gets the InputPaneShowing message, the pane is beginning to animate in.

    if (occludedRect.Top < 400)
```

© 2014 Microsoft. All rights reserved.

```
    {
        // In this scenario, the developer decides to remove some elements (perhaps
        // a fixed navbar) and dim the screen to give focus to the text element.
        var elementsToRemove = document.getElementsByName("extraneousElements");

        // The app developer is not using default framework animation.
        _StartElementRemovalAnimations(elementsToRemove);
        _StartScreenDimAnimation();
    }

    // This developer doesn't want the framework's focused element visibility
    // code/animation to override the custom logic.
    e.ensuredFocusedElementInView = true;
}

// Handle the hiding event.
function onInputPaneHiding(e)
{
    // In this case, the Input Pane is dismissing. The developer can use
    // this message to start animations.
    if (_ExtraElementsWereRemoved())
    {
        _StartElementAdditionAnimations();
    }

    // Don't use framework scroll- or visibility-related
    // animations that might conflict with the app's logic.
    e.ensuredFocusedElementInView = true;
}
```

C#

```
public class MyApplication
{
    public MyApplication()
    {
        // Grab the input pane for the main application window and attach
        // touch keyboard event handlers.
        Windows.Foundation.Application.InputPane.GetForCurrentView().Showing
            += new EventHandler(_OnInputPaneShowing);
        Windows.Foundation.Application.InputPane.GetForCurrentView().Hiding
            += new EventHandler(_OnInputPaneHiding);
    }

    private void _OnInputPaneShowing(object sender, IInputPaneVisibilityEventArgs
eventArgs)
    {
        // If the size of this window is going to be too small, the app uses
        // the Showing event to begin some element removal animations.
        if (eventArgs.OccludedRect.Top < 400)
        {
            _StartElementRemovalAnimations();

            // Don't use framework scroll- or visibility-related
            // animations that might conflict with the app's logic.
            eventArgs.EnsuredFocusedElementInView = true;
        }
    }

    private void _OnInputPaneHiding(object sender, IInputPaneVisibilityEventArgs
eventArgs)
```

© 2014 Microsoft. All rights reserved. | May 13, 2014

```
    {
        if (_ResetToDefaultElements())
        {
            eventArgs.EnsuredFocusedElementInView = true;
        }
    }

    private void _StartElementRemovalAnimations()
    {
        // This function starts the process of removing elements
        // and starting the animation.
    }

    private void _ResetToDefaultElements()
    {
        // This function resets the window's elements to their default state.
    }
}
```

© 2014 Microsoft. All rights reserved. | May 13, 2014

Guidelines for visual feedback



Description

Use visual feedback to show users when their interactions with a Windows Store app are detected, interpreted, and handled. Visual feedback can help users by encouraging interaction. It indicates the success of an interaction, which improves the user's sense of control. It also relays system status and reduces errors.

Dos and don'ts

- Provide visual feedback no matter how brief the contact. This helps the user to:
  - Confirm that the touch screen is working.
  - Identify whether the target is touch-enabled or responsive.
  - Identify whether the user missed their intended target.
- Display feedback immediately for all interaction events.
- Provide feedback that consists of subtle, intuitive cues that don't distract users.
- Ensure touch targets stick to the fingertip during all manipulations.
- Enable selection of items with the swipe gesture when panning is constrained to one direction.
- Don't use touch visualizations in situations where they might interfere with the use of the app. For more info, see **ShowGestureFeedback**.
- Don't display feedback unless it is absolutely necessary. Keep the UI clean and uncluttered by not showing visual feedback unless you are adding value that is not available elsewhere. Never display tooltips if they repeat text that is already visible. Tooltips should be reserved for specific occasions, such as truncated text (text with ellipsis) that is not displayed when the item is selected, or where additional information is required to understand or use your app.
- Don't use the press and hold gesture for anything other than informational UI.

  Important: Press and hold can be used for selection in cases where both horizontal and vertical panning is enabled.

- Don't customize the visual feedback behaviors of the built-in Windows 8 gestures, as this can create an inconsistent and confusing user experience.
- Don't show visual feedback during panning or dragging; the actual movement of the object on the screen is sufficient. But if the content area doesn't pan or scroll, then use visualizations to indicate the boundary conditions.
- Don't display feedback for a control that isn't identified as the target. Visual feedback is critical when relying on touch input for activities that require accuracy and precision based on location. Displaying feedback every time you detect touch input helps the user understand any custom targeting heuristics defined by your app and its controls.

© 2014 Microsoft. All rights reserved. | May 13, 2014

- Don't use feedback behavior intended for one input type with another. For example, a keyboard focus rectangle should be used only with keyboard input, not touch.

Additional usage guidance

Contact visualizations are especially critical for touch interactions that require accuracy and precision. For example, your app should clearly indicate the location of a tap to let a user know if they missed their target, how much they missed it by, and what adjustments they must make.

Use the platform controls exposed through the Windows Store apps language frameworks (Windows Store apps using JavaScript and Windows Store apps using C++, C#, or Visual Basic) to get Windows 8 visualizations for free. If your app features custom interactions that require customized feedback, you should ensure the feedback is appropriate, spans input devices, and doesn't distract a user from their task. This can be a particular issue in game or drawing apps, where the visual feedback might conflict with or obscure critical UI.

Important

We don't recommend changing the interaction behavior of the built-in gestures.

Feedback UI

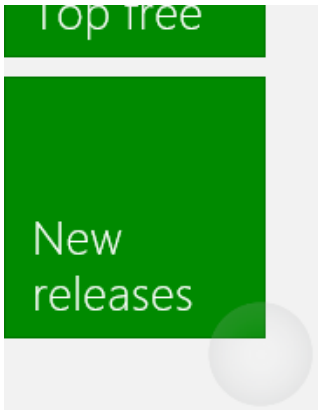
Feedback UI is generally dependent on the input device (touch, touchpad, mouse, pen/stylus, keyboard, and so on). For example, the built-in feedback for a mouse usually involves moving and changing the cursor, while touch and pen require contact visualizations, and keyboard input and navigation uses focus rectangles and highlighting.

Use **ShowGestureFeedback** to set feedback behavior for the platform gestures.

If customizing feedback UI, ensure you provide feedback that supports, and is suitable for, all input modes.

Here are some examples of built-in contact visualizations in Windows 8.

© 2014 Microsoft. All rights reserved. | May 13, 2014



Touch visualization



Mouse/touchpad visualization



Pen visualization



Keyboard visualization

Informational UI (popups)

One of the primary forms of visual feedback is informational UI (or disambiguation UI). Informational UI identifies and displays info about an object, describes functionality and how to access it, and provides guidance where necessary.

Here are the different types of informational UI supported by Windows Store apps.

- Tooltips
- Rich tooltips
- Menus
- Message dialogs
- Flyouts

Informational UI is particularly useful for overcoming fingertip occlusion (obstruction) and improving touch interactions with your app. It even has a built-in gesture dedicated to it: press and hold.

Press and hold is a timed interaction, which are typically discouraged in Windows 8. A timed interaction is acceptable in this case as it is used as a tool for learning and exploration. The recommended length of time depends on the type of informational UI. Here are the recommended time thresholds.

Informational UI type	Timing	Activation	Usage
Occlusion tooltip (for scrubbing and small targets)	0 ms	Yes	For rapid clarification of actions. Typically used for commands.
Occlusion tooltip (for actions)	200 ms	Yes	

© 2014 Microsoft. All rights reserved. | May 13, 2014

Rich tooltip	~2000 ms	No	For slower, more deliberate exploration and learning. Typically used with collection items.
Self-revealing interaction	~2000 ms	No	
Context menu	~2000 ms	No	Exposes a limited set of commands related to the selected object.
Flyouts	~2000 ms	No	Exposes a limited set of commands related to the selected object.

Tooltips

Use tooltips to reveal more information about a control before asking the user to perform an action.

Tooltips appear automatically when a user performs a press and hold gesture (or a hover event is detected) on a control or object. The tooltip disappears when the contact or cursor moves off the control or object. A tooltip can include text and images, but is not interactive.

Occlusion tooltips for small targets

Occlusion tooltips describe the occluded target. These tooltips are useful when targeting and activating items smaller than a standard touch target size, such as hyperlinks on a webpage.

You can replace these tooltips with an informational pop-up after a certain time threshold has passed. For example, use an occlusion tooltip to show the occluded text of the hyperlink and then replace the tooltip with a pop-up containing the URL.

Occlusion tooltips for actions and commands

These tooltips describe the action or command that occurs when a user lifts their finger from an element. These tooltips are useful when targeting and activating a button or similar control.

A small-target tooltip can be followed by an action tooltip after a certain time threshold has passed. In this case, the small-target tooltip should expand to include the additional info in the action tooltip.

Rich tooltip

These tooltips reveal secondary info about an element. For example, a rich tooltip could be a text description of an image, the full text of a truncated title, or other info relevant to the target.

Rich tooltips typically contain info that doesn't need to be made available immediately and, in some cases, might be distracting if shown too quickly. A longer time threshold lets users be more deliberate about obtaining the info.

© 2014 Microsoft. All rights reserved. | May 13, 2014

After a rich tooltip is displayed, the object is no longer activated when the user lifts their finger. The reason for this is that info gleaned from the tooltip might influence the user to not to activate the item.

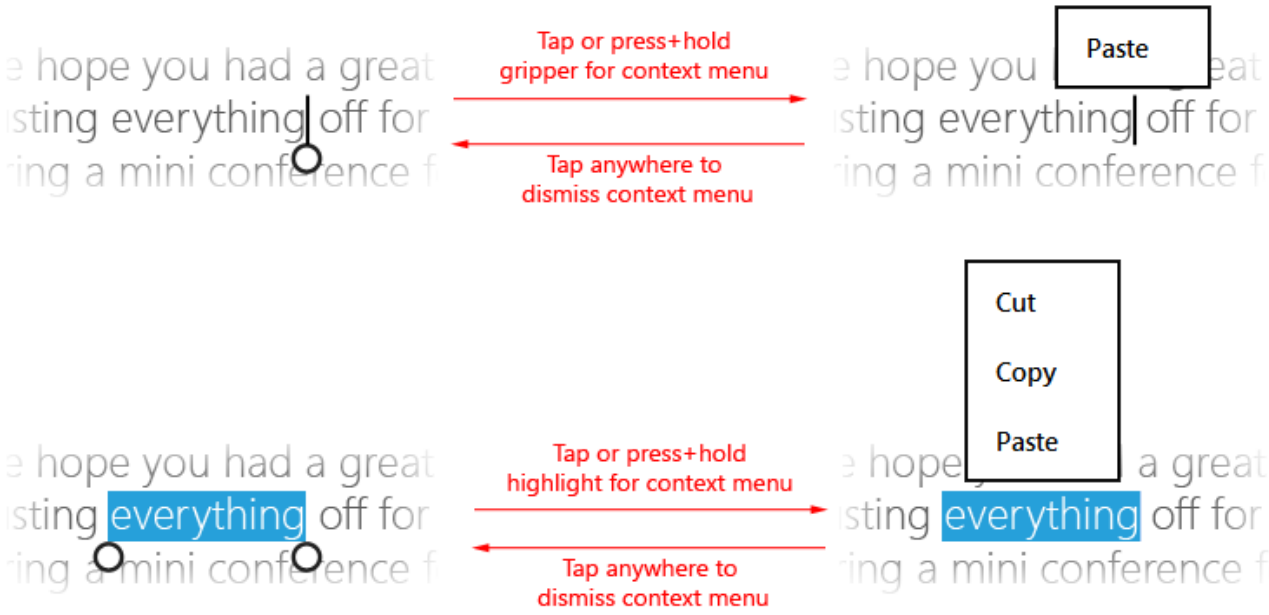
We recommend that the visual design and info in the rich tooltip be distinct and more substantial than that of a standard tooltip.

Context menu

The context menu (**PopupMenu**) is a lightweight menu that gives users immediate access to actions (like clipboard commands) on text or UI objects in Windows Store apps.

The touch-optimized context menu consists of two parts. A visual cue, the hint, is displayed as a result of a hold interaction. Then, the context menu itself is displayed after the hint disappears and the user lifts their finger.

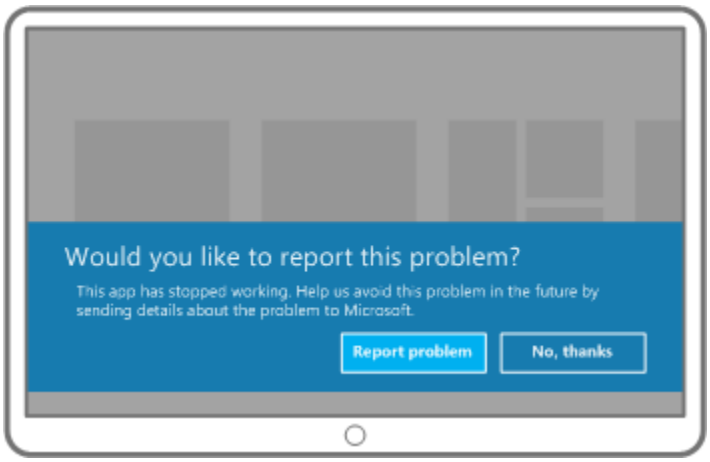
The following images demonstrate how to invoke the default context menu for text by tapping within a selection or on a gripper (press and hold can also be used).



Message dialog

Use message dialogs (**MessageDialog**) to prompt users for a response, based on user action or app state, before continuing. Explicit user interaction is required and input to the app is blocked until the user responds.

© 2014 Microsoft. All rights reserved. | May 13, 2014



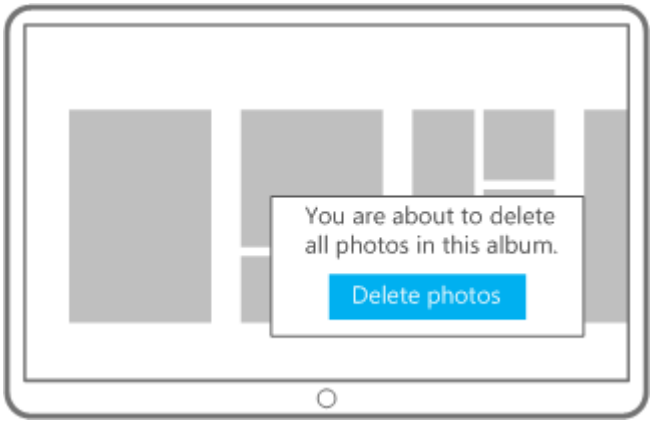
Here are some typical reasons to display a message dialog.

- Present urgent information
- Ask a question before continuing execution
- Display error messages

Flyout

A flyout (**Flyout**) is a lightweight UI panel displayed on a tap, click, or other activation that is used to present the user with information, questions, or a menu of options related to the current activity. It can be light dismissed (it disappears when the user touches or clicks outside the flyout panel or presses ESC). In other words, a flyout can be ignored.

Unlike tooltips, flyouts can accept input. Unlike message dialogs, the app is still active and accepting input.

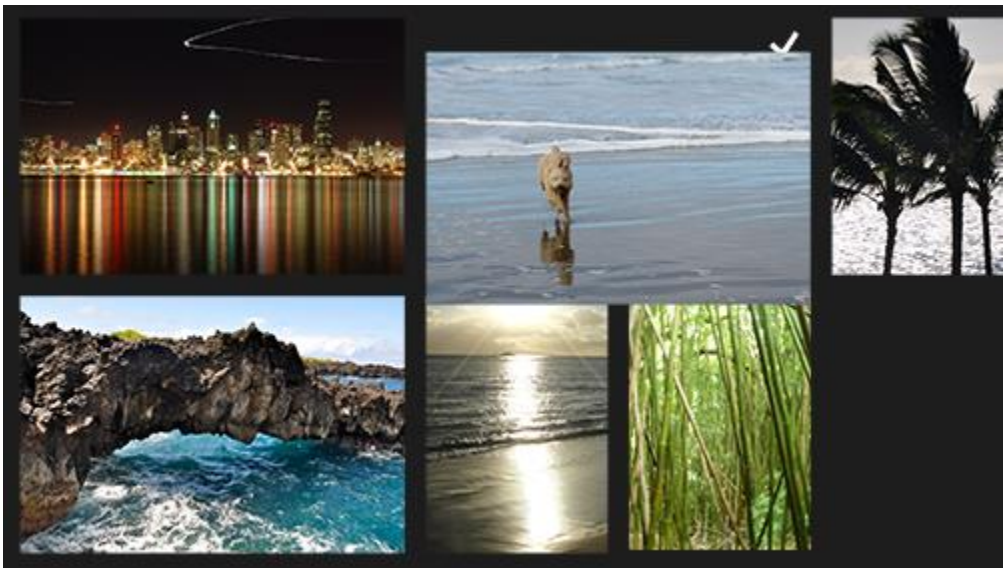


Self-revealing UI

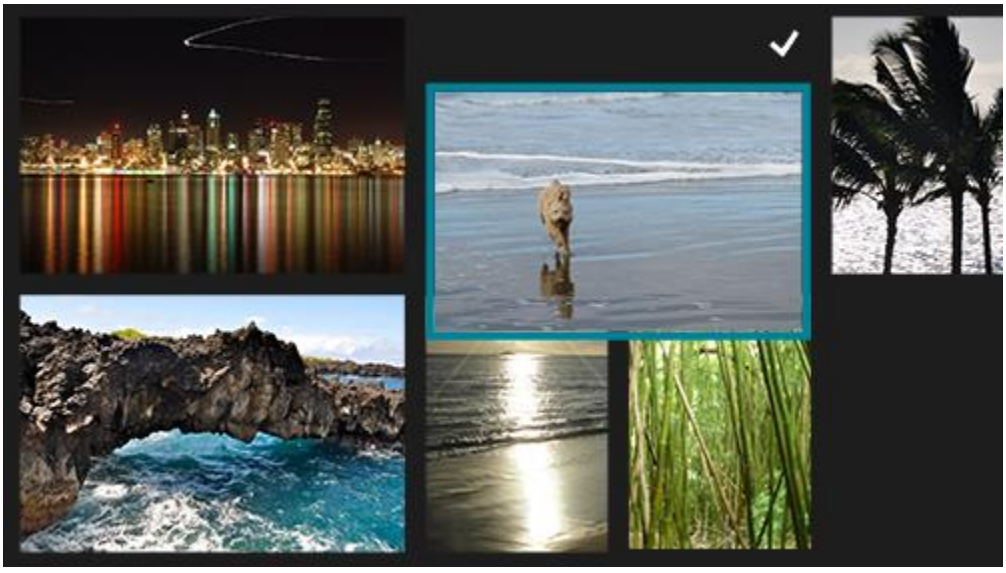
A self-revealing interaction is an informative visual cue or animation that demonstrates how to perform an action with a target object and provides a preview of the result of that action.

© 2014 Microsoft. All rights reserved. | May 13, 2014

These next few images show the self-revealing interaction for a cross-slide selection on the Start screen. When a user touches an app tile (without dragging the tile) the tile slides down (as if being dragged) to reveal the selection check mark that would appear if the app were actually selected.



Press finger down on an item to start the self-revealing interaction for selection. The self-revealing interaction demonstrates what action will be performed on the item.



Without lifting the finger, swipe to actually select the item.

© 2014 Microsoft. All rights reserved. | May 13, 2014



If the user continues to slide their finger, the self-revealing visualization changes to show that the object can now be moved.

After a self-revealing interaction is displayed, the object is no longer activated when the user lifts their finger.

© 2014 Microsoft. All rights reserved. | May 13, 2014