Tell us about your PDF experience.

Design basics for Desktop applications

Article • 02/09/2021

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

Use these articles to help you design the user experience for your Windows desktop apps:

- How to Design a Great User Experience

  Recommendations for achieving power and simplicity through carefully balanced feature selection and presentation.

- UX checklist

  Some common mistakes and inconsistencies to watch out for in your user interface design.

(For info on designing Universal Windows Platform (UWP) apps, see Design basics for UWP apps.)

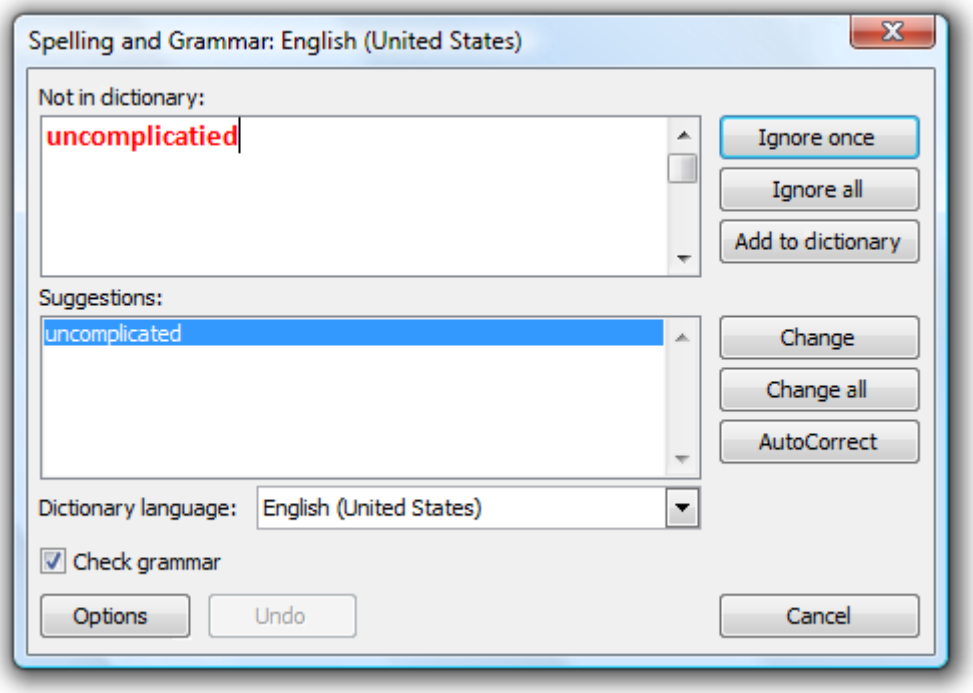
How to design a great user experience for desktop applications

Article • 02/09/2021

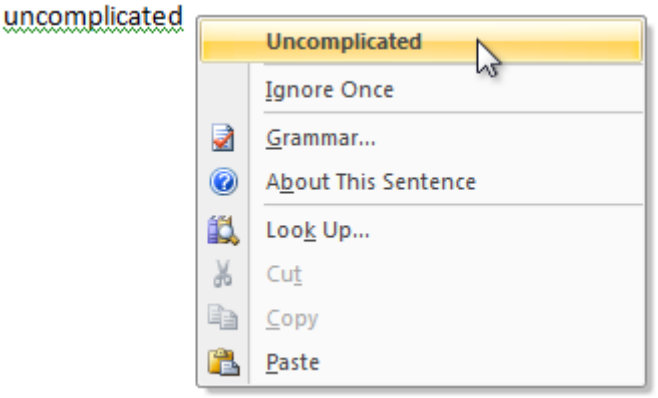
> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

A great desktop application is powerful and, at the same time, simple. Through carefully balanced feature selection and presentation, you can achieve both power and simplicity.

Powerful:



Powerful and simple:



The ideal Windows-based application is both powerful and simple. Of course you want your application to be powerful and of course you want it to be simple, but can you achieve both? There is a natural tension between these goals, but that tension is far from irreconcilable. You can achieve both power and simplicity through carefully balanced feature selection and presentation.

What makes an application powerful?

What does "power" really mean in terms of software? An application might be considered powerful if it is jam-packed full of features, having a tremendous breadth of functionality in an attempt to be all things to all users. Such a design is not likely to be successful because an untargeted feature set is unlikely to satisfy the needs of anyone. This is not the type of power that we are after.

An application is powerful when it has the right combination of these characteristics:

- **Enabling.** The application satisfies the needs of its target users, enabling them to perform tasks that they couldn't otherwise do and achieve their goals effectively.
- **Efficient.** The application enables users to perform tasks with a level of productivity and scale that wasn't possible before.
- **Versatile.** The application enables users to perform a wide range of tasks effectively in a variety of circumstances.
- **Direct.** The application feels like it is directly helping users achieve their goals, instead of getting in the way or requiring unnecessary steps. Features like shortcuts, keyboard access, and macros improve the sense of directness.
- **Flexible.** The application allows users complete, fine-grained control over their work.
- **Integrated.** The application is well integrated with Microsoft Windows, allowing it to share data with other applications.
- **Advanced.** The application has extraordinary, innovative, state-of-the-art features that are not found in competing solutions.

Some of these characteristics depend upon the perception of the user and are relative to users' current capabilities. What is considered powerful may change over time, so today's advanced search feature might be commonplace tomorrow.

All these characteristics can be combined into our definition of power:

An application is powerful when it enables its target users to realize their full potential efficiently.

Thus, the ultimate measure of power is productivity, not the number of features.

Different users need help in achieving their full potential in different ways. What is enabling to some users might harm versatility, directness, and control for others. Well-designed software must balance these characteristics appropriately. For example, a desktop publishing system designed for nonprofessionals might use wizards to walk users through complex tasks. Such wizards enable the target users to perform tasks that they otherwise wouldn't be able to perform. By contrast, a desktop publishing system for professionals might focus on directness, efficiency, and complete control. For users of such an application, wizards may be limiting and frustrating.

If you do only one thing...

Understand your target users' goals and craft a feature set that enables them to achieve those goals productively.

What makes a user experience simple?

We define simplicity as follows:

Simplicity is the reduction or elimination of an attribute of a design that target users are aware of and consider unessential.

In practice, simplicity is achieved by selecting the right feature set and presenting the features in the right way. This reduces the unessential, both real and perceived.

Simplicity is dependent upon the perception of users. Consider how the effect of an automatic transmission depends on a user's perspective:

- For the typical driver (the target user), an automatic transmission eliminates the need for a manual gear shift and clutch, making a car much easier to drive. A manual gear shift and clutch are not essential to the task of driving, so they are removed to achieve simplicity.
- For a professional race car driver, having direct control over the transmission is essential to being competitive. An automatic transmission negatively affects the

car's performance, so it is not regarded as resulting in simplicity.
- For a mechanic, an automatic transmission is a more complex mechanism, and therefore isn't easier to repair or maintain than a manual transmission. Unlike the mechanic, the target user is blissfully unaware of this internal complexity.

While different users regard the automatic transmission differently, it's successful because it eliminates the need for unessential knowledge, skill, and effort from its target users. For the typical driver, automatic transmission is a great feature because it just works.

Simplicity vs. ease of use

Simplicity, when correctly applied, results in ease of use. But simplicity and ease of use are not the same concepts. Ease of use is achieved when users can perform a task successfully on their own without difficulty or confusion within a suitable amount of time. There are many ways to achieve ease of use, and simplicity—the reduction of the unessential—is just one of them.

All users, no matter how sophisticated, want to get their work done with a minimum amount of unnecessary effort. All users—even advanced users—are primarily motivated to get their work done, not to learn about computers or your application.

Simplicity is the most effective way to achieve ease of use, and ease of use equals use. Complex, hard-to-use features just don't get used. By contrast, simple, elegant designs that perform their function well are a joy to use. They invoke a positive, emotional response.

For example, consider the wireless networking support in Microsoft Windows XP. Microsoft could have added a wizard to walk users through the configuration process. This approach would have resulted in ease of use but not simplicity, because an unessential feature (the wizard) would have been added. Instead, Microsoft designed wireless networking to configure itself automatically. Users ultimately don't care about the configuration details, so long as it "just works" reliably and securely. This combination of power and simplicity in wireless networking technology has led to its popularity and rapid adoption.

If you do only one thing...

Start your design process with the simplest designs that do the job well.

If you're not satisfied with your current design, start by stripping away all the unessential elements. You will find that what remains is usually quite good.

Obtaining simplicity while maintaining power

Design principles

To obtain simplicity, always design for the probable, not the possible.

The possible

Design decisions based on what's possible lead to complex user interfaces like the Registry Editor, where the design assumes that all actions are equally possible and as a result require equal effort. Because anything is possible, user goals aren't considered in design decisions.

The probable

Design decisions based on the probable lead to simplified, goal- and task-based solutions, where the likely scenarios receive focus and require minimal effort to perform.

The simplicity design principle

To obtain simplicity, focus on what is likely; reduce, hide, or remove what is unlikely; and eliminate what is impossible.

What users will do is far more relevant to design than what they might do.

Design techniques

To obtain simplicity while maintaining power, choose the **right set of features**, locate the features **in the right places**, and **reduce the effort** to use them. This section gives some common techniques to achieve these goals.

Choosing the right feature set

"Perfection is achieved, not when there is nothing more to add,

but when there is nothing left to take away." —Antoine de Saint-Exupery

The following design techniques give your users the features they need while achieving simplicity through actual reduction or removal:

- **Determine the features your users need.** Understand your users' needs through goal, scenario, and task analysis. Determine a set of features that realizes these objectives.

- **Remove unnecessary elements.** Remove elements that aren't likely to be used or have preferable alternatives.
- **Remove unnecessary redundancy.** There might be several effective ways to perform a task. To achieve simplicity, make the hard decision and choose the best one for your target users instead of providing all of them and making the choice an option.
- **Make it "just work" automatically.** The element is necessary, but any user interaction to get it to work is not because there is an acceptable default behavior or configuration. To achieve simplicity, make it work automatically and either hide it from the user completely or reduce its exposure significantly.

Streamlining the presentation

"The ability to simplify means to eliminate the unnecessary

so that the necessary may speak." —Hans Hofmann

Use the following design techniques to preserve power, while achieving simplicity through the perception of reduction or removal:

- **Combine what should be combined.** Put the essential features that support a task together so that a task can be performed in one place. The task's steps should have a unified, streamlined flow. Break down complex tasks into a set of easy, clear steps, so that "one" place might consist of several UI surfaces, such as a wizard.
- **Separate what should be separated.** Not everything can be presented in one place, so always have clear, well-chosen boundaries. Make features that support core scenarios central and obvious, and hide optional functionality or make it peripheral. Separate individual tasks and provide links to related tasks. For example, tasks related to manipulating photos should be clearly separated from tasks related to managing collections of photos, but they should be readily accessible from each other.
- **Eliminate what can be eliminated.** Take a printout of your design and highlight the elements used to perform the most important tasks. Even highlight the individual words in the UI text that communicate useful information. Now review what isn't highlighted and consider removing it from the design. If you remove the item, would anything bad happen? If not, remove it!
- Consistency, configurability, and generalization are often desirable qualities, but they can lead to unnecessary complexity. Review your design for misguided efforts in consistency (such as having redundant text), generalization (such as having any number of time zones when two is sufficient), and configurability (such as options that users aren't likely to change), and eliminate what can be eliminated.

- **Put the elements in the right place.** Within a window, an element's location should follow its utility. Essential controls, instructions, and explanations should all be in context in logical order. If more options are needed, expose them in context by clicking a chevron or similar mechanism; if more information is needed, display an infotip on mouse hover. Place less important tasks, options, and Help information outside the main flow in a separate window or page. The technique of displaying additional detail as needed is called progressive disclosure.
- **Use meaningful high-level combinations.** It is often simpler and more scalable to select and manipulate groups of related elements than individual elements. Examples of high-level combinations include folders, themes, styles, and user groups. Such combinations often map to a user goal or intention that isn't apparent from the individual elements. For example, the intention behind the High Contrast Black color scheme is far more apparent than that of a black window background.
- **Select the right controls.** Design elements are embodied by the controls you use to represent them, so selecting the right control is crucial to efficient presentation. For example, the font selection box used by Microsoft Word shows both a preview of the font as well as the most recently used fonts. Similarly, the way Word shows potential spelling and grammar errors in place is much simpler than the dialog box alternative, as shown in the beginning of this article.

Reducing effort

"Simple things should be simple.

Complex things should be possible."—Alan Kay

The following design techniques result in reduced effort for users:

- **Make tasks discoverable and visible.** All tasks, but especially frequent tasks, should be readily discoverable within the user interface. The steps required to perform tasks should be visible and should not rely on memorization.
- **Present tasks in the user's domain.** Complex software requires users to map their problems to the technology. Simple software does that mapping for them by presenting what is natural. For example, a red-eye reduction feature maps directly to the problem space and doesn't require users to think in terms of details like hues and gradients.
- **Put domain knowledge into the program.** Users shouldn't be required to access external information to use your application successfully. Domain knowledge can range from complex data and algorithms to simply making it clear what type of input is valid.

- **Use text that users understand.** Well-crafted text is crucial to effective communication with users. Use concepts and terms familiar to your users. Fully explain what is being asked in plain language so that users can make intelligent, informed decisions.
- **Use safe, secure, probable defaults.** If a setting has a value that applies to most users in most circumstances, and that setting is both safe and secure, use it as the default value. Make users specify values only when necessary.
- **Use constraints.** If there are many ways to perform a task, but only some are correct, constrain the task to those correct ways. Users should not be allowed to make readily preventable mistakes.

Simplicity does not mean simplistic

"Everything should be made as simple as possible,

but not simpler."—Albert Einstein

We believe that simplicity is crucial to an effective, desirable user experience—but it is always possible to take a good thing too far. The essence of simplicity is the reduction or elimination of the unessential. Removal of the essential just produces a poor design. If your "simplification" results in users becoming frustrated, confused, unconfident, or unable to complete tasks successfully, you have removed too much.

Simplicity does mean more effort for you

"I have only made this letter longer because I have

not the time to make it shorter."—Blaise Pascal

Obtaining simplicity while preserving power often requires significant internal complexity. It is usually easier to design software that exposes all the technology plumbing than to design one that hides it—the latter requires an excellent understanding of your target users and their goals. Removing a feature requires discipline, as does deciding against adding that cool feature that really isn't practical. Simplicity requires making hard design choices instead of making everything configurable. Complex software often results from a misconception about users: that they value unused features or overly complex features they can't understand.

Powerful and simple

Power is all about enabling your users and making them productive. Simplicity is all about removing the unessential and presenting features the right way. By understanding

your target users and achieving the right balance of features and presentation, you can design Windows-based applications that do both.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

UX checklist for desktop applications

Article • 03/24/2021

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

Here's a collection of some of the important guidelines in UX Guide. You can use this as a checklist to make sure your program user interface gets these important items right.

Windows

- **Support the minimum Windows effective resolution of 800x600 pixels.** For critical user interfaces (UIs) that must work in safe mode, support an effective resolution of 640x480 pixels. Be sure to account for the space used by the taskbar by reserving 48 vertical relative pixels for windows displayed with the taskbar.
- **Optimize resizable window layouts for an effective resolution of 1024x768 pixels.** Automatically resize these windows for lower screen resolutions in a way that is still functional.
- **Be sure to test your windows in 96 dots per inch (dpi) (at 800x600 pixels), 120 dpi (at 1024x768 pixels), and 144 dpi (at 1200x900 pixels) modes.** Check for layout problems, such as clipping of controls, text, and windows, and stretching of icons and bitmaps.
- **For programs with touch and mobile use scenarios, optimize for 120 dpi.** High-dpi screens are currently prevalent on touch and mobile PCs.
- **If a window is an owned window, initially display it "centered" on top of the owner window.** Never underneath. For subsequent display, consider displaying it in its last location (relative to the owner window) if doing so is likely to be more convenient.
- **If a window is contextual, always display it near the object that it was launched from.** However, place it out of the way (preferably offset down and to the right) so that the object isn't covered by the window.

Layout

- **Size controls and panes within a window to match their typical content.** Avoid truncated text and their associated ellipses. Users should never have to interact with a window to view its typical content—reserve resizing and scrolling for unusually large content. Specifically check:
  - **Control sizes.** Size controls to their typical content, making controls wider, taller, or multi-line if necessary. Size controls to eliminate or reduce scrolling in windows that have plenty of available space. Also, there should never be truncated labels, or truncated text within windows that have plenty of available space. However, to make text easier to read, consider limiting line widths to 65 characters.
  - **Column widths.** Make sure list view columns have suitable default, minimum, and maximum sizing. Choose list views default column widths that don't result in truncated text, especially if there is space available within the list view.
  - **Layout balance.** The layout of a window should feel roughly balanced. If the layout feels left-heavy, consider making controls wider and moving some controls more to the right.
  - **Layout resize.** When a window is resizable and data is truncated, make sure larger window sizes show more data. When data is truncated, users expect resizing windows to show more information.
- **Set a minimum window size if there is a size below which the content is no longer usable.** For resizable controls, set minimum resizable element sizes to their smallest functional sizes, such as minimum functional column widths in list views.

Text

- **Use ordinary, conversational terms when you can.** Focus on the user goals, not technology. This is especially effective if you are explaining a complex technical concept or action. Imagine yourself looking over the user's shoulder and explaining how to accomplish the task.
- **Be polite, supportive, and encouraging.** The user should never feel condescended to, blamed, or intimidated.
- **Remove redundant text.** Look for redundant text in window titles, main instructions, supplemental instructions, content areas, command links, and commit buttons. Generally, leave full text in main instructions and interactive controls, and remove any redundancy from the other places.
- **Use title-style capitalization for titles, and sentence-style capitalization for all other UI elements.** Doing so is more appropriate for the Windows tone.
  - **Exception:** For legacy applications, you may use title-style capitalization for command buttons, menus, and column headings if necessary to avoid mixing capitalization styles.

- **For feature and technology names, be conservative in capitalizing.** Typically, only major components should be capitalized (using title-style capitalization).
- **For feature and technology names, be consistent in capitalizing.** If the name appears more than once on a UI screen, it should always appear the same way. Likewise, across all UI screens in the program, the name should be consistently presented.
- **Don't capitalize the names of generic user interface elements,** such as toolbar, menu, scroll bar, button, and icon.
  - **Exceptions:** Address bar, Links bar, ribbon.
- **Don't use all capital letters for keyboard keys.** Instead, follow the capitalization used by standard keyboards, or lowercase if the key is not labeled on the keyboard.
- **Ellipses mean incompleteness.** Use ellipses in UI text as follows:
  - **Commands.** Indicate that a command needs additional information. Don't use an ellipsis whenever an action displays another window—only when additional information is required. Commands whose implicit verb is to show another window don't take an ellipsis, such as Advanced, Help, Options, Properties, or Settings.
  - **Data.** Indicate that text is truncated.
  - **Labels.** Indicate that a task is in progress (for example, "Searching...").

Tip: Truncated text in a window or page with unused space indicates poor layout or a default window size that is too small. Strive for layouts and default window sizes that eliminate or reduce the amount of truncated text. For more information, see Layout.

- **Don't use blue text that isn't a link, because users may assume that it is a link.** Use bold or a shade of gray where you'd otherwise use colored text.
- **Use bold sparingly to draw attention to text users must read.**
- **Use the main instruction to explain concisely what users should do in a given window or page.** Good main instructions communicate the user's objective rather than focusing just on manipulating the UI.
- **Express the main instruction in the form of an imperative direction or specific question.**
- **Don't place periods at the end of control labels or main instructions.**
- **Use one space between sentences.** Not two.

Controls

- **General**
  - **Label every control or group of controls. Exceptions:**
    - Text boxes and drop-down lists can be labeled using prompts.

    - Subordinate controls use the label of their associated control. Spin controls are always subordinate controls.
  - **For all controls, select the safest (to prevent loss of data or system access), most secure value by default.** If safety and security aren't factors, select the most likely or convenient value.
  - **Prefer constrained controls.** Use constrained controls like lists and sliders whenever possible, instead of unconstrained controls like text boxes, to reduce the need for text input.
  - **Reconsider disabled controls.** Disabled controls can be hard to use because users literally have to deduce why they are disabled. Disable a control when users expect it to apply and they can easily deduce why the control is disabled. Remove the control when there is no way for users to enable it or they don't expect it to apply, or leave it enabled, but provide an error message when it is used incorrectly.
    - **Tip:** If you aren't sure whether you should disable a control or provide an error message, start by composing the error message that you might provide. If the error message contains helpful information that target users aren't likely to quickly deduce, leave the control enabled and provide the error. Otherwise, disable the control.
- **Command buttons**
  - **Prefer specific labels over generic ones.** Ideally users shouldn't have to read anything else to understand the label. Users are far more likely to read command button labels than static text.
    - **Exception:** Don't rename the Cancel button if the meaning of cancel is unambiguous. Users shouldn't have to read all the buttons to determine which button cancels an action. However, rename Cancel if it is unclear what action is being canceled, such as when there are several pending actions.
  - **When asking a question, use labels that match the question.** For example, provide Yes and No responses to a yes or no question.
  - **Don't use Apply buttons in dialog boxes that aren't property sheets or control panel items.** The Apply button means apply the pending changes, but leave the window open. Doing so allows users to evaluate the changes before closing the window. However, only property sheets and control panel items have this need.
  - Label a button Cancel if canceling returns the environment to its previous state (leaving no side effect); otherwise, label the button Close (if the operation is complete), or Stop (if the operation is in progress) to indicate that it leaves the current changed state intact.
- **Command links**
  - **Always present command links in a set of two or more.** Logically, there is no reason to ask a question that has only one answer.

    - **Provide an explicit Cancel button.** Don't use a command link for this purpose. Quite often, users realize that they don't want to perform a task. Using a command link to cancel would require users to read all the command links carefully to determine which one means to cancel. Having an explicit Cancel button allows users to cancel a task efficiently.
    - **If providing an explicit Cancel button leaves a single command link, provide both a command link to cancel and a Cancel button.** Doing so makes it clear that users have a choice. Phrase this command link in terms of how it differs from the first response, instead of just "Cancel" or some variation.
- **"Don't show this <item> again" check boxes**
    - **Consider using a "Don't show this <item> again" option to allow users to suppress a recurring dialog box, only if there isn't a better alternative.** It is better always to show the dialog if users really need it, or simply eliminate it if they don't.
    - **Replace <item> with the specific item.** For example, Don't show this reminder again. When referring to a dialog box in general, use Don't show this message again.
    - **Clearly indicate when user input will be used for future default values** by adding the following sentence under the option: Your selections will be used by default in the future.
    - **Don't select the option by default. If the dialog box really should be displayed only once, do so without asking.** Don't use this option as an excuse to annoy users—make sure the default behavior isn't annoying.
    - **If users select the option and click Cancel, this option does take effect.** This setting is a meta-option, so it doesn't follow the standard Cancel behavior of leaving no side effect. Note that if users don't want to see the dialog in the future, most likely they want to cancel it as well.
- **Links**
    - **Don't assign an** access key. Links are accessed using the Tab key.
    - **Don't add "Click" or "Click here" to the link text.** It isn't necessary because a link implies clicking.
- **Tooltips**
    - **Use tooltips to provide labels for unlabeled controls.** You don't have to give labeled controls tooltips simply for the sake of consistency.
    - **Tooltips may also provide more detail for labeled toolbar buttons if doing so is helpful.** Don't just repeat or give a wordy restatement of what is already in the label.
    - **Avoid covering the object the user is about to view or interact with.** Always place the tip on the side of the object, even if that requires separation between the pointer and the tip. Some separation isn't a problem as long as the relationship between the object and its tip is clear.

    - **Exception:** Full name tooltips used in lists and trees.
  - **For collections of items, avoid covering the next object that the user is likely to view or interact with.** For horizontally arranged items, avoid placing tips to the right; for vertically arranged items, avoid placing tips below.
- **Progressive disclosure**
  - **Use More/Fewer progressive disclosure buttons to hide advanced or rarely used options, commands, and details that users typically don't need.** Don't hide commonly used items, because users might not find them. But make sure whatever is hidden has value.
  - If the surface always displays some options, commands, or details, use the following label pairs:
    - **More/Fewer options.** Use for options or a mixture of options, commands, and details.
    - **More/Fewer commands.** Use for commands only.
    - **More/Fewer details.** Use for information only.
    - **More/Fewer <object name>.** Use for other object types, such as folders.
  - Otherwise:
    - **Show/Hide options.** Use for options or a mixture of options, commands, and details.
    - **Show/Hide commands.** Use for commands only.
    - **Show/Hide details.** Use for information only.
    - **Show/Hide <object name>.** Use for other object types, such as folders.
- **Progress bars**
  - **Use determinate progress bars for operations that require a bounded amount of time,** even if that amount of time cannot be accurately predicted. Indeterminate progress bars show that progress is being made, but provide no other information. Don't choose an indeterminate progress bar based only on the possible lack of accuracy alone.
  - **Provide a time remaining estimate if you can do so accurately.** Time remaining estimates that are accurate are useful, but estimates that are way off the mark or bounce around significantly aren't helpful. You may need to perform some processing before you can give accurate estimates. If so, don't display potentially inaccurate estimates during this initial period.
  - **Don't restart progress.** A progress bar loses its value if it restarts (perhaps because a step in the operation completes) because users have no way of knowing when the operation will complete. Instead, have all the steps in the operation share a portion of the progress and have the progress bar go to completion once.
  - **Provide useful progress details.** Provide additional progress information, but only if users can do something with it. Make sure the text is displayed long enough for users to be able to read it.

    - **Don't combine a progress bar with a busy pointer.** Use one or the other, but not both at the same time.
- **Prompts**
    - **Use a prompt when screen space is at such a premium that using a label or instruction is undesirable,** such as on a toolbar.
    - **The prompt is primarily for identifying the purpose of the text box or combo box in a compact way.** It must not be crucial information that the user needs to see while using the control.
    - **The prompt text must not be confused with real text.** To do this:
        - Draw the prompt text in italic gray and the actual input text in roman black.
        - The prompt text should not be editable and should disappear once users click in or tab into the text box.
            - **Exception:** If the text box has default input focus, the prompt is displayed, and it disappears once the user starts typing.
    - Don't use ending punctuation or ellipsis.
- **Notifications**
    - **Use notifications for events that are unrelated to the current user activity, don't require immediate user action, and users can freely ignore.**
    - **Don't abuse notifications:**
        - **Use notifications only if you need to.** When you display a notification, you are potentially interrupting users or even annoying them. Make sure that interruption is justified.
        - **Use notifications for non-critical events or situations that don't require immediate user action.** For critical events or situations that require immediate user action, use an alternative UI element (such as a modal dialog box).
        - **Don't use notifications for feature advertisements!**

Keyboard

- **Assign initial input focus to the control that users are most likely to interact with first,** which is often the first interactive control. If the first interactive control isn't a good choice, consider changing the window's layout.
- **Assign tabs stops to all interactive controls, including read-only edit boxes. Exceptions:**
    - Group sets of related controls that behave as a single control, such as radio buttons. Such groups have a single tab stop.
    - Properly contain groups so that the arrow keys cycle both forward and backward within the group and stay within the group.

- **Tab order should flow from left to right, top to bottom.** Generally, tab order should follow reading order. Consider making exceptions for commonly used controls by putting them earlier in the tab order. Tab should cycle through all the tab stops in both directions without stopping. Within a group, tab order should be in sequential order, without exceptions.
- **Within a tab stop, the arrow key order should flow from left to right, top to bottom,** without exceptions. The arrow keys should cycle through all items in both directions without stopping.
- **Present the commit buttons in the following order:**
  - OK/[Do it]/Yes
  - [Don't do it]/No
  - Cancel
  - Apply (if present)

where [Do it] and [Don't do it] are specific responses to the main instruction.

- **Don't confuse access keys with shortcut keys.** While both access keys and shortcut keys provide keyboard access to UI, they have different purposes and guidelines.
- **Whenever possible, assign unique access keys to all interactive controls or their labels.** Read-only text boxes are interactive controls (because users can scroll them and copy text), so they benefit from access keys. **Don't assign access keys to:**
  - OK, Cancel, and Close buttons. Enter and Esc are used for their access keys. However, always assign an access key to a control that means OK or Cancel, but has a different label.
- **Assign shortcut keys to the most commonly used commands.** Infrequently used programs and features don't need shortcut keys because users can use access keys instead.
- **Don't make a shortcut key the only way to perform a task.** Users should also be able to use the mouse or the keyboard with Tab, arrow, and access keys.
- **Don't assign different meanings to well-known shortcut keys.** Because they are memorized, inconsistent meanings for well-known shortcuts are frustrating and error prone.
- **Don't try to assign system-wide program shortcut keys.** Your program's shortcut keys will have effect only when your program has input focus.

Mouse

- **Never require users to click an object to determine if it is clickable.** Users must be able to determine clickability by visual inspection alone.

    - Primary UI (such as commit buttons) must have a static click affordance. Users shouldn't have to hover to discover primary UI.
    - Secondary UI (such as secondary commands or progressive disclosure controls) can display their click affordance on hover.
    - Text links should statically suggest link text, and then display their click affordance (underline or other presentation change, with hand pointer) on hover.
    - Graphics links only display a hand pointer on hover.
- **Use the hand (or "link select") pointer only for text and graphic links.** Otherwise, users would have to click on objects to determine if they are links.

Dialog boxes

- **Modal dialog boxes require interaction, so use them for things that users must respond to before continuing with their task.** Make sure the interruption is justified, such as for critical or infrequent, one-off tasks that require completion. Otherwise, consider modeless alternatives.
- **Modeless dialog boxes don't require interaction, so use them when users need to switch between a dialog box and the owner window.** They are best used for frequent, repetitive, or ongoing tasks. However, ribbons, toolbars, and palette windows are often better alternatives.

Property Sheets

- **Make sure the properties are necessary.** Don't clutter your property pages with unnecessary properties just to avoid making hard design decisions.
- **Present properties in terms of user goals instead of technology.** Just because a property configures a specific technology doesn't mean that you must present the property in terms of that technology.
    - If you must present settings in terms of technology (perhaps because your users recognize the technology's name), include a brief description of the user benefit.
- **Use specific, meaningful tab labels.** Avoid generic tab labels that could apply to any tab, such as General, Advanced, or Settings.
- **Avoid General pages.** You aren't required to have a General page. Use a General page only if:
    - The properties apply to several tasks and are meaningful to most users. Don't put specialized or advanced properties on a General page, but you can make them accessible through a command button on the General page.

    - The properties don't fit a more specific category. If they do, use that name for the page instead.
- **Avoid Advanced pages.** Use an Advanced page only if:
    - The properties apply to uncommon tasks and are meaningful primarily to advanced users.
    - The properties don't fit a more specific category. If they do, use that name for the page instead.
- **Don't use tabs if a property window has only a single tab and isn't extensible.** Use a regular dialog box with OK, Cancel, and an optional Apply button instead. However, extensible property windows (which can be extended by third parties) must use tabs.

Wizards

- **Consider lightweight alternatives first, such as dialog boxes, task panes, or single pages.** Wizards are a heavy UI, best used for multi-step, infrequently performed task. You don't have to use wizards—you can provide helpful information and assistance in any UI.
- **Use Next only when advancing to the next page without commitment.** Advancing to the next page is considered a commitment when its effect can't be undone by clicking Back or Cancel.
- **Use Back only to correct mistakes.** Aside from correcting mistakes, users shouldn't have to click Back to make progress in a task.
- **When users are committing to a task, use a commit button that is a specific response to the main instruction (for example, Print, Connect, or Start).** Don't use generic labels like Next (which doesn't imply commitment) or Finish (which isn't specific) for committing a task. The labels on these commit buttons should make sense on their own. Always start commit button labels with a verb.
  Exceptions:
    - Use Finish when the specific responses are still generic, such as Save, Select, Choose, or Get.
    - Use Finish to change a specific setting or a collection of settings.
- **Use command links only for choices, not commitments.** Specific commit buttons indicate commitment far better than command links in a wizard.
- **When using command links, hide the Next button, but leave the Cancel button.**
- **Use Close for Follow-Up and Completion pages.** Don't use Cancel because closing the window won't abandon any changes or actions done at this point. Don't use Done because it isn't an imperative verb.
- **Don't use "wizard" in wizard names.** For example, use "Connect to a Network" instead of "Network Setup Wizard." However, it's acceptable to refer to wizards as

wizards. For example: "If you're setting up a network for the first time, you can get help by using the Connect to a Network wizard."
- **Preserve user selections through navigation.** For example, if the user makes changes, clicks Back, then Next, those changes should be preserved. Users don't expect to have to re-enter changes unless they explicitly chose to clear them.

Wizard pages

- **Focus on efficient decision making.** Reduce the number of pages to focus on essentials. Consolidate related pages, and take optional pages out of the main flow. Having users click Next completely through your wizard may seem like a good experience at first, but if users never need to change the defaults, the pages are probably unnecessary.
- **Don't use Welcome pages—make the first page functional whenever possible.** Use an optional Getting Started page only when:
  - The wizard has prerequisites that are necessary to complete the wizard successfully.
  - Users may not understand the purpose of the wizard based on its first Choice page, and there isn't room for further explanation.
  - The main instruction for Getting Started pages is "Before you begin:".
- **On pages in which users are asked to make choices, optimize for the most likely cases.** These kinds of pages should present actual choices, not just instructions.
  - If you don't use a Getting Started page, explain the purpose of the wizard at the top of the first page of choices.
- **Use Commit pages to make it clear when users are committing to the task.** Usually the Commit page is the last page of choices, and the Next button is relabeled to indicate the task being committed.
  - Don't use Summary pages that merely summarize the user's previous selections, unless the task is risky (involving security, or loss of time or money) or there is a good chance that users might not understand their selections and need to review them.
- **Don't use Congratulations pages that do nothing but end the wizard.** If the wizard results are clearly apparent to users, just close the wizard on the final commit button.
  - Use Follow-Up pages when there are related tasks that users are likely to do as follow-up. Avoid familiar follow-up tasks, such as "Send an e-mail message."
  - Use Completion pages only when the results aren't visible and there's no better way to provide feedback for task completion.
  - Wizards that have Progress pages must use a Completion page or Follow-Up page to indicate task completion. For long-running tasks, close the wizard on

the Commit page and use notifications to give feedback.

Error messages

- **Don't give error messages when users aren't likely to perform an action or change their behavior** as the result of the message. If there is no action users can take, or if the problem isn't significant, suppress the error message.
- **Whenever possible, propose a solution so users can fix the problem.** However, make sure the proposed solution is likely to solve the problem. Don't waste users' time by suggesting possible, but improbable, solutions.
- **Be specific.** Avoid vague wording, such as syntax error and illegal operation. Provide specific names, locations, and values of the objects involved.
- **Don't use phrasing that blames the user or implies user error.** Avoid using you and your in the phrasing. While the active voice is generally preferred, use the passive voice when the user is the subject and might feel blamed for the error if the active voice were used.
- **Don't use OK for error messages.** Users don't view errors as being OK. If the error message has no direct action, use Close instead.
- **Don't use the following words:**
  - Error, failure (use problem instead)
  - Failed to (use unable to instead)
  - Illegal, invalid, bad (use incorrect or not valid instead)
  - Abort, kill, terminate (use stop instead)
  - Catastrophic, fatal (use serious instead)

These terms are unnecessary and contrary to the encouraging tone of Windows. Instead, an error icon, when used correctly, sufficiently communicates that there is a problem.

- **Don't accompany error messages with sound effects.** Doing so is jarring and unnecessary.

Warning messages

- **Warnings describe a condition that might cause a problem in the future.** Warnings aren't errors or questions, so don't phrase routine questions as warnings.
- **Don't give warning messages when users aren't likely to perform an action or change their behavior as the result of the message.** If there is no action users can take, or if the condition isn't significant, suppress the warning message.

Confirmations

- **Don't use unnecessary confirmations.** Use confirmations only when:
  - There is a clear reason not to proceed and a reasonable chance that sometimes users won't.
  - The action has significant consequences or cannot be easily undone.
  - The action has consequences that users might not be aware of.
  - Proceeding with the action requires users to make a choice that doesn't have a suitable default.
  - Given the current context, users are likely to have performed an action in error.
- **Phrase confirmations as a yes or no question, and provide yes or no answers.** Unlike other types of dialog boxes, confirmations are designed to prevent users from making quick decisions. If users don't put thought into their response, a confirmation has no value.

Icons

- **All icons should adhere to the** Aero-style icon guidelines. Replace all Windows XP-style icons.
- **Choose standard icons based their message type, not the severity of the underlying issue:**
  - **Error.** An error or problem that has occurred.
  - **Warning.** A condition that might cause a problem in the future.
  - **Information.** Useful information.

If an issue combines different message types, focus on the most important aspect that users need to act on.

- **Icons must always match the main instruction or other corresponding text.**
- **Error icons aren't needed for non-critical user input problems.** However, icons are needed for in-place errors, because otherwise such contextual feedback would be too easy to overlook.
- **Don't use warning icons to "soften" non-critical errors.** Errors aren't warnings; apply the error icon guidelines instead.
- **For question dialogs, use warning icons only for questions with significant consequences.** Don't use warning icons for routine questions.

Help

- **Link to specific, relevant Help topics.** Don't link to the Help home page, the table of contents, a list of search results, or a page that just links to other pages. Avoid linking to pages structured as a large list of frequently asked questions, because it forces users to search for the one that matches the link they clicked. Don't link to

specific Help topics that aren't relevant and helpful to the task at hand. Never link to empty pages.

- **Don't put Help links on every window or page for the sake of consistency.** Providing a Help link in one place doesn't mean that you have to provide them everywhere.
- **Whenever possible, phrase Help links text in terms of the primary question answered by the Help content.** Don't use the "Learn more about" or "Get help with this" phrasing.
- **Use the entire Help link for the link text,** not just the keywords.
- **Use complete sentences.**
- **Don't use ending punctuation or ellipses, except for question marks.**
- **If the Help content is online, make that clear in the link text.** Doing so helps make the result of the links predictable.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Guidelines

Article • 02/09/2021

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

These sections comprise the detailed user experience guidelines for Windows-based desktop applications.

In this section

Expand table

Topic	Description
Controls	Controls are UI elements that your users interact with on your app's main window area. See visual examples of controls in Windows-based, desktop apps and get links to guidelines for each control.
Commands	Commands are actions users can take while using your app. Learn the guidelines for adding commands to your app's menus, ribbons, and toolbars.
Text	Text includes any text users can see in your app. Review these guidelines on the use of UI text, style, and tone for your apps.
Messages	Messages are any kind of message users need or want to see as they use your app. Learn how to present errors, warning, confirmations, and notifications in your app.
Interaction	Interaction is the variety of ways users interact with your app, including touch, keyboard, mouse, and so on. Use these guidelines to create intuitive and distinctive experiences that are optimized for touch but work consistently across input devices.
Windows	Windows are the main "canvases" or UI surfaces of your desktop app, including the main windows itself and pop-ups, dialogs, and wizards. Follow these guidelines when deciding which surface to use and how best to use them.
Visuals	Visuals include the visual elements other than the controls. These guidelines help you make decisions about layout, fonts, color, icons, and so on in your app.

Topic	Description
Experiences	Experiences are the common experiences and use cases for all apps, like set up, first run, and printing. Learn about the best practices for creating these experiences and communicating your app brand.
Windows Environment	The Windows environment is the onscreen work area provided by Windows, analogous to a physical desktop, and the operating system's core extension points. Learn how to leverage the desktop, taskbar, notification area, control panels, help, and user account control for your app.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

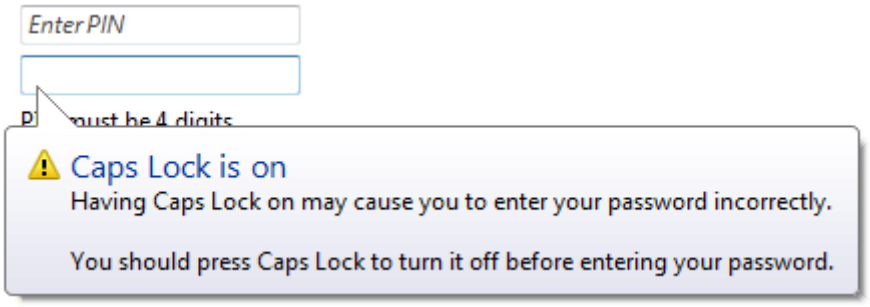
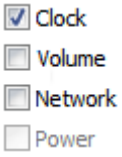
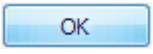
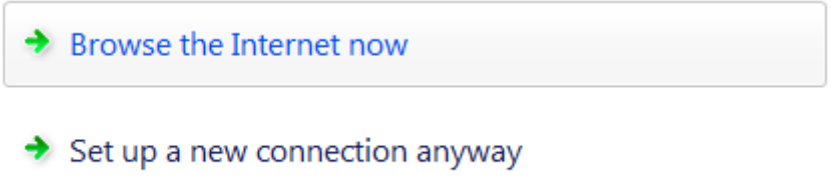
Controls (Design basics)

Article • 06/04/2021

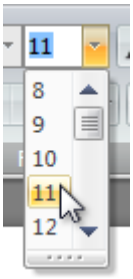
> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

Controls are UI elements that your users interact with on your app's main window area. See visual examples of controls in Windows-based, desktop apps and get links to guidelines for each control.

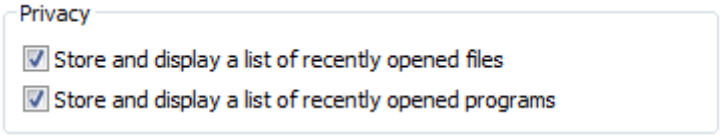
Expand table

Examples
 Balloons inform users of a non-critical problem or special condition in a control.
 Check boxes allow users to make a decision between two or more clearly differing choices.
 Command buttons allow users to perform an immediate action.
 Command links allow users to make a choice among a set of mutually exclusive, related choices.

Examples



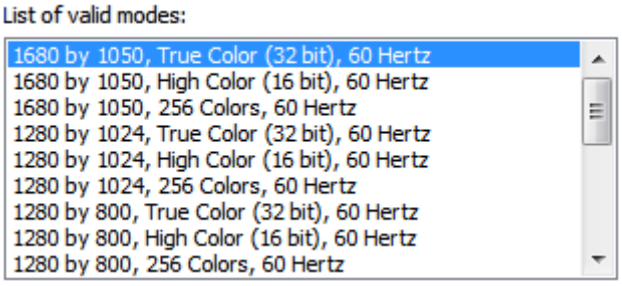
Drop-down lists and combo boxes allow users to make a choice among a list of mutually exclusive values.



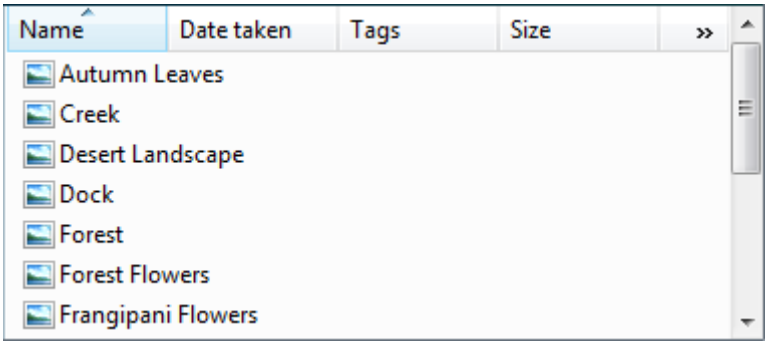
Group boxes allow users to see relationships among a set of related controls.

Check for updates

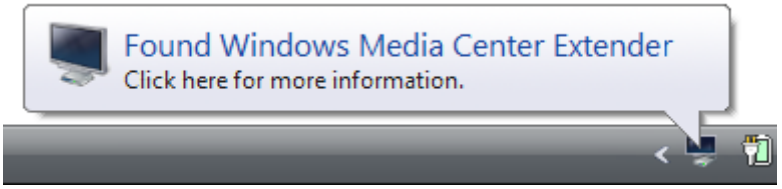
Links allow users to navigate to another page, window, or Help topic; display a definition; initiate a command; or choose an option.



List boxes allow users to select from a set of values presented in a list that is always visible. With a single-selection list box, users select one item from a list of mutually exclusive values. With a multiple-selection list box, users select zero or more items from a list of values.

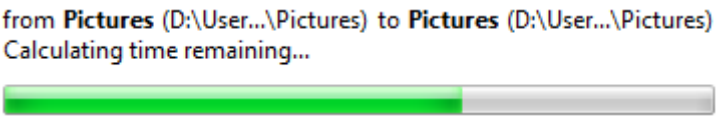


List views allow users to view and interact with a collection of data objects, using either single selection or multiple selection.



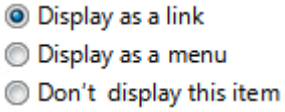
Notifications inform users of events that are unrelated to the current user activity.

Examples



Progress bars allow users to follow the progress of a lengthy operation.

Progressive disclosure controls allow users to show or hide additional information including data, options, or commands.

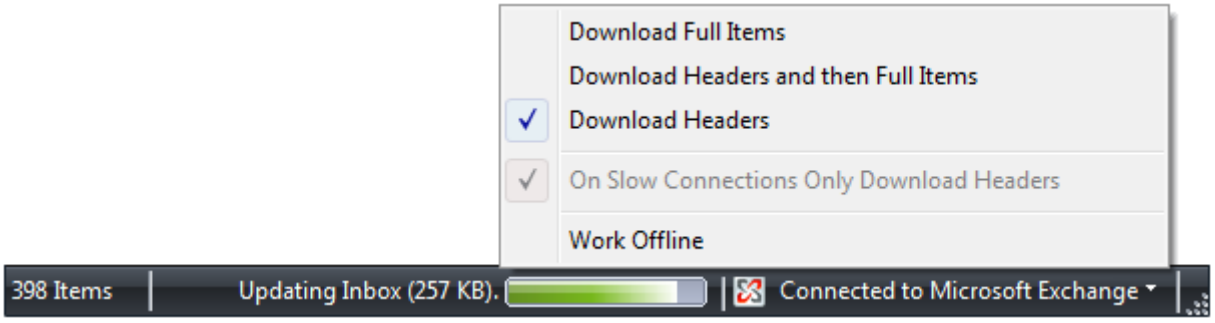


Radio buttons allow users to make a choice among a set of mutually exclusive, related choices.

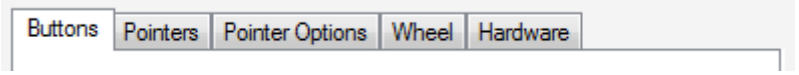
Search boxes provide users a way to locate specific objects or text quickly.

Sliders allow users to choose from a continuous range of values.

Spin controls allow users to change incrementally the value within its associated numeric text box.



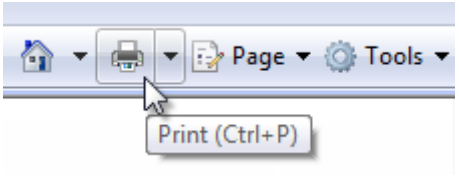
Status bars display information about the state of the current window, background tasks, or other contextual information.



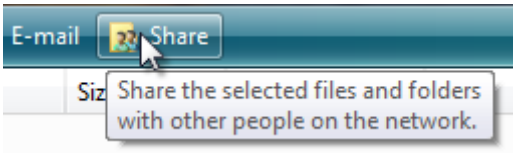
Tabs present users with related information on separate labeled pages.

Text boxes allow users to display, enter, or edit a text or numeric value.

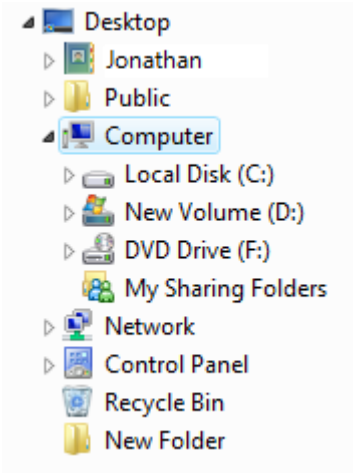
Examples



Tooltips label an unlabeled control.



Infotips describe an object to which the user is pointing.



Tree views allow users to view and interact with a hierarchically arranged collection of objects, using either single selection or multiple selection.

Feedback

Was this page helpful? Yes No

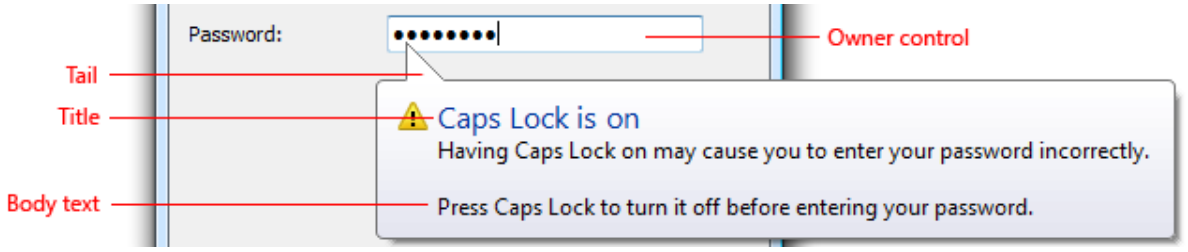
Provide product feedback | Get help at Microsoft Q&A

Balloons

Article • 06/04/2021

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

A balloon is a small pop-up window that informs users of a non-critical problem or special condition in a control.



A typical balloon.

Balloons have an icon, a title, and body text, all of which are optional. Unlike tooltips and infotips, balloons also have a tail that identifies their source. Usually the source is a control if so, it is referred to as the owner control.

While balloons inform users of non-critical problems, they don't prevent problems although the owner control might. Any unhandled problems must be handled by the owner user interface (UI) when users attempt to commit to the action.

Balloons are usually used with text boxes, or controls that use text boxes for changing values, such as combo boxes, list views, and tree views. Other kinds of controls are sufficiently well constrained, and don't need the additional feedback balloons afford. Furthermore, if there is a problem with other types of controls, it often involves inconsistency between multiple controls a situation for which balloons aren't suitable. Only text-entry controls are both unconstrained and a common source of single-point errors.

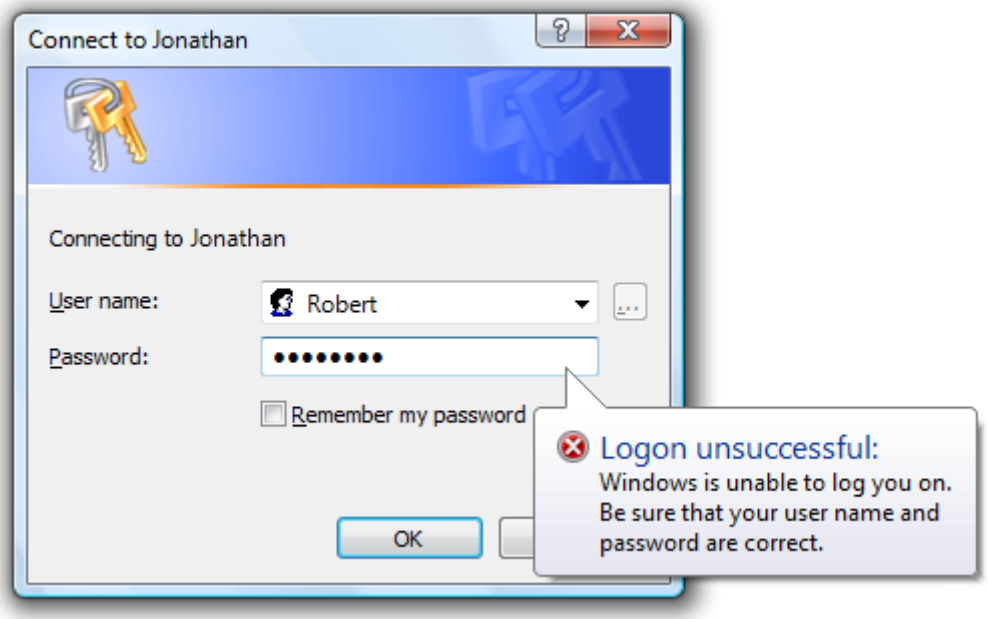
A notification is a specific type of balloon displayed by a notification area icon.

Note: Guidelines related to notifications, tooltips and infotips, and error messages are presented in separate articles.

Is this the right control?

To decide, consider these questions:

- **Does the information describe a problem or special condition?** If not, use another control. Don't use balloons to display supplemental information for a control; consider using static text,infotips, progressive disclosure, or prompts instead.
- **Can the problem or special condition be detected immediately** either on input or when the owner control loses input focus? If not, use an error message displayed in a task dialog or message box.
- **For problems, is the problem critical?** If so, use an error message displayed in a task dialog or message box. Such error messages require interaction (which is suitable for critical errors), whereas balloons don't.
- **For special conditions, is the condition valid yet likely to be unintended?** If so, balloons are appropriate. For conditions not valid, it is better to prevent them in the first place. For likely intended conditions, there is no need to do anything.
- **Can the problem or special condition be expressed concisely?** If not, use another control. Balloons can't have detailed explanations or provide supplemental information.
- **Does the information describe the control currently being hovered over?** If so, use a tip instead, unless users may need to interact with the message.
- **Is the information related to the user's current activity?** If not, consider using a notification or dialog box instead. Users are likely to overlook balloons outside the current activity, and, by default, balloons time out after 10 seconds.
- **Does the information come from a single, specific source?** If a problem or condition has multiple sources or no specific source, use an in-place message or a dialog box instead.



Incorrect:

In this example, the problem could be with the user name or the password, but reporting the problem with a balloon visually suggests that only the password is the

problem. Consequently, the feedback from entering an incorrect user name is misleading.

Balloons are an alternative to infotips, dialog boxes, and in-place messages. In contrast to tooltips and infotips:

- Balloons can be displayed independently of the current pointer location, so they have a tail that indicates their source.
- Balloons have a title, body text, and an icon.
- Balloons can be interactive, whereas it is impossible to click on a tip.

In contrast to modal dialog boxes:

- Balloons don't steal input focus or require interaction.
- Balloons identify a single, specific source. Modal dialogs can have multiple sources, or no specific source at all.

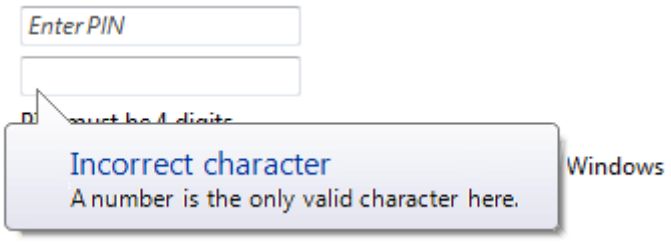
In contrast to in-place messages:

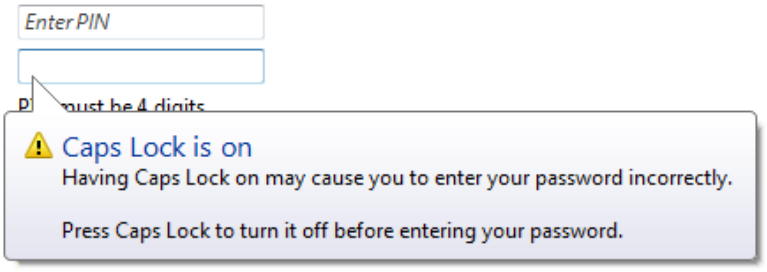
- Balloons are more noticeable.
- Balloons don't require available screen space or the dynamic layout that is required to display in-place messages.
- Balloons remove themselves automatically after a timeout.

Usage patterns

Balloons have these usage patterns:

Expand table

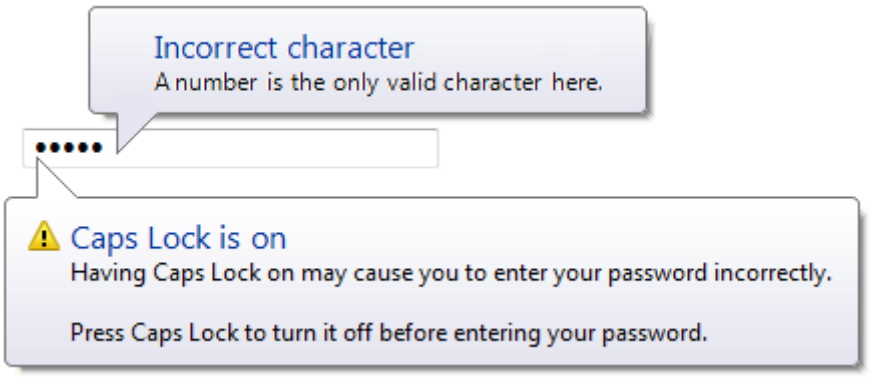
Usage	Example
Input problem A non-critical user input problem coming from a single owner control, usually a text box.	using balloons for error messages doesn't steal input focus, yet is still very noticeable if the owner control has input focus. to correct the problem, the user may have to change or reenter the input; but if the owner control ignores incorrect input, the user may not have to make any changes at all. because the problem isn't critical, no error icon is necessary.  A balloon used to report a non-critical user input problem.

Usage	Example
Special condition The owner control is in a state that affects input. This state is likely unintended and the user may not realize input is affected.	use balloons to prevent frustration by alerting users of special conditions as soon as they happen (for example, exceeding maximum input size or setting caps lock on by mistake). it is important to give such feedback without stealing input focus or forcing interaction because these conditions might be intentional. these balloons are especially important for password and pin boxes, where users are otherwise working with minimal feedback. these balloons have a warning icon.  A balloon used to report a special condition.

Guidelines

When to display

- **Display the balloon as soon as the problem or special condition is detected, even if repeatedly, without any noticeable delay.**
  - For problems involving individual characters or the maximum input size, display the balloon immediately on input.
  - For problems involving the input value (including requiring a non-blank value), display the balloon when the owner control loses input focus. Otherwise, displaying balloons while users are entering potentially valid input can be distracting and annoying.
- **Display only one balloon at a time.** Displaying multiple balloons can be overwhelming. If a single event results in multiple problems, either present all the problems at once or report only the most important problem.



Incorrect:

In this example, two problems are incorrectly presented at the same time.

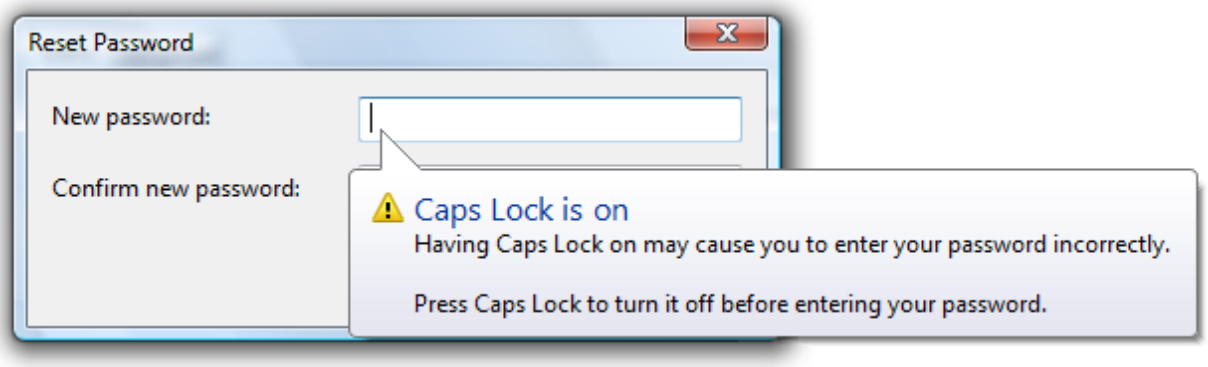
How long to display

- **Remove a balloon when:**
  - The problem is resolved or special condition is removed.
  - The user enters valid data (for input problems).
  - The balloon times out. By default balloons remove themselves after 10 seconds, although users can change this by modifying the SPI_MESSAGEDURATION system parameter.
- **Remove the timeout if users can't continue until the problem is resolved.Developers:** In Win32, you can set the display time with the TTM_SETDELAYTIME message.

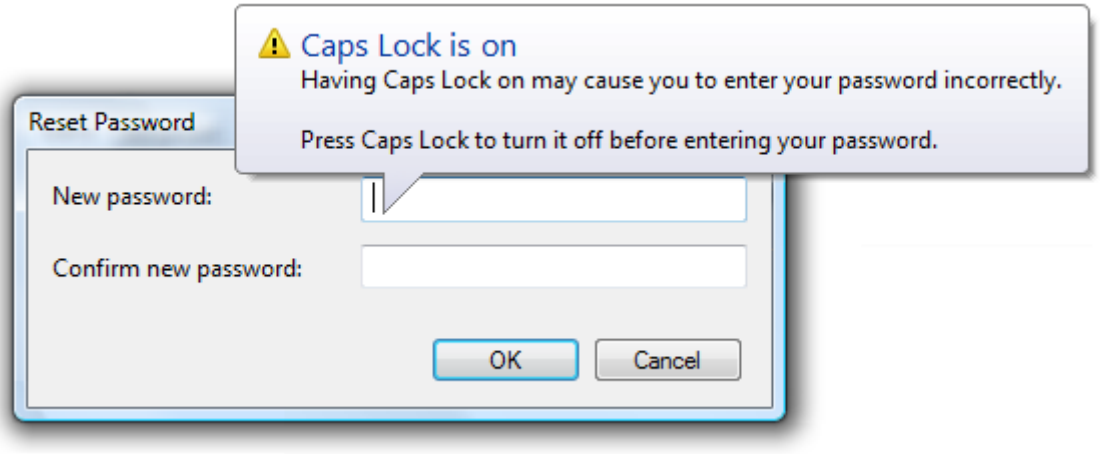
How to display

- **Display balloons below their owner control.** Doing so allows users to view the context, including the owner control and its label. Microsoft Windows automatically adjusts balloon positions so that they are completely on screen. The default behavior is to display a balloon above its owner control, as done with notifications.

Correct:



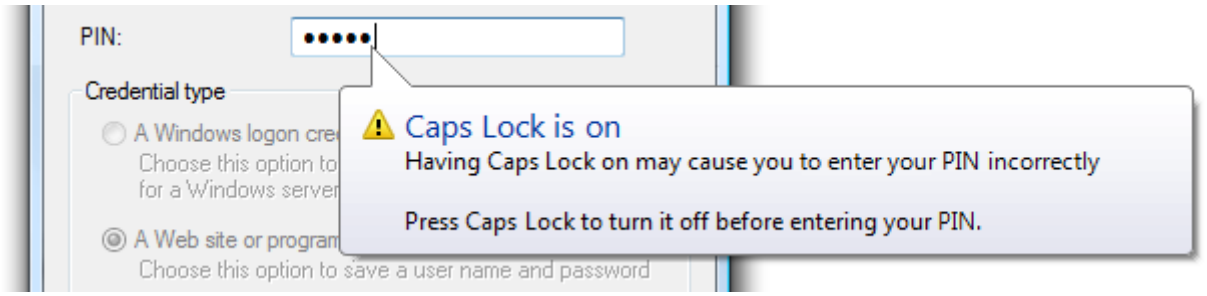
Incorrect:



In the incorrect example, the balloon is awkwardly displayed above its owner control.

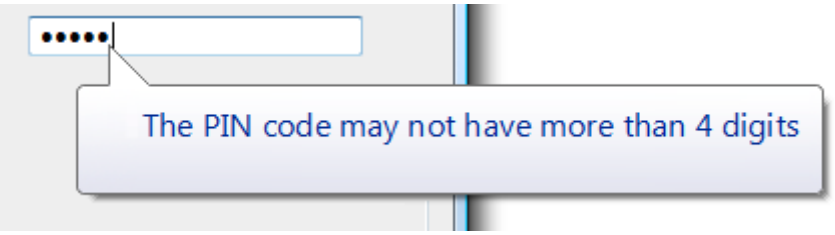
Password and PIN text boxes

- **Use a balloon to indicate that Caps Lock is on**, using the text in the following example:



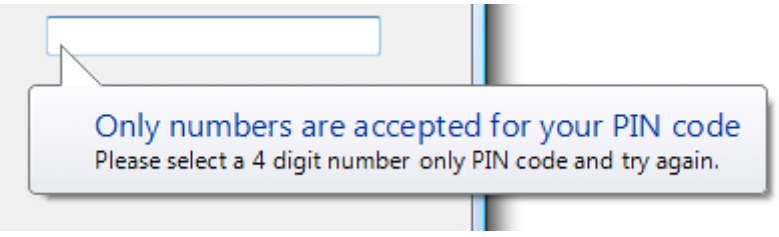
In this example, a balloon indicates that Caps Lock is on in a PIN text box.

- **Use a balloon to indicate when users attempt to exceed the maximum input size.** Reaching the maximum input size is much less obvious in password and PIN text boxes than ordinary text boxes.



In this example, a balloon indicates that the user is attempting to exceed the maximum input size.

- **Use a balloon to indicate when users input incorrect characters.** However, it is better not to have such restrictions because they reduce the security of the password or PIN. To prevent information disclosure, the balloon should mention only documented facts about valid passwords or PINs.

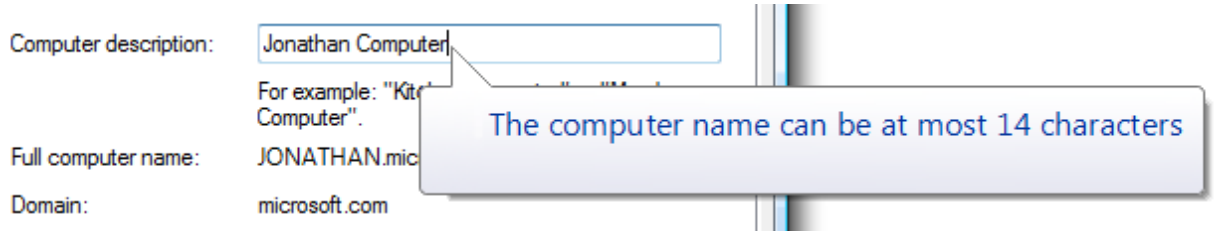


In this example, a balloon indicates that the PIN requires numbers.

Other text boxes

- **Consider using a balloon to indicate when users attempt to exceed the maximum input size for critical, short text boxes aimed at novice users.** Examples include user names and account names. Text boxes beep when users attempt to

exceed the maximum input, but novice users might not understand the meaning of the beep.



In this example, a balloon indicates that the user attempted to exceed the maximum input size.

Interaction

- **When users click a balloon, just dismiss the balloon without displaying any other UI or having any other side effect.** Unlike notifications, balloons shouldn't have close buttons.

Icons

- Choose the icon based on the usage pattern:

Expand table

Pattern	Icon
Input problem	No icon. Not using an error icon here is consistent with the Windows tone guidelines.
Special condition	The standard 16x16 pixel warning icon.

Accessibility

When used properly, balloons enhance accessibility. For balloons to be accessible:

- Only display balloons that relate to the user's current activity.
- Keep the balloon text concise. Doing so makes the balloon text easier to read for users with low vision, and minimizes the interruption when read by screen readers.
- Redisplay the balloon whenever the problem or condition recurs.

Text

Title text

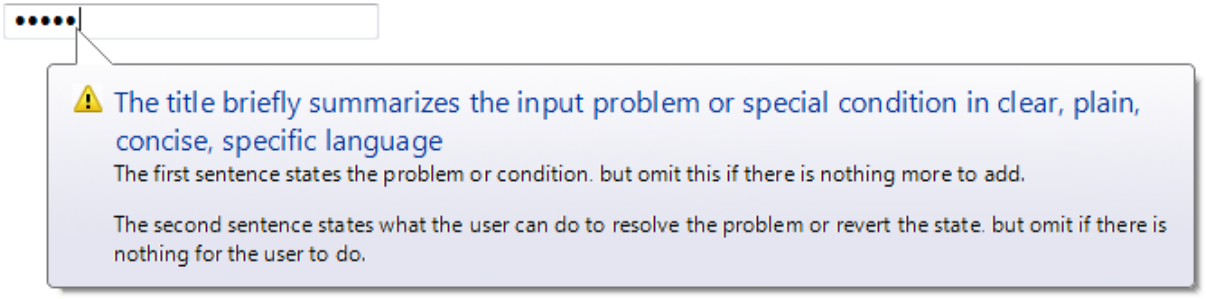
- **Use title text that briefly summarizes the input problem or special condition in clear, plain, concise, specific language.** Users should be able to understand the

purpose of the balloon quickly and with minimal effort.

- **Use text fragments or complete sentences without ending punctuation.**
- **Use sentence-style capitalization.** For more info, see the glossary.
- **Use no more than 48 characters (in English) to accommodate localization.** The title has a maximum length of 63 characters and must be able to expand by at least 30 percent to accommodate localization.

Body text

- **Use the first sentence of the body text to state the problem or condition in a way that is clearly relevant to the user.** Don't repeat the information in the title. Omit this if there is nothing more to add.
- **Use the second sentence to state what the user can do to resolve the problem or revert the state.** In accordance with the Style and Tone guidelines, there's no need to use the word Please in this statement. Put two line breaks between the first and second sentences.



This example shows the standard balloon text layout.

- **Explain how to resolve the problem or revert the state even if that explanation is obvious,** but omit redundancy between the problem statement and its resolution.
  Exceptions:
  - Omit the resolution if it can't be expressed concisely or without significant redundancy.
  - Omit the resolution if there is nothing for the user to do, such as when incorrect characters are ignored.
- **Use complete sentences with ending punctuation.**
- **Use sentence-style capitalization.**
- **Use no more than 200 characters (in English) to accommodate localization.** The body text has a maximum length of 255 characters and must be able to expand by at least 30 percent to accommodate localization.

Documentation

When referring to balloons:

- Use the exact title text, including its capitalization.
- Refer to the component as a balloon, not as a notification or an alert.
- When possible, format the title text using bold text. Otherwise, put the title in quotation marks only if required to prevent confusion.

Feedback

Was this page helpful? Yes No

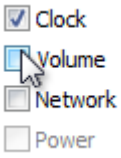
Provide product feedback | Get help at Microsoft Q&A

Check Boxes

Article • 08/25/2021

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

With a check box, users make a decision between two clearly opposite choices. The check box label indicates the selected state, whereas the meaning of the cleared state must be the unambiguous opposite of the selected state. Consequently, **check boxes should be used only to toggle an option on or off or to select or deselect an item.**



A typical group of check boxes.

> **Note**
>
> Guidelines related to layout are presented in a separate article.

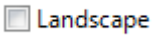
Is this the right control?

To decide, consider these questions:

- **Is the check box used to toggle an option on or off or to select or deselect an item?** If not, use another control.
- **Are the selected and cleared states clear and unambiguous opposites?** If not, use radio buttons or a drop-down list so that you can label the states independently.
- **When used in a group, does the group comprise independent choices, from which users may choose zero or more?** If not, consider controls for dependent choices, such as radio buttons and check box tree views.

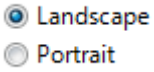
- **When used in a group, does the group comprise dependent choices, from which users must choose one or more?** If so, use a group of check boxes and handle the error when none of the options are selected.
- **Is the number of options in a group 10 or fewer?** Since the screen space used is proportional to the number of options, keep the number of check boxes to 10 or fewer. For more than 10 options, use a check box list.
- **Would a radio button be a better choice?** Where check boxes are suitable only for turning an option on or off, radio buttons can be used for completely different options. If both solutions are possible:
  - Use radio buttons if the meaning of the cleared check box isn't completely obvious.

    Incorrect:

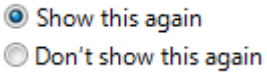
    In this example, the opposite choice from Landscape isn't clear, so the check box isn't a good choice.

    Correct:

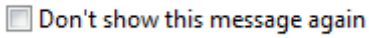
    In this example, the choices are not opposites so radio buttons are the better choice.
  - Use radio buttons on wizard pages to make the alternatives clear, even if a check box is otherwise acceptable.
  - Use radio buttons if you have enough screen space and the options are important enough to be a good use of that screen space. Otherwise, use a check box or a drop-down list.

    Incorrect:

    In this example, the options aren't important enough to use radio buttons.

    Correct:



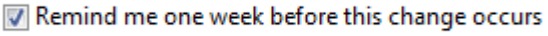
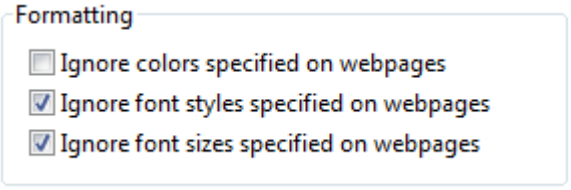
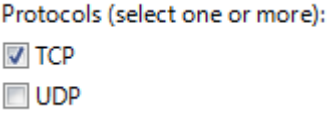
In this example, a check box is an efficient use of screen space for this peripheral option.

- Use a check box if there other check boxes on the window.
- **Does the option present a program option, rather than data?** The option's values shouldn't be based on context or other data. For data, use a check box list or multiple-selection list.

Usage patterns

Check boxes have several usage patterns:

Expand table

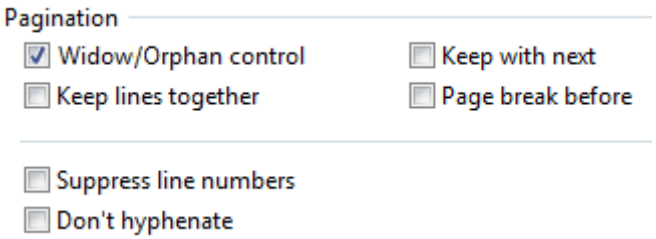
Usage	Example
An individual choice A single check box is used to select an individual choice.	 A single check box is used for an individual choice.
Independent choices (zero or more) A group of check boxes is used to select from a set of zero or more choices.	unlike single-selection controls such as radio buttons, users can select any combination of options in a group of check boxes.  A group of check boxes is used for independent choices.
Dependent choices (one or more) A group of check boxes can also be used to select from a set of one or more choices.	**you may need to represent a selection of one or more dependent choices**. because microsoft?windows doesn't have a control that directly supports this type of input, the best solution is to use a group of check boxes and handle the error when none of the options are selected.  A group of check boxes is used where at least one protocol must be selected.

Usage	Example
Mixed choice In addition to their selected and cleared states, check boxes also have a mixed state for multiple selection to indicate that the option is set for some, but not all, objects.	 A mixed-state check box.

Guidelines

General

- **Group related check boxes**. Combine related options and separate unrelated options into groups of 10 or fewer, using multiple groups if necessary.

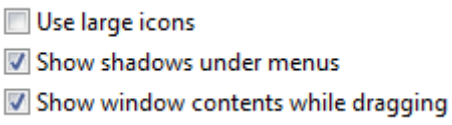
  An example of groups of related, independent options.

- **Reconsider using group boxes to organize groups of check boxes** this often results in unnecessary screen clutter.

- **List check boxes in a logical order**, such as grouping highly related options together or placing most common options first, or following some other natural progression. Alphabetical ordering isn't recommended because it is language dependent, and therefore not localizable.

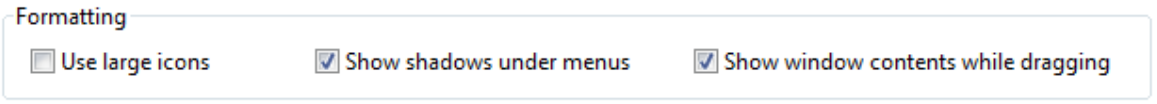
- **Align check boxes vertically, not horizontally**. Horizontal alignment is harder to read.

  Correct:

  In this example, the check boxes are correctly aligned.

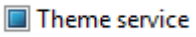
  Incorrect:



In this example, the horizontal alignment is harder to read.

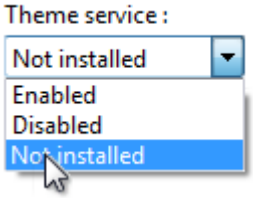
- **Don't use the mixed state to represent a third state.** The mixed state is used to indicate that an option is set for some, but not all, child objects. Users shouldn't be able to set a mixed state directly rather the mixed state is a reflection of the child objects. The mixed state isn't used as a third state for an individual item. To represent a third state, use radio buttons or a drop-down list instead.

  Incorrect:

  In this example, the mixed state is supposed to indicate that the Theme service isn't installed.

  Correct:

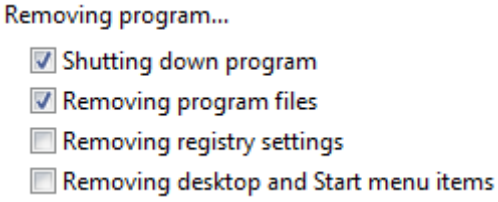
  

  In this example, users can choose from a list of three clear options.

- **Clicking a mixed state check box should cycle through all selected, all cleared, and the original mixed states.** For forgiveness, it's important to be able to restore the original mixed state because the settings might be complex or unknown to the user. Otherwise, the only way to restore the mixed state with confidence would be to cancel the task and start over.

- **Don't use check boxes as a progress indicator**. Use a progress indicator control instead.

  Incorrect:

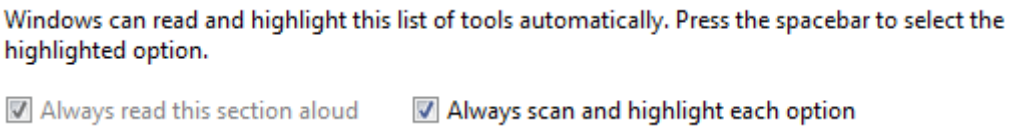
In this example, check boxes are used incorrectly as a progress indicator.

Correct:

Example of a typical progress bar.

- **Show disabled check boxes using the correct selection state.** Even though users can't change them, disabled check boxes convey information so they should be consistent with results.

  Incorrect:

  In this example, the "Always read this section aloud" option should be cleared because the section isn't read when the option is disabled.

- **Don't use the selection of a check box to**:
  - Perform commands.
  - Display other windows, such as a dialog box to gather more input.
  - Dynamically display other controls related to the selected control (screen readers cannot detect such events).

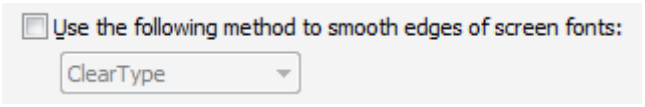
Don't show this <item> again

- **Consider using a Don't show this <item> again option to allow users to suppress a recurring dialog box only if there isn't a better alternative.** Try to determine beforehand if users really need the dialog; if they do, always show the dialog, and if they don't, eliminate the dialog.

For more guidelines and examples, see Dialog Boxes.

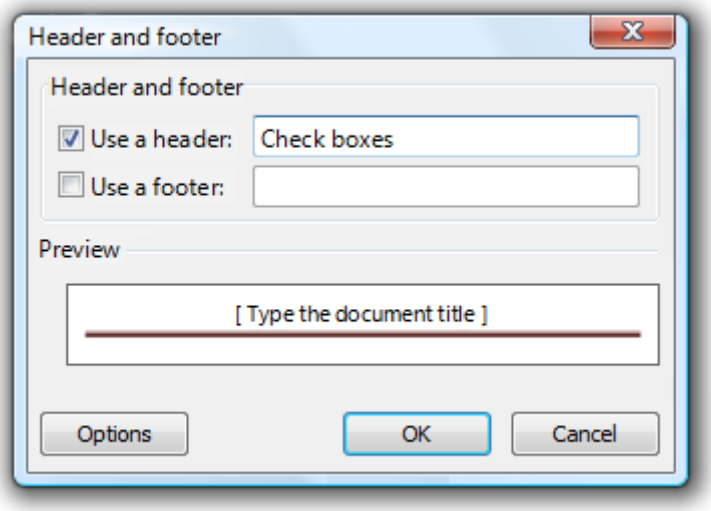
Subordinate controls

- Place subordinate controls to the right of or below (indented, flush with the check box label) the check box and its label. End the check box label with a colon.



In this example, the check box and its subordinate control share the check box label and its access key.

- **Leave dependent editable text boxes and drop-down lists enabled if they share the check box's label.** When users type or paste anything into the box, select the corresponding option automatically. Doing so simplifies the interaction.



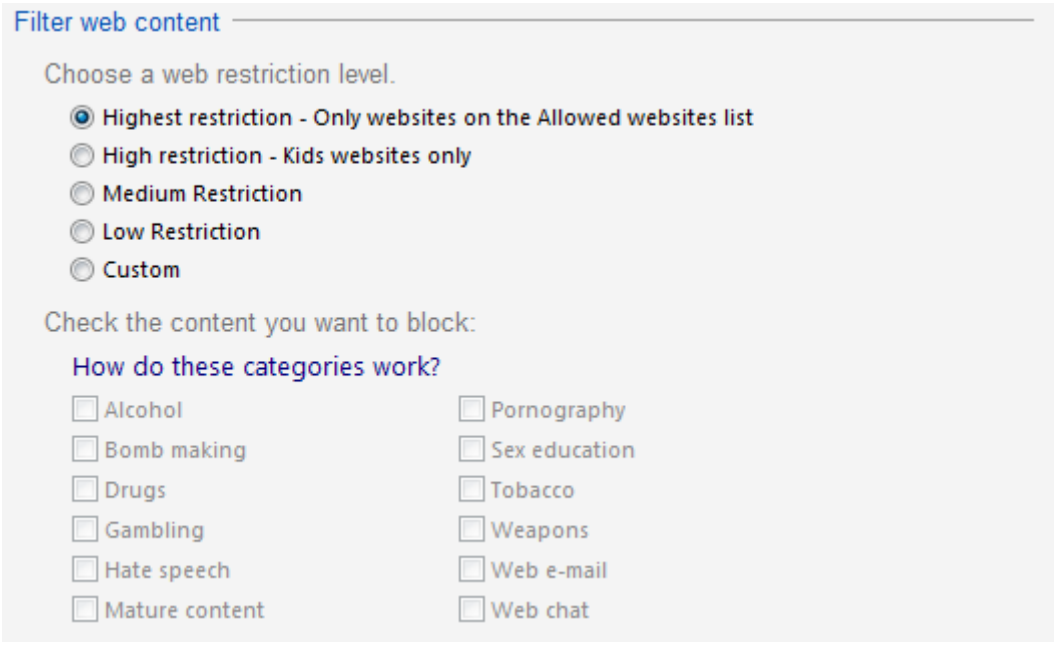
In this example, entering a header or footer automatically selects the option.

- If you nest check boxes with radio buttons or other check boxes, **disable these subordinate controls until the high-level option is selected**. Doing so avoids confusion about the meaning of the subordinate controls.

- Make subordinate controls to a check box contiguous with the check box in the tab order.

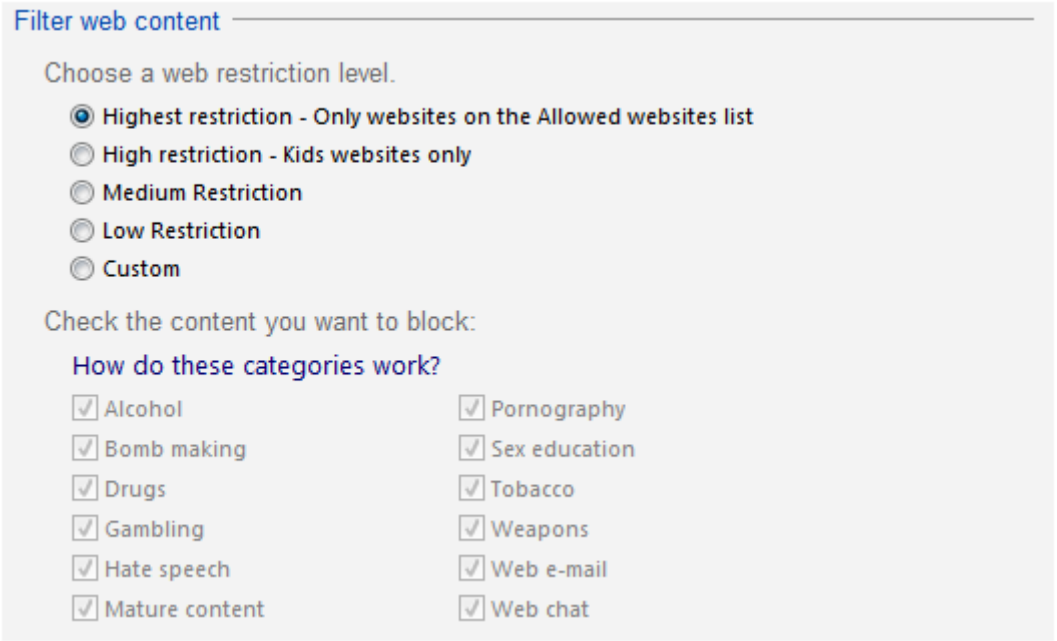
- **If selecting an option implies selecting subordinate check boxes, explicitly select those check boxes to make the relationship clear.**

  Incorrect:



In this example, the subordinate check boxes aren't selected.

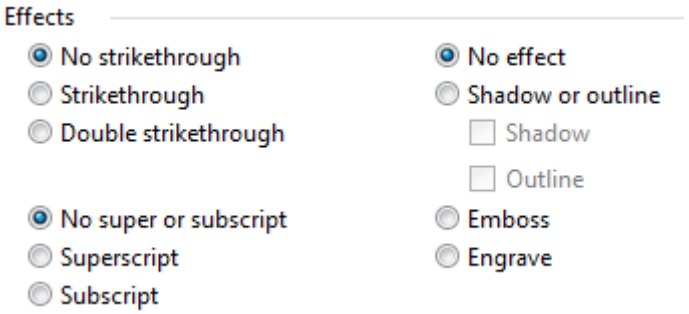
Correct:



In this example, the subordinate check boxes are selected, making their relationship to the selected option clear.

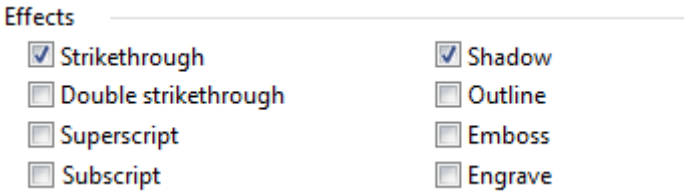
- **Use dependent check boxes if the alternatives add unnecessary complexity**. While check boxes should be independent options, sometimes alternatives such as radio buttons add unnecessary complexity.

  Correct:



In this example, the use of radio buttons is accurate, but creates unnecessary complexity.

Better:



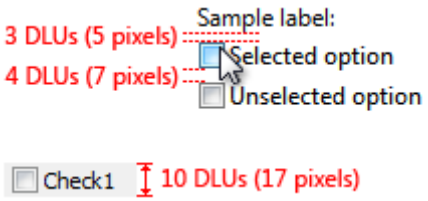
In this example, the use of check boxes is simpler and allows users to focus on selecting the desired options instead of on their complex relationship.

Important: Apply this guideline only in extremely rare circumstances, when showing the dependencies adds significant complexity without adding clarity. In the previous example, it is unlikely that users would attempt to choose both superscript and subscript, and if they did, it would be easy to understand that they were exclusive options.

Default values

- If a check box is for a user option, **set the safest (to prevent loss of data or system access), most secure and private state by default.** If safety and security aren't factors, select the most likely or convenient value.

Recommended sizing and spacing



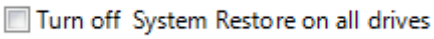
Recommended sizing and spacing for check boxes.

Labels

Check box labels

- Label every check box.
- Assign a unique access key to each label. For guidelines, see Keyboard.
- Use sentence-style capitalization.
- Write the label as a phrase or an imperative sentence, and use no ending punctuation.
  - **Exception:** If a check box label also labels a subordinate control that follows it, end the label with a colon.
- Write the label so that it describes the selected state of the check box.
- For a group of check boxes, use parallel phrasing and try to keep the length about the same for all labels.
- For a group of check boxes, focus the label text on the differences among the options. If all the options have the same introductory text, move that text to the group label.
- Use positive phrasing. Don't phrase a label so that selecting a check box means not to perform an action.
  - **Exception:Don't show this <item> again** check boxes.

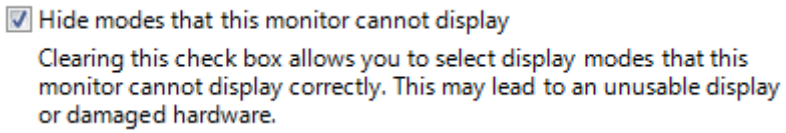
  Incorrect:

  In this example, the option doesn't use positive phrasing.
- Describe just the option with the label. Keep labels brief so it's easy to refer to them in messages and documentation. If the option requires further explanation, provide the explanation in a static text control using complete sentences and ending punctuation.

  > **Note**
  >
  > Adding an explanation to one check box in a group doesn't mean that you have to provide explanations for all check boxes in the group. Provide the

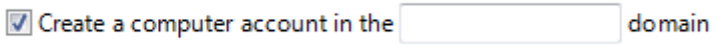
relevant information in the label if you can, and use explanations only when necessary. Don't merely restate the label for consistency.



In this example, a check box label has additional explanatory text beneath it.

- If an option is strongly recommended, consider adding "(recommended)" to the label. Be sure to add to the control label, not the supplemental notes.
- If you must use multi-line labels, align the top of the label with the check box.
- Don't use a subordinate control, the values it contains, or its units label to create a sentence or phrase. Such a design isn't localizable because sentence structure varies with language.

  Incorrect:

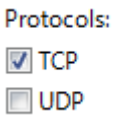


  In this example, the text box is incorrectly placed inside the check box label.

Check box group labels

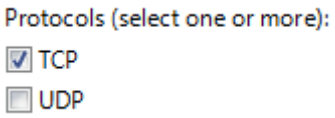
- Use the group label to explain the purpose of the group, not how to make the selection. Assume that users know how to use check boxes. For example, don't say "Select any of the following choices".
- End each label with a colon.
- Don't assign an access key to the label. Doing so isn't necessary and it makes the other access keys harder to assign.
- For a selection of one or more dependent choices, explain the requirement on the label.

  Correct:



  In this example, users might think that they can only make one selection.

Better:



In this example, it's clear that users can make more than one selection.

Documentation

When referring to check boxes:

- Use the exact label text, including its capitalization, but don't include the access key underscore or colon. Include the word check box.
- Refer to a check box as a check box, not option, checkbox, or just box, because box alone is ambiguous for localizers.
- To describe user interaction, use select and clear.
- When possible, format the label using bold text. Otherwise, put the label in quotation marks only if required to prevent confusion.

  Example: Select the **Underline** check box.

Feedback

Was this page helpful? Yes No

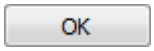
Provide product feedback | Get help at Microsoft Q&A

Command Buttons in Windows 7

Article • 10/18/2022

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

With command buttons in Windows 7, users initiate immediate actions.



A typical command button.

The default command button is invoked when users press the Enter key. It is assigned by the developer, but any command button becomes the default when users tab to it.

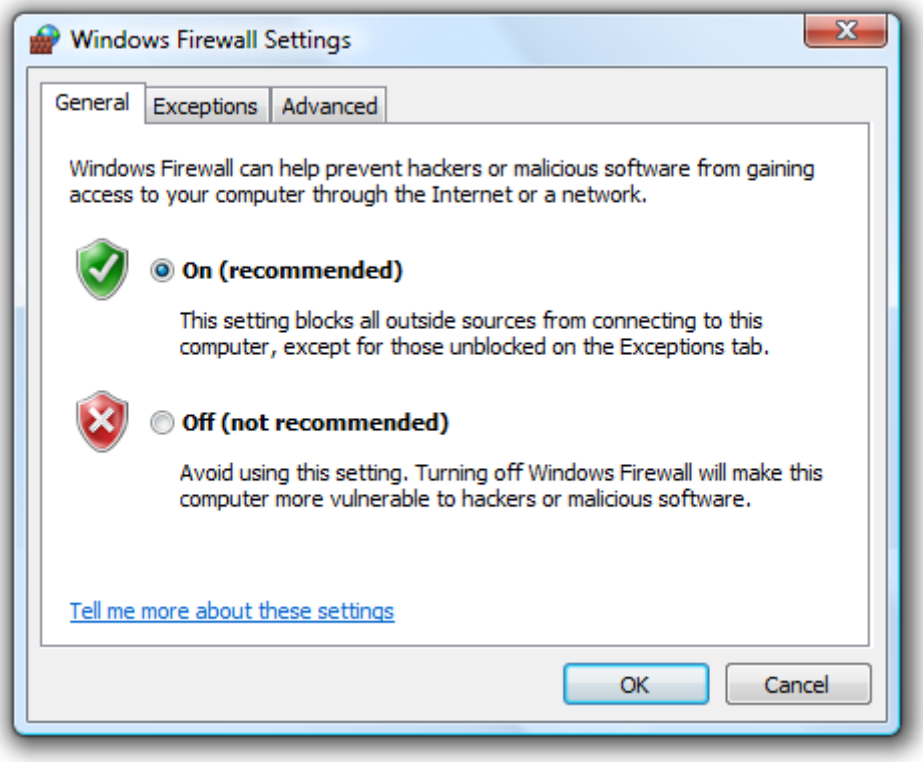
> **Note**
>
> Guidelines related to layout are presented in a separate article.

Is this the right control?

To decide, consider these questions:

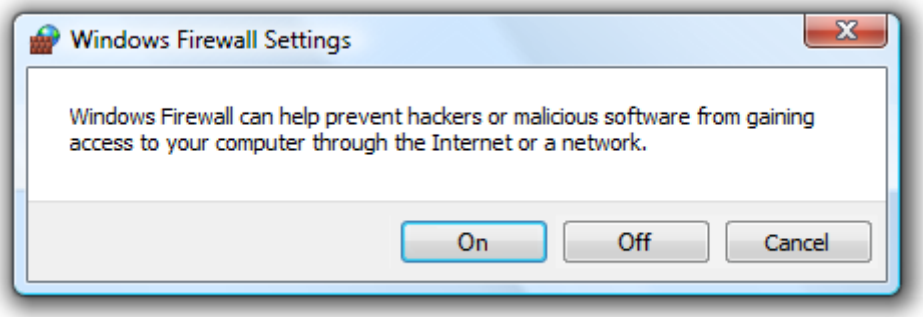
- **Is the command button used to initiate an immediate action?** If not, use another control.
- **Would a link be a better choice?** Use a link if:
  - The action is to navigate to another page, window, or Help topic. **Exception**: Wizard navigation uses Back and Next command buttons.
  - The command is embedded in a body of text.
  - The command is secondary in nature. That is, it does not relate to the primary purpose of the window. In this case, either a lightweight command button or link would be appropriate.
  - The command is part of a menu or group of related links.
  - The label is lengthy, consisting of five or more words, thus giving a command button an awkward appearance.

- **Would a combination of radio buttons and generic command buttons be a better choice?** Often radio buttons are used in conjunction with generic command buttons (OK, Cancel) in place of a set of specific command buttons when any of the following are true:
    - There are five or more possible actions.
    - Users need to view additional information before making a decision.
    - Users need to interact with the choices (perhaps to see additional information) before making a decision.
    - Users view the choices as options instead of different commands.



Correct:

In this example, radio buttons are combined with OK and Cancel buttons to provide additional information about the options.



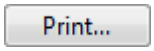
Incorrect:

In this example, command buttons alone make it difficult for users to make an informed decision.

Design concepts

Using ellipses

While command buttons are used for immediate actions, more information might be needed to perform the action. **Indicate a command that needs additional information (including confirmation) by adding an ellipsis at the end of the button label**.



In this example, the Print... command displays a Print dialog box to gather more information.

By contrast, in this example the Print command prints a single copy of a document to the default printer without any further user interaction.

Proper use of ellipses is important to indicate that users can make further choices before performing the action, or even cancel the action entirely. The visual cue offered by an ellipsis allows users to explore your software without fear.

This doesn't mean you should use an ellipsis whenever an action displays another window. Use an ellipsis only when additional information is required to perform the action. Consequently, **any command button whose implicit verb is to "show another window" doesn't take an ellipsis**, such as with the commands About, Advanced, Help (or any other command linking to a Help topic), Options, Properties, or Settings.

Generally, ellipses are used in user interfaces to indicate incompleteness. Commands that show other windows aren't incomplete; they must display another window and additional information isn't needed to perform their action. This approach eliminates screen clutter in situations where ellipses have little value.

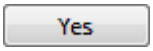
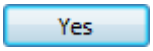
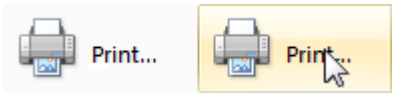
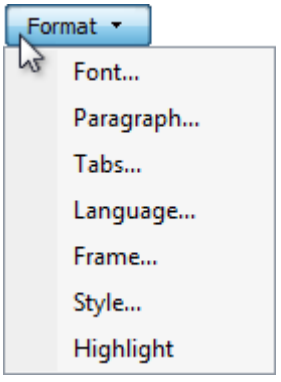
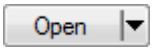
Note: When determining if a command button needs an ellipsis, don't use the need to elevate privileges as a factor. Elevation isn't information needed to perform a command (rather, it's for permission) and the need to elevate is indicated with the security shield.

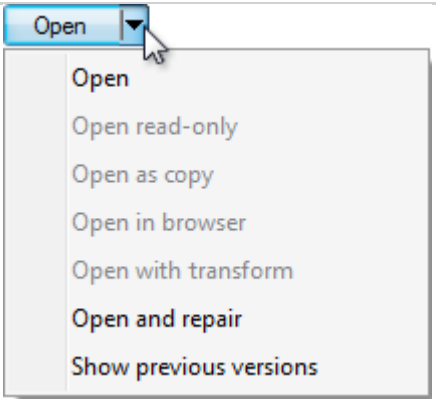
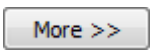
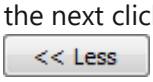
If you do only one thing... Use a concise, specific, self-explanatory label that clearly describes the action that the command button performs, and use an ellipsis when appropriate.

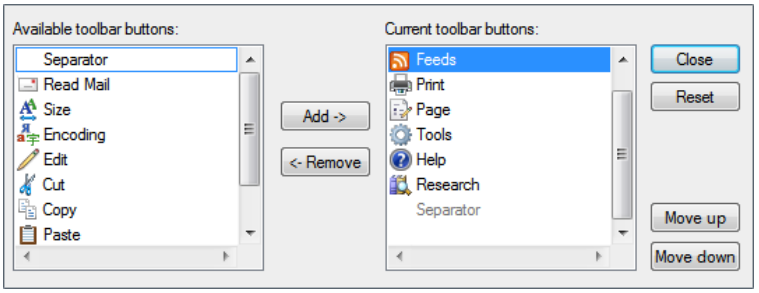
Usage patterns

Command buttons have several usage patterns:

Expand table

Usage	Example
Standard command buttons You can use standard command buttons to initiate an immediate action.	 A standard command button.
Default command buttons The default command button in a window indicates the command button that will be activated when users press the Enter key.	 A default command button. Any command button becomes the default when users tab to it. If the input focus is on a control that isn't a command button, the command button with the default button attribute becomes the default. Only one command button in a window can be the default.
Lightweight command buttons A lightweight command button is similar to a standard command button, except its button frame is shown only on mouse hover.	 In this example, the command has a very lightweight appearance (similar to a link) until the user hovers over the command, at which point it is drawn with a button frame. You can use lightweight command buttons in situations where you would use a standard command button, but you want to avoid always showing the button frame. Lightweight command buttons are ideal for commands that you want to underemphasize and using a link wouldn't be appropriate.
Menu buttons Use a menu button when you need a menu for a small set of related commands.	 A menu button with a small set of related commands. Use a menu button when a menu bar is undesirable, such as in a dialog box, toolbar, or other window that doesn't have a menu bar. A single downward-pointing triangle indicates that clicking the button will drop down a menu.
Split buttons Use a split button to consolidate a set of variations of a command, especially when one of the commands is used most of the time.	 a collapsed split button. like a menu button, a single downward-pointing triangle indicates that clicking the rightmost portion of the button will drop down a menu.

Usage	Example
	 a dropped-down split button. in this example, a split button is used to consolidate six variations of the open command. the regular open command is used most of the time, so users normally don't need to see the other commands. using a split button saves a significant amount of screen space, while also providing powerful choices. unlike a menu button, clicking the left portion of the button performs the action on the label directly. split buttons are effective in situations where the next action with a specific tool is likely to be the same as the last action. in this case, the label is changed to the last action, as with a color picker: In this example, the label is changed to the last action.
Browse buttons Use a browse button to display a dialog box to help users select a valid value.	dialog boxes launched by a browse button help users select files, folders, computers, users, colors, and so on. they are typically combined with an unconstrained control such as a text box. they're usually labeled browse, other, or more, and always have an ellipsis to indicate that more information is required. a text box with a browse button. for windows that have many browse buttons, you can use a short version: A short browse button.
Progressive disclosure buttons Use a progressive disclosure button to show or hide infrequently used options.	hiding infrequently used options until they are needed is called progressive disclosure. double chevrons are used to indicate progressive disclosure, and they point in the direction in which the revealing or hiding will take place:  after the button is clicked, its label changes to indicate that the next click will have the opposite effect: 

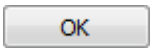
Usage	Example
	For more information and examples, see Progressive Disclosure Controls.
Directional buttons Use a directional button to indicate the direction in which an action will take place.	in this case, a single angle bracket is used instead of a double chevron:  A directional button indicates the direction of action.

Guidelines

General

- **Display a busy pointer if the result of clicking a command button isn't instantaneous.** Without feedback, users might assume that the click didn't happen and click again.
- If the same command button appears in more than one window, **try to use the same label text and access key, and locate it in approximately the same place in each window when practical.**
- **For command buttons with text labels, use a minimum button width and the standard command button height.** For more information, see Recommended sizing and spacing.
- For each window **make the command buttons the same width**. If that's impractical, limit the number of different widths for command buttons with text labels to two.
- When another control interoperates with a command button, such as a text box with a Browse button, **denote the relationship by placing the command button in one of three places:**
  - To the right of and top-aligned with the other control.
  - Below and left-aligned with the other control.
  - Vertically centered between controls that interoperate (such as Add and Remove buttons between two interoperating list boxes).
- If multiple command buttons interoperate with the same control, **vertically stack them to the right of and top-aligned with the other control, or horizontally place them left-aligned under the control.**

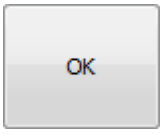
- When command buttons are subordinate to other controls, **use the above placement and disable the subordinate command button until the superior control is selected.**
- **Don't use narrow, short, or tall command buttons with text labels** because they tend to look unprofessional. Try to work with the default widths and heights.

Correct: 

In this example, the button size is standard and looks professional.

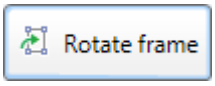
Incorrect:

In this example, the button is too small.

Incorrect: 

In this example, the button has too much space around the label.

- **Avoid combining text labels and graphics on command buttons.** Combining text and graphics usually adds unnecessary visual clutter and does not improve the user's comprehension. Consider combining text and graphics only when the graphic aids in comprehension, such as when it is a standard symbol for the command or it helps users visualize the results of the command. Otherwise, prefer text, but use either text or graphics.

Correct: 

In this example, the arrow graphic helps users visualize the results of the command.

Correct:

In this example, standard symbols are combined with text to aid comprehension

Incorrect:

In this example, the cancel graphic adds nothing to the text.

- **Don't use command buttons to set state**. Use radio buttons or check boxes instead. Command buttons are only for initiating actions.

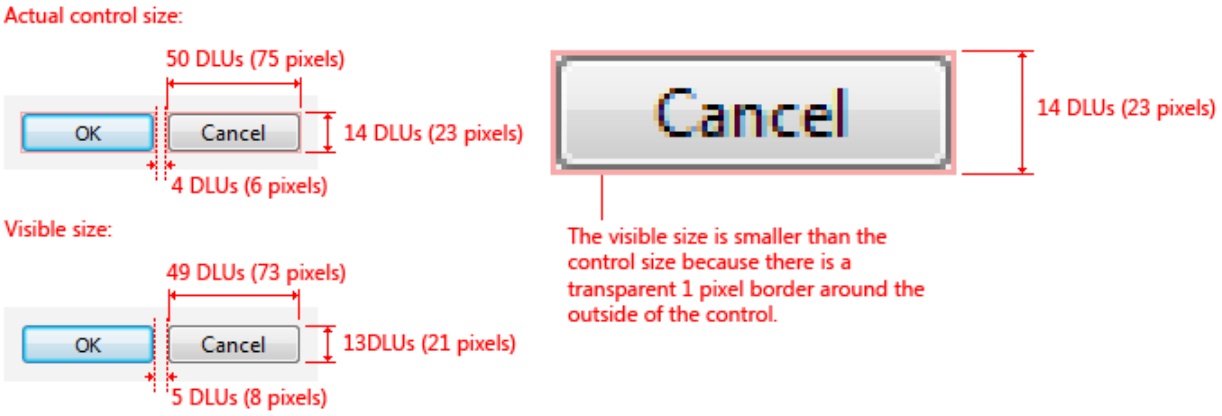
Split buttons

- **Make the most likely command the default behavior**. If there is more than one likely command, choose one that doesn't require additional information.
- **If the most likely command is the last user selection, change the button label to the last selection.**
- **Display the default command using bold text in the menu**. Doing so makes it easier for users to find the default command, especially when the default command is dynamic or the split button uses a graphic instead of a text label.

Default values

- Include a default command button on every dialog box. **Select the safest (to prevent loss of data or system access) and most secure command to be the default**. If safety and security aren't factors, select the most likely or convenient command.
- **Don't make a destructive action the default command button** unless there is an easy way to undo the command.

Recommended sizing and spacing



Recommended sizing and spacing for command buttons.

Labels

- Label every command button.

- If the button has a graphic label only, assign its Name property to an appropriate text label. This enables assistive technology products such as screen readers to provide users with alternative information about the graphic.

This example shows graphic buttons; internally, these buttons are labeled Previous, Next, and Copy.

- For short browse buttons (labeled "..."), the internal label should be Browse.
- Assign a unique access key. For guidelines, see Keyboard.

  Exceptions:
  - Don't assign access keys to OK and Cancel buttons, because Enter is the access key for the default button (which is usually the OK button), and Esc is the access key for Cancel. Doing so makes the other access keys easier to assign.
  - Don't assign access keys to short browse buttons (labeled "..."), because they can't be assigned uniquely.

- Prefer specific labels over generic ones. Ideally users shouldn't have to read anything else to understand the label. Users are far more likely to read command button labels than static text.
  - **Exception:** Don't rename the Cancel button if the meaning of cancel is unambiguous. Users shouldn't have to read all the buttons to determine which button cancels an action. However, rename Cancel if it is unclear what action is being canceled, such as when there are several pending actions.

  Acceptable:

  In this example, OK and Cancel are acceptable but unspecific labels.

  Better:

  In this example, Burn CD is more specific than OK.

  Incorrect:

  In this example, Cancel should be used instead of Don't Burn CD.

- Start labels with an imperative verb and clearly describe the action that the button performs. Don't use ending punctuation.

- **Exception:** The following standard labels are acceptable without verbs: Advanced, Back, Details, Forward, Less, More, New, Next, No, OK, Options, Previous, Properties, Settings, and Yes.

- While short labels are preferred, use enough text to explain the command sufficiently. Use a direct object (a noun after the verb) when the object is not apparent from context. Ideally users shouldn't have to read anything else to understand the label.

  Acceptable:

  In this example, a short label is acceptable if its meaning in context is readily apparent.

  Better: (if Add isn't clear)

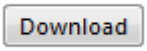
  In this example, adding a noun to the verb aids users' comprehension.

  Best: (if Add or Add items aren't clear)

  In this example, the label is self-explanatory.

- Use sentence-style capitalization. Doing so is more appropriate for Windows toneWindows tone and the use of short phrases for command buttons.
  - **Exception:** For legacy applications, you may use title-style capitalization if necessary to avoid mixing capitalization styles.

- Don't use now in command button labels because the immediacy of the command can be taken for granted.
  - **Exception:** When necessary, use now to differentiate commands that start a task from commands that perform a task immediately.

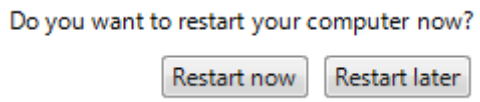
  In this example, clicking the command button goes to a window or page that allows users to download.

In this example, clicking the command button performs the download.

Only one command in a task flow should be labeled with now. So, for example, a **Download now** command should never be followed by another **Download now** command.

- Don't use later in command button labels if it implies an action that won't happen. For example, don't use Install later (in contrast to Install now) unless that command installs at a later time. Instead, use either Don't install or Cancel.

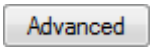
  Incorrect:

  In this example, Restart Later incorrectly implies that command restarts automatically at a later time.

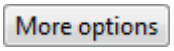
- Use an Advanced button only for options that are relevant to advanced users or require advanced user knowledge. Don't use an Advanced button for features that are considered technologically advanced. For example, a printer's stapling feature is not an advanced option, but its color management system is.

  Incorrect: (if the options really aren't advanced)

  In this example, Advanced is misleading.

  Correct:

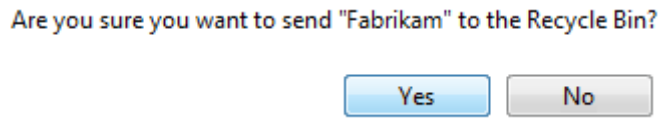
  

  In this example, the label is more specific and accurate.

- For command buttons that open other windows, choose a label that uses part or all of the secondary window's title bar text. For example, a command button labeled Browse might open a dialog box entitled Browse for Folder. Using the same terminology throughout the task helps to keep users oriented.

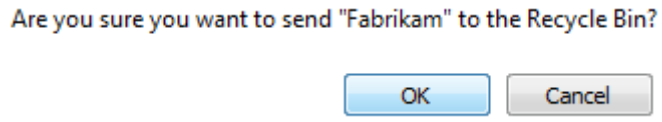
- When asking a question, use labels that match the question. Don't use OK/Cancel to answer Yes/No questions.

  Correct:



In this example, the buttons answer the question.

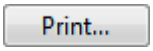
Incorrect:



In this example, the buttons don't answer the question.

- End the label with an ellipsis if the command requires additional information to execute.
  - **Exception:** Graphic labels don't take an ellipsis.

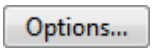
  Correct: (if a Print options dialog is displayed)

  In this example, after the button is clicked, the Print options dialog is displayed and requires more information from the user.

- Don't use an ellipsis when the successful completion of the action is simply to display another window. The following commands never take an ellipsis: About, Advanced, Options, Properties, Help.

  Incorrect:

  In this example, after the button is clicked, the Options dialog is displayed, but more information from the user is not required.

- In case of ambiguity (for example, the command label lacks a verb), decide based on the most likely user action. If simply viewing the window is a common action, don't use an ellipsis.

  Correct:

  More colors...

  Version information

In the first example, users are most likely going to choose a color, so using an ellipses is correct. In the second example, users are most likely going to view the version information, making ellipses unnecessary.

- For browse buttons, use short browse buttons (labeled "...") when there are more than two browse buttons in a window. Always use the short version when you want to display a browse button in a grid.
- For directional buttons, use a single angle bracket and have it point in the direction where the action takes place.

The following table shows some common command button labels and their usage.

Expand table

Button label	Meaning	Access key
Back	In wizards and task flows, go to the previous page.	'B'
Browse...	Display a dialog box to look for a file or object.	'B' or 'r'
Options	Display the choices available to users for customizing a program.	'O'
Pause	In progress dialog boxes, suspend the task.	'P'
Personalize	Customize a core experience that is crucial to the user's personal identification with a program.	'P'
Preferences	Don't use. Use Options instead.	Not applicable.
Properties	Display the attributes and settings for an object.	'P' or first 'r'
Save	Save a group of settings, or save a file or object using its current name.	'S'
Save as...	Save a file or object using a specified name.	Second 'a'
Settings	Don't use. Use Options instead.	Not applicable.
Troubleshoot	Don't use. Use a specific Help link instead.	Not applicable.

For guidelines about commit button labels (OK, Cancel, Yes/No, Close, Stop, Apply, Next, Finish, Done), see User Interface Text.

Documentation

When referring to command buttons:

- Use the exact label text, including its capitalization, but don't include the access key underscore or ellipsis. Don't include the word button.
- To describe user interaction, use click.
- When possible, format the label using bold text. Otherwise, put the label in quotation marks only if required to prevent confusion.

Example: Click **Print** to print the document.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

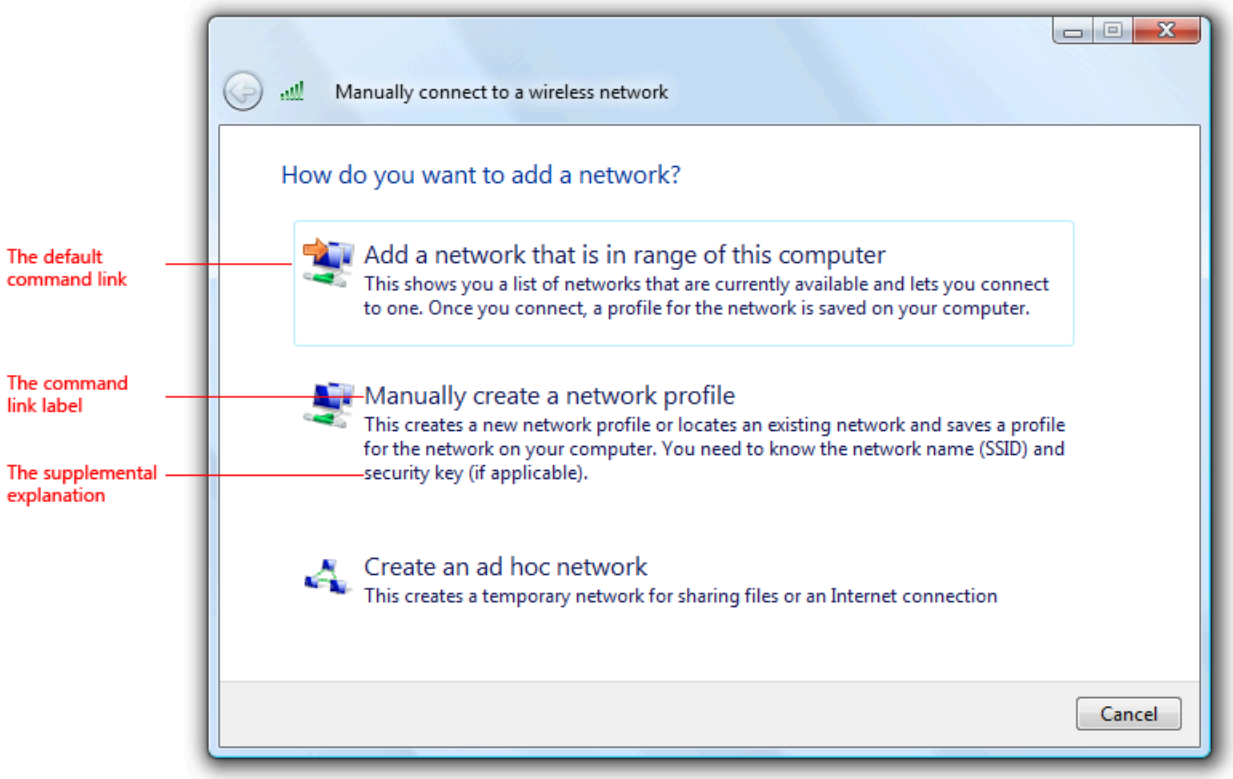
Command Links

Article • 06/04/2021

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

With command links, users select a single response to a main instruction and by doing so, move on to the next step in a task.

Command links have a clean, lightweight appearance that allows for descriptive labels, and are displayed with either a standard arrow or custom icon, and an optional supplemental explanation.



A typical set of command links.

Command links are similar to radio buttons in that they are used to select from a set of mutually exclusive, related choices. Like radio buttons, command links are always presented in sets, never individually. In appearance, command links have the lightweight appearance similar to regular links, without a frame or other strong click affordance. Command links are also similar to command buttons, in that they can be the default "command button" and they can have an access key assigned. Like commit buttons, on

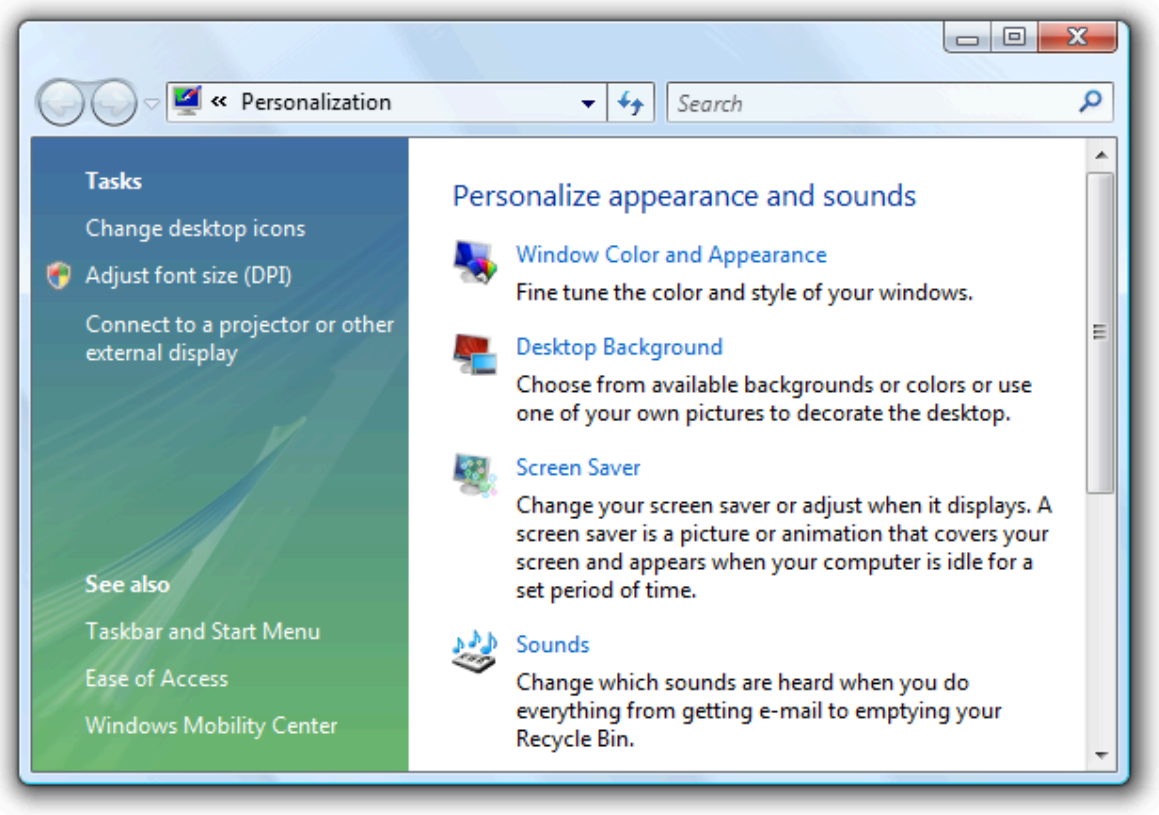
click they either close the window (for dialog boxes) or advance to the next page (for wizards and pages flows).

> ⓘ Note
>
> Guidelines related to links and layout are presented in separate articles.

Is this the right control?

To decide, consider these questions:

- **Are the options responses to the main instruction and related to the primary purpose of the window or page?** Must users respond to them to do something other than just navigating to a different page? If not, use another control such as command buttons or links. Command links aren't appropriate for secondary or optional choices, or pure navigation.

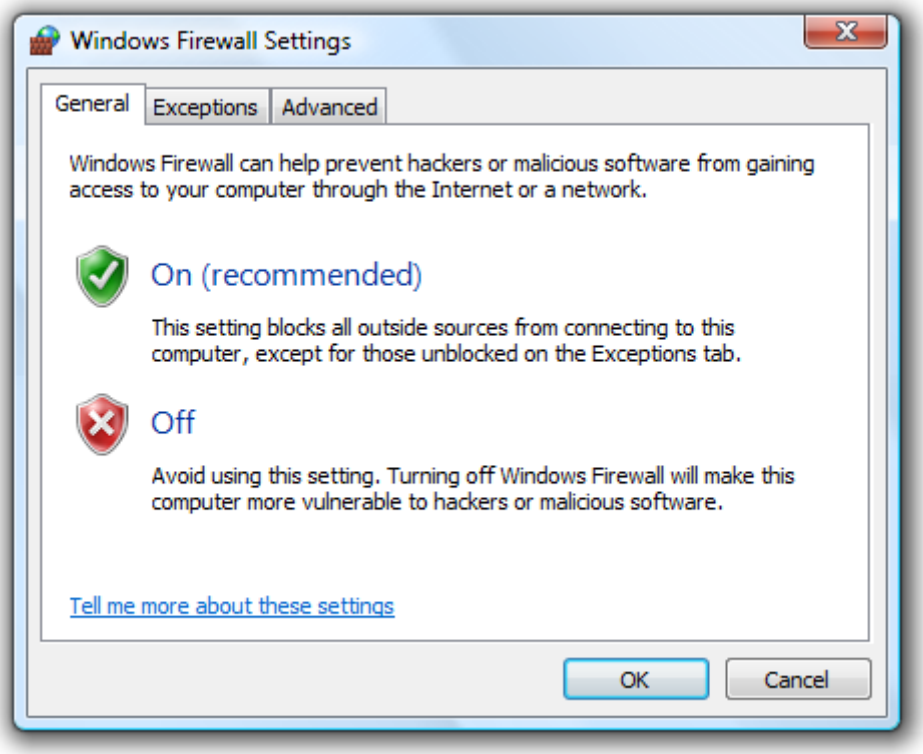
  

  While the Personalization Control Panel item looks like it is using command links, the options are regular links because this hub page is for pure navigation.

- **Is the control used to choose one response from a set of mutually exclusive responses?** If not, use another control. To let users choose individual commands, use command buttons or links.

- **For dialog boxes, does clicking the control close the window?** If not, use a control that doesn't require closing the window, such as radio buttons, command buttons, or links.

  Incorrect:

  Command links can't be used in property windows or tabbed dialogs because clicking the control closes the window.

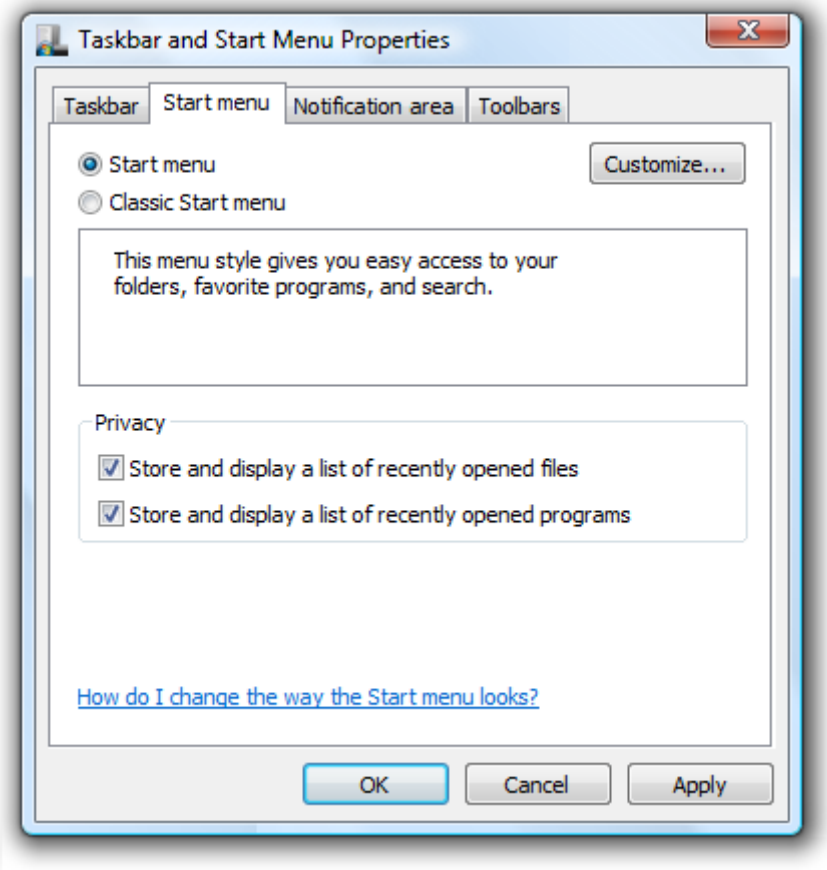
- **For wizards and page flows, does clicking advance to the next page without commitment?** Don't use command links to commit to a task; use commit buttons instead. Because command links look like links and users associate links with navigation within a page flow, links aren't appropriate for commit pages because users should always be able to back out.

- **For wizards and page flows, are other pages using command links?** If so, and all other factors being equal, prefer command links for consistency across pages.

- **Is the number of responses between two and five?** There should never be a single command link. Because command links are large controls and the screen space used is proportional to the number of options, keep the number of responses to five or fewer. For six or more options, use radio buttons, regular links, or a single-selection list view.

In this example, the AutoPlay feature in Microsoft Windows uses a list view.

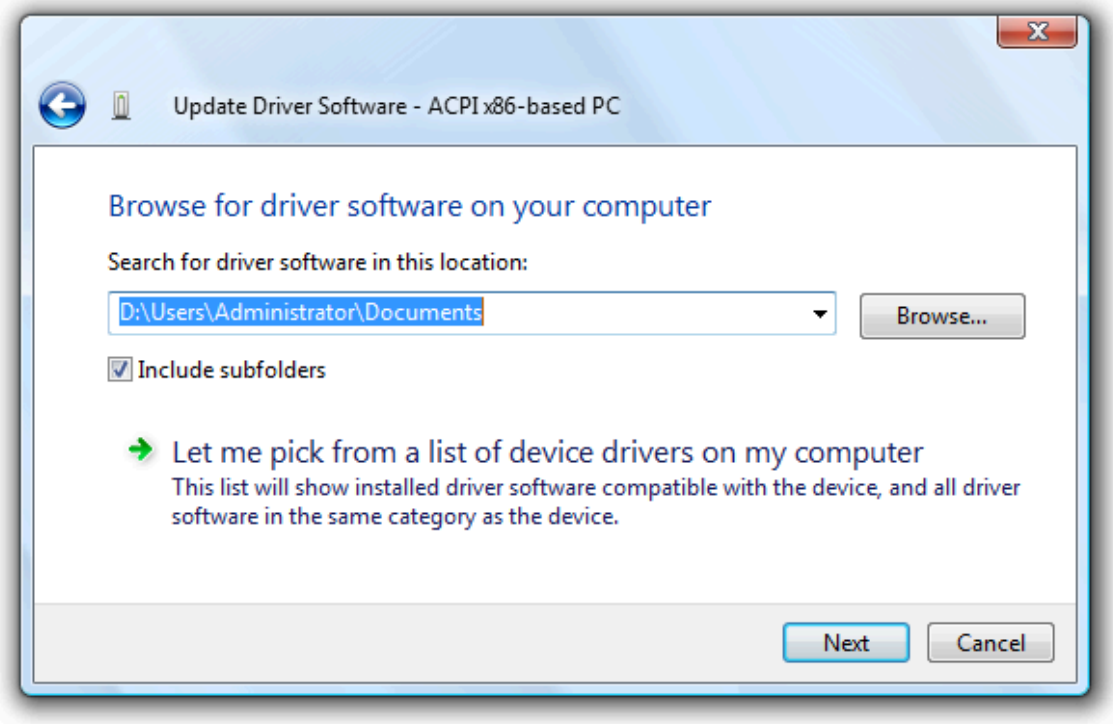
- **Would a combination of radio buttons and a commit button be a better choice?** Radio buttons are a better choice when any of the following are true:
    - **There is a strong default option that you want most users to select.** Users are less likely to change a default radio button than a default command link especially in a wizard, where users are accustomed to clicking Next to accept appropriate defaults. On the other hand, command links are a better choice if you want to encourage users to make an explicit choice.
    - **Users need to interact with the choices (perhaps to see additional information) before making a decision.** For example, selecting a radio button might display a description about the option dynamically.



In this example, selecting a radio button displays a description of the option.

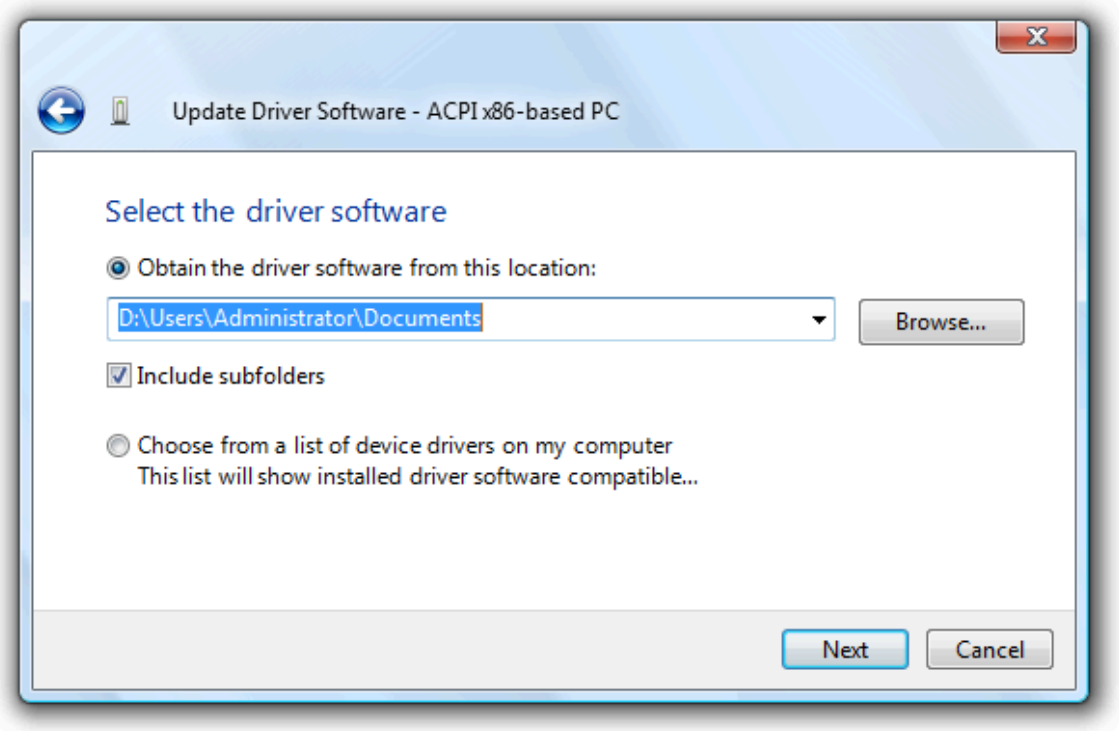
- **There are secondary or related options on the page.** Command links tend to dominate the page, making it easy to overlook everything else. Furthermore, once a command link is clicked, it's impossible to select secondary options.

  Incorrect:



In this example, there are two different ways to respond to the main instruction. A command link wasn't used for the first response because it would be difficult to select secondary options.

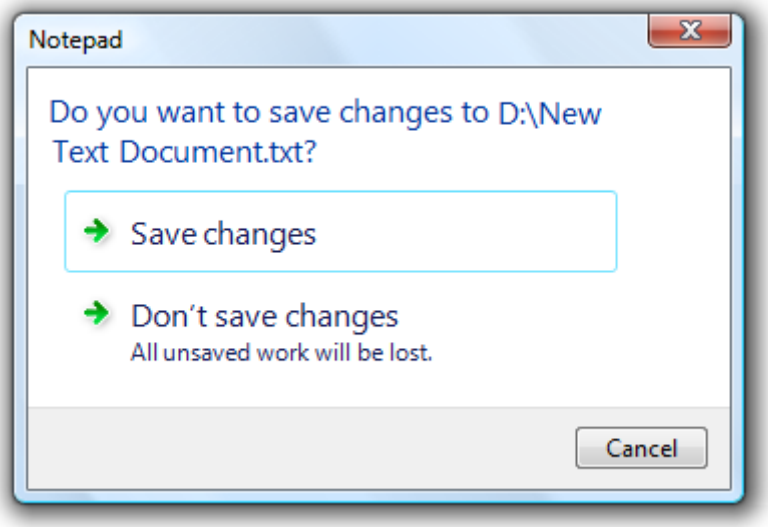
Correct:



In this example, radio buttons make the responses clear, while allowing users to select secondary options.

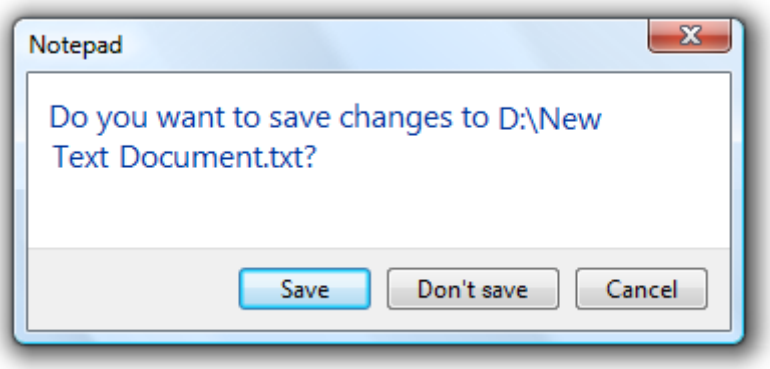
- **For dialog boxes, would a group of commit buttons be a better choice?**
  Command links work better when the options require longer, more explanatory responses and supplemental explanations, but a group of commit buttons is a better choice if there are a few simple options.

  Incorrect:



In this example, using command links for simple commands makes the dialog box unnecessarily complicated.

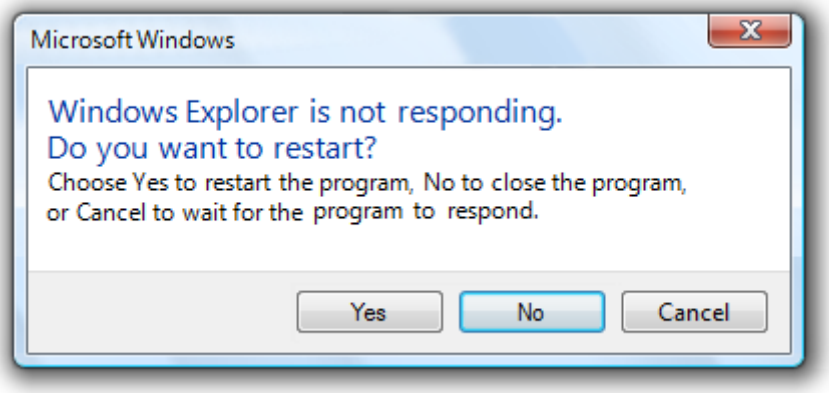
Correct:



In this example, using simple commit buttons gets right to the point.

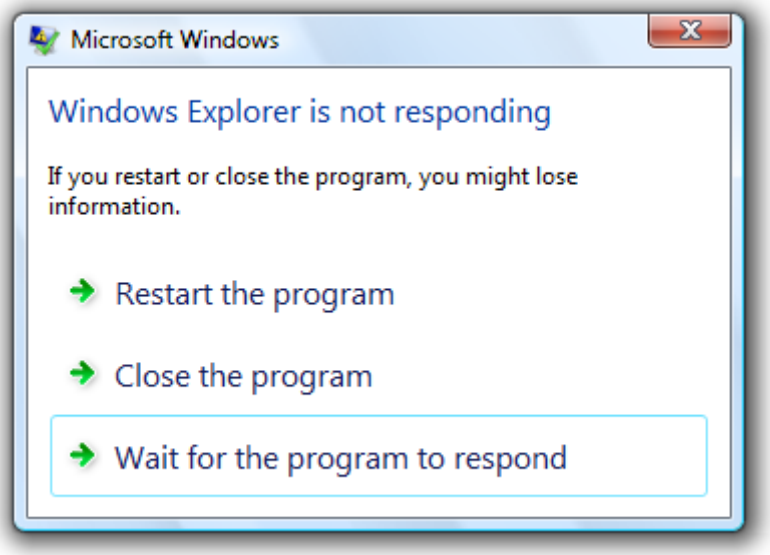
However, self-explanatory command links are always better choice when text is used to explain commit buttons.

Incorrect:



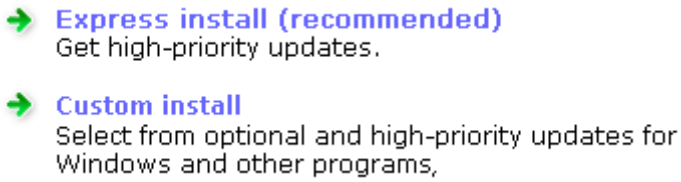
In this example, text is used to explain the commit buttons.

Correct:



In this example, the command links are self-explanatory.

> **Note**
>
> Command links require Windows Vista or later, so they aren't suitable for earlier versions of Windows. You can use regular links as a substitute.

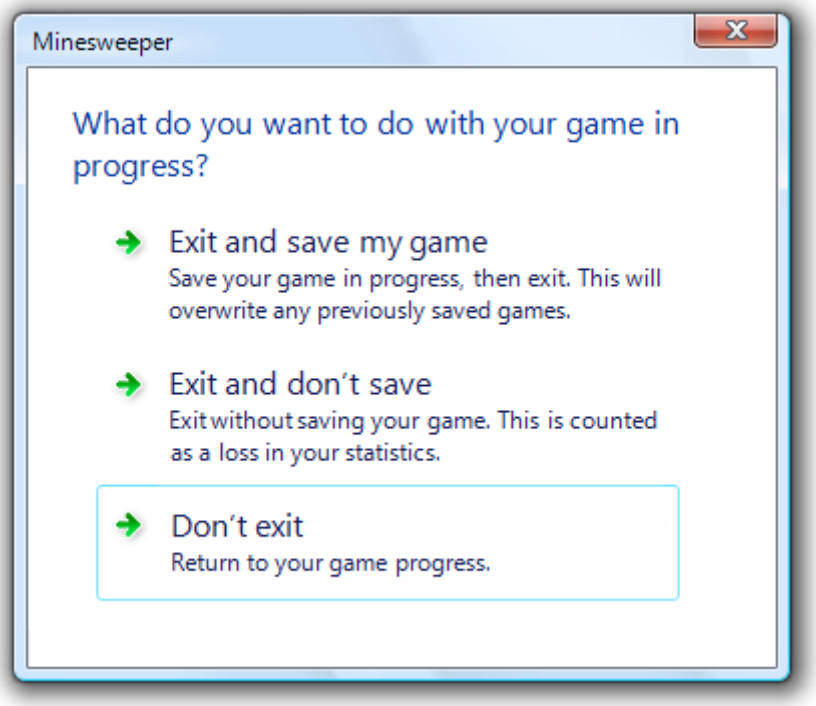


In this example, regular links with an icon and a supplemental explanation are used as a substitute for command links in Windows XP.

Design concepts

Just because command links allow you to use more descriptive labels and optional supplemental explanations doesn't mean you should. Consider the following example:

Incorrect:



This dialog box is seriously over-communicating.

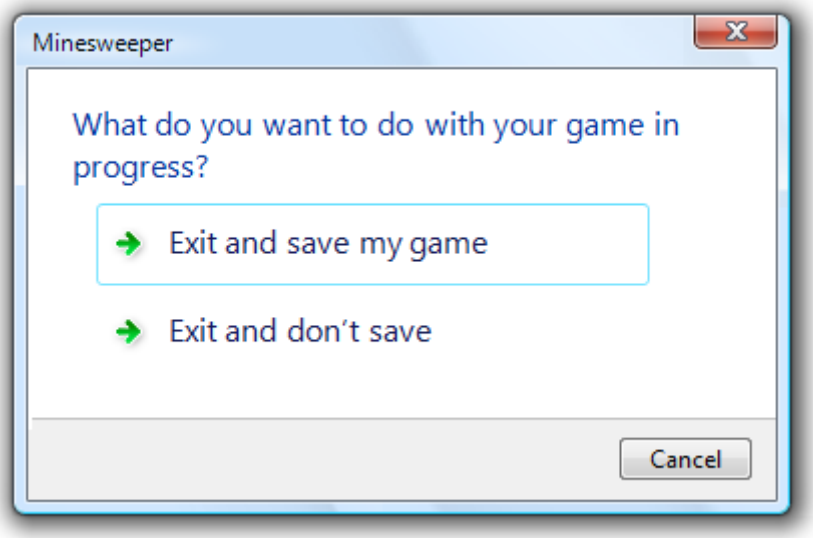
This dialog box takes a simple question and unnecessarily complicates it with command link text. Users don't want to read all that text for such simple questions.

We can simplify this dialog box by applying three command link guidelines:

- **Don't use a supplemental explanation that is a wordy restatement of the command link.** Use a supplemental explanation only when you can't make a command link self-explanatory. Providing a supplemental explanation for one command link doesn't mean that you have to provide them for all commands.
- **Select the safest (to prevent loss of data or system access) and most secure response to be the default.** If safety and security aren't factors, select the most likely or convenient response.
- **Provide an explicit Cancel button.** Don't use a command link for this purpose.

By applying these guidelines, we can eliminate the unnecessary supplemental explanations, make the most convenient response the default, and provide an explicit Cancel button.

Better:

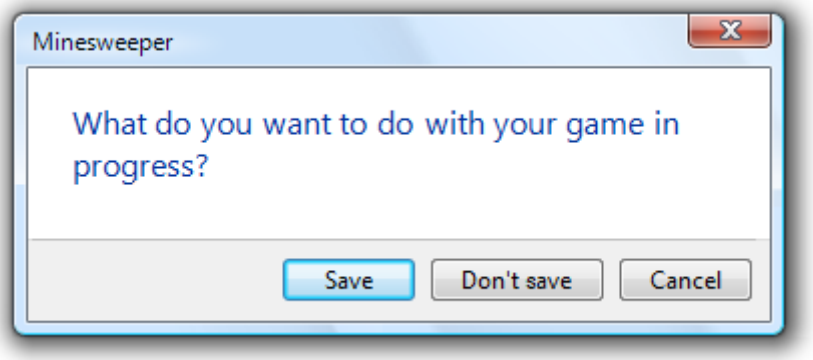


An improved version with simpler command links.

While it's true that this version doesn't explain explicitly that not saving is counted as a loss, few users will change their decision based on this information, making this a good tradeoff.

This dialog box could be made even better by analyzing whether or not command links are even the right control to use in this case. Commit buttons are actually a better choice, because longer, more explanatory responses aren't needed.

Best:



The correct version uses commit buttons to get right to the point.

Command links have many advantages, but when used unwisely they lead to over-communication. For dialog boxes, consider using commit buttons first and use command links only if commit buttons don't do the job well.

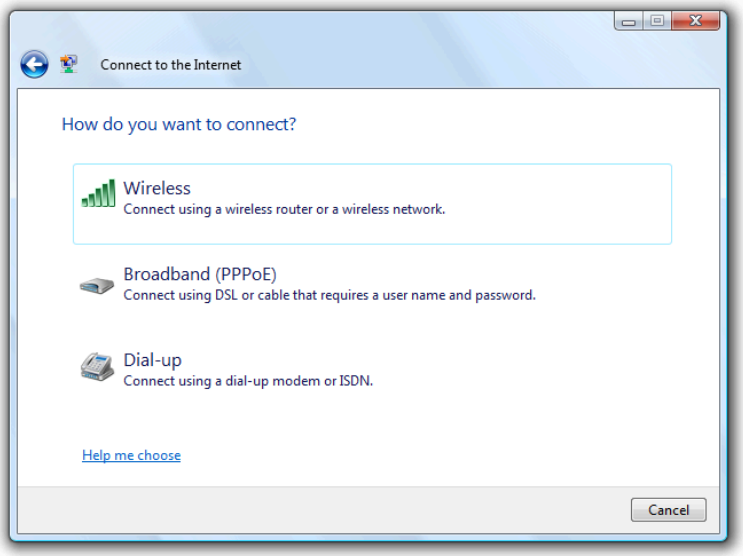
When used appropriately, command links should simplify and clarify your UI. If the results are the opposite, take a step back, review the alternatives, and focus on what you really need to communicate.

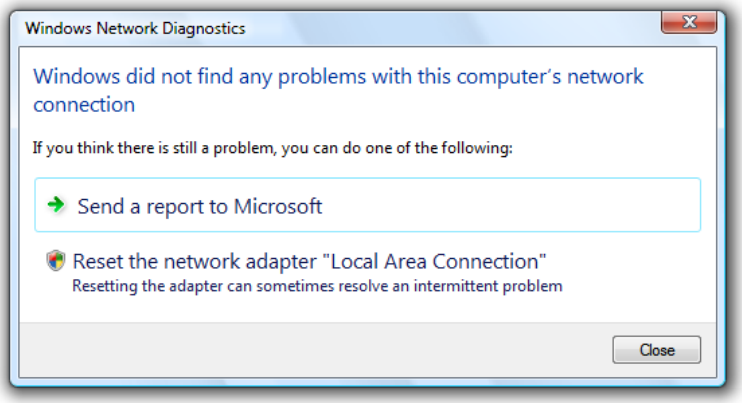
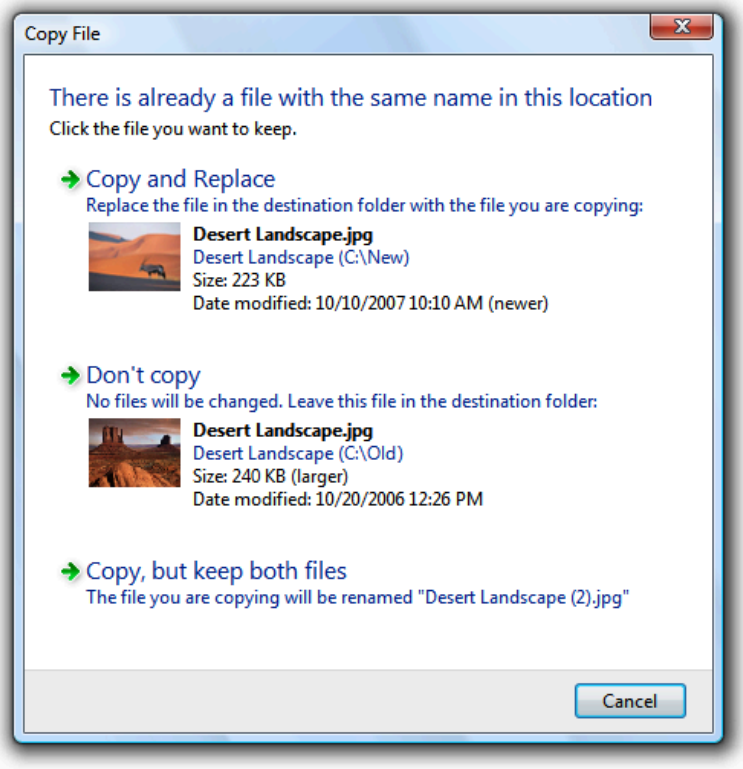
If you do only one thing... Don't use command links to over-communicate. Command links should simplify and clarify the communication, not make it more complicated.

Usage patterns

Command links have several usage patterns:

Expand table

Usage	Example
Page responses Command links are used to respond to the main instruction and advance to the next page.	with this pattern, the command links replace the next button, but there is still a cancel button. Page responses don't imply commitment. because command links look like links and users associate links with navigation within a page flow, links aren't appropriate for commit pages. users should always be able to back out.  In this example, command links are used to give descriptive responses to the main instruction. While radio buttons could be used here, command links allow users to respond with a single click.
Dialog box responses Command links are used to respond to the main instruction and close the dialog box.	with this pattern, the command links replace the commit buttons (such as ok), but there is still a cancel button. Unlike page flows, there is no way to back out of a dialog box-based response once it has been made. consequently, dialog box command links imply commitment.

Usage	Example
	 In this example, command links are used to give descriptive responses to the main instruction. While radio buttons could be used here, command links allow users to choose with a single click.
Detailed responses A page or dialog response that includes detailed information.	on occasion, users may need more detailed information to choose their response.  In this example, detailed command links are used so that users can make informed decisions. The thumbnails and file details help users decide.

Guidelines

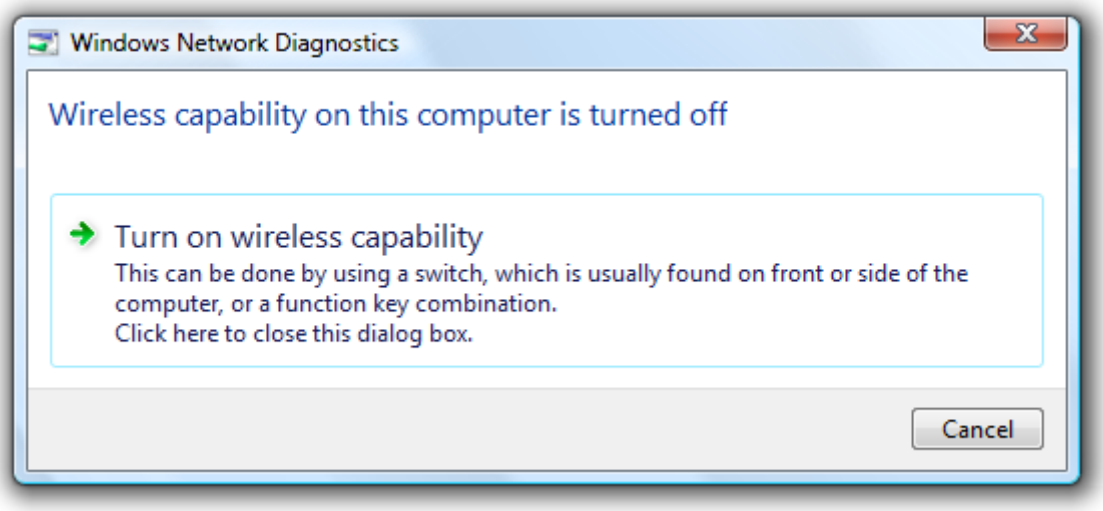
Interaction

- **Display a busy pointer if the result of clicking a command link isn't instantaneous.** Without feedback, users might assume that the click didn't happen and click again.

Presentation

- **Always present command links in a set of two or more.** Logically, there is no reason to ask a question that has only one answer.

  Incorrect:

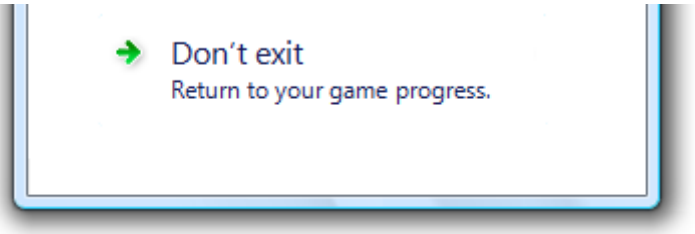
  

  In this example, the dialog box appears to be offering the user a choice, but there is just an instruction. This should be an informational dialog instead.

- **Present the most commonly used command links first.** The resulting order should roughly follow the likelihood of use, but also have a logical flow.
  - **Exception:** Command links that result in doing everything should be placed first.

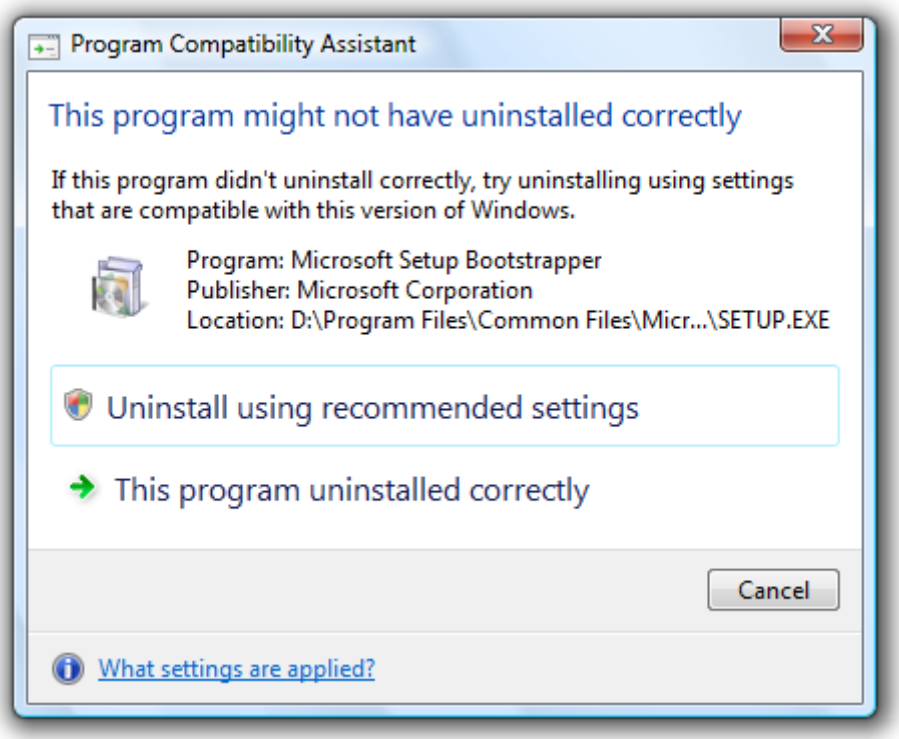
- **Provide an explicit Cancel button.** Don't use a command link for this purpose. Quite often users realize that they don't want to perform a task. Using a command link to cancel would require users to read all the command links carefully to determine which one means cancel. Having an explicit Cancel button allows users to cancel a task efficiently.

  Incorrect:

In this example, the Don't exit command link should be a Cancel button.

- **If providing an explicit Cancel button leaves a single command link, provide both a command link to cancel and a Cancel button.** Doing so makes it clear that users have a choice. Phrase this command link in terms of how it differs from the first response, instead of just "Cancel" or some variation.



In this example, the second command link indicates that the user has a choice, but all it does is cancel. However, it is phrased in terms of how it differs from the first command link.

- **Use Close instead of Cancel if you can't return the environment to its previous state, leaving no side effect.**

- **Don't display disabled command links.** If a command link doesn't apply to the current context, remove it instead. If removing all the command links that don't apply leaves a single command link, either eliminate the window or page, or display a confirmation if explicit user consent is needed.

Icons

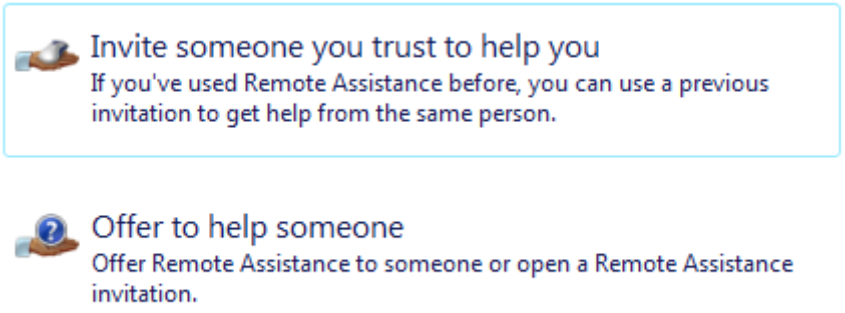
- **All command links need an icon.** The icons help users distinguish command links from regular links and user interface text.

- **Use the arrow icon only for command links.** Regular links shouldn't use the arrow icon unless they are being used as a substitute for command links in Windows XP.

- **Use the security shield icon to indicate that a response requires immediate elevation.** For additional guidelines on using the security shield icon, see the User Account Control.

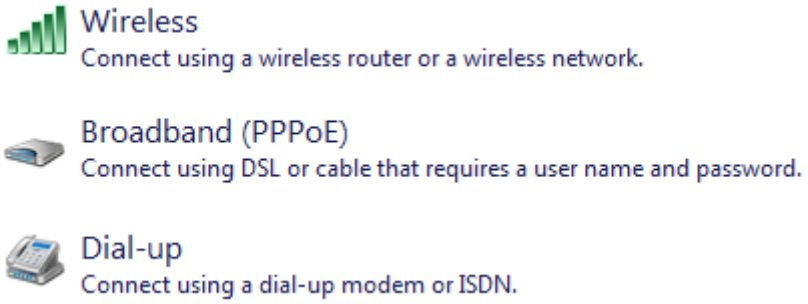
- **Use custom icons only if they help users visually identify and differentiate the options.** Don't use custom icons if they aren't immediately recognizable or meaningful.

  Incorrect:

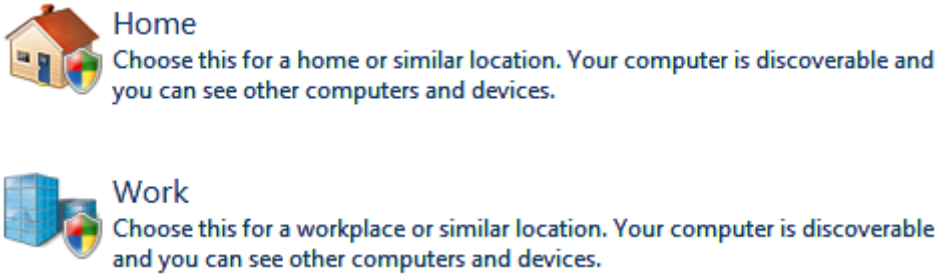
  

  In this example, the custom icons aren't immediately recognizable.

- **For custom icons, use 16x16 or 32x32 pixel icons.** Use the larger icons if there is sufficient space and they benefit visually from the larger size. If you need security shield overlays, use 32x32 or 48x48 pixel icons.

  This example uses 32x32 pixel custom icons.

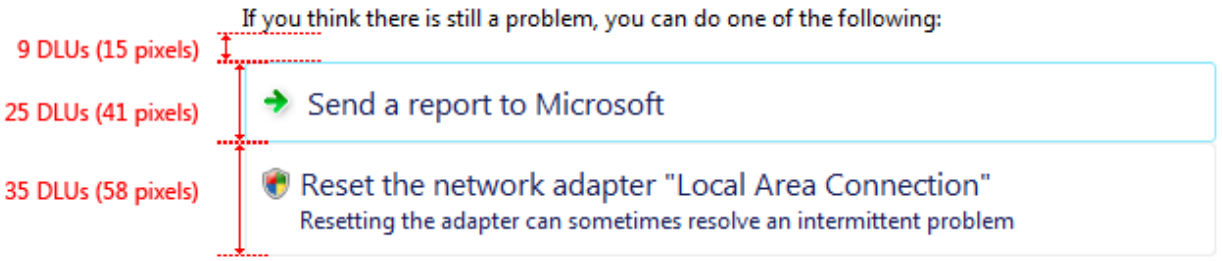
  This example uses 48x48 pixel custom icons, with a security shield overlay.

- **Avoid mixing custom icons with the standard arrow icon on a window or a page.** If you use a custom icon on a surface, try to use all custom icons. However, prefer the standard arrow icon over meaningless custom icons.

Default values

- **Select the safest (to prevent loss of data or system access) and most secure response to be the default.** If safety and security aren't factors, select the most likely or convenient response.
- **When practical, make the first response the default option** because users often expect that unless that order isn't logical.
- **For dialog boxes, don't make a destructive action the default command link** unless there is an easy way to undo the action.

Recommended sizing and spacing



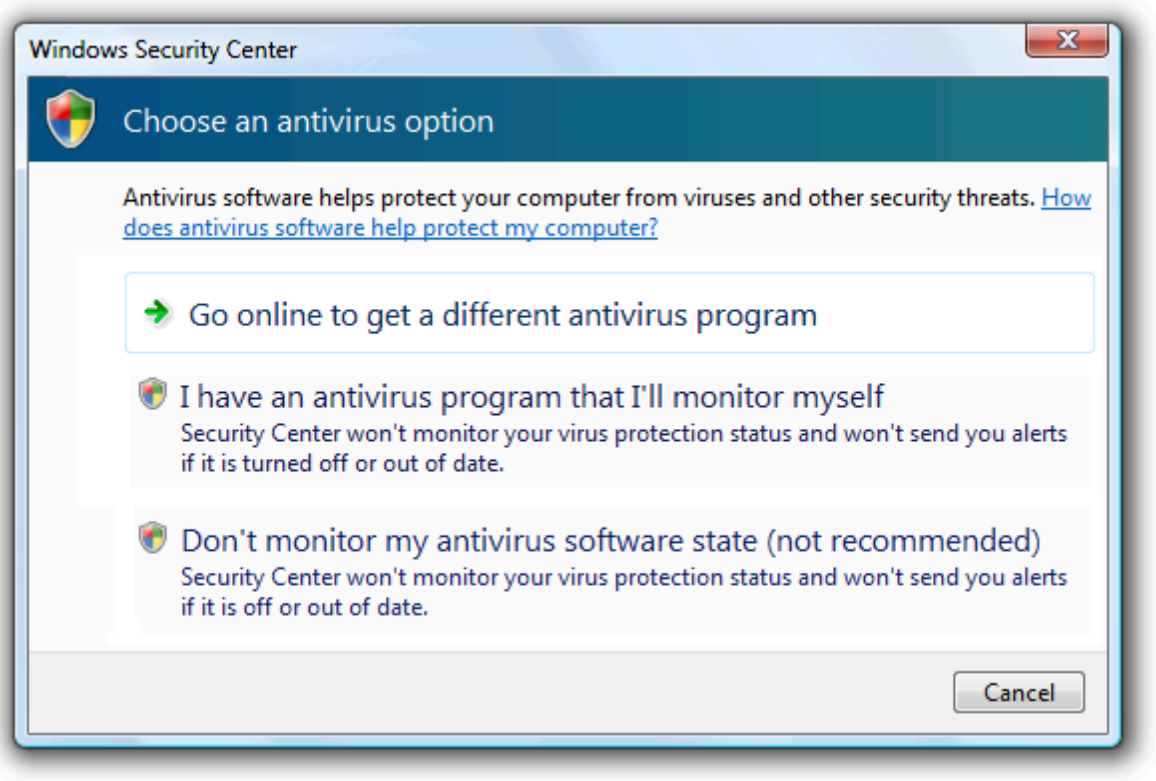
Labels

> ⓘ Note
>
> Because command links are responses to a main instruction, you should craft a good main instruction before determining its responses.

Command link labels

- **Choose a concise label that clearly communicates and differentiates what the command link does.** It should be self-explanatory and correspond to the main instruction. Focus the labels on the differences among the responses. Users shouldn't have to figure out what the command link really means or how it differs from other command links.

  Incorrect:



In this example, what is the difference between the second and third responses? Aren't you glad there's a Cancel button?

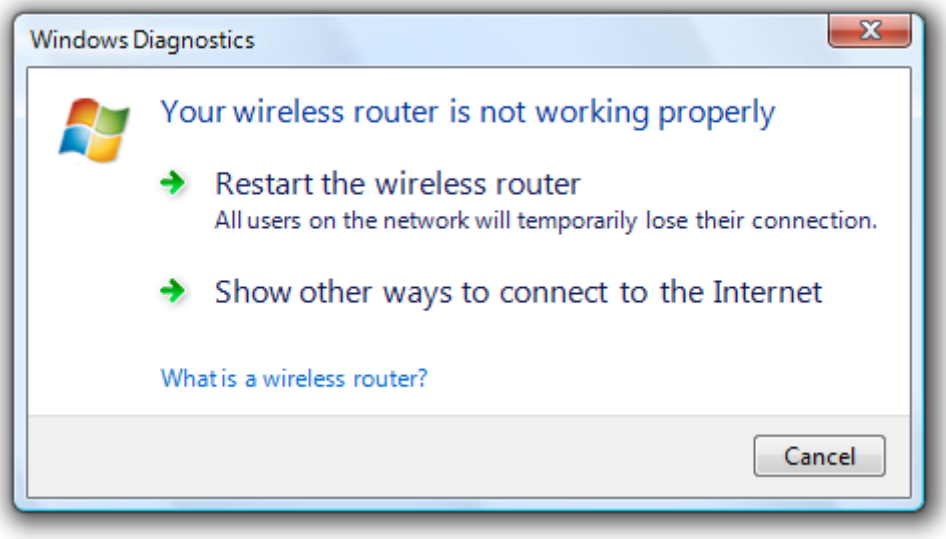
- **Focus command link labels on helping users make the right decision.** Omit details that don't affect the choice. The labels don't have to be a complete specification of what will happen.
- **Start command links with a verb.** Don't use click, however, because the label should communicate what the command link does, not how it works.
  - **Exception:** If all the command links begin with the same verb or phrase, eliminate the redundant verb or phrase.
- In general, **use positive phrasing** (providing a choice to do something). Negative phrasing (providing a choice not to do something) is acceptable if it makes the labels easier to understand.
- **Use parallel phrasing and single line labels.** Long labels discourage reading and shouldn't be necessary. Also, moderately sized labels are easier to refer to in documentation.
- **Use sentence-style capitalization.**
- **Don't use ending punctuation unless the label is a question.**
- **Assign a unique access key.** For guidelines, see Keyboard.

- **Don't use ellipses.** Ellipses mean that more information might be needed to perform the action. Properly used command links don't need ellipses because they have an immediate effect.
- **If a response is strongly recommended, add "(recommended)" to the label.** Be sure to add to the label, not the supplemental explanation.
- **If a response is intended only for advanced users, consider adding "(advanced)" to the label.** Be sure to add to the label, not the supplemental explanation.

Tip: You can evaluate command links by imagining that a friend stated the main instruction, and you responded with the command links. If responding with the command links would be unnatural or awkward, revise the command links and possibly the main instruction.

Supplemental explanations

- If a command link requires further explanation, **provide a supplemental explanation**. Supplemental explanations describe why users might want to choose a response or what happens if a response is chosen.



  In this example, the supplemental explanation describes the implications of the option.

- **Don't use a supplemental explanation that is wordy restatement of the command link.** Use a supplemental explanation only when you can't make a command link self-explanatory. Providing a supplemental explanation for one command link doesn't mean that you have to provide them for all.
- **Focus supplemental explanations on helping users make the right decision.** Omit details that don't affect the choice. The supplemental explanations don't have to be a complete specification of what will happen.

- **Use parallel phrasing and at most three lines of text.** Long supplemental explanations discourage reading and shouldn't be necessary.
- **Use complete sentences and ending punctuation.**

Command link group labels

- **Don't use group labels.** Main instructions act as the group label for command links.

Documentation

When referring to command links:

- Use the exact label text, including its capitalization, but don't include the access key underscore.
- If the label includes an object name, either omit the object name or use placeholder text.
- To describe the user interaction, use click.
- When possible, format the label using bold text. Otherwise, put the label in quotation marks only if required to prevent confusion.

Examples: To copy the picture, click **Copy and Replace**.

Click **Reset the network adapter**. (For a command link labeled "Reset the network adaptor *adaptor name*".)

Feedback

Was this page helpful? Yes No

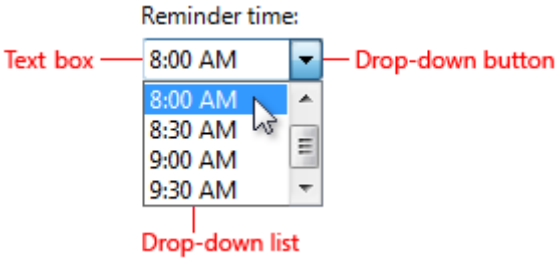
Provide product feedback | Get help at Microsoft Q&A

Windows 7 Drop-down Lists & Combo Boxes

Article • 02/08/2022

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

With a drop-down list or combo box, users make a choice among a list of mutually exclusive values. Users can choose one and only one option. With a standard drop-down list, users are limited to choices in the list, but with a combo box they can enter a choice that isn't in the list.



A typical combo box.

The following terms are important to understand as you read this article:

- A standard list box is a box containing a list of multiple items, with multiple items visible.
- 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.
- A combo box is a combination of a standard list box or a drop-down list and an editable text box, thus allowing users to enter a value that isn't in the list.
  - An editable drop-down list is a combination of a drop-down list and an editable text box.
  - An editable list box is a combination of a standard list box and an editable text box.

> **Note**

Guidelines related to layout are presented in a separate article.

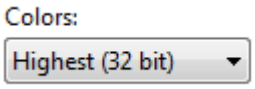
Is this the right control?

To decide, consider these questions:

- **Is the control used to choose one option from a list of mutually exclusive values?** If not, use another control. To choose multiple options, use a standard multiple-selection list, check box list, list builder, or add/remove list instead.
- **Are the options commands?** If so, use a menu button or split button instead. Use drop-down lists and combo boxes for objects (nouns) or attributes (adjectives), but not commands (verbs).
- **Does the list present data, rather than program options?** Either way, a drop-down list or combo box is a suitable choice. By contrast, radio buttons are suitable only for a small number of program options.

Drop-down lists

- **Is there a default option that is recommended for most users in most situations?** Is seeing the selected option far more important than seeing the alternatives? Consider using a drop-down list if you don't want to encourage users to make changes by hiding the alternatives. If not, consider radio buttons, a single-selection list, or an editable list box, which give more emphasis to the alternative choices.

  In this example, the highest color quality is the best choice for most users, so a drop-down list is a good choice to downplay the alternatives.

- **Do you want to draw attention to the option?** If so, consider radio buttons, a single-selection list, or an editable list box, which tend to draw more attention by taking more screen space. Because drop-down lists are compact, they are good choices for options that you want to underemphasize.

- **Is screen space at a premium?** If so, use a drop-down list because the screen space used is fixed and independent of the number of choices.

- **Are there other drop-down lists on the window?** If so, consider using a drop-down list for consistency.

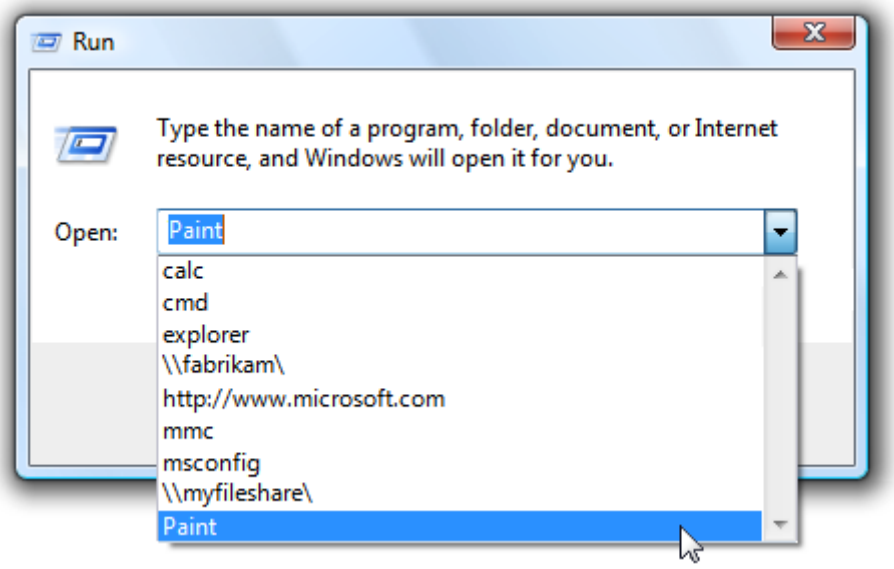
Editable drop-down lists

In addition to the principles just provided for drop-down lists, the following also apply:

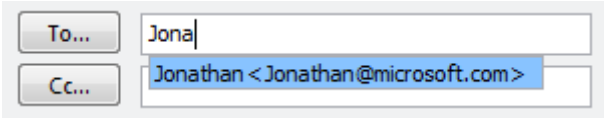
- **Are the possible choices constrained?** If so, use a normal drop-down list instead. Combo boxes are for unconstrained input, in which users may need to enter a value not currently in the list. Because the input is unconstrained, if users enter text that isn't valid you must handle the error with an error message.

- **Can you enumerate the most likely choices in advance**? If not, use a text box instead.

- **Is the drop-down list being used to list previous user input?** Unless users need to review the complete list of previous input, use a text box with the auto-complete option instead.

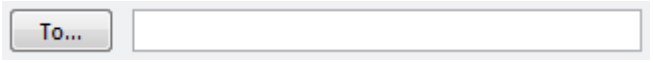
  

  In this example, users may need to review their previous input, so an editable drop-down list is a good choice.

  In this example, a text box with auto-complete is a good choice.

- **Will users need assistance in selecting valid values?** If so, use a text box with a Browse button instead.

  In this example, users can click "To" to help them select valid values.

- **Is it important to encourage users to review the alternative choices or invite change?** If so, consider using an editable list box instead. With an editable drop-

down list, users aren't going to be aware of the alternatives until the list is dropped.

- **Do users need to locate an item rapidly in a large list?** (Win32 only) If so, use a combo box because users can select an item by typing its full name. By contrast, the Win32 drop-down list selects items based only by the last character typed (so typing "Jun" into a list of months would match November, not June). In this case, use a combo box even if the possible choices are constrained.

Editable list boxes

- **Are the possible choices constrained?** If so, use a single-selection list or normal drop-down list instead. Combo boxes are for unconstrained input, where users may need to enter a value not currently in the list. Because the input is unconstrained, if users enter text that is not valid you must handle the error with an error message.
- **Can you enumerate the most likely choices in advance?** If not, use a text box instead.
- **Is it important to encourage users to review the alternative choices or invite change?** If not, consider an editable drop-down list instead.
- **Do you want to draw attention to the option?** If not, consider an editable drop-down list instead. Because drop-down lists are compact, they are good choices for options that you want to underemphasize.
- **Is screen space at a premium?** If so, use an editable drop-down list because the screen space used is fixed and independent of the number of choices.

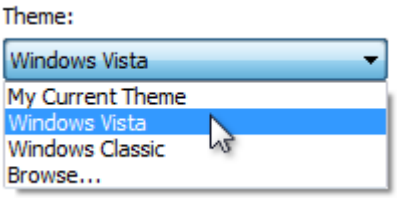
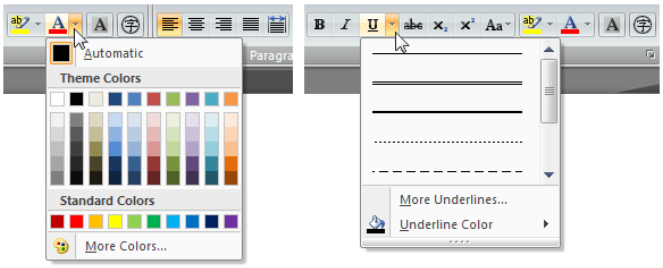
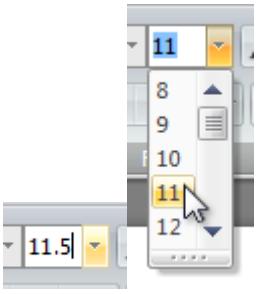
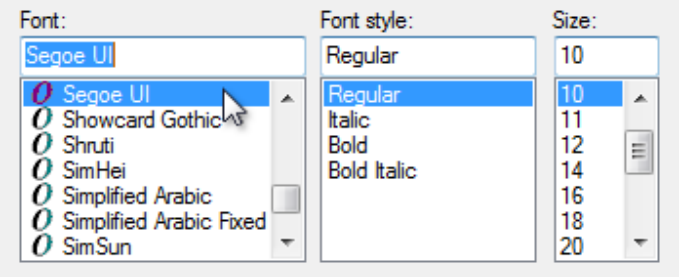
For drop-down lists, **the number of items in the list isn't a factor in choosing the control** because they scale from thousands of items all the way down to one. Editable drop-down lists scale from thousands of items down to none, because users can enter a value that isn't in the list. Because drop-down lists can be used for data, the number of items might not be known in advance and perhaps cannot be guaranteed. **Always include at least three items in editable list boxes to justify the additional screen space.**

Usage patterns

Drop-down lists and combo boxes have several usage patterns:

Expand table

Usage	Example
Drop-down list a standard drop-down list, with a fixed set of predetermined	When closed, only the selected item is visible. When users click the drop-down button, all the options

Usage	Example
values.	become visible. To change the value, users can open the list and click another value. In this example, the list is in its normal state.  In this example, the list has been dropped down.
Preview drop-down list a drop-down list that previews the results of the selection to help users choose.	 In these examples, the drop-down lists preview the results of the selection.
Editable drop-down list a drop-down combo box, which allows users to enter a value that isn't in the drop-down list.	 Examples of an editable drop-down list in edit and dropped-down modes. Use this control when you want to give the flexibility of a text box, yet want to assist users by providing a convenient list of likely choices.
Editable list boxes a regular combo box, which allows users to enter a value that isn't in the always visible list.	 In these examples, the editable list boxes are always displayed. This control is a better choice than the editable drop-down list when it is important to encourage users to review the alternative choices or invite change.

Guidelines

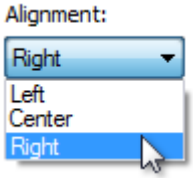
General

- **Don't use the change of a drop-down list or combo box to**:
  - Perform commands.
  - Display other windows, such as a dialog box to gather more input.
  - Dynamically display other controls related to the selected control (screen readers cannot detect such events).

Presentation

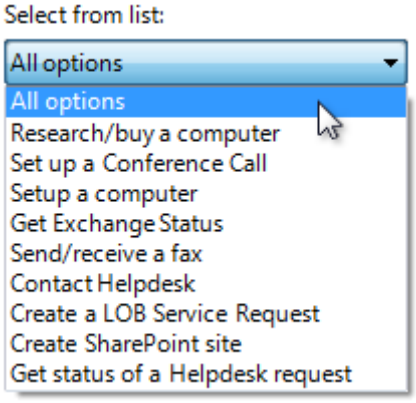
- **Sort list items in a logical order**, such as grouping highly related options together, placing most common options first, or using alphabetical order. Sort names in alphabetical order, numbers in numeric order, and dates in chronological order. Lists with 12 or more items should be sorted alphabetically to make items easier to find.

Correct:



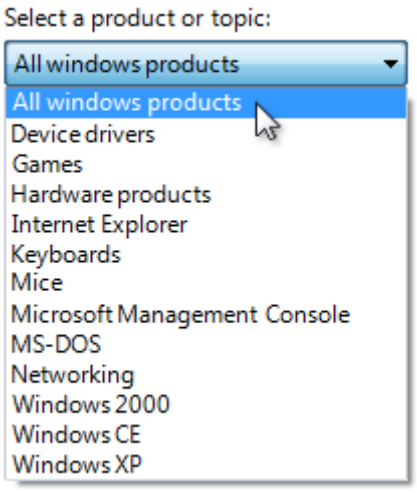
In this example, the list items are sorted by their spatial relationship.

Incorrect:



In this example, there are so many list items that they need to be sorted in alphabetical order.

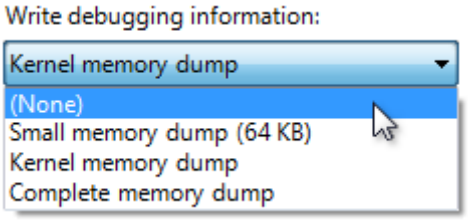
Correct:



In this example, the list items are sorted in alphabetical order except for the option that represents all items.

- **Place options that represent All or None at the beginning of the list, regardless of the sort order of the remaining items.**

- **Enclose meta-options in parentheses.**



In this example, "(None)" is a meta-option because it is not a valid value for the choice rather it describes that the option itself isn't being used.

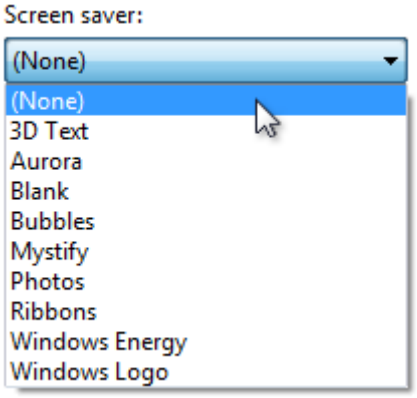
- **When disabling a drop-down list or combo box, also disable any associated labels and command buttons.**

Drop-down lists

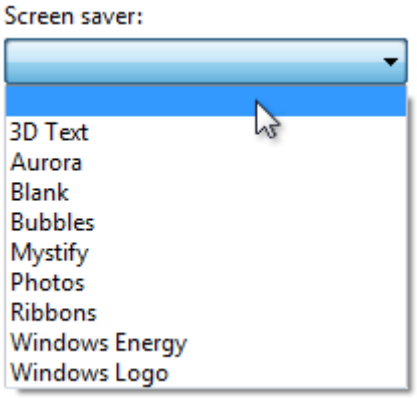
- When a single drop-down list is used to change the view of an associated control, **change the view immediately on selection instead of requiring a separate command button.** Use a separate command button only if the list takes a significant amount of time to render. However, list headers and menu buttons are the preferred controls for this purpose.

- **Don't have blank list items use meta-options instead**. Users don't know how to interpret blank items, whereas the meaning of meta-options is explicit.

Correct:



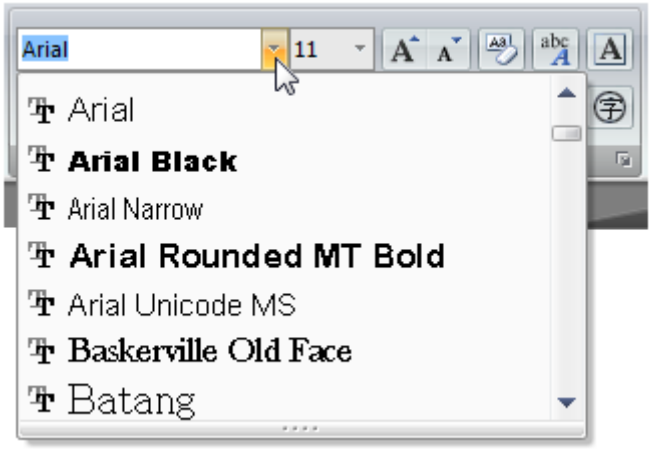
Incorrect:



In the incorrect example, the meaning of the blank option is unclear.

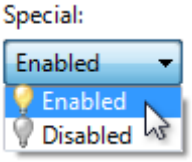
Preview drop-down lists

- **Use previews in the list items when it is better to show with images than describe using text alone.**



In this example, the preview explains the options far better than text alone.

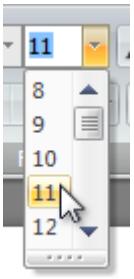
- **Don't use unnecessary, unhelpful icons in previews**.

Incorrect: 

In this example, the preview icons are unnecessary because they don't communicate any information.

Combo boxes

- **Limit the length of the input text when you can.** For example, if the valid input is a number between 0 and 999, use a combo box that is limited to three characters.
- **If there are many possible options, focus the list contents on the most likely options**. Because users can enter values that aren't in the list, combo boxes don't have to list all choices, just the likely choices or a representative sample.



In this example, many valid choices aren't listed, such as 15, or half-size fonts such as 9.5.

Default values

- **Select the safest (to prevent loss of data or system access) and most secure option by default.** If safety and security aren't factors, select the most likely or convenient option.
  - **Exception:** Display a blank default value if the control represents a property in a mixed state, which happens when displaying a property for multiple objects that don't have the same setting.

Prompts

A prompt is a label or short instruction placed inside an editable drop-down list as its default value. Unlike static text, prompts disappear from the screen once users type something into the combo box or it gets input focus.

A typical prompt.

Use a prompt when:

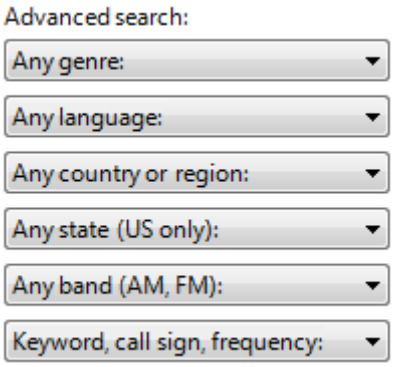
- Screen space is at such a premium that using a label or instruction is undesirable, such as on a toolbar.
- The prompt is primarily for identifying the purpose of the list in a compact way. It must not be crucial information that users need to see while using the combo box.

Don't use prompts just to direct users to select something from the list or to click buttons. For example, prompts like Select an option or Enter a filename and then click Send are unnecessary.

When using prompts:

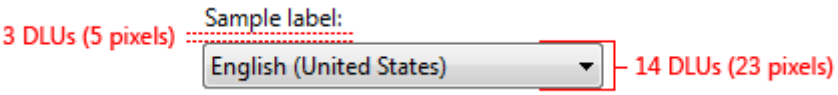
- Draw the prompt text in italic gray and real text in normal black. The prompt text must not be confused with real text.
- Keep the prompt text concise. You can use fragments instead of full sentences.
- Use sentence-style capitalization.
- Don't use ending punctuation or ellipsis.
- The prompt text should not be editable, and should disappear once users click in or tab into the text box.
  - **Exception:** The prompt is displayed if the text box has default input focus, and only disappears once the user starts typing.
- The prompt text is restored if the text box is still empty when it loses input focus.

Incorrect:



In this example, screen space is not at a premium; once an editable drop-down list is filled out, it is difficult for users to remember what it is for; and the prompt text is editable and drawn the same way as real text.

Recommended sizing and spacing



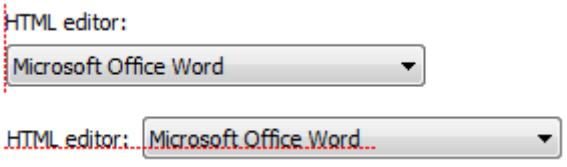
Recommended sizing and spacing for drop-down lists and combo boxes.

- **Choose a width appropriate for the longest valid data.** Drop-down lists cannot be scrolled horizontally, so users can see only what is visible in the control. (Note, however, that combo boxes can have AutoScroll functionality enabled.)
- **Include an additional 30 percent** (up to 200 percent for shorter text) for any text (but not numbers) that will be localized.
- **Choose a list length that eliminates unnecessary vertical scrolling.** Because drop-down lists are displayed on demand, their lists should show up to 30 items. Editable list boxes (those that don't have a drop-down button) should show between 3 and 12 items.

Labels

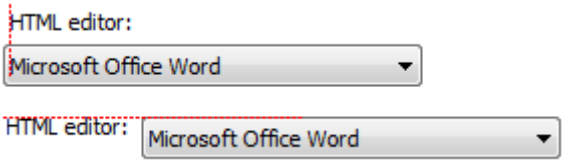
Control labels

- Write the label as a word or phrase, not as a sentence, and end it with a colon.
  Exceptions:
  - Editable drop-down lists with prompts located where space is at a premium.
  - If a drop-down list or combo box is subordinate to a radio button or check box and is introduced by its label ending with a colon, don't put an additional label on the control.
- Assign a unique access key for each label. For guidelines, see Keyboard.
- Use sentence-style capitalization.
- Position the label either to the left of or above the control, and align the label with the left edge of the control. If label is on the left, vertically align the label text with the control text.

  Correct:

  In this example, the label is correctly aligned with the control text.



Incorrect:

In this example, the label is incorrectly aligned with the control text.

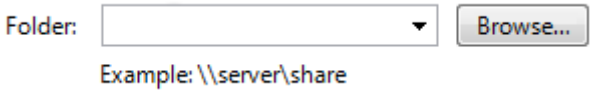
- You may specify units (seconds, connections, and so on) in parentheses after the label.
- Don't make the content of the drop-down list or combo box (or its units label) part of a sentence, because this is not localizable.

Option text

- Assign a unique name to each option.
- Use sentence-style capitalization, unless an item is a proper noun.
- Write the label as a word or phrase, not as a sentence, and use no ending punctuation.
- Use parallel phrasing, and try to keep the length about the same for all options.

Instructional text

- If you need to add instructional text about a drop-down list or combo box, add it above the label. Use complete sentences with ending punctuation.
- Use sentence-style capitalization.
- Additional information that is helpful but not necessary should be kept short. Place this information either in parentheses between the label and colon, or without parentheses below the control.



This example shows additional information placed below the control.

Documentation

When referring to drop-down lists:

- Use the exact label text, including its capitalization, but don't include the access key underscore or colon; include either list or box, whichever is clearer.
- For list options, use the exact option text, including its capitalization.

- In programming and other technical documentation, refer to drop-down lists as drop-down lists. Everywhere else, use either list or box, whichever is clearer.
- To describe user interaction, use click.
- When possible, format the label and list options using bold text. Otherwise, put the label and options in quotation marks only if required to prevent confusion.

Example: In the **Font size** list, click **Large fonts**.

When referring to combo boxes:

- Use the exact label text, including its capitalization, but don't include the access key underscore or colon; include the word box.
- For list options, use the exact option text including its capitalization.
- In programming and other technical documentation, refer to combo boxes as combo boxes. Everywhere else, use box.
- To describe user interaction, use enter.
- When possible, format the label and list options using bold text. Otherwise, put the label and options in quotation marks only if required to prevent confusion.

Example: In the **Font** box, enter the font you want to use.

Feedback

Was this page helpful? Yes No

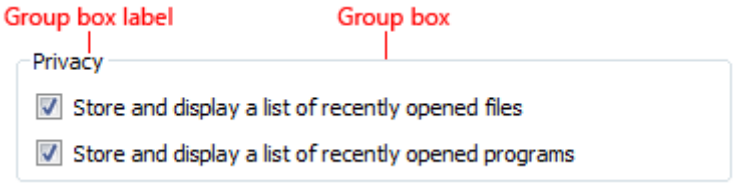
Provide product feedback | Get help at Microsoft Q&A

Group Boxes

Article • 02/09/2021

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

A group box is a labeled rectangular frame that surrounds a set of related controls. A group box is a way to show relationships visually; aside from possibly providing an access key for a group of controls, it provides no functionality.



A typical group box.

> **Note**
>
> Guidelines related to layout are presented in a separate article.

Is this the right control?

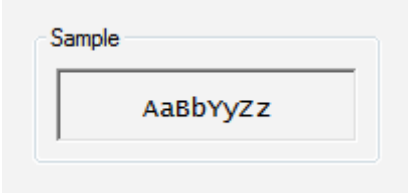
While group boxes are a strong visual means of indicating relationships, overusing them adds visual clutter and greatly reduces the space available on a surface. They are visually heavy and should be used sparingly—only when there isn't a better solution.

A design trend in Windows is a simpler, cleaner appearance by eliminating unnecessary lines.

To decide whether a group box is necessary, consider these questions:

- **Is there more than one control in the group?** If not, use a plain text label instead. A rare exception is to use a group box with a single control to maintain consistency with other group boxes on the same surface.

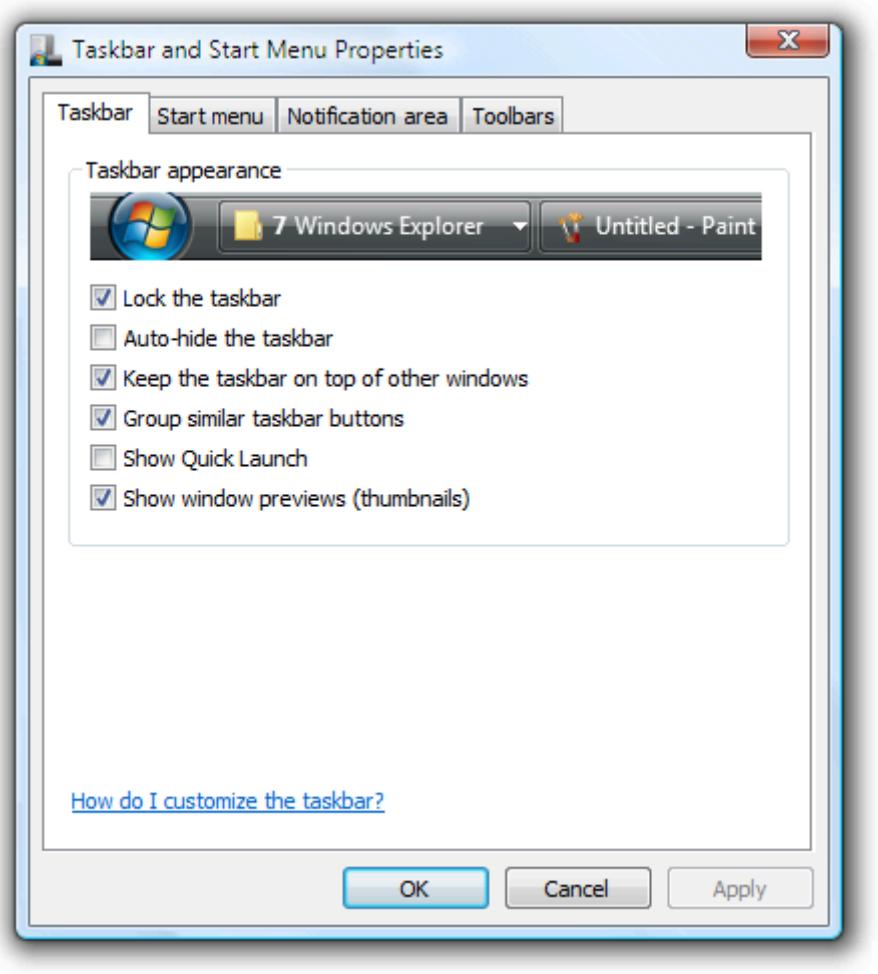
Incorrect:



In this example, the group box has only a single control.

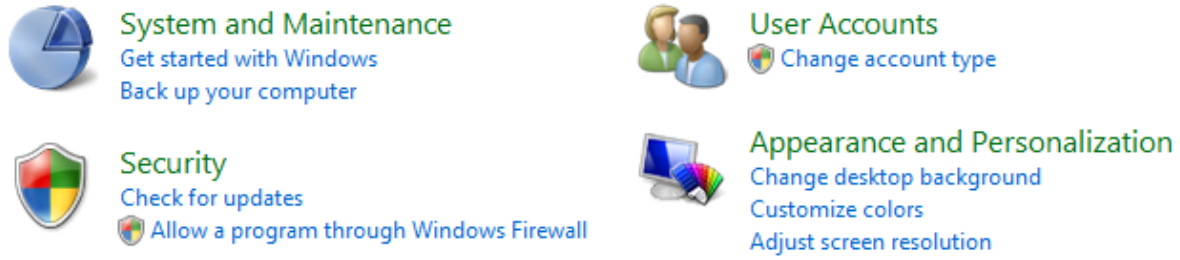
- **Are the controls related? Does showing the relationship add clarity?** If not, present the controls separately outside of a group box.
- **Are all the controls inside the group?** If so, indicate the relationship on the larger surface, such as the parent dialog box or page.

Incorrect:



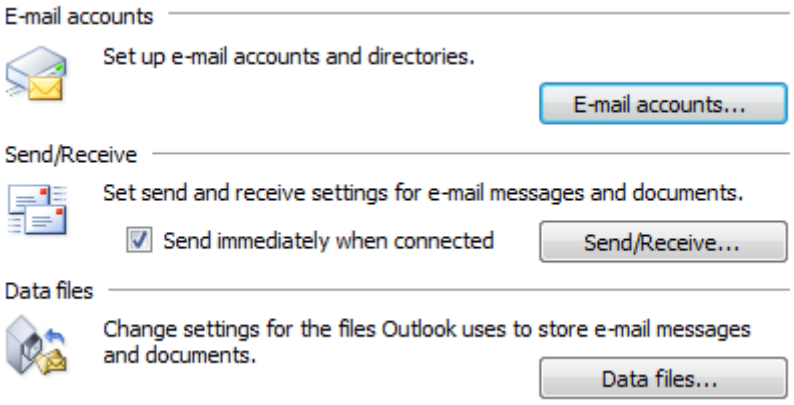
In this example, all the controls (aside from the commit buttons) in the dialog box are within the group box.

- **Can you effectively communicate the relationships using layout alone?** If so, use layout instead. You can place related controls next to each other and put extra spacing between unrelated controls. You can also use headings and indenting to show hierarchical relationships.

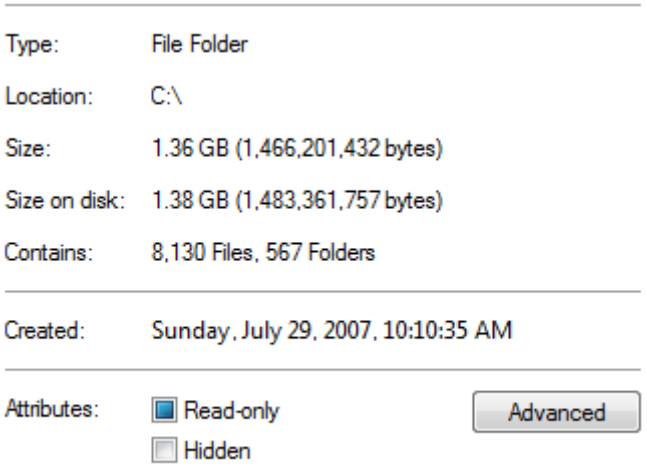


In this example, layout alone is used to show control relationships.

- **Can you effectively communicate the relationships using a separator?** If so, use a separator instead. A separator is a horizontal line that unifies the controls below it. Separators provide a simpler, cleaner look. However, unlike group boxes, they work best when they span the full width of the surface.
    - **Developers:** You can implement a separator with an etched rectangle with a height of one.



In this example, labeled separators are used to show control relationships.



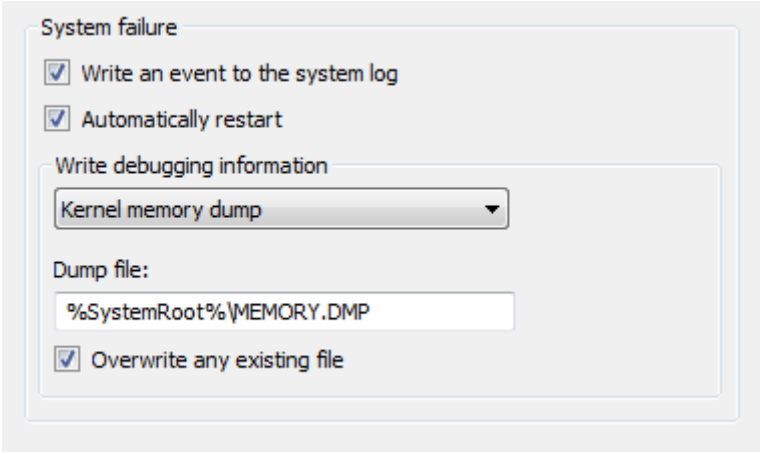
In this example, unlabeled separators are used to show control relationships.

- **Can you effectively communicate the relationships without text?** If so, consider using graphic elements such as backgrounds or aggregators.

Guidelines

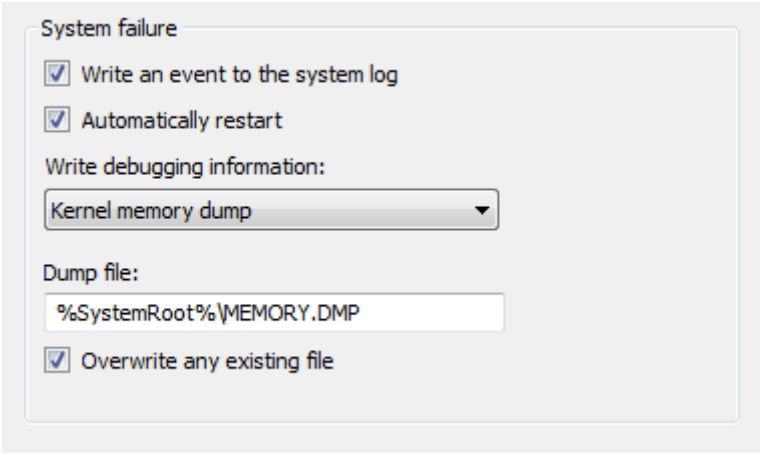
- **Don't nest group boxes.** Use layout to show relationships within a group box.

Incorrect:



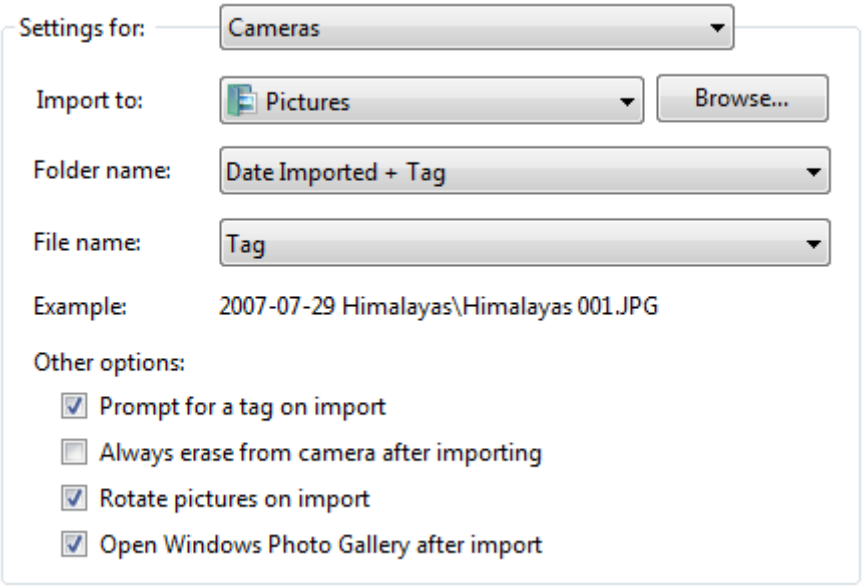
In this example, the nested group boxes result in unnecessary visual clutter.

Correct:



In this example, the same control relationship is shown using layout instead.

- Don't put controls in group box labels.
  - **Exception:** You can use a check box as a group box label if all of the controls inside the box are enabled and disabled by the check box.



Incorrect:

In this example, a drop-down list is incorrectly placed on a group box. This example should use tabs instead.

- **Don't disable group boxes.** To indicate that a group of controls doesn't currently apply, disable all the controls within the group box, but not the group box itself. This approach is more accessible and can be supported consistently by all UI frameworks.

Labels

- Label all group boxes.
- Don't assign an access key to the label. Doing so is unnecessary and makes the other access keys harder to assign. Instead, assign access keys to the controls within the group box.
  - **Exception:** If a surface has many controls, there may not be enough access keys available. If so, reduce the number of access keys by assigning them to group boxes instead of the controls within the group boxes.
- Use sentence-style capitalization.
- Write the label using a noun or a noun phrase, not as a sentence, and use no ending punctuation, including colons.
- Use parallel phrasing for group box labels within the same surface.
- Keep group box labels concise. Don't use instructional text as the label. You can have instructional text within the group box, however.
- Don't repeat the group box label in control labels within the box. For example, if the group box is labeled Alignment, label the option buttons Left, Right, and so on, not Left alignment or Right alignment.
- Don't refer to group boxes in user interface text.

Documentation

When referring to group boxes:

- Refer to group boxes only in programmer and other technical documentation. For group box, use two lowercase words.
- Everywhere else, it is unnecessary to include the name of the group box in a procedure unless a dialog box contains more than one option with the same name. In such cases, use under with the group box name.
- When possible, format the label using bold text. Otherwise, put the label in quotation marks only if required to prevent confusion.

Example: Under **Effects**, select **Hidden**.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Links

Article • 06/04/2021

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

With a *link*, users can navigate to another page, window, or Help topic; display a definition; initiate a command; or choose an option. A link is text or a graphic that indicates that it can be clicked, typically by being displayed using the visited or unvisited link system colors. Traditionally, links are underlined as well, but that approach is often unnecessary and falling out of favor to reduce visual clutter.

When users hover over a link, the link text appears as underlined (if it wasn't already) and the pointer shape changes to a hand.

A text link is the lightest weight clickable control, and is often used to reduce the visual complexity of a design.

> **Note**
>
> Guidelines related to command links and layout are presented in separate articles.

Is this the right control?

To decide, consider these questions:

- **Is the link used to navigate to another page, window, or Help topic; display a definition; initiate a command; or choose an option?** If not, use another control.
- **Would a command button be a better choice?** Use a command button if:
  - The control initiates an immediate action, including displaying a window, and that command relates to the primary purpose of the window.
  - A window is displayed to gather input or making choices, even if for a secondary command.
  - The label is short, consisting of four or fewer words, thus avoiding the awkward appearance of long buttons.
  - The command is not inline.

- The control appears within a group of other related command buttons.
- The action is destructive or irreversible. Because users associate links with navigation (and the ability to back out), links aren't appropriate for commands with significant consequences.
- Similarly, in a wizard or task flow, the command represents commitment. In such windows, command buttons suggest commitment whereas links suggest navigating to the next step.

Design concepts

Making links recognizable

Links lack affordance, which means **their visual properties don't suggest how they are used** and are understood only through experience. Links without an underline and link system colors appear as normal text; the only way to ascertain their behavior is from their presentation, their context, or by positioning the pointer over them.

Surprisingly, this lack of affordance is often a motivation for using links because they appear so lightweight, thereby reducing the visual complexity of a design. Links eliminate the visually heavy frame used by command buttons and border used by other controls. For example, while you might use command buttons to make primary commands obvious, you might choose links for secondary commands to de-emphasize them.

The challenge is then to keep enough visual clues so users can recognize the links. The fundamental guideline is **users must be able to recognize links by visual inspection alone they shouldn't have to hover over an object or click it to determine if it is a link**.

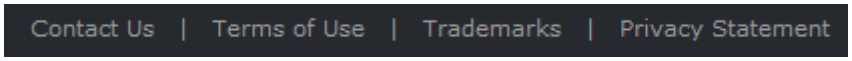
Users can recognize a link by visual inspection alone if the link uses the link system colors and at least one of the following visual clues:

- Underlined text.
- A graphic or bullet, such as with the text with icon link pattern.
- Placement within a standard navigation, option, or command location, such as the content area of a window, or in a navigation bar, menu bar, toolbar, or page footer.

Users can also recognize a link by visual inspection with the following visual clues, but these clues aren't sufficient by themselves:

- Text that suggests clicking, such as a command starting with an imperative verb like Show, Print, Copy, or Delete.
- Placement within a block of normal text.

Of course, users can always determine a link through interaction either hovering or clicking. If discovery of a link isn't required for any significant tasks, you can de-emphasize such links.



In this example, Contact Us, Terms of Use, Trademarks, and Privacy Statement are links. They are intentionally de-emphasized because they aren't required for any important tasks. The only clues that they are links are that they have a mouse pointer on hover and are positioned in a standard navigation area at the bottom of the window.

Making links specific, relevant, and predictable

Link text should indicate the result of clicking on the link.

Specific links are more compelling to users than general links, so **use link labels that give specific descriptive information about the result of clicking on the link**. However, make sure that your link text isn't so specific that it is misleading and discourages proper use.

Concise links are more likely to be read than verbose links. **Eliminate unnecessary text and detail.** Link labels don't have to be comprehensive.

To evaluate your link text:

- Make sure the link text reflects the scenarios that the link supports.
- Make sure the results of the link are predictable. Users shouldn't be surprised by the results.

If you do only two things...

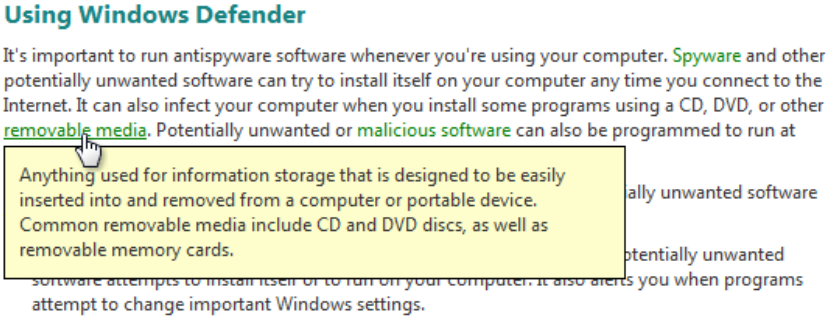
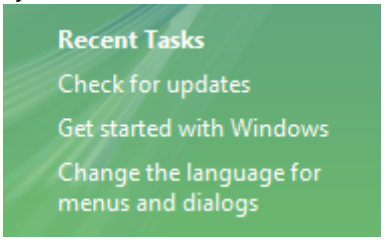
1. Make links discoverable by visual inspection alone. Users shouldn't have to interact with your program to find links.

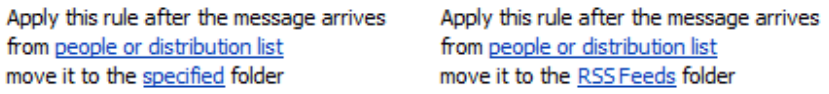
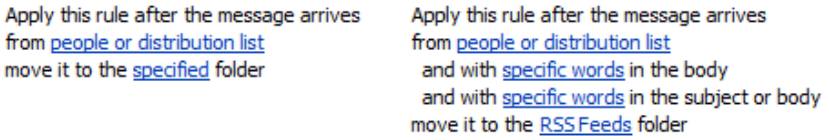
2. Use links that give specific descriptive information about the result of clicking on the link, using as much text as necessary. Users should be able to accurately predict the result of a link from its link text and optional infotip.

Usage patterns

Links have several functional patterns:

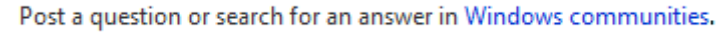
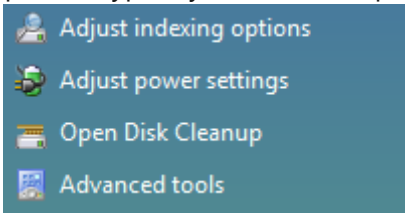
Expand table

Usage	Example
Navigation links A link used to navigate to another page or window.	Clicking the link navigates inplace to another page, as in a browser window or wizard; or displays a new window. In contrast to task links, the navigation doesn't initiate a task but simply navigates to another place or proceeds with a task already in progress. Navigation implies safety because the user can always go back. News headlines In this example, clicking the link navigates to the News headlines page.
Task links A link used to initiate a new command.	Clicking the link either performs a command immediately, or displays a dialog box or page to gather more input. In contrast to navigation links, task links initiate a new task instead of continuing with an existing task. Tasks don't imply safetyusers can't revert to the previous state with a Back command. Task links are so called to prevent confusion with command links. Login In this example, clicking the link initiates a login command.
Help links A text link used to display a Help topic.	Clicking the link displays a Help article in a separate window. What is a strong password? In this example, clicking the link displays a Help window with the given topic.
Definition links a text link used to display a definition in an infotip when the user clicks on or hovers over the link.	this pattern is useful for defining terms that may not be known to your users without adding screen clutter.  In this example, the infotip definition is displayed.
Menu links a set of task links used to create a menu.	because the context of the menu indicates a set of links, the text is usually not underlined (except on hover) and might not use the link system colors.  In this example, a set of links creates a menu.
Option links a selected option or its	unlike regular text links, the link changes its text to reflect the currently selected option and is always drawn using the unvisited link

Usage	Example
placeholder, where clicking the link invokes a command to change that option.	color.  the example on the left shows a rule from the microsoft outlook rules wizard with placeholder options. after users click the links and select some options, the right-hand example updates the link text to show the results. using option links is particularly suitable if the options have a variable format.  the example on the right shows that outlook rules have a variable format. The example on the left shows an option link. It becomes a drop-down list when selected, as shown on the right.

Links also have several presentation patterns:

Expand table

Usage	Example
Plain text links consist only of text.	this presentation is the most flexible because it can be used anywhere, including inline.  In this example, the text color clearly identifies an inline link.
Text with icon links text with a preceding icon that indicates its function.	because the graphic provides an additional visual indication of a link, it is easier to recognize as a link than a plain text link that isn't underlined. this pattern typically uses a 16x16 pixel icon.  in this example, the icons provide an additional visual indication of a link. In this example, the standard triangular Play symbol indicates that this text is a command.

Usage	Example
Graphics-only links consist only of a graphic.	given the lack of a text link, there is no link color or underline to indicate the link. these links depend on either the graphic design to suggest clicking, or text within the graphic that suggests an action when users click. graphic-only links sometimes have a mouse over effect to indicate the link. this approach helps, but isn't discoverable by visual inspection alone. In this example, the link isn't discoverable by visual inspection alone. **Due to their potential recognition and localization problems, graphics-only links are not recommended as the only way to perform a task.**

Guidelines

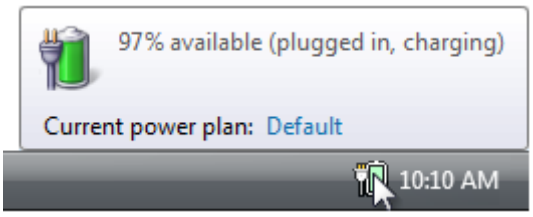
Interaction

- **Display a busy pointer if the result of clicking a link isn't instantaneous.** Without feedback, users might assume that the click didn't happen and click again.

Color

- **Use the theme or link system colors for visited and unvisited links.** The meaning of these colors is consistent across all programs. If for any reason users don't like these colors (perhaps for accessibility reasons), they can change them themselves.
- **For navigation links, use different colors for visited and unvisited links.** Keep the history of visited links only for the duration of the program instance. The visited color is important to indicate where users have already been, preventing them from unintentionally revisiting the same pages repeatedly.
- **For other types of links, don't use the visited link color.** There isn't sufficient value in identifying "visited" commands, for example.
- **Don't color text that isn't a link because users may assume that it is a link.** Use bold or a shade of gray where you'd otherwise use colored text.
- **Exception**: You can use colored text if all links are either underlined or placed within standard navigation or command locations.

  Incorrect:



In this example, blue text is incorrectly used for text that isn't a link.

- **Use background colors that contrast with the link colors.** The window system color is always a good choice.

  Incorrect:

  In this example, the background color provides poor contrast with the link color.

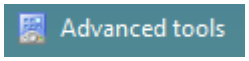
Underlining

- **For links that are necessary to perform a primary task, provide visual clues so that users can recognize links by visual inspection alone.** These clues include underlining, graphics or bullets, and standard link locations. Users shouldn't have to hover over an object or attempt to click on it to determine if it is a link. Use underlined text if the link isn't obvious from its context.
- **Don't underline text that isn't a link because users may assume that it is a link.** Use italics where you'd otherwise use underlined text. Reserve underlining only for links.
- **When printing, don't print underlines or link colors.** Printed links have no value and are potentially confusing.

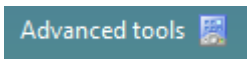
Text with icon links

- **Use the arrow icon only for command links.** Regular links shouldn't use the arrow icon unless they are being used as a substitute for command links in Windows XP.
- **Place the icon to the left of the text.** The icon needs to lead into the text visually.

Correct:



Incorrect:



In the incorrect example, the icon doesn't lead into the text.

- **Make the result of clicking the icon the same as clicking the text.** Doing otherwise would be unexpected and confusing.

Graphics-only links

- **Don't use graphics-only links.** Users have difficultly recognizing them as links and any text within the graphic (used to indicate their action when clicked) creates a localization problem.

Navigation links

- **Make sure navigation links don't require commitment.** Users should always be able to return to the initial state, either by using Back for inplace navigation or Cancel to close a new window.
- **Link to specific content rather than general content.** For example, it is better to link to the relevant section of a document than to link to the beginning.
- **Use a link only if the linked material is relevant, helpful, and not redundant.** Use restraint in navigation links don't use them just because you can.
- **If a link navigates to an external site, put the URL in the infotip** so that users can determine the target of the link.
- **Link only the first occurrence of the link text.** Redundant links are unnecessary and can make text difficult to read.

  Correct:

  The Pictures folder makes sharing your pictures easy. You can use the tasks in Pictures to send your pictures in e-mail or publish them in a secure, private location on the Web. You can also print your pictures directly from the Pictures folder.

  Incorrect:

  The Pictures folder makes sharing your pictures easy. You can use the tasks in Pictures to send your pictures in e-mail or publish them in a secure, private location on the Web. You can also print your pictures directly from the Pictures folder.

  In the correct example, only the first occurrence of the relevant text is linked.

Exceptions:

- **If an instruction has a link, put the link in the instruction.**

  Using strong passwords is very important. For more information, see Strong Passwords.

  In this example, the link is in the instruction instead of the first occurrence.

- **Link to later occurrences if they are far away from the first.** For example, you can link redundantly in different sections within a Help topic.

Task links

- **Use task links for commands that aren't destructive or are easily reversible.** Because users associate links with navigation (and the ability to back out), links aren't appropriate for commands with significant consequences. Commands that display a dialog box or a confirmation are a good choice.

  Correct:

  Start

  Stop

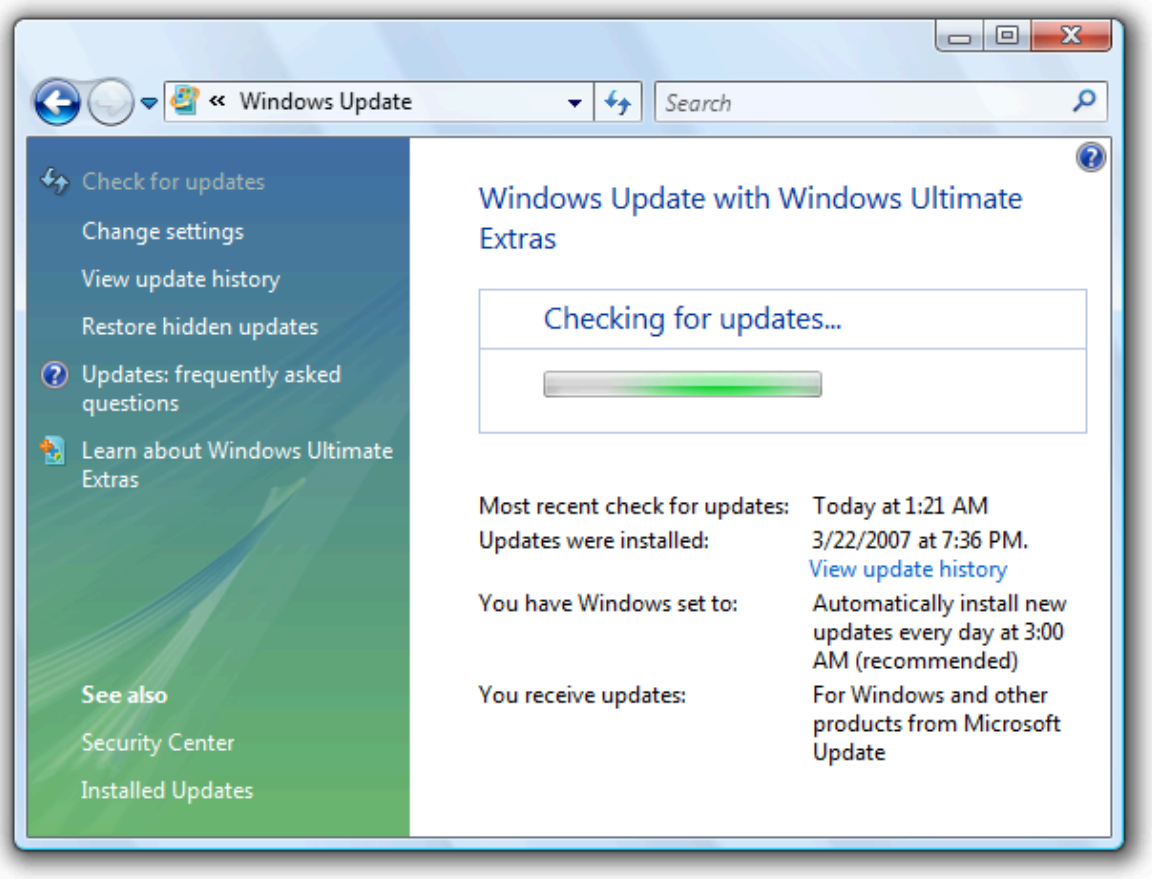
  Incorrect:

  Delete file

  In the incorrect example, the command is destructive.

Menu links

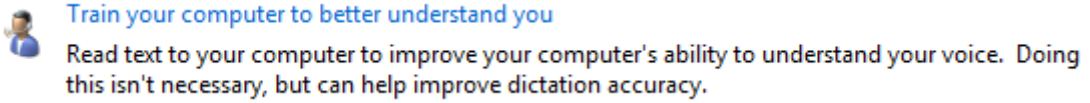
- **Group related navigation and task links into menus.** A menu of related links placed within a standard navigation or command location makes it easier to find and understand the links than when they're placed separately.
- **For selection-dependent menus, remove menu links that don't apply.** Don't disable them. Doing so eliminates clutter and users won't miss links that require selection.
- **For selection-independent menus, disable menu links that don't apply.** Don't remove them. Doing so makes the menus more stable and such links easier to find.



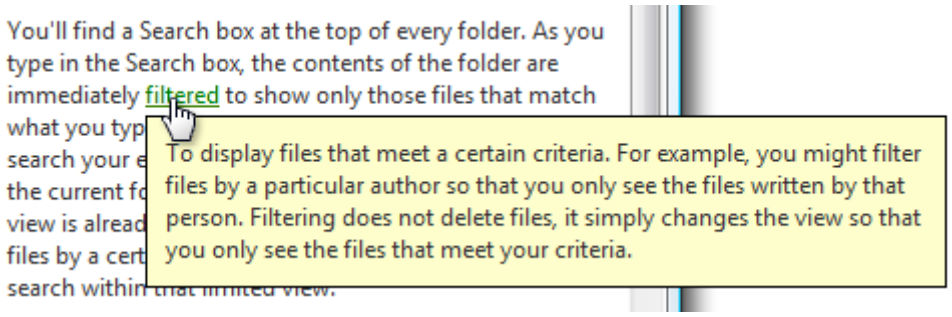
In this example from Windows Update, an update is being performed, so the Check for updates command is disabled rather than removed.

Link infotips

- If a link requires further explanation, **provide the explanation in either a supplemental explanation in a separate text control or an** infotip, but not both. Use complete sentences and ending punctuation. Providing both is unnecessary if the text is the same, and confusing if the text is different.



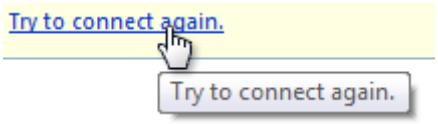
In this example, a supplemental explanation provides further information about the link.



In this example, an infotip provides further information.

- **Don't provide an infotip that is merely a restatement of the link text.**

  Incorrect:

  In this example, the infotip risks annoying users by its repetitiveness.

Text

- Don't assign an access key. Links are accessed using the Tab key.
- **Use links that give specific descriptive information about the result of clicking on the link**, using as much text as necessary. The link text should indicate the result of clicking on the link. **Users should be able to accurately predict the result of a link from its link text and optional infotip.**

  Incorrect:

  In this example, even though the link appears important, its label is too general. Users are more likely to click a more specific link.

- For inline links:
  - Preserve the capitalization and punctuation of the text.
  - Don't include ending punctuation in the link unless the text is a question.
  - Link on the most relevant part of the text and choose link text that is large enough to be easy to click.

    Correct:

    Go to a newsgroup.

    Incorrect:

    Go to a newsgroup.

In these examples, "Go" isn't the most relevant part of the text and it isn't large enough to make a good click target, whereas "newsgroup" is.

- **Avoid putting two different inline links next to each other.** Users are likely to believe they are a single link.

  Incorrect:

  For more information, see UX guidelines.

  In this example, "UX" and "guidelines" are two different links.

- For independent links (not inline):
  - Use sentence-style capitalization.
  - Don't use ending punctuation unless the link is a question.
  - Use all the text as the link.

- Use links that are clearly differentiated from the other links on the screen. Users should be able to accurately predict and differentiate between link targets.

  Incorrect:

  Find antivirus software

  Get antivirus software

  Correct:

  How to know if antivirus software is installed

  Install antivirus software

  In the incorrect example, the distinction between the two links is unclear.

- Don't add Click or Click here to the link text. It isn't necessary because a link implies clicking. Also, Click here and here alone convey no information about the link when read by a screen reader.

  Incorrect:

  Click here for description.

  Correct:

  Description

  In the incorrect examples, "click here" goes without saying and conveys no information about the link.

Navigation links

- **Start the link with a noun and clearly describe where clicking the link will go.** Don't use ending punctuation. On occasion you may need to start navigation links with a verb, but don't use verbs that reiterate navigation that is already implied by the fact of linking, such as View, Open, or Go to.
- **Present a navigation link as a URL if it navigates to a Web page and you expect the target users to recall the URL and type it into a browser.** If possible, design such URLs to be short and easy to remember.
- **If the link includes a URL to a Web site starting with "www," omit the https:// protocol name and use lowercase text.**

  Incorrect:

  https://www.microsoft.com

  `www.microsoft.com`

  Correct:

  microsoft.com

  In the incorrect examples, the "https://" and "www" go without saying.

Task links

- **Start the link with an imperative verb and clearly describe the task that the link performs.** Don't use ending punctuation.
- **End the link with an ellipsis if the command needs additional information (including a confirmation) for successful completion.** Don't use an ellipsis when the successful completion of the task is to display another window only when additional information is needed to perform the task.

  Print...

  In this example, the Print... command link displays a Print dialog box to gather more information.

  Print

  By contrast, in this example a Print command link prints a single copy of a document to the default printer without any further user interaction.

Proper use of ellipses is important to indicate that users can make further choices before performing the task, or can cancel the task entirely. The visual cue offered by an ellipsis allows users to explore your software without fear.

- **If necessary, end a task link with "now" to distinguish it from a navigation link.**

  Download files

  Download files now

  In this example, "Download files" navigates to a page for downloading files, whereas "Download files now" actually performs the command.

Help links

For guidelines and examples, see Help.

Link infotips

- Use full sentences and ending punctuation.

For more guidelines and examples, see Tooltips and Infotips.

Documentation

When referring to links:

- Use the exact link text, including its capitalization, but don't include the ellipsis.
- To describe user interaction, use click.
- When possible, format the link text using bold text. Otherwise, put the link text in quotation marks only if required to prevent confusion.

Example: To start the scan, click **Scan a computer**.

Feedback

Was this page helpful? Yes No

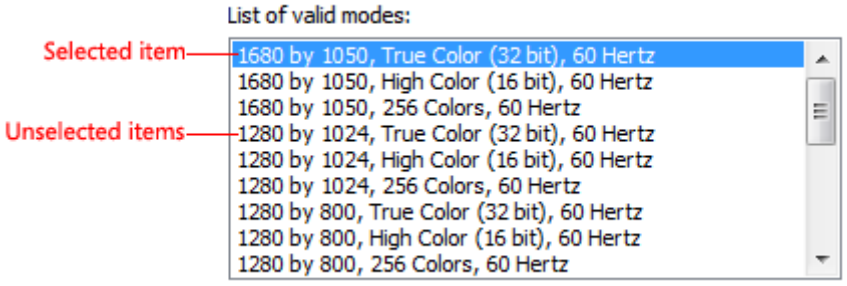
Provide product feedback | Get help at Microsoft Q&A

List Boxes

Article • 08/26/2021

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

With a list box, users can select from a set of values presented in a list that is always visible. With a single-selection list box, users select one item from a list of mutually exclusive values. With a multiple-selection list box, users select zero or more items from a list of values.



A typical single-selection list box.

> **Note**
>
> Guidelines related to layout and list views are presented in separate articles.

Is this the right control?

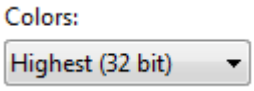
To decide, consider these questions:

- **Does the list present data, rather than program options?** Either way, a list box is a suitable choice regardless of the number of items. By contrast, radio buttons or check boxes are suitable only for a small number of program options.
- **Do users need to change views, group, sort by columns, or change column widths and order?** If so, use a list view instead.
- **Does the control need to be a drag source or a drop target?** If so, use a list view instead.

- **Do the list items need to be copied to or pasted from the clipboard?** If so, use a list view instead.

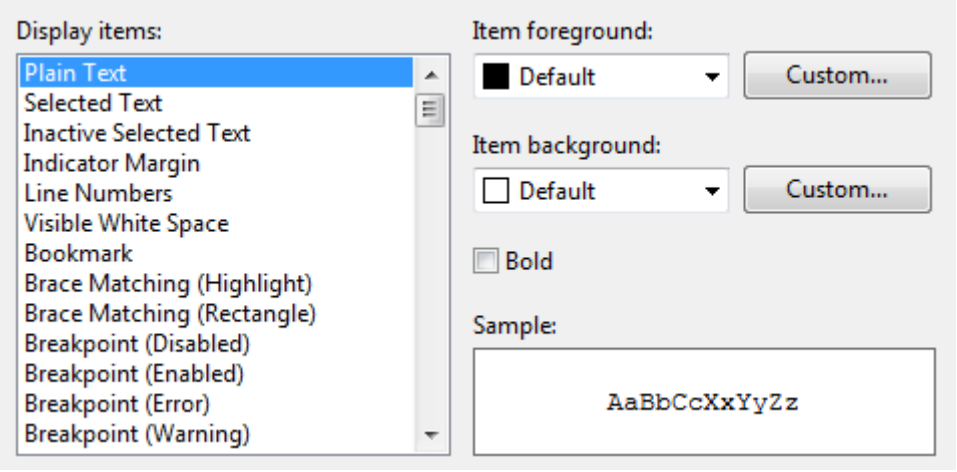
Single-selection lists

- **Is the control used to choose one option from a list of mutually exclusive values?** If not, use another control. To choose multiple options, use a standard multiple-selection list, check box list, list builder, or add/remove list instead.
- **Is there a default option that is recommended for most users in most situations?** Is seeing the selected option far more important than seeing the alternatives? If so, consider using a drop-down list if you don't want to encourage users to make changes by hiding the alternatives.



In this example, the highest color quality is the best choice for most users, so a drop-down list is a good choice to downplay the alternatives.

- **Does the list require constant interaction?** If so, use a single-selection list to simplify the interaction.



In this example, users are constantly changing the selected item in the Display items list to set the foreground and background colors. Using a drop-down list in this case would be very tedious.

- **Does the setting seem like a relative quantity?Would users benefit from instant feedback** on the effect of setting changes? If so, consider using a slider instead.
- **Is there a significant hierarchical relationship between the list items?** If so, use a tree view control instead.
- **Is screen space at a premium?** If so, use a drop-down list instead because the screen space used is fixed and independent of the number of list items.

Standard multiple-selection lists and check box lists

- **Is multiple selection essential to the task or commonly used?** If so, use a check box list to make multiple selection obvious, especially if your target users aren't advanced. Many users won't realize that a standard multiple-selection list supports multiple selection. Use a standard multiple-selection list if the check boxes would draw too much attention to multiple selection or result in too much screen clutter.
- **Is the stability of the multiple selection important?** If so, use a check box list, list builder, or add/remove list because clicking changes only a single item at a time. With a standard multiple selection list, it's very easy to clear all the selections even by accident.
- **Is the control used to choose zero or more items from a list of values?** If not, use another control. For choosing one item, use a single-selection list instead.

Preview lists

- **Are the options easier to select with images than with text alone?** If so, use a preview list.

List builders and add/remove lists

- **Is the control used to choose zero or more items from a list of values?** If not, use another control. For choosing one item, use a single-selection list instead.
- **Does the order of the selected items matter?** If so, the list builder and add/remove list patterns support order, whereas the other multiple-selection patterns do not.
- **Is it important for users to see a summary of all the selected items?** If so, the list builder and add/remove list patterns display only the selected items, whereas the other multiple-selection patterns do not.
- **Are the possible choices unconstrained?** If so, use an add/remove list so that users can choose values not currently in the list.
- **Does adding a value to the list require a specialized dialog box for choosing objects?** If so, use an add/remove list and display the dialog box when users click Add.
- **Is screen space at a premium?** If so, use an add/remove list instead because it uses less screen space by not always showing the set of options.

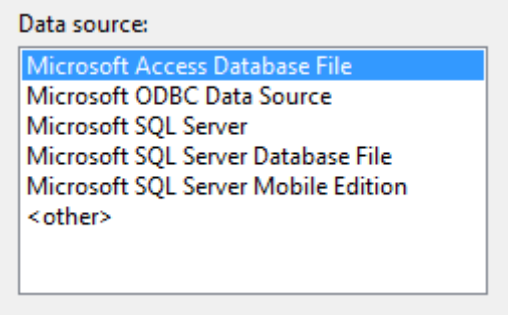
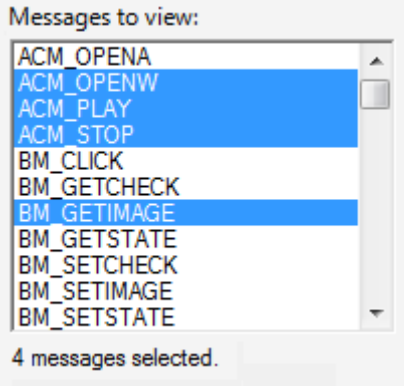
For list boxes, **the number of items in the list isn't a factor in choosing the control** because they scale from thousands of items all the way down to one for single-selection lists (and none for multiple-selection lists). Because list boxes can be used for data, the number of items might not be known in advance.

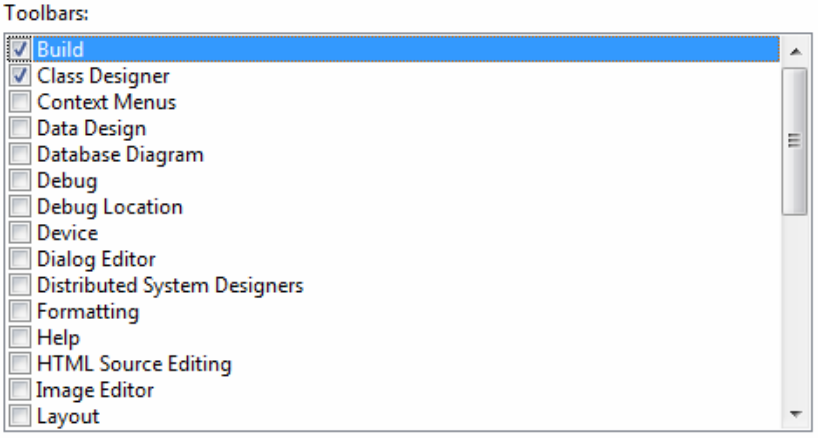
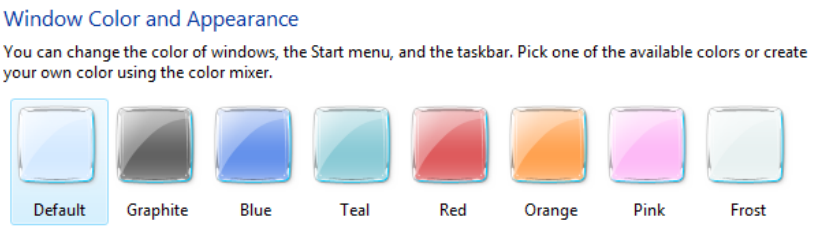
Note: Sometimes a control that looks like a list box is implemented using a list view and vice versa. In such cases, apply the guidelines based on usage, not implementation.

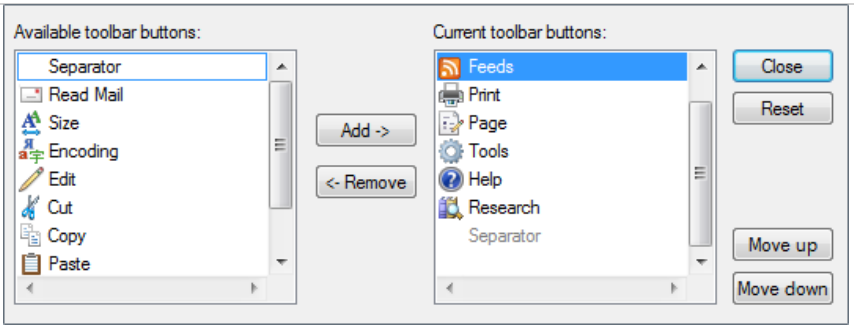
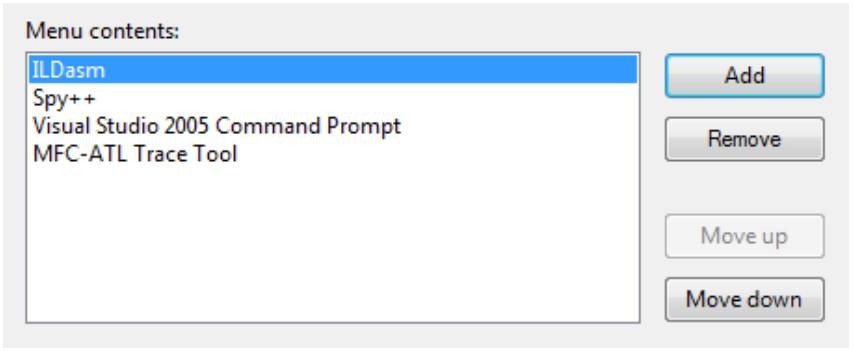
Usage patterns

List boxes have several usage patterns:

Expand table

Label	Value
Single-selection lists Allow users to select one item at a time.	 In this example, users can select only one display item.
Standard multiple-selection lists Allow users to select any number of items, including none.	Standard multiple-selection lists have exactly the same appearance as single-selection lists, so there is no visual clue that a list box supports multiple selection. Because users have to discover this ability, this list pattern is best used for tasks where multiple selection isn't essential and is rarely used. There are two different multiple-selection modes: multiple and extended. **Extended selection mode** is by far the more common, where the selection can be extended by dragging or with Shift+click and Ctrl+click to select groups of contiguous and non-adjacent values, respectively. In the **multiple-selection mode**, clicking any item toggles its selection state regardless of the Shift and Ctrl keys. Given this unusual behavior, multiple-selection mode is deprecated and you should use check box lists instead.  In this example, users can select any number of items using the multiple-selection mode.
Check box lists Like standard multiple-	Unlike standard multiple-selection lists, the check boxes clearly indicate that multiple selection is possible. Use this list pattern for

Label	Value
selection list boxes, check box lists allow users to select any number of items, including none.	tasks where multiple selection is essential or commonly used.  In this example, users typically select more than one item so a check box list is used. Given this clear indication of multiple selection, you might assume that check box lists are preferable to standard multiple-selection lists. In practice, few tasks require multiple selection or use it heavily; using a check box list in such cases draws too much attention to selection. Consequently, **standard multiple-selection lists are far more common.**
Preview lists Can be single or multiple selection, but they show a preview of the effect of the selection rather than just text.	 In this example, a preview of each option clearly shows the effect of the choice, which is more effective than using text alone.
List builders Allow users to create a list of choices by adding one item at a time, and optionally setting the list order.	A list builder consists of two single-selection lists: the list on the left is a fixed set of options and the list on the right is the list being built. There are two command buttons between the lists: • An **Add** button that moves the currently selected option to the list being built, inserted before the selected item. (Double-clicking on an option item has the same effect.) • A **Remove** button that removes the selected item from the built list and returns it to the option list. (Double-clicking on an item in the built list has the same effect.) The built list may optionally have **Move Up** and **Move Down** commands to order the list items.

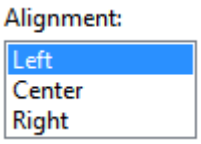
Label	Value
	 In this example, a list builder is used to create a toolbar by selecting items from a set of available options and setting their order.
Add/remove lists Allow users to create a list of choices by adding one or more items at a time, and optionally setting the list order (like list builders).	Unlike a list builder, clicking **Add** displays a dialog box to select items to add to the list. Using a separate dialog box allows for significant flexibility in choosing items you can use a specialized object picker or even a common dialog. Compared to the list builder, this variation is more compact but requires slightly more effort to add items.  In this example, users can add or remove tools from a menu, as well as set order. While the list builder and add/remove list patterns are significantly heavier than the other multiple-selection lists, they offer two unique advantages: • Users have control over the list order, both while building the list and after. • Users can review a summary of the selected items, which can be a significant benefit if the number of choices is large. Their disadvantages are that they require much more screen space and can be difficult to use when creating a large list of items from scratch. Consequently, they are best used to create short lists or modify lists that already exist.

Guidelines

Presentation

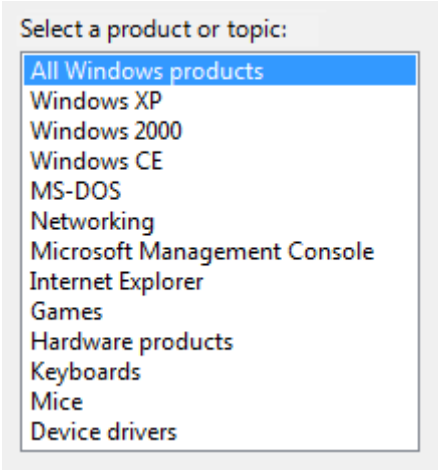
- **Sort list items in a logical order,** such as grouping related options together, placing most frequently used items first, or using alphabetical order. Sort names in alphabetical order, numbers in numeric order, and dates in chronological order. Lists with 12 or more items should be sorted alphabetically to make items easier to find.

Correct:



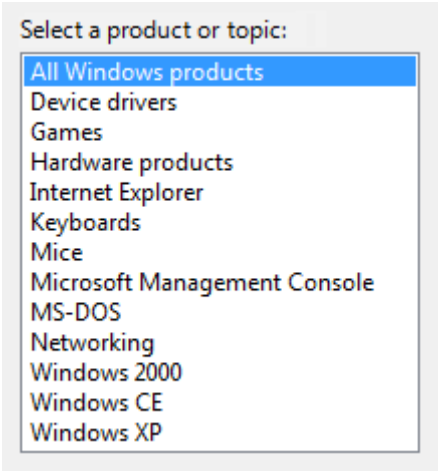
In this example, the list box items are sorted by their spatial relationship.

Incorrect:



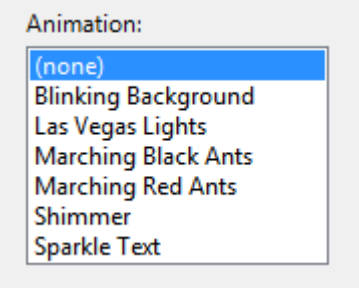
In this example, there are so many list items that they should be sorted in alphabetical order.

Correct:



In this example, the list items are easier to find because they are sorted in alphabetical order. However, the item "All Windows products" is at the beginning of the list, regardless of its sort order.

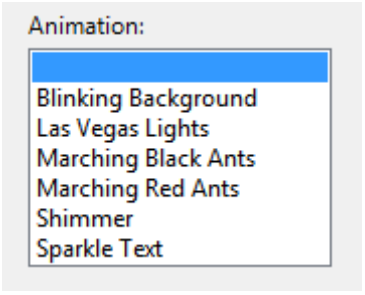
- **Place options that represent All or None at the beginning of the list**, regardless of sort order of the remaining items.
- **Enclose meta-options in parentheses.**



In this example, "(none)" is a meta-option because it is not a valid value for the choice rather it indicates that the option itself isn't being used.

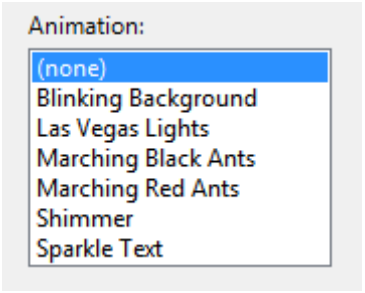
- **Don't have blank list items use meta-options instead.** Users don't know how to interpret blank items, whereas the meaning of meta-options is explicit.

Incorrect:



In this example, the meaning of the blank item is unclear.

Correct:

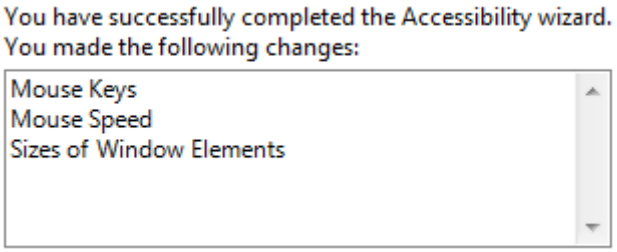


In this example, the "(none)" meta-option is used instead.

Interaction

- **Consider providing double-click behavior.** Double-clicking should have the same effect as selecting an item and performing its default command.
- **Make double-click behavior redundant.** There should always be a command button or context menu command that has the same effect.
- **If users can't do anything with the selected items, don't allow selection.**

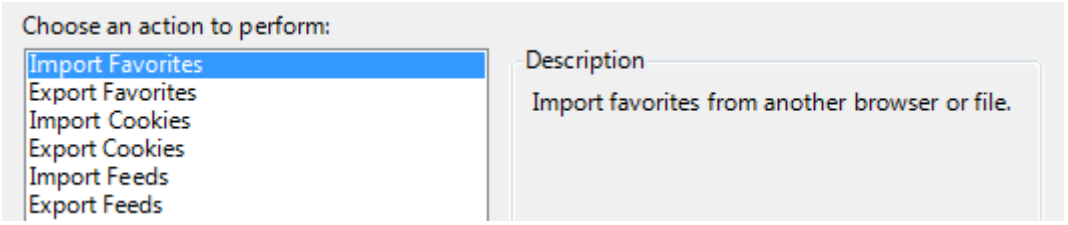
Correct:



This list box displays a read-only list of changes; there is no need for selection.

- **When disabling a list box, also disable any associated labels and command buttons.**
- **Don't use the change of the selected item in a list box to:**
  - Perform commands.
  - Display other windows, such as a dialog box to gather more input.
  - Dynamically display other controls related to the selected control (screen readers cannot detect such events). **Exception:** You can dynamically change static text used to describe the selected item.

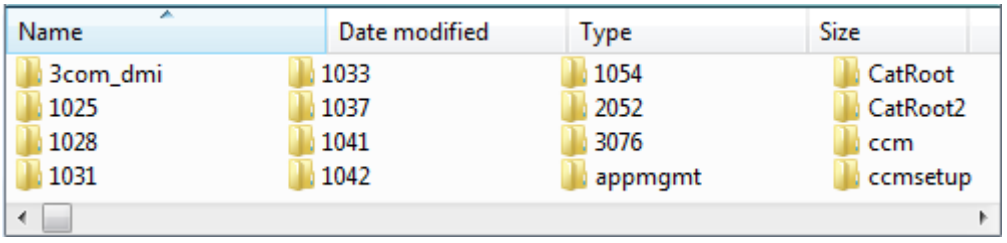
Acceptable:



In this example, changing the selected item changes the description.

- **Avoid horizontal scrolling.** Multicolumn lists rely on horizontal scrolling, which is generally harder to use than vertical scrolling. Multicolumn lists that require horizontal scrolling may be used when you have many alphabetically sorted items and sufficient screen space for a wide control.

Acceptable:

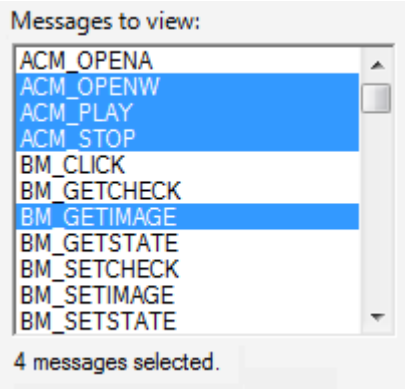


In this example, multiple columns that require horizontal scrolling are used because there are many items and plenty of available screen space for a wide control.

Multiple-selection lists

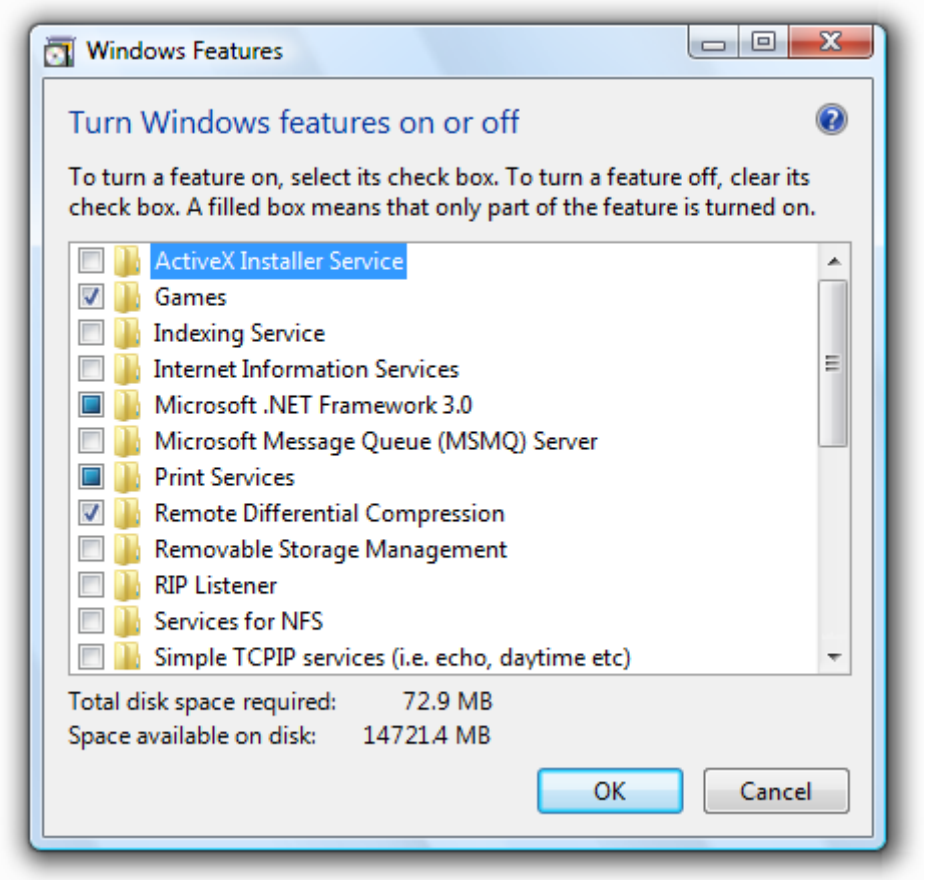
- **Consider displaying the number of selected items below the list,** especially if users are likely to select several items. This information not only gives useful

feedback, but it also clearly indicates that the list box supports multiple selection.



In this example, the number of selected items is displayed below the list.

- You can provide other selection metrics that might be more meaningful, such as the resources required for the selections.



In this example, the disk space required to install the components is more meaningful than the number of items selected.

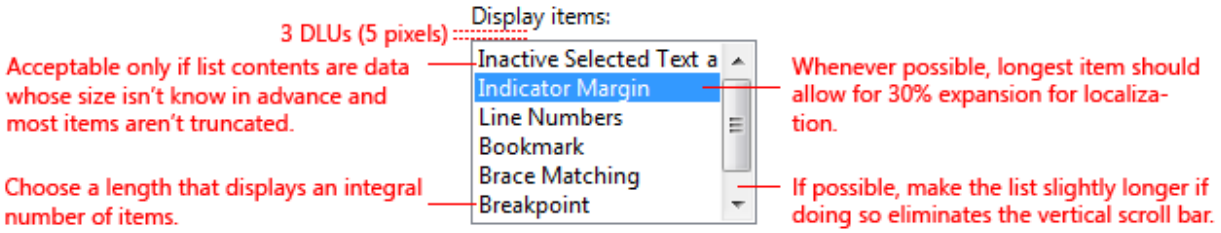
- If there are potentially many list items and selecting or clearing all of them is likely, add Select all and Clear all command buttons.
- For standard multiple-selection lists, don't use multiple-selection mode because this selection mode has been deprecated. For equivalent behavior, use a check box list instead.

Default values

- **Select the safest (to prevent loss of data or system access) and most secure option by default.** If safety and security aren't factors, select the most likely or convenient option.

Exception: Don't select any items if the control represents a property in a mixed state, which happens when displaying a property for multiple objects that don't have the same setting.

Recommended sizing and spacing



Recommended sizing and spacing for list boxes.

- **Choose a list box width appropriate for the longest valid data.** Standard list boxes cannot be scrolled horizontally, so users can see only what is visible in the control.
- **Include an additional 30 percent** (up to 200 percent for shorter text) for any text (but not numbers) that will be localized.
- **Choose a list box height that displays an integral number of items.** Avoid truncating items vertically.
- **Choose a list box height that eliminates unnecessary vertical scrolling.** List boxes should display between 3 and 20 items without the need for scrolling. Consider making a list box slightly longer if doing so eliminates the vertical scroll bar. Lists with potentially many items should display at least five items to facilitate scrolling by showing more items at a time and making the scroll bar easier to position.
- **If users benefit from making the list box larger, make the list box and its parent window resizable.** Doing so allows users to adjust the list box size as needed. However, resizable list boxes should display no fewer than three items.

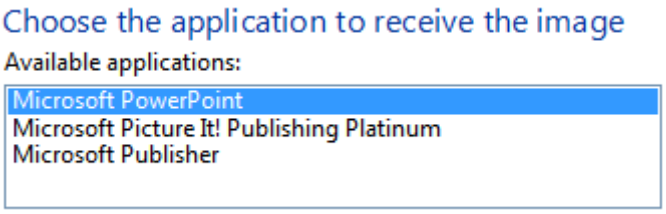
Labels

Control labels

- All list boxes need labels. Write the label as a word or phrase, not as a sentence; use a colon at the end of the label.

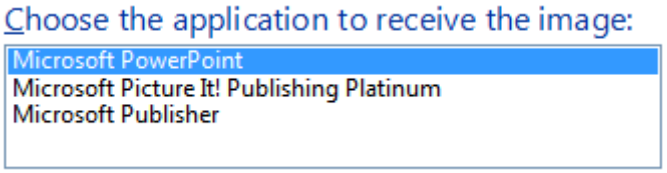
Exception: Omit the label if it is merely a restatement of a dialog box's main instruction. In this case, the main instruction takes the colon (unless it's a question) and access key.

Acceptable:



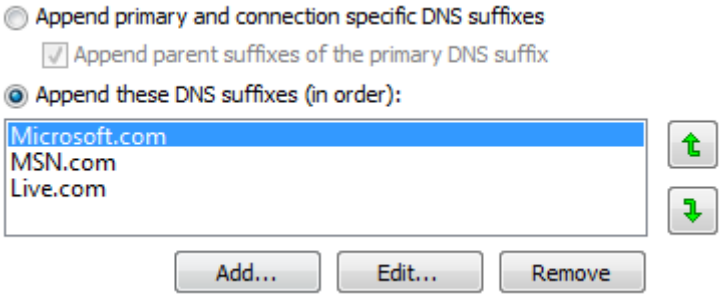
In this example, the list box label just restates the main instruction.

Better:



In this example, the redundant label is removed, so the main instruction takes the colon and access key.

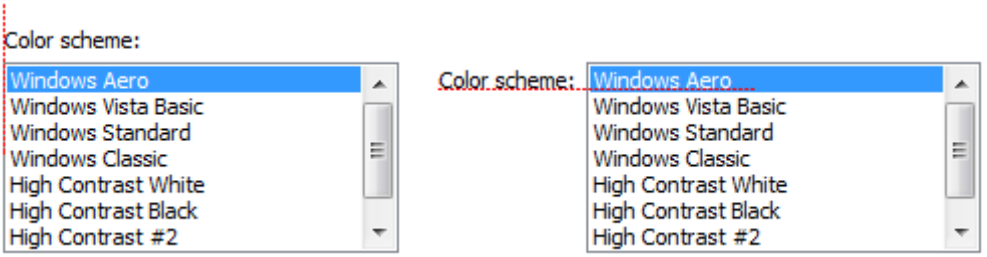
- If a list box is subordinate to a radio button or check box and is introduced by that control's label ending with a colon, don't put an additional label on the list box control.



In this example, the list box is subordinate to a radio button and shares its label.

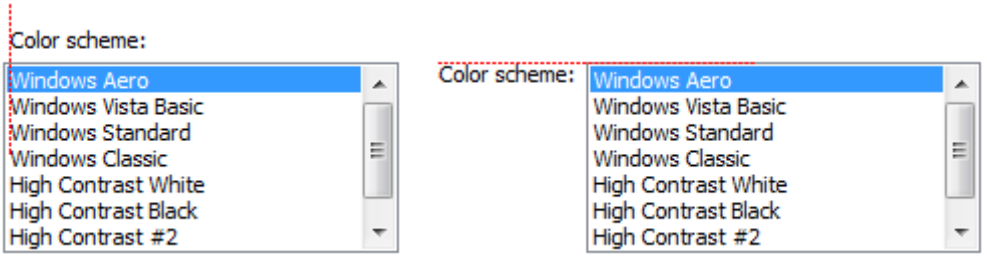
- Assign a unique access key. For guidelines, see Keyboard.
- Use sentence-style capitalization.
- Position the label either to the left of or above the control, and align the label with the left edge of the control.
  - If label is on the left, vertically align the label text with the first line of text in the control.

Correct:



In these examples, the label on top aligns with the left edge of the list box and the label on the left aligns with the text in the list box.

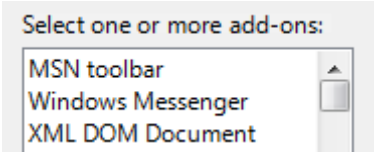
Incorrect:



In these incorrect examples, the label on top aligns with the text in the list box and the label on the left aligns with the top of the list box.

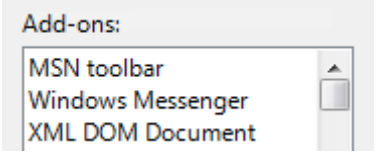
- For multiple-selection list boxes, use a label that clearly indicates multiple selection is possible. Check box list labels can be less explicit.

Correct:



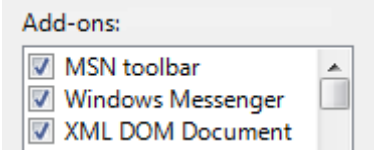
In this example, the label clearly indicates that multiple selection is possible.

Incorrect:



In this example, the label provides no obvious information about multiple selection.

Best:



In this example, the check boxes clearly indicate that multiple selection is possible, so the label doesn't have to be explicit.

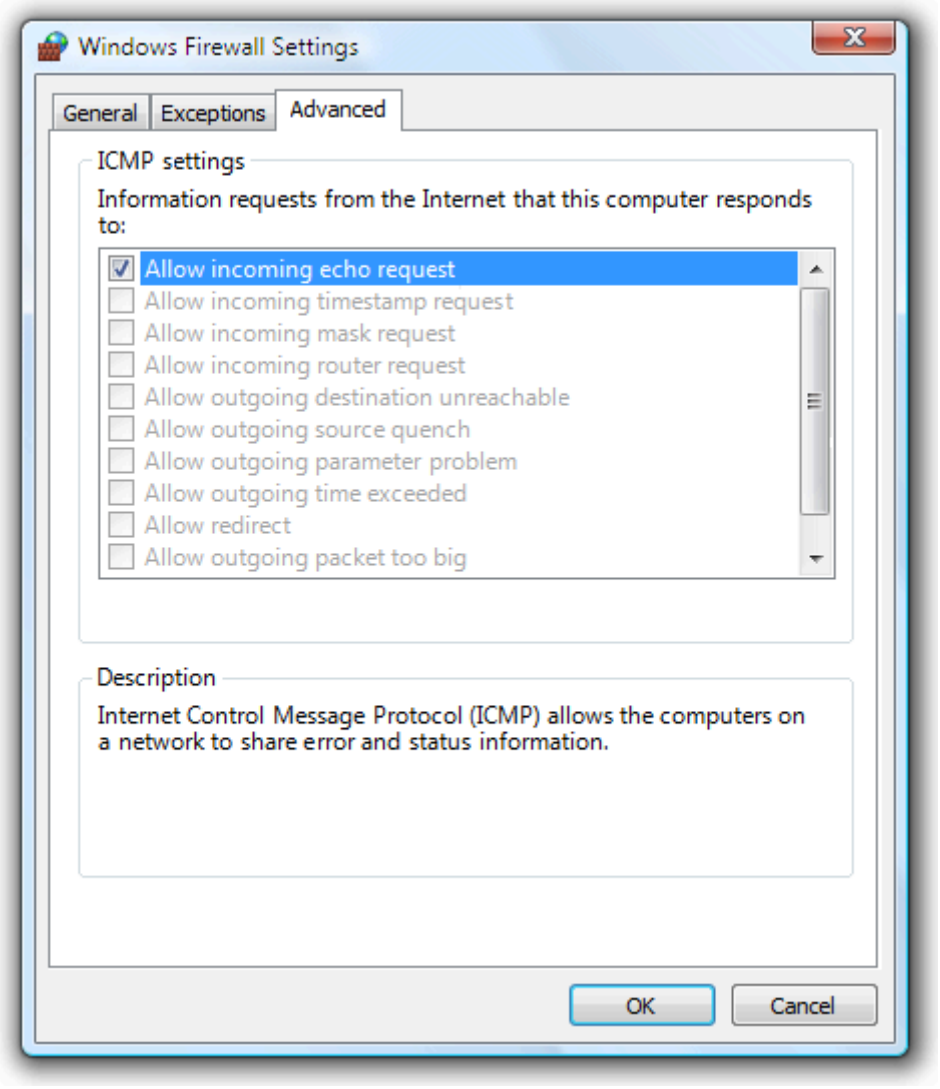
- You may specify units (seconds, connections, and so on) in parentheses after the label.

Option text

- Assign a unique name to each option.
- Use sentence-style capitalization, unless an item is a proper noun.
- Write the label as a word or phrase, not as a sentence, and use no ending punctuation.
- Use parallel phrasing, and try to keep the length about the same for all options.

Instructional and supplemental text

- If you need to add instructional text about a list box, add it above the label. Use complete sentences with ending punctuation.
- Use sentence-style capitalization.
- Additional information that is helpful but not necessary should be kept short. Place this text either in parentheses between the label and colon, or without parentheses below the control.



In this example, supplemental text is placed below the list.

Documentation

When referring to list boxes:

- Use the exact label text, including its capitalization, but don't include the access key underscore or colon. Include the word list. Don't refer to a list box as a list box or a field.
- For list items, use the exact item text, including its capitalization.
- In programming and other technical documentation, refer to list boxes as list boxes. Everywhere else, use list.
- To describe user interaction, use select.
- When possible, format the label and list items using bold text. Otherwise, put the label and items in quotation marks only if required to prevent confusion.

Example: In the **Go to what** list, select **Bookmark**.

Feedback

Was this page helpful? Yes No

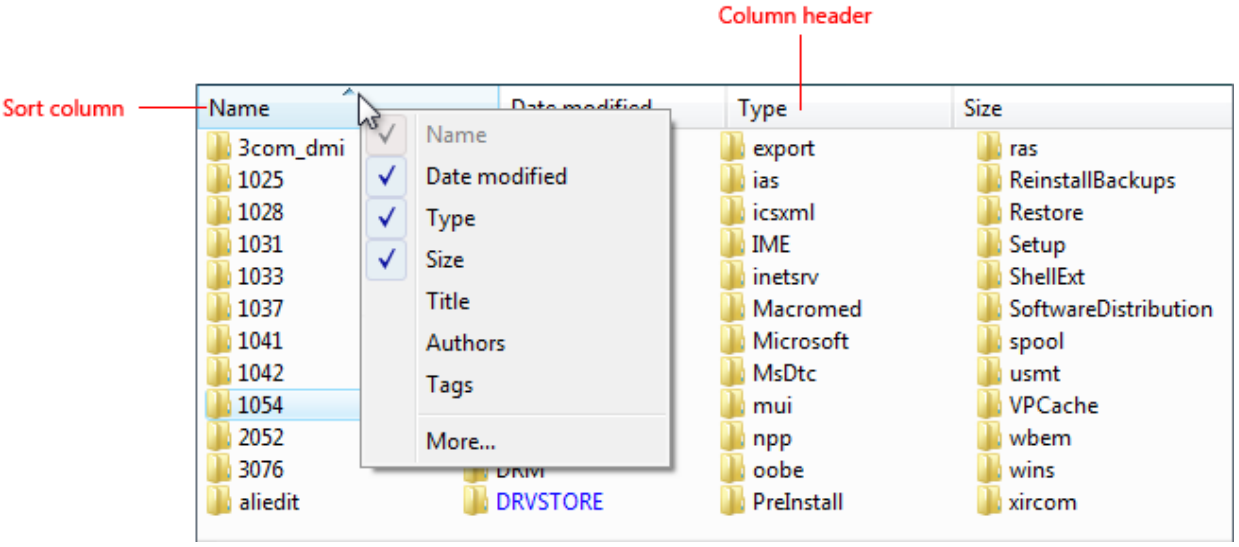
Provide product feedback | Get help at Microsoft Q&A

List Views

Article • 08/26/2021

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

With a list view, users can view and interact with a collection of data objects, using either single selection or multiple selection.



A typical list view.

List views have more flexibility and functionality than list boxes. Unlike list boxes, they support changing views, grouping, multiple columns with headings, sorting by columns, changing column widths and order, being a drag source or a drop target, and copying data to and from the clipboard.

> **Note**
>
> Guidelines related to layout and list boxes are presented in separate articles.

Is this the right control?

A list view is more than just a more flexible and functional list box: its extra functionality results in different usage. The following table shows the comparison.

Expand table

Usage	List boxes	List views
Data type	Both data and program options.	Data only.
Contents	Labels only.	Labels and auxiliary data, possibly in multiple columns.
Interaction	Used for making selections.	Can be used for making selections, but often used for displaying and interacting with data. Can be a drag source or a drop target.
Presentation	Fixed.	Users can change views, group, sort by columns, and change column widths and order.

To decide if this is the right control, consider these questions:

- **Does the list present data, rather than program options?** If not, consider using a list box instead.
- **Do users need to change views, group, sort by columns, or change column widths and order?** If not, use a list box instead.
- **Does the control need to be a drag source or a drop target?** If so, use a list view.
- **Do the list items need to be copied to or pasted from the clipboard?** If so, use a list view.

Check box list views

- **Is the control used to choose zero or more items from a list of data?** To choose one item, use single selection instead.
- **Is multiple selection essential to the task or commonly used?** If so, use a check box list view to make multiple selection obvious, especially if your target users aren't advanced. If not, use a standard multiple-selection list view if the check boxes would draw too much attention to multiple selection or result in too much screen clutter.
- **Is the stability of the multiple selection important?** If so, use a check box list, list builder, or add/remove list because clicking changes only a single item at a time. With a standard multiple selection list, it's very easy to clear all the selections even by accident.

> **Note**
>
> Sometimes a control that looks like a list view is implemented using a list box, and vice versa. In such cases, apply the guidelines based on the usage, not on the

implementation.

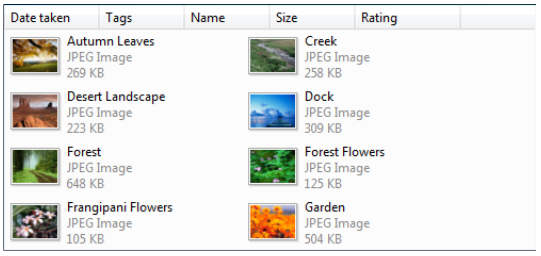
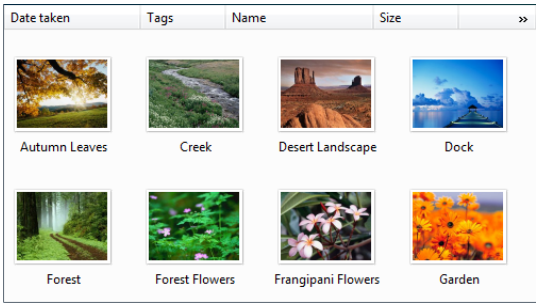
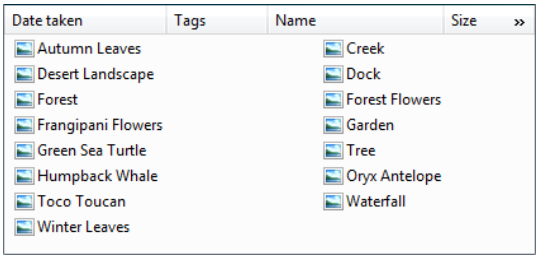
Usage patterns

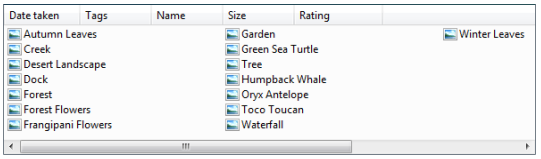
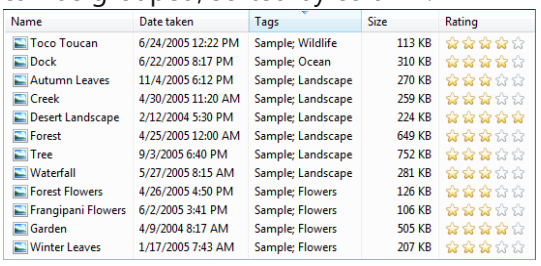
All views support single selection, where users can select only one item at a time, and multiple selection, where users can select any number of items, including none. List views support extended selection mode, where the selection can be extended by dragging or with Shift+click or Ctrl+click to select groups of contiguous or non-adjacent values, respectively. Unlike list boxes, they don't support multiple selection mode, where clicking any item toggles its selection state regardless of the Shift and Ctrl keys.

Standard list view

The list view control supports five standard views:

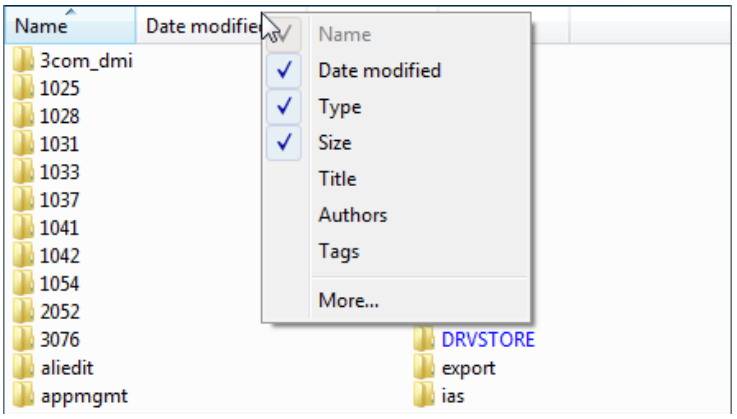
Expand table

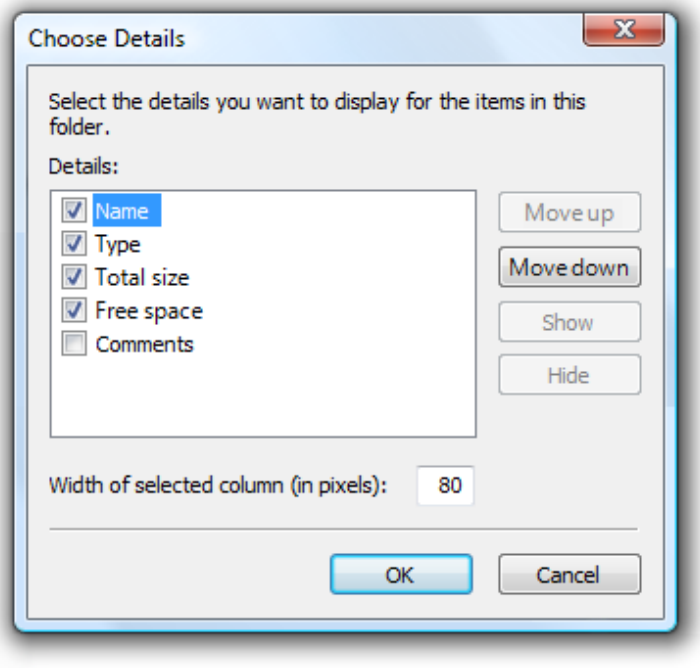
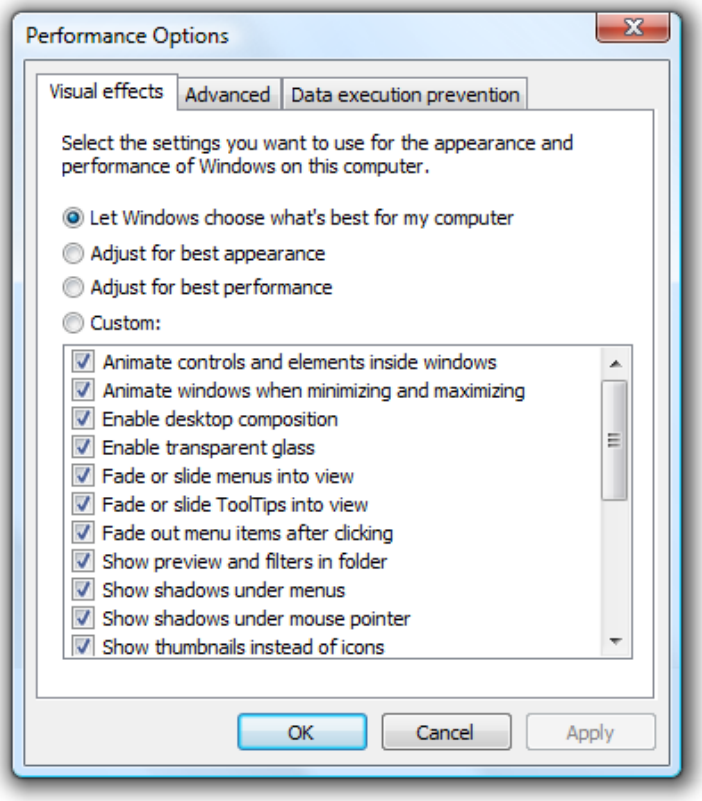
Usage	Example
Tile each item appears as a medium icon, with a label and optional details to the right.	 Tile view shows medium icons with labels and optional details on the right.
Large icon each item appears as an extra large, large, or medium icon with a label below it.	 Large Icon view shows each item as a large icon with a label below it.
Small icon each item appears as a small icon with a label to the right.	 Small Icon view shows each item as a small

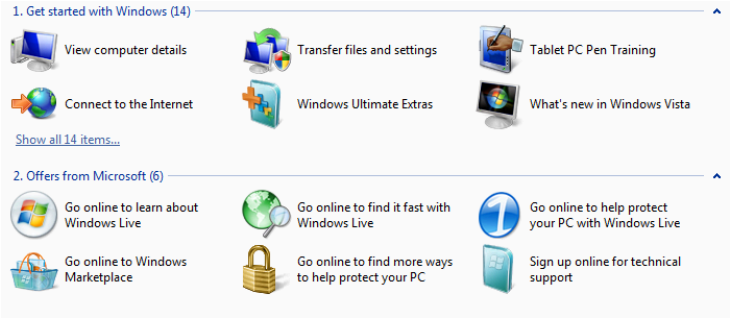
Usage	Example
	icon with its label on the right.
List each item appears as a small icon with a label to the right.	in list mode, this view orders items in columns and uses a horizontal scrollbar. by contrast, the icon view modes order items in rows and use a vertical scrollbar.  List mode shows each item as a small icon with its label on the right.
Details each item appears as a row in a tabular format. the leftmost column contains both the item's optional icon and label, and the subsequent columns contain additional information, such as item properties.	additionally, columns can be added or removed, and reordered and resized. rows can be grouped, sorted by column.  Details view shows each item as a line in a table format.

List view variations

Expand table

Label	Value
Column chooser List views sometimes have so many columns that it isn't practical to show them all. In this case, the best approach is to display the most useful columns by default and allow users to add or remove columns as needed.	 Right-clicking the column heading displays a context menu that allows users to add or remove columns.

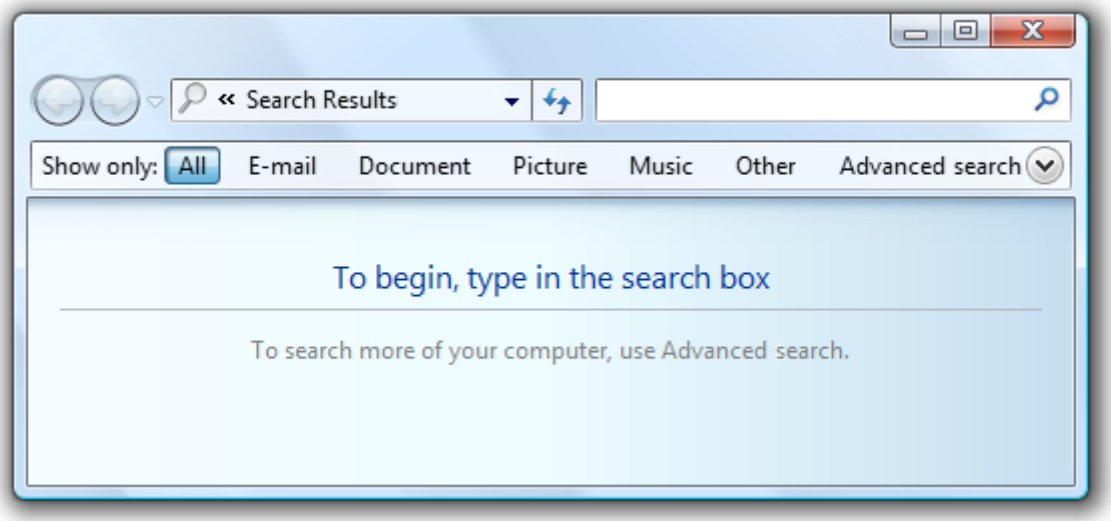
Label	Value
	 Clicking More in the column header context menu displays the Choose Columns dialog box, which allows users to add or remove columns as well as reorder them.
Check box list view Allow users to select multiple items.	Multiple-selection list views have exactly the same appearance as single-selection list views, so there is no visual clue that they support multiple selection. A check box list view can be used to clearly indicate that multiple selection is possible. Consequently, this pattern should be used for tasks where multiple selection is essential or commonly used. 

Label	Value
	In this example, a Small Icon view uses check boxes because multiple selection is essential to the task.
List views with groups Organize the data into groups.	While Details views often support sorting the data by any of the columns, list views further allow users to organize the items into groups. Some benefits of grouping are: • Groups works in all views (except list), so, for example, users could group an extra large icons view of albums by artist. • Groups can be high-level collections, which are often more meaningful than grouping directly off the data. For example, Windows Explorer groups dates into Today, Yesterday, Last week, Earlier this year, and A long time ago.  In this example, the Windows Welcome Center shows grouped items in a list view.

Guidelines

Presentation

- **Sort list items in a logical order.** Sort names in alphabetical order, numbers in numeric order, and dates in chronological order.
- **If appropriate, allow users to change the sort order.** User sorting is important if the list has many items or if there are scenarios where items are found more effectively using a sort order other than the default.
- **Use the Always Show Selection attribute** so that users can readily determine the selected item, even when the control doesn't have focus.
- **Avoid presenting empty list views.** If users create a list, initialize the list with instructions or example items that users might need.

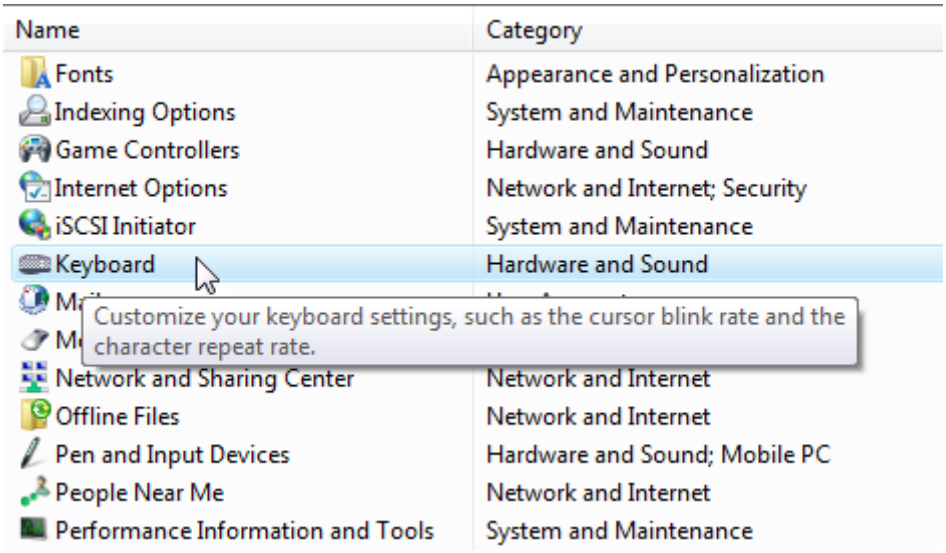


In this example, the Search list view initially presents instructions.

- **If users can change views, group, sort by columns, or change columns and their widths and order, make those settings persist so they take effect the next time the list view is displayed.** Make them persist on a per-list view, per-user basis.

Interaction

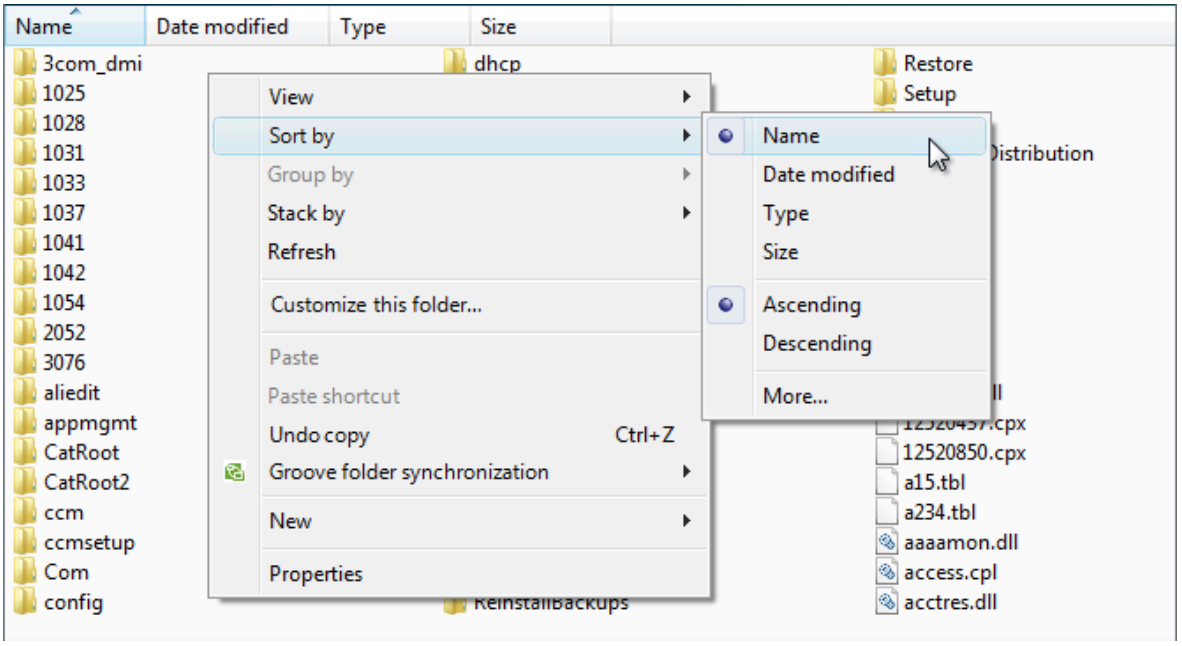
- **Use single-click to select the list item the user is pointing to.Exception:** For the command link list pattern, single-click selects the item and either closes the window or navigates to the next page.
- **Consider providing double-click behavior.** Double clicking should have the same effect as selecting an item and performing its default command.
- **Make double-click behavior redundant.** There should always be a command button or context menu command that has the same effect.
- If a list item requires further explanation, **provide the explanation in an** infotip. Use complete sentences and ending punctuation.



In this example, an infotip is used to provide further information.

- **Provide context menus of relevant commands.** Such commands include Cut, Copy, Paste, Remove or Delete, Rename, and Properties.

- **If users can change the sort order and grouping, provide Sort By and Group By context menus.** The first click on a column name sorts or groups the list in the ascending order for that column, the second click sorts or groups in descending order. Use the previous order (from another column) as the secondary key.



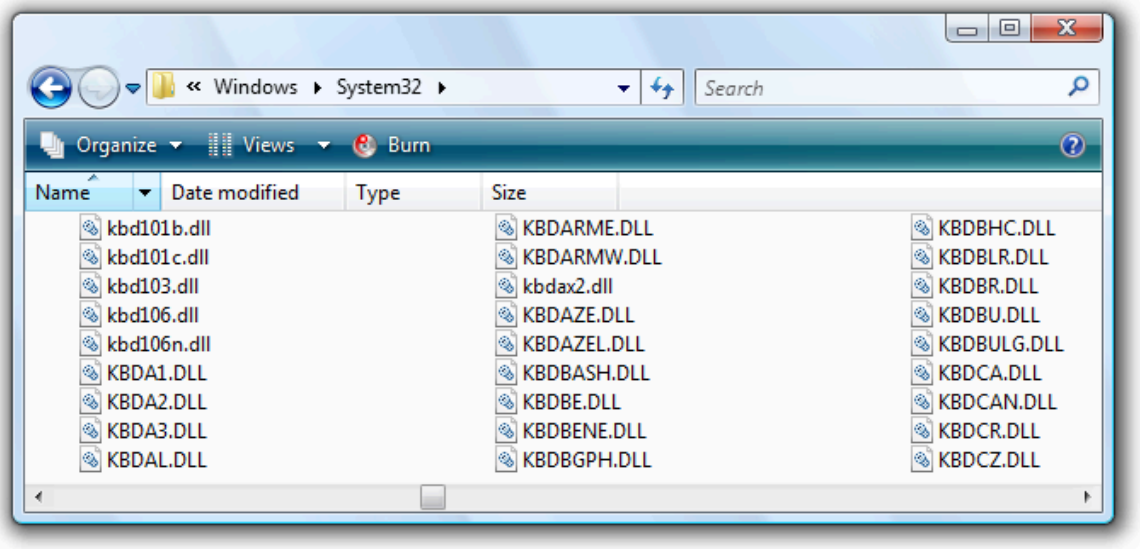
In this example, the Sort By context menu changes the sort order. Clicking Name once sorts by name in ascending order. Clicking Name again sorts by name in descending order.

- **Make the list view column header accessible using the keyboard.**
  - **Developers:** You can do this by setting focus on the column header control. This capability is new to Windows Vista.

- **When disabling a list view, also disable any associated labels and command buttons.**

- **Avoid horizontal scrolling.** The List mode uses horizontal scrolling. This mode is usually the most compact, but horizontal scrolling is generally harder to use than vertical scrolling. Consider using the Small Icon view instead if compactness isn't important. However, List mode is a good choice when there are many alphabetically sorted items and sufficient screen space for a wide control.

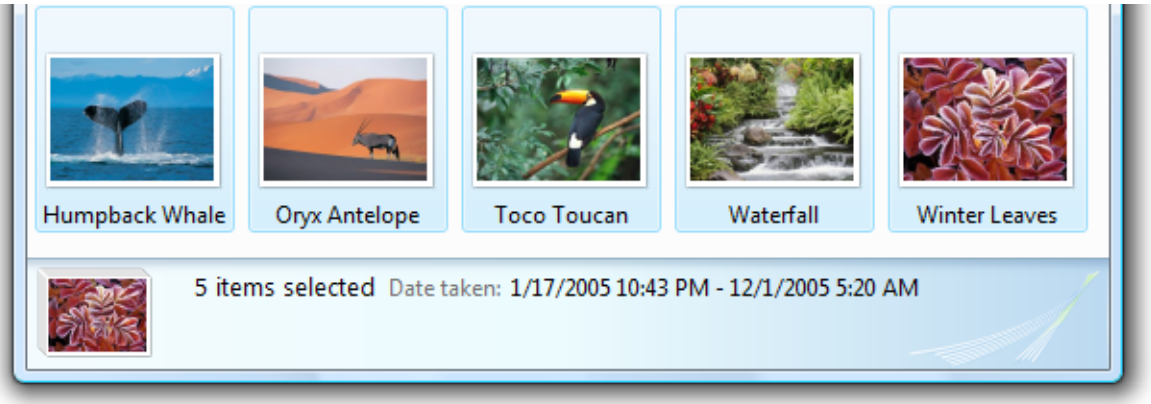
  Acceptable:



  In this example, List mode is used because there are many items and plenty of available screen space for a wide control.

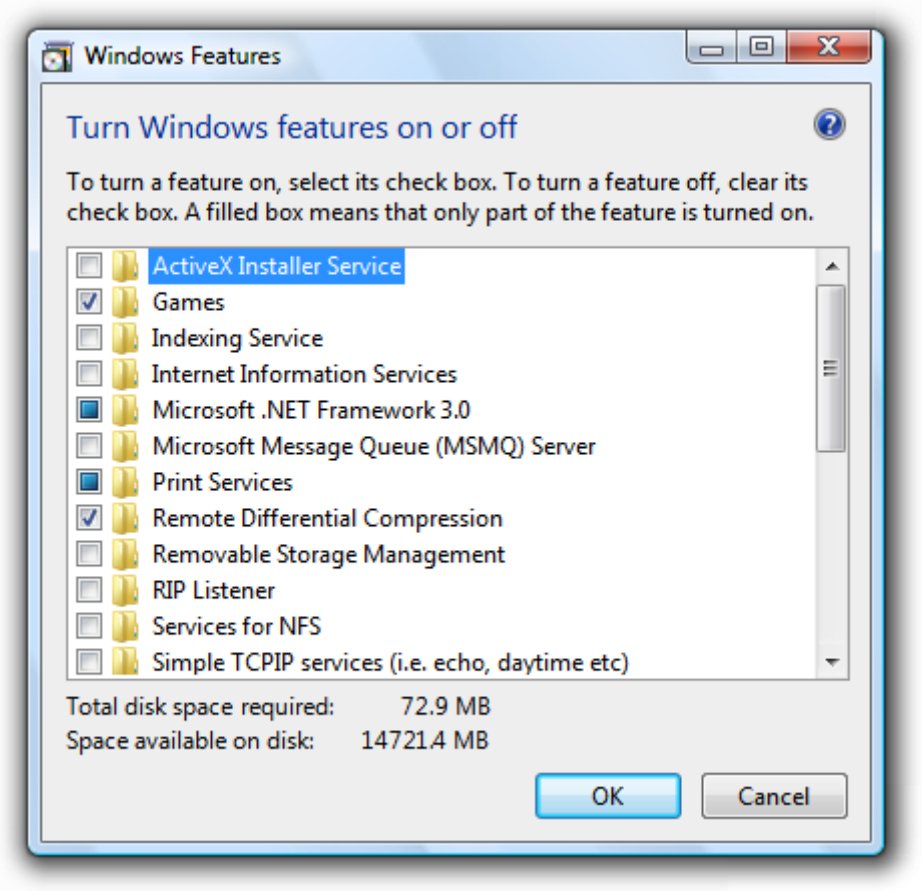
Multiple-selection lists

- **Consider displaying the number of selected items below the list**, especially if users are likely to select several items. This information not only gives useful feedback, but it also clearly indicates that the list view supports multiple selection.



  In this example, the number of selected items is displayed below the list.

- Alternatively instead of the number of selected items, you can give other selection metrics that might be more meaningful, such as the resources required for the selections.



  In this example, the disk space required to install the components is more meaningful than the number of components selected.

- For check box list views, if there are potentially many items and selecting or clearing all of them is likely, add Select all and Clear all command buttons.

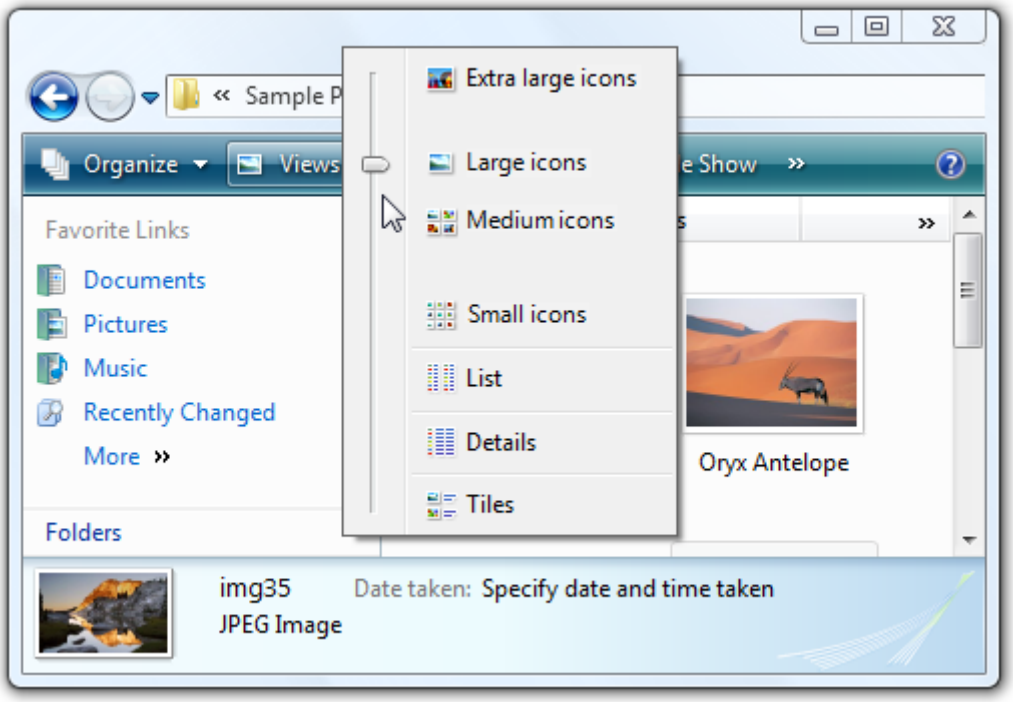
- **Use mixed-state check boxes to indicate partial selection of the items in a container.** The mixed state is not used as a third state for an individual item.

Changing views

If users can change views:

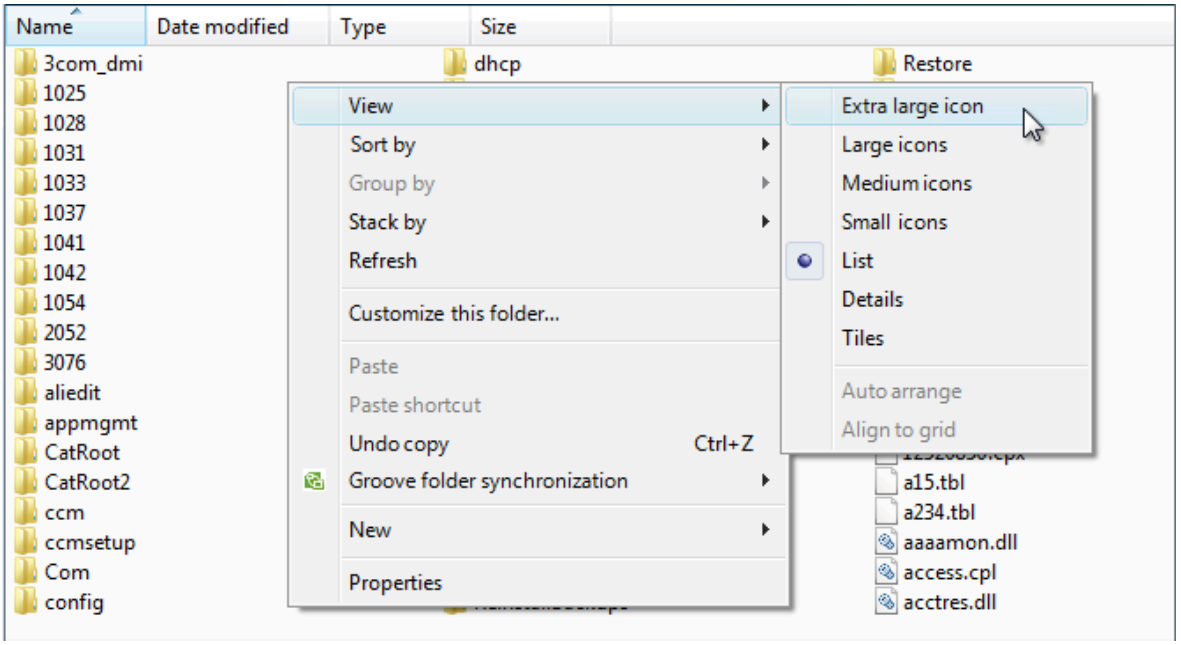
- **Choose the most convenient view by default.** Any changes users make should be made persistent on a per-list view, per-user basis.

- **Change the view using a split button, menu button, or drop-down list.** Whenever practical, use a split button on the toolbar and change the button label to reflect the current view.



In this example, a split button on the toolbar is used to change views.

- **Provide a View context menu.**



In this example, a View context menu is used to change views.

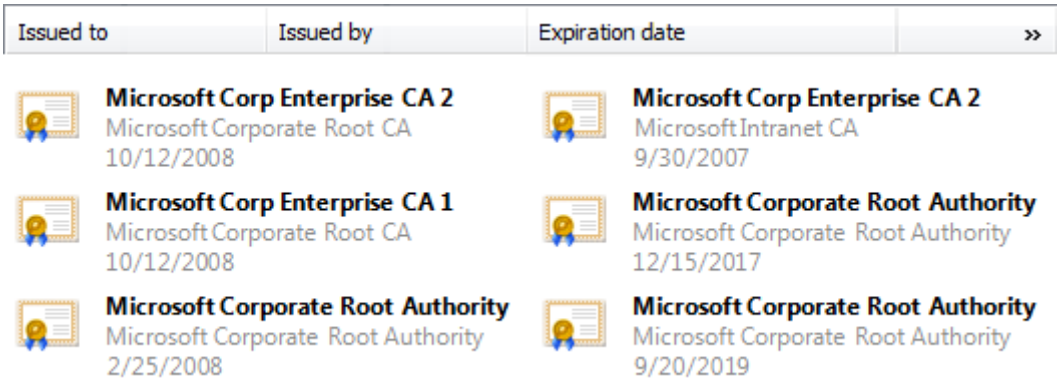
Details views

- **Consider using tiles view to improve readability.**

  Acceptable:

In this example, there is too much data and the window, list, and columns are too small, making the list items hard to read.

Better:

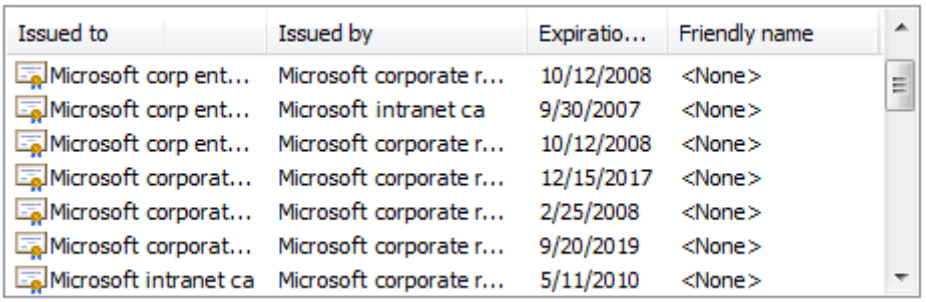


In this example, Tile view displays the data without truncation.

- **Choose default column widths appropriate for the longest data.** List views automatically truncate long data with ellipses, so the column widths are appropriate if few ellipses are displayed by default. While users can resize columns, prefer other solutions:
  - Size each column width to fit its data.
  - Size the control width to fit its columns plus any likely scrollbars.
  - If necessary, use horizontal scrolling.
  - Have truncated data only for odd-sized items or as a last resort.

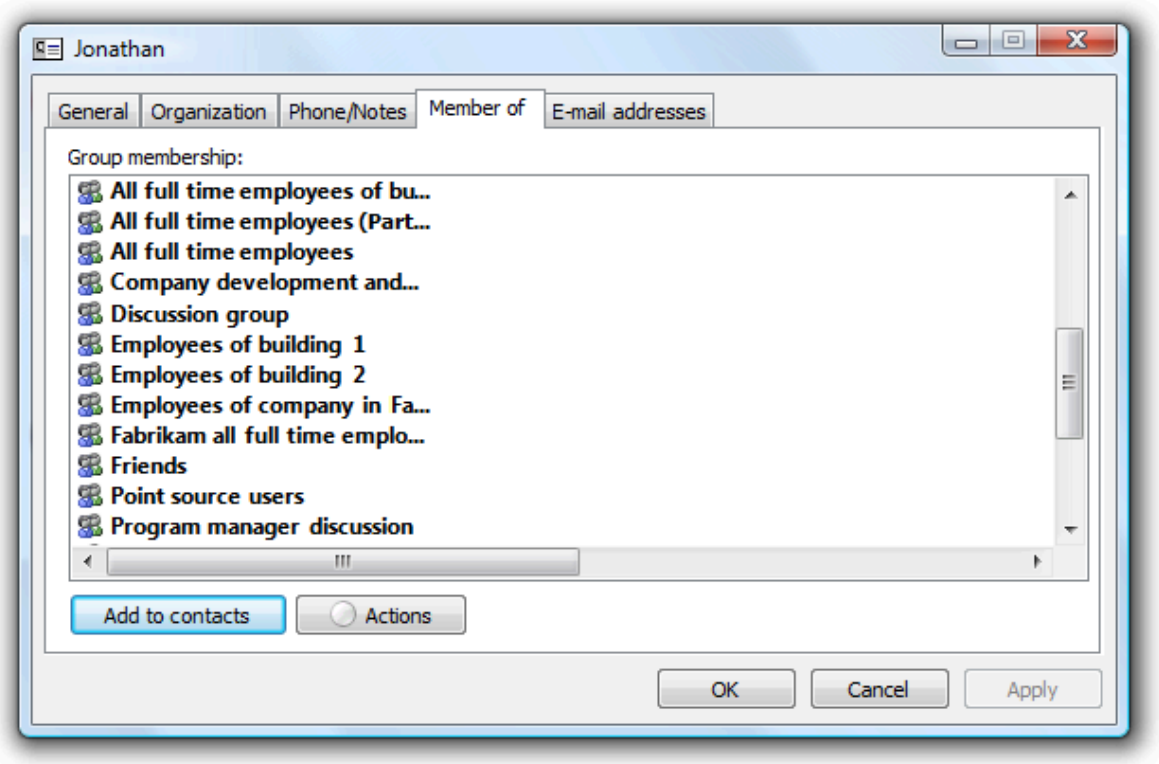
  If normal-sized data must be truncated by default, make the window and list view resizable. Include an additional 30 percent (up to 200 percent for shorter text) for any text (but not numbers) that will be localized.

  Incorrect:



In this example, most data is truncated. The many ellipses clearly indicate that the control and column widths are too small for the data.

Incorrect:



In this example, data is truncated without reason.

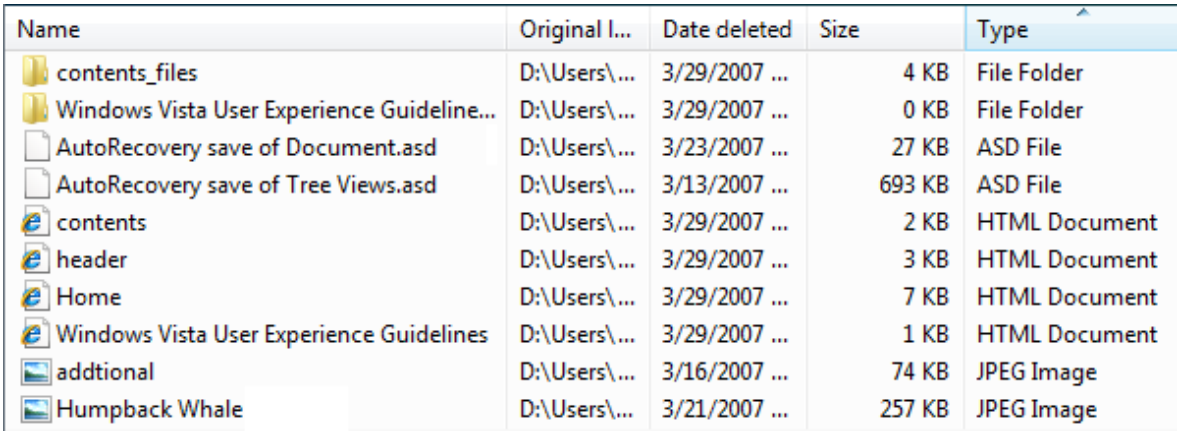
- **Choose an appropriate default column order.** Generally, order the columns as follows:
  - First, the item name or identifying data.
  - Next, other data useful in differentiating the list items.
  - Next, the most useful (preferably short or fixed length) data.
  - Next, less useful (preferable short or fixed length) data.
  - Last, long, variable-length data.

  Long, variable length-data is placed in the last columns to reduce the need for horizontal scrolling. Within these categories, place related information together in a logical sequence.

- **When appropriate, allow users to add and remove columns, as well as change the order.** Display the most useful columns by default. This is achieved with the Header Drag Drop attribute.

- Choose an alignment appropriate for the data. Use the following rules:
  - Right-align numbers, currencies, and times.
  - Left-align text, IDs (even if numeric), and dates.

- For sortable column headings, **the first click on a heading sorts the list in ascending order for the column, the second click sorts in descending order.** Use the previous sort order (from another column) as the secondary sort key.



Name	Original l...	Date deleted	Size	Type
contents_files	D:\Users\...	3/29/2007 ...	4 KB	File Folder
Windows Vista User Experience Guideline...	D:\Users\...	3/29/2007 ...	0 KB	File Folder
AutoRecovery save of Document.asd	D:\Users\...	3/23/2007 ...	27 KB	ASD File
AutoRecovery save of Tree Views.asd	D:\Users\...	3/13/2007 ...	693 KB	ASD File
contents	D:\Users\...	3/29/2007 ...	2 KB	HTML Document
header	D:\Users\...	3/29/2007 ...	3 KB	HTML Document
Home	D:\Users\...	3/29/2007 ...	7 KB	HTML Document
Windows Vista User Experience Guidelines	D:\Users\...	3/29/2007 ...	1 KB	HTML Document
addtional	D:\Users\...	3/16/2007 ...	74 KB	JPEG Image
Humpback Whale	D:\Users\...	3/21/2007 ...	257 KB	JPEG Image

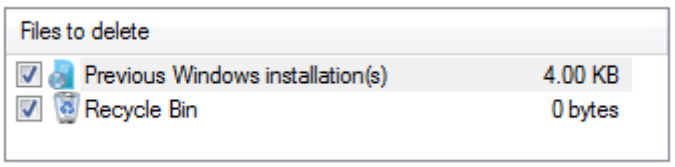
  In this example, the Name column was clicked first, then the Type column. As a result, Type in ascending order is the primary sort key, and Name in ascending order is the secondary.

- **Use the Full Row Select attribute** so that users can readily determine the selected items in all columns.

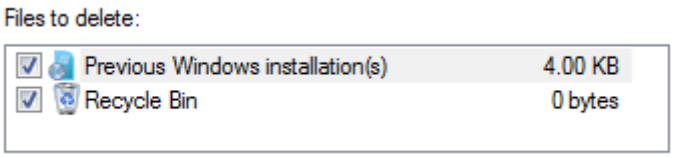
- **Don't use a sortable column header unless the data can be sorted.**

- **Don't use a column header if there is only one column and there is no need to reverse sort.** Use a label instead to identify the data.

  Incorrect:

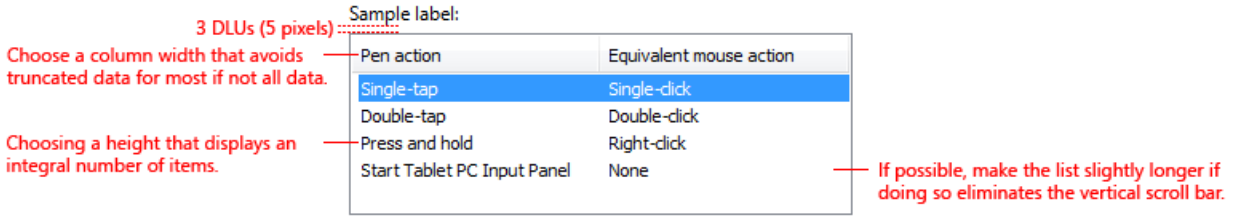


  Correct:



In the correct example, a label is used instead of a column header.

Recommended sizing and spacing



Recommended sizing and spacing for list views.

- **Choose a list view height that displays an integral number of items.** Avoid truncating items vertically.
- **Choose a list view size that eliminates unnecessary vertical and horizontal scrolling in all supported views.** List views should display between 3 and 20 items. Consider making a list view slightly larger if doing so eliminates a scroll bar. Lists with potentially many items should display at least five items to facilitate scrolling by showing more items at a time and making the scroll bar easier to position.
- **If users benefit from making the list view larger, make the list view and its parent window resizable.** Doing so allows users to adjust the list view size as needed. However, resizable list views should display no fewer than three items.

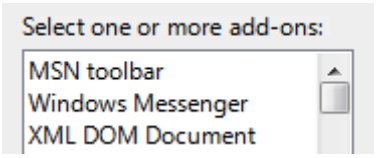
Labels

Control labels

- All list views need labels. Write the label as a word or phrase, not as a sentence, ending with a colon using static text.
- Assign a unique access key for each label. For guidelines, see Keyboard.
- Use sentence-style capitalization.
- Position the label above the control and align the label with the left edge of the control.

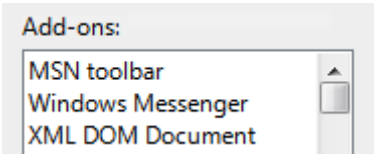
- For multiple-selection list views, write the label that clearly indicates multiple selection is possible. Check box list view labels can be less explicit.

  Correct:

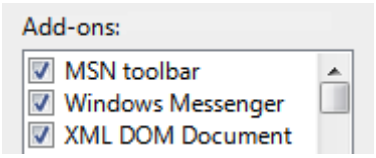
  In this example, the label clearly indicates that multiple selection is possible.

  Incorrect:

  In this example, the label provides no information about multiple selection.

  Acceptable:

  In this example, the check boxes clearly indicate that multiple selection is possible, so the label doesn't have to be explicit.

- You may specify units (seconds, connections, and so on) in parentheses after the label.

Heading labels

- Keep the heading labels brief (three words or fewer).
- Use a single noun or noun phrase with no ending punctuation.
- Use sentence-style capitalization.
- Align the heading the same way as the data.

Group labels

- Use the following group labels for high-level collections:
  - Names: Use first letter of name or letter ranges.
  - Sizes: Unspecified, 0 KB, 0-10 KB, 10-100 KB, 100 KB - 1 MB, 1-16 MB, 16-128 MB

    - Dates: Today, Yesterday, Last week, Earlier this year, and A long time ago.
- Otherwise, group labels use the exact text of the data being grouped, including capitalization and punctuation.

Data text

- Use sentence-style capitalization.

Instructional text

- If you need to add instructional text about a list view, add it above the label. Use complete sentences with ending punctuation.
- Use sentence-style capitalization.
- Additional information that is helpful but not necessary should be kept short. Place this information either in parentheses between the label and colon or without parentheses below the control.

Documentation

When referring to list views:

- Use the exact label text including its capitalization but don't include the access key underscore or colon, and include the word list. Don't refer to a list box as a list box, list view, or field.
- For list data, use the exact data text including its capitalization.
- Refer to list views as list views only in programming and other technical documentation. Everywhere else use list.
- To describe user interaction, use select for the data, and click for the headings.
- When possible, format the label and list options using bold text. Otherwise, put the label and options in quotation marks only if required to prevent confusion.

Example: In the **Programs and services** list, select **File and printer sharing**.

When referring to check boxes in a list view:

- Use the exact label text including its capitalization, and include the word check box. Don't include the access key underscore.
- To describe user interaction, use select and clear.
- When possible, format the label using bold text. Otherwise, put the label in quotation marks only if required to prevent confusion.

Example: Select the **Underline** check box.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Progress Bars

Article • 11/17/2021

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

With a progress bar, users can follow the progress of a lengthy operation. A progress bar may either show an approximate percentage of completion (determinate), or indicate that an operation is ongoing (indeterminate).

Usability studies have shown that users are aware of response times of over one second. Consequently, you should consider operations that take two seconds or longer to complete to be lengthy and in need of some type of progress feedback.

A typical progress bar.

> **Note**
>
> Guidelines related to layout are presented in a separate article.

Is this the right control?

To decide, consider these questions:

- **Will the operation complete in about five seconds or less?** If so, use an activity indicator instead, because displaying a progress bar for such a short duration would be distracting. If the operation usually takes five seconds or less but sometimes takes more, start with a busy pointer and convert to a progress bar after five seconds.
- **Is an indeterminate progress bar used to wait for the user to complete a task?** If so, don't use a progress bar. Progress bars are for computer progress, not user progress.

- **Is an indeterminate progress bar combined with an animation?** If so, use just the animation instead. The indeterminate progress bar is effectively a generic animation and adds no value to the animation.

- **Is the operation a very lengthy (longer than two minutes) background task for which users are more interested in completion than progress?** If so, use a notifications instead. In this case, users do other tasks in the meantime and are not monitoring the progress. Using a notification allows users to perform other tasks without disruption. Examples of such lengthy operations include printing, backup, system scans, and bulk data transfers or conversions.

- **When the operation is complete, will users be able to replay the results?** If so, use a slider instead. Examples of such operations include video and audio recording and playback.

  In this example, a slider is used to indicate progress while playing sound. Doing so allows users to replay the results later.

Design concepts

During a lengthy operation, users need a general idea of what the operation is doing. They also need to know:

- That a lengthy operation has started.
- That progress is being made and that the operation will eventually complete (and therefore hasn't locked up).
- The approximate percentage of the operation that has been completed (and therefore the percentage remaining).
- If they should cancel the operation if it isn't worth continuing to wait.
- If they should continue to wait or do something else while the operation completes.

Use determinate progress bars for operations that require a bounded amount of time, even if that amount of time cannot be accurately predicted. Indeterminate progress bars show that progress is being made, but provide no other information. Don't choose an indeterminate progress bar based only on the possible lack of accuracy alone.

For example, suppose an operation requires five steps and each of those steps requires a bounded amount of time, but the amount of time for each step can vary greatly. In this case, use a determinate progress bar and show progress when each step completes proportional to the amount of time each step usually takes. Use an indeterminate progress bar only if a determinate progress bar would cause users to conclude incorrectly that the operation has locked up.

If you do only one thing...

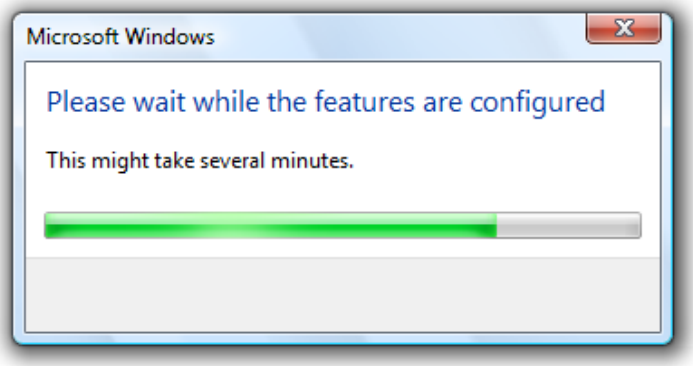
Make sure that you provide progress feedback for lengthy operations and that the above information is clearly communicated. Use determinate progress bars whenever possible.

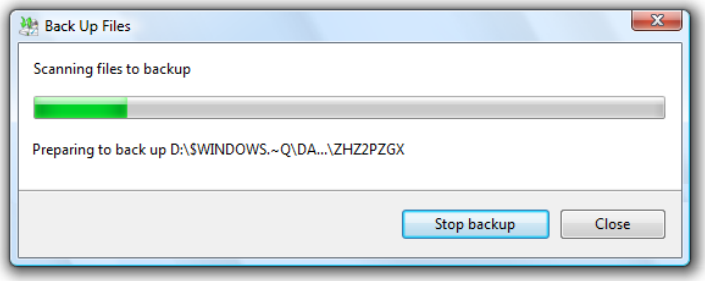
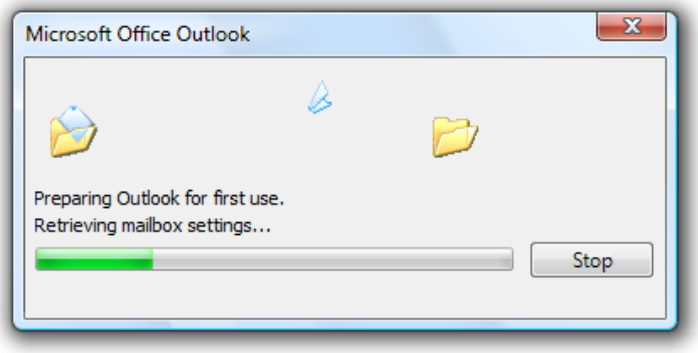
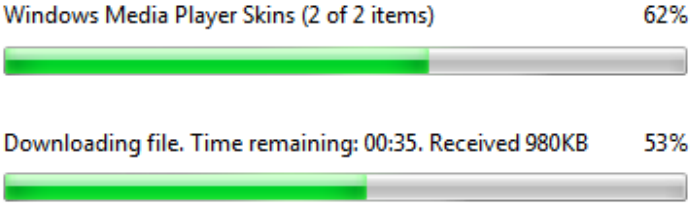
Usage patterns

Progress bars have several usage patterns:

Determinate progress bars

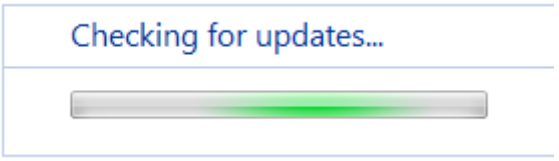
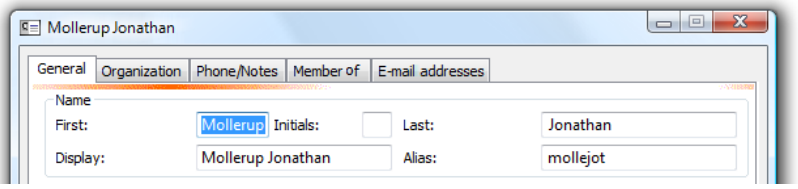
Expand table

Label	Value
Modal determinate progress bars Indicate an operation's progress by filling from left to right and filling completely when the operation is complete.	Because this feedback is modal, users cannot perform other tasks in the window (or its parent if displayed in a modal dialog box) until the operation is complete.  In this example, the progress bar gives feedback during configuration.

Label	Value
Modal determinate progress bars with a Cancel or Stop button Allow users to halt the operation, perhaps because the operation is taking too long or isn't worth the wait.	 In this example, users can click Stop to halt the operation and leave the environment in its current state.
Modal determinate progress bars with a Cancel or Stop button and animation Allow users to halt the operation, and include an animation to help users visualize the effect of an operation.	 In this example, users can click Stop to halt the operation and leave the environment in its current state.
Modal determinate double progress bars Indicate the progress of a multi-step operation by showing the progress of the current step in the first progress bar, and the overall progress in the second bar.	Because the first progress bar provides little additional information and can be quite distracting, this pattern is not recommended. Instead, have all the steps in the operation share a portion of the progress and have a single progress bar go to completion once.  In this example, the first progress bar shows the progress of the current step and the second progress bar shows the overall progress. **Note:** This pattern is usually unnecessary and should be avoided.
Modeless determinate progress bars Indicate an operation's progress by filling from left to right and filling completely when the operation is complete.	Unlike with modal progress bars, users can perform other tasks while the operation is in progress. These progress bars can be displayed in context or on a status bar. In this example, Windows Internet ExplorerWindows Internet Explorer displays its progress for loading a Web page on the status bar. Users can perform other tasks while the page is loading.

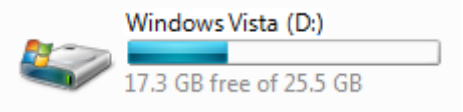
Indeterminate progress bars

Expand table

Progress Bar Type	Description
Modal indeterminate progress bars Indicate an operation is in progress by showing an animation that continuously cycles across the bar from left to right.	Used only for operations whose overall progress cannot be determined, so there is no notion of completeness. Determinate progress bars are preferable because they indicate the approximate percentage of the operation that has been completed, and help users determine if the operation is worth continuing to wait. They are also less visually distracting.  In this example, Windows Update uses a modal indeterminate progress bar to indicate progress while it looks for updates.
Modeless indeterminate progress bars Indicate an operation is in progress by showing an animation that continuously cycles across the bar from left to right.	Unlike modal progress bars, users can perform other tasks while the processing is in progress. these progress bars can be displayed in context or on a status bar.  In this example, Microsoft Outlook uses a modeless indeterminate progress bar while filling in contact properties. Users can continue to use the property window while this work is in progress.

Meters

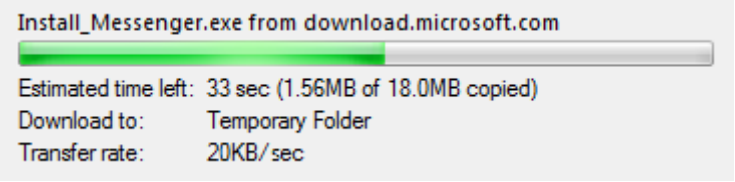
Expand table

Type	Description
Meters Indicate a percentage that is not related to progress.	This pattern isn't a progress bar, but it is implemented using the progress bar control. meters have a distinct look to differentiate them from true progress bars.  In this example, the meter shows the percentage of disk drive space used.

Guidelines

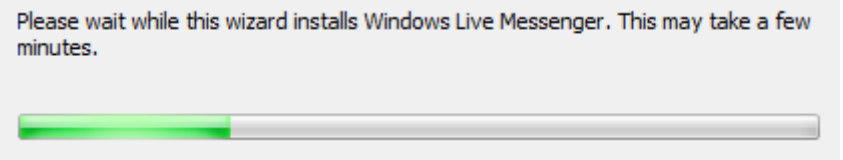
General

- **Provide progress feedback when performing a lengthy operation.** Users should never have to guess if progress is being made.
- **Clearly indicate real progress.** The progress bar must advance if progress is being made. If the range of expected completion times is large, consider using a non-linear scale to indicate progress for the longer times. You don't want users to conclude that your program has locked up when it hasn't.
- **Clearly indicate lack of progress.** The progress bar must not advance if no progress is being made. You don't want users to wait indefinitely for an operation that is never going to complete.
- **Provide useful progress details.** Provide additional progress information, but only if users can do something with it. Make sure the text is displayed long enough for users to be able to read it.

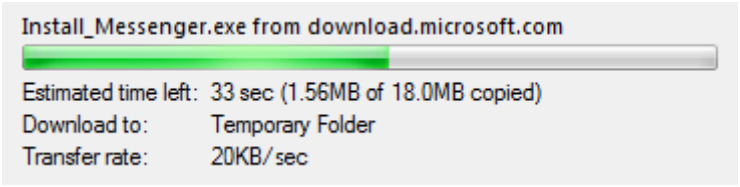
  In this example, users can see the transfer rate. The low transfer rate here suggests the need for using a high-bandwidth network connection.
- **Don't provide unnecessary details.** Generally users don't care about the details of the operation being performed. For example, users of a setup program don't care about the specific file being copied or that system components are being registered because they have no expectations about these details. Typically, a well-labeled progress bar alone provides sufficient information, so provide additional progress information only if users can do something with it. Providing details that users don't care about makes the user experience overly complicated and technical. If you need more detailed information for debugging, don't display it in release builds.

  Correct:

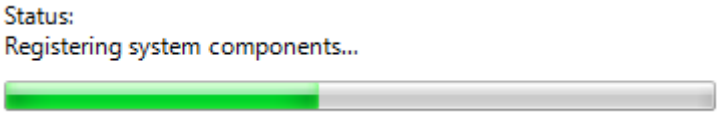
In this example, the labeled progress bar is all that is needed.

Correct:



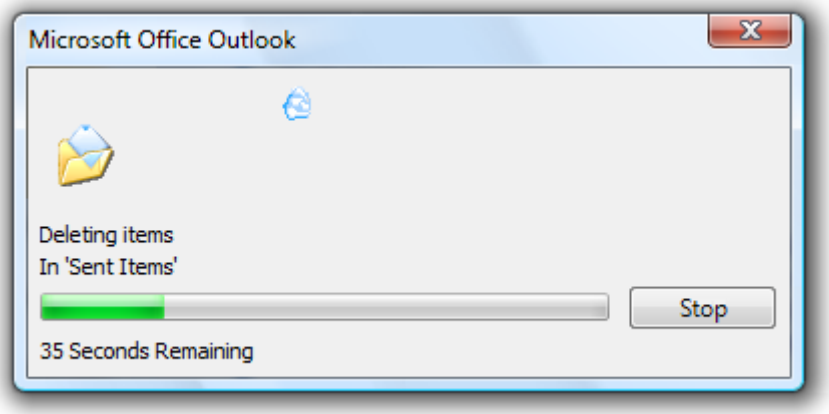
In this example, Windows Explorer is copying files the user selected, so displaying the filenames being copied is meaningful.

Incorrect:



In this example, a setup program is providing details that are meaningless to the user.

- **Provide useful animations.** If done well, animations improve the user experience by helping users visualize the operation. Good animations have more impact than text alone. For example, the progress bar for the Outlook Delete command displays the Recycle Bin for the destination if the files can be recovered, but no Recycle Bin if the files can't be recovered.



In this example, the lack of a Recycle Bin reinforces that the files are being permanently deleted. This additional information wouldn't be communicated as effectively using text alone.

- **Don't use unnecessary animations.** Animations can be misleading because they usually run in a separate thread from the actual task and therefore can suggest progress even if the operation has locked up. Also, if the operation is slower than expected, users sometimes assume that the animation is part of the reason.

Consequently, only use animations when there is a clear justification; don't use them to try to entertain users.

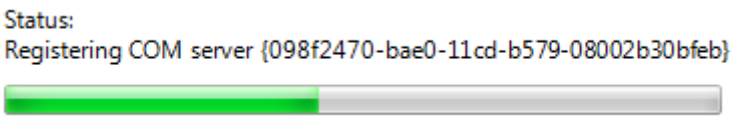
- **Position animations centered over the progress bar.** Put the animation above the progress bar labels, if you have any. If there is a Cancel or Stop button to the right of the progress bar, include the button when determining the center.

- **Play a sound effect at the completion of an operation only if it is very lengthy (longer than two minutes), infrequent, and important.** If the user is likely to walk away from an important operation while it is processing, a sound effect restores the user's attention. Using a sound effect upon completion in other circumstances would be a distracting annoyance.

- **Don't steal input focus to show a progress update or completion.** Users often switch to other programs while waiting and don't want to be interrupted. Background tasks must stay in the background.

- **Don't worry about technical support.** Because the feedback provided by progress bars isn't necessarily accurate and is fleeting, progress bars aren't a good mechanism for providing information for technical support. Consequently, if the operation can fail (as with a setup program), don't provide additional progress information that is only useful to technical support. Instead, provide an alternative mechanism such as a log file to record technical support information.

  Incorrect:

  In this example, the progress bar is showing details intended for technical support.

- **Don't put the percentage complete or any other text on a progress bar.** Such text isn't accessible and isn't compatible with using themes.

  Incorrect:

  In this example, the percentage text on the progress bar isn't accessible.

- **Don't combine a progress bar with a busy pointer.** Use one or the other, but not both at the same time.

- **Don't use vertical progress bars.** Horizontal progress bars have a more natural mapping and better flow.

Determinate progress bars

- **Use determinate progress bars for operations that require a bounded amount of time,** even if that amount of time cannot be accurately predicted. Indeterminate progress bars show that progress is being made, but provide no other information. Don't choose an indeterminate progress bar based only on the possible lack of accuracy alone.

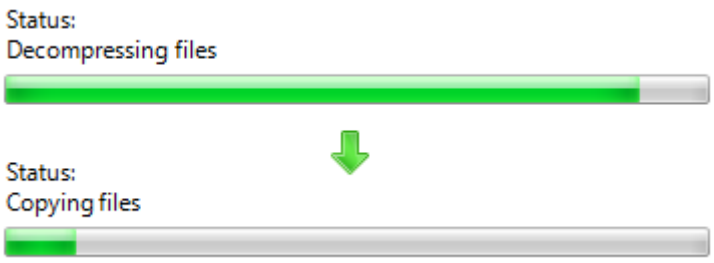
- **Clearly indicate the progress phase.** The progress bar must be able to indicate if the operation is in the beginning, middle, or end of an operation. For example, progress bars that immediately shoot to 99 percent completion, then stay there for a long time are particularly uninformative and annoying. In these cases, the progress bar should be set initially to at most 33 percent to indicate that the operation is still in the beginning phase.

- **Clearly indicate completion.** Don't let a progress bar go to 100 percent unless the operation has completed.

- **Provide a time remaining estimate if you can do so accurately.** Time remaining estimates that are accurate are useful, but estimates that are way off the mark or bounce around significantly aren't helpful. You may need to perform some processing before you can give accurate estimates. If so, don't display potentially inaccurate estimates during this initial period.

- **Don't restart progress.** A progress bar loses its value if it restarts (perhaps because a step in the operation completes) because users have no way of knowing when the operation will complete. Instead, have all the steps in the operation share a portion of the progress and have the progress bar go to completion once.

  Incorrect:

  In this example, the operation moved to the step of copying files and reset the progress bar for that step. Now users have no idea how much progress has been

made or how much time is left.

- **Don't back up progress.** As with a restart, a progress bar loses its value if it backs up. Always increase progress monotonically. However, you can have a time remaining estimate that increases (as well as decreases) because the rate of progress may vary.

Indeterminate progress bars

- **Use indeterminate progress bars only for operations whose overall progress cannot be determined.** Use indeterminate progress bars for operations that require an unbounded amount of time or that access an unknown number of objects. Use timeouts to give bounds to time-based operations.
- **Convert to a determinate progress bar once the overall progress can be determined.** For example, if it takes significantly longer than two seconds to determine the number of objects, you can use an indeterminate progress bar while the objects are counted, and then convert to a determinate progress bar.
- **Don't combine indeterminate progress bars with percent complete or time remaining estimates.** If you can provide this information, use a determinate progress bar instead.
- **Don't combine indeterminate progress bars with animations.** An indeterminate progress bar is effectively a generic animation, so you should use one or the other but never both.

  Correct:

  In this example, only an animation is used to show that an operation is ongoing.

Modeless progress bars

- **If users can do something productive while the operation is in progress, provide modeless feedback.** You might need to disable a subset of functionality that requires the operation to complete.
- **If the window has an address bar, display the modeless progress in the address bar.**

  In this example, modeless progress is shown in the address bar.
- Otherwise, **if the window has a status bar, display the modeless progress in the status bar.** Put any corresponding text to its left in the status bar.

  In this example, modeless progress is shown in the status bar.

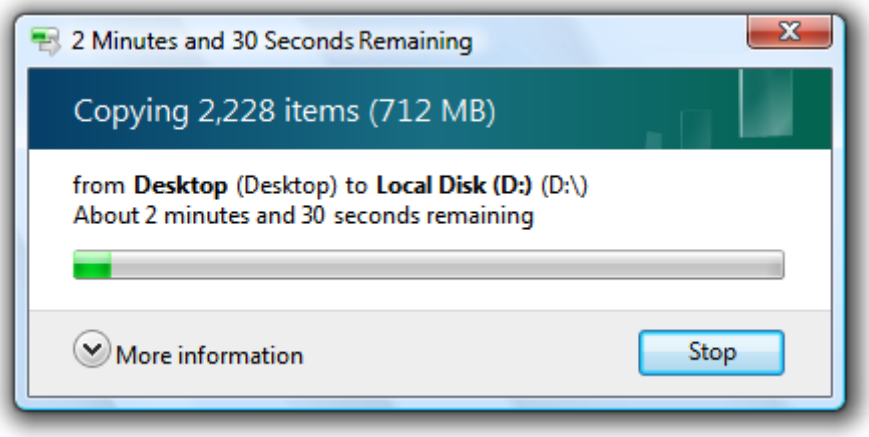
Modal progress bars

- **Place modal progress bars on progress pages or** progress dialog boxes.
- **Provide a command button to halt the operation if it takes more than a few seconds to complete, or has the potential never to complete.** Label the button Cancel if canceling returns the environment to its previous state (leaving no side effects), otherwise label the button Stop to indicate that it leaves the partially completed operation intact. You can change the button label from Cancel to Stop in the middle of the operation if at some point it isn't possible to return the environment to its previous state. Center the command button vertically with the progress bar instead of aligning their tops.

  Correct:

  In this example, halting the network connection has no side effect so Cancel is used.

  Correct:



In this example, halting the copy leaves any copied files, so the command button is labeled Stop.

Incorrect:

In this example, halting the search leaves no side effect, so the command button should be labeled Cancel.

Time remaining

For determinate progress bars:

- **Use the following time formats.** Start with the first of the following formats where the largest time unit isn't zero, and then change to the next format once the largest time unit becomes zero.

  For progress bars:

  If related information is shown in a colon format:

  Time remaining: h hours, m minutes

  Time remaining: m minutes, s seconds

  Time remaining: s seconds

  If screen space is at a premium:

  h hrs, m mins remaining

  m mins, s secs remaining

s seconds remaining

Otherwise:

h hours, m minutes remaining

m minutes, s seconds remaining

s seconds remaining

For title bars:

hh:mm remaining

mm:ss remaining

0:ss remaining

This compact format shows the most important information first so that it isn't truncated on the taskbar.

- **Make estimates accurate, but don't give false precision.** If largest unit is hours, give minutes (if meaningful) but not seconds.

  Incorrect:

  hh hours, mm minutes, ss seconds

- **Keep the estimate up-to-date.** Update time remaining estimates at least every 5 seconds.

- **Focus on the time remaining** because that is the information users care about most. Give total elapsed time only when there are scenarios where elapsed time is helpful (such as when the task is likely to be repeated). If the time remaining estimate is associated with a progress bar, don't have percent complete text because that information is conveyed by the progress bar itself.

- **Be grammatically correct.** Use singular units when the number is one.

  Incorrect:

  1 minutes, 1 seconds

- **Use sentence-style capitalization.**

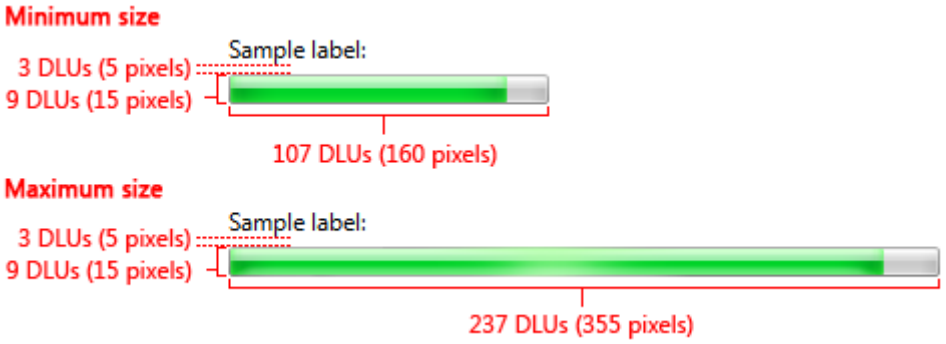
Progress bar colors

- **Use red or yellow progress bars only to indicate the progress status, not the final results of a task.** A red or yellow progress bar indicates that users need to take some action to complete the task. If the condition isn't recoverable, leave the progress bar green and display an error message.
- **Turn the progress bar red when there is a user recoverable condition that prevents making further progress.** Display a message to explain the problem and recommend a solution.
- **Turn the progress bar yellow to indicate either that the user has paused the task or that there is a condition that is impeding progress** but progress is still taking place (as, for example, with poor network connectivity). If the user has paused, change the Pause button label to Resume. If progress is impeded, display a message to explain the problem and recommend a solution.

Meters

- **Use progress bars only for progress.** Use meters to indicate percentages that aren't related to progress.

Recommended sizing and spacing



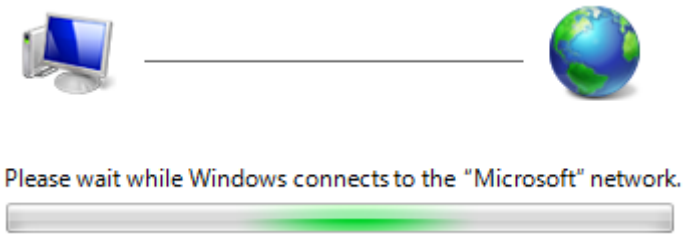
Recommended sizing and spacing for progress bars.

- Always use the recommended progress bar height.
  - **Exception:** You may use a different height if the parent window doesn't support the recommended height.
- Use the minimum width if you want to make the progress bar unobtrusive.
- Don't use widths longer than the maximum recommended. The progress bar doesn't have to fill the available space.
- Center the progress bar horizontally if the window is much wider than the maximum recommended width.

Labels

Progress bar labels

- **Use a concise label with a static text control to indicate what the operation is doing.** Start the label with a verb (for example, Copying) and end with an ellipsis. This label may change dynamically if the operation has multiple steps or is processing multiple objects.
- Don't assign a unique access key because the control isn't interactive.
- Use sentence-style capitalization.
- If the operation was not directly initiated by the user, you can include an additional label to give the context and apologize for the interruption. Start this extra label with the phrase, Please wait while. This label should not change during the operation.

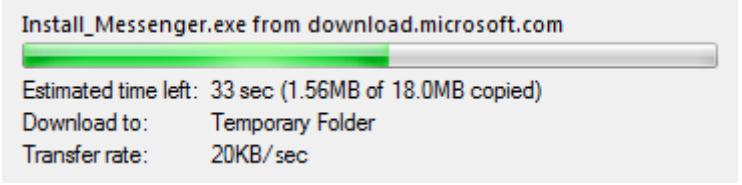
  

  In this example, the user is being asked to please wait because the user didn't directly initiate the operation.
- Position the label above the progress bar and align the label with the left edge of the progress bar.

Progress bar details

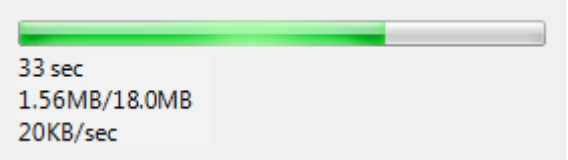
- Provide details in static text, preceding the data with a label ending with a colon. Specify units (seconds, kilobytes, and so on) after the details text.

  Correct:

  In this example, the details are properly labeled.

  Incorrect:



In this example, the details aren't labeled, thus requiring users to determine their meaning.

- Use sentence-style capitalization.
- Position the details below the progress bar and align the label with the left edge of the progress bar.
- Don't give the percentage completed or remaining because that information is conveyed by the progress bar itself.

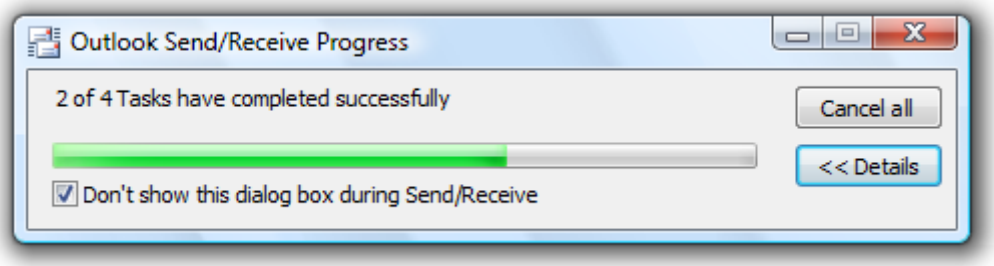
Cancel button

- Label the button Cancel if canceling returns the environment to its previous state (leaving no side effect); otherwise, label the button Stop to indicate that it leaves the partially completed operation intact.
- You can change the button label from Cancel to Stop in the middle of the operation if at some point it isn't possible to return the environment to its previous state.

Progress dialog box titles

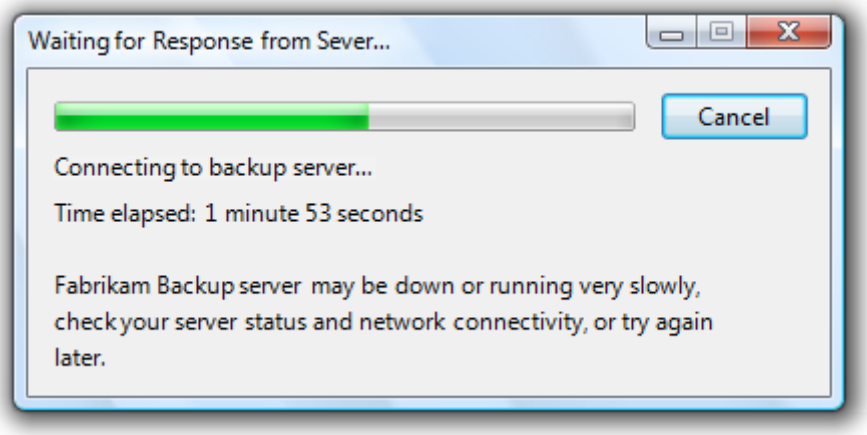
- If the progress bar is displayed in a modal dialog box, the dialog box title should be the name of the program or the name of the operation. Don't use what should be the progress bar label for the dialog box title.

  Correct:



  In this example, the task name is used for the dialog box title.

  Incorrect:



In this example, the dialog box title text is a restatement of the progress bar label. The program name should be used instead.

- If the progress bar is displayed in a modeless dialog box, optimize the title for display on the taskbar by concisely placing the distinguishing information first. Example: "66% Complete."

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Progressive Disclosure Controls

Article • 04/29/2022

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

With a progressive disclosure control, users can show or hide additional information including data, options, or commands. Progressive disclosure promotes simplicity by focusing on the essential, yet revealing additional detail as needed.

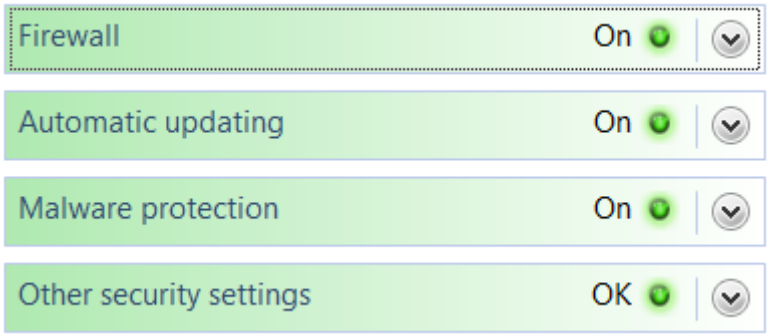
Examples of progressive disclosure controls.

> **Note**
>
> Guidelines related to layout, menus, and toolbars are presented in separate articles.

Is this the right control?

To decide, consider these questions:

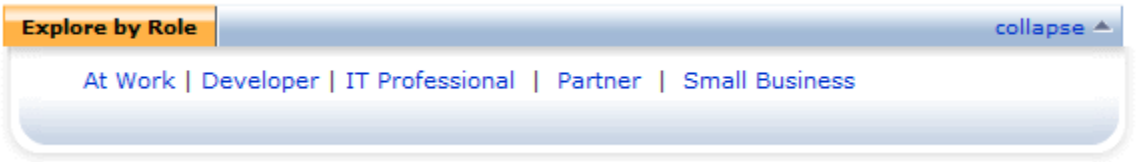
- **Do users need to see the information in some but not all scenarios, or some but not all of the time?** If so, displaying the information using progressive disclosure simplifies the baseline experience, yet allows users to access the information easily.



In this example, the Windows Security app displays the important security status all the time, but uses progressive disclosure to display details on demand.

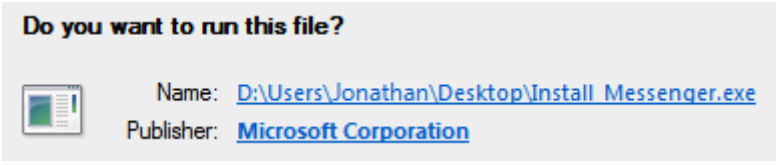
- **If the information is displayed by default, are users ever likely to choose to hide it?** Are there scenarios where users will need more space? Are users sufficiently motivated to customize the user interface (UI)? If not, display the information without using progressive disclosure.

  Incorrect:

  In this example, users won't be motivated to hide the information.

- **Is the additional information advanced, substantial, complex, or related to an independent subtask?** If so, consider displaying the information in a separate window using command buttons or links instead of using a progressive disclosure control. (Additional information is advanced if it is intended for advanced users. It's complex if it makes other information hard to read or lay out.)

  In this example, information about the software's name and publisher is meaningful primarily to advanced users, so links to separate windows are used.

- **Is the additional information a sentence or sentence fragment that describes what an item does or how it can be used?** If so, consider using a tooltip or infotip.

- **Is the additional information related to the current task, but independent of the currently displayed information?** If so, consider using tabs instead. However, collapsible lists are often preferable to tabs because they are more flexible and scalable.

- **Is showing or hiding the additional information essentially a data filter?** If so, consider using a drop-down list or check boxes instead to apply the filter to the entire list.

Design concepts

The goals of progressive disclosure are to:

- **Simplify a UI** by focusing on the essential, yet revealing additional detail as needed.
- **Simplify a UI's appearance** by reducing the perception of clutter.

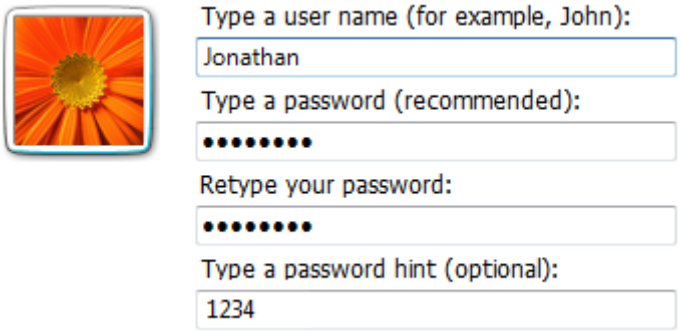
Both goals can be achieved by using progressive disclosure controls, where users click to see more detail. However, you can achieve the second goal of simplifying the appearance without using explicit progressive disclosure controls by:

- **Showing contextual detail only in context.** For example, you can show contextual commands or toolbars automatically when relevant to the selected object or mode.

- **Reducing the weight of affordances for secondary UI.** Affordances are visual properties that suggest how objects are used. The trend is to have UI that users can interact with in place, but to have all such UI drawn to scream "click me!" leads to too much visual clutter. For secondary UI, it is often better to use subtle affordances and give the full effects on mouse over.

  In this example, the Rating field is interactive, but doesn't appear so until mouse hover.

- **Showing follow-up steps only after prerequisites are done.** This approach is best used with familiar tasks where users can confidently take the first steps.

  In this example, the user name and password page initially shows only the user name and optional password boxes. The confirmation and hint boxes are displayed after the user enters a password.

While progressive disclosure is a great way to simplify UIs, it has these risks:

- **Lack of discoverability.** Users may assume that if they can't see something, it doesn't exist. Users may not hover or click if they don't see what they are looking for. There is always a chance that users might not click things like More options.
- **Lack of stability.** Progressive disclosure should be expected or at least feel natural. If controls unexpectedly appear and disappear, the resulting UI can feel unstable.

Progressive disclosure controls

Progressive disclosure controls are usually displayed without direct labels that describe their behavior, so users must be able to do the following based on the control's visual appearance alone:

- Recognize that the control provides progressive disclosure.
- Determine if the current state is expanded or collapsed.
- Determine if additional information, options, or commands are needed to perform the task.
- Determine how to restore the original state, if desired.

While users can determine the above by trial and error, you should try to make such experimentation unnecessary.

Progressive disclosure controls have a fairly weak affordance, which means their visual properties suggest how they are used, albeit weakly. The following table compares the appearance of the common progressive disclosure controls:

Expand table

Control	Purpose	Appearance	Glyph indicates
Chevrons	**Show all:** Show or hide the remaining items in completely or partially hidden content. Items are either shown in place (using a single chevron) or in a pop-up menu (using a double chevron).	Chevrons point in the direction where the action will occur.	Future state
Arrows	**Show options:** Show a pop-up command menu.	Arrows point in the direction where the action will occur.	Future state
Plus and minus	**Expand containers:** Expand or collapse container content in place when navigating through a hierarchy.	Plus and minus symbols don't point, but the action always occurs to their right.	Future state

Control	Purpose	Appearance	Glyph indicates
controls			
Rotating triangles	**Show details:** Show or hide additional information in place for an individual item. They are also used to expand containers.	Rotating triangles somewhat resemble rotating levers, so they point in the direction where the action has occurred.	Present state

If you do only one thing...

Users should be able to predict a progressive disclosure control's behavior correctly by inspection alone. To achieve this, select the appropriate usage patterns and apply their appearance, location, and behavior consistently.

Usage patterns

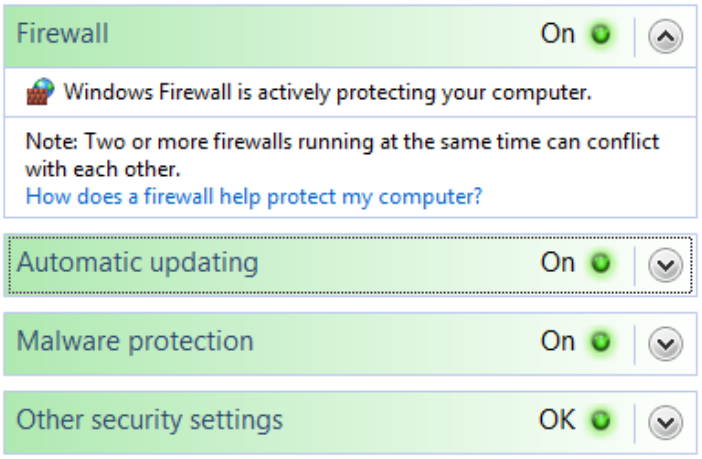
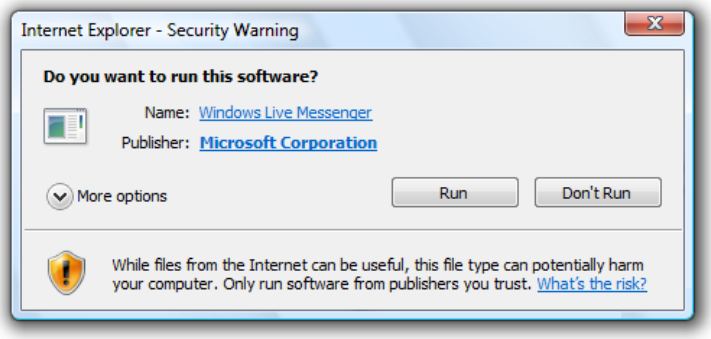
Progressive disclosure controls have several usage patterns. Some of them are built into common controls.

Chevrons

Chevrons show or hide the remaining items in completely or partially hidden content. Usually the items are shown in place, but they can also be shown in a pop-up menu. When in place, the item stays expanded until the user collapses it.

Chevrons are used in the following ways:

Expand table

Usage	Example
In-place UI the associated object receives input focus and the single chevron is activated with the space bar.	 In these examples, the in-place single chevrons are positioned to the right of their associated control.
Command buttons with external labels the command button receives input focus and the single chevron is activated with the space bar.	 In this example, the single chevron button is labeled and positioned to the left of the label. With this pattern, the button would be difficult to understand without its label.
Command buttons with internal labels the command button receives input focus and is activated with the space bar.	 In these examples, regular command buttons have the double chevron positioned to suggest their meaning.

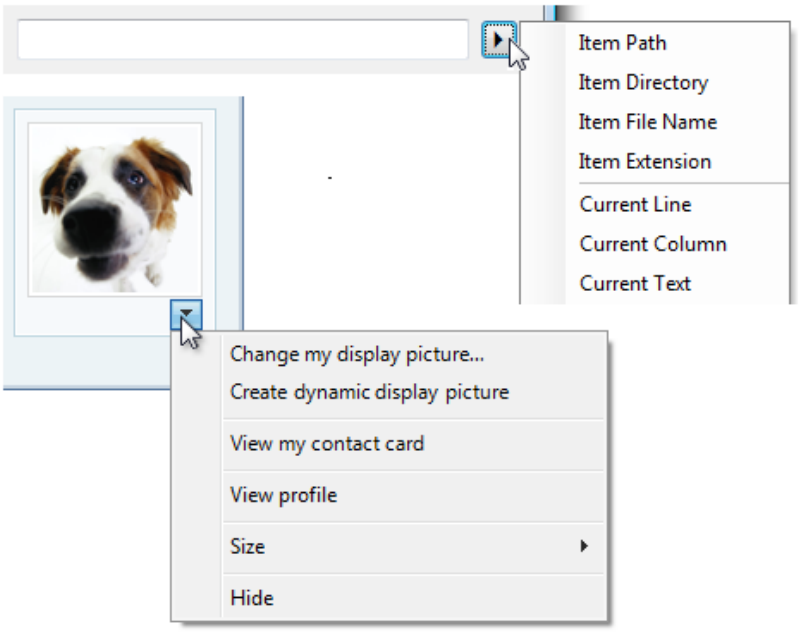
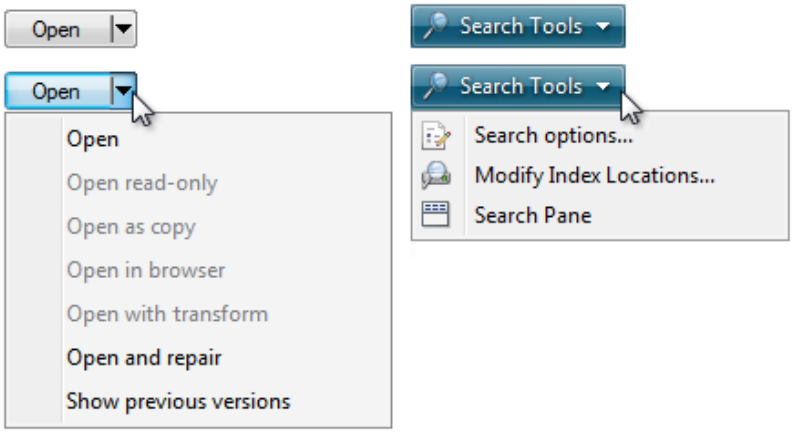
Arrows

Arrows show a pop-up command menu. The item stays expanded until the user makes a selection or clicks anywhere.

If the arrow button is an independent control, it receives input focus and is activated with the space bar. If the arrow button has a parent control, the parent receives input focus and the arrow is activated with Alt+down arrow and Alt+up arrow keys, as with the drop-down list control.

Arrows are used in the following ways:

Expand table

Usage	Example
Separate buttons the arrow is in a separate button control.	 In these examples, separate arrow buttons positioned to the right indicate a command menu.
Command buttons the arrow is part of a command button.	 In these examples, menu buttons and split buttons have the arrows positioned to the right of the text.

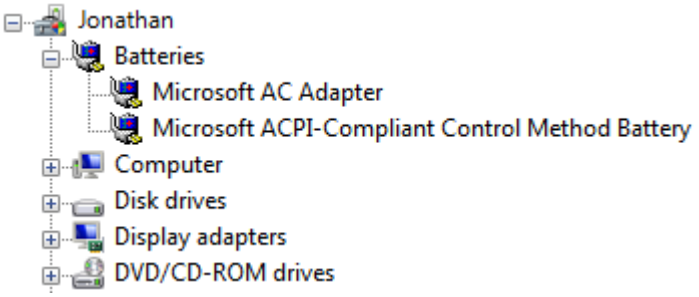
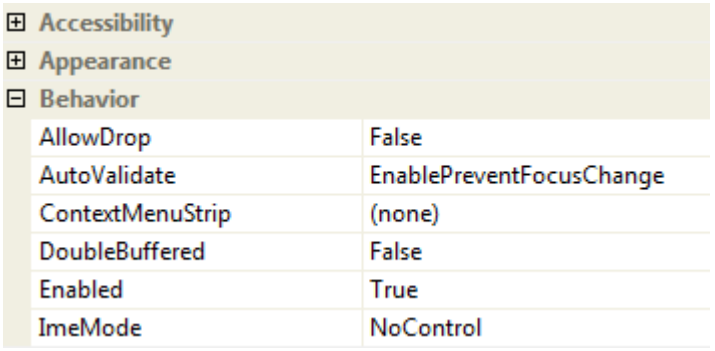
Plus and minus controls

Plus and minus controls expand or collapse to show container content in place when navigating through a hierarchy. The item stays expanded until the user collapses it. Although these look like buttons, their behavior is in-place.

The associated object receives input focus. The plus is activated with the right arrow key, and the minus with the left arrow key.

Plus and minus controls are used in the following ways:

Expand table

Usage	Example
Collapsible trees a multi-level hierarchy to show container content.	 In this example, the plus and minus controls are positioned to the left of the associated container.
Collapsible lists a two-level hierarchy to show container content.	 In this example, the plus and minus controls are positioned to the left of the associated list header.

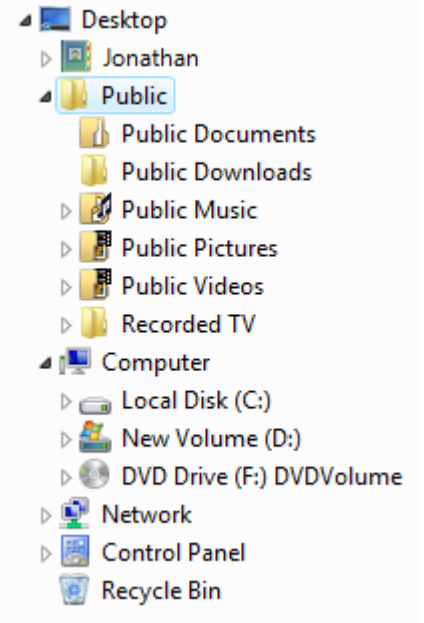
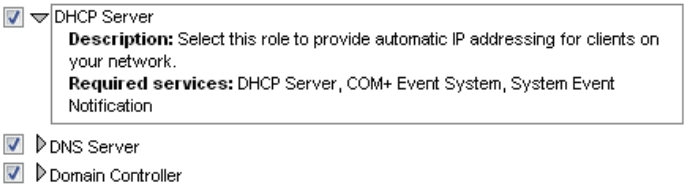
Rotating triangles

Rotating triangles show or hide additional information in place for an individual item. They are also used to expand containers. The item stays expanded until the user collapses it.

The associated object receives input focus. The collapsed (right-pointing) triangle is activated with the right arrow key, and the expanded (downward-pointing) triangle with the left arrow key.

Rotating triangles are used in the following ways:

Expand table

Usage	Example
Collapsible trees a multi-level hierarchy to show container content.	 In this example, the rotating triangles are positioned to the left of the associated container.
Collapsible lists a two-level hierarchy to show additional information in place.	 In this example, the rotating triangles are positioned to the left of their associated list items.

Preview arrows

Like chevrons, additional information is shown or hidden in place. The item stays expanded until the user collapses it. Unlike chevrons, the glyphs have a graphical representation of the action, typically with an arrow indicating what will happen.

In these examples from Microsoft Windows Media Player, the glyphs have arrows that suggest the action that will happen.

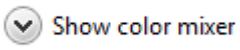
Preview arrows are best reserved for situations where a standard chevron doesn't adequately communicate the control's behavior, such as when the disclosure is complex or there is more than one type of disclosure.

Guidelines

General

- **Select the progressive disclosure pattern based on its usage.** For a description of each usage pattern, see the previous table.
- **Don't use links for progressive disclosure controls.** Use only the progressive disclosure controls presented in the Usage patterns section. However, do use links to navigate to Help topics.

  Correct:

  Incorrect:

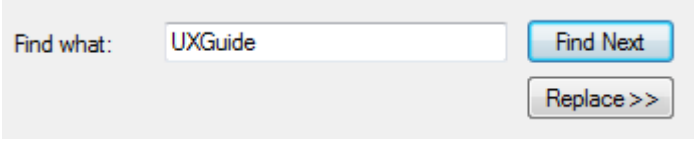
  In the incorrect example, a link is used to show more options in place. This usage would be correct if the link navigated to another page or dialog box, or displayed a Help topic.

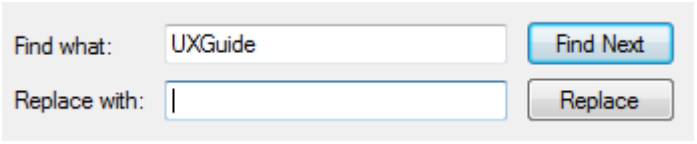
Interaction

- **For chevrons and arrows that aren't directly labeled, use tooltips to describe what they do.**

  In this example, the tooltip indicates the effect of an unlabeled chevron control.
- **If a user expands or collapses an item, make the state persist so it takes effect the next time the window is displayed**, unless users are likely to prefer starting in the default state. Make the state persist on a per-window, per-user basis.
- **Make sure that all expanded content can be collapsed and vice versa, and that the inverse operation is obvious.** Doing so encourages exploration and reduces frustration. The best way to make the inverse operation obvious is to keep the control in the same fixed location. If you need to move the control, keep it in the same relative location within a visually distinct area.

  Incorrect:





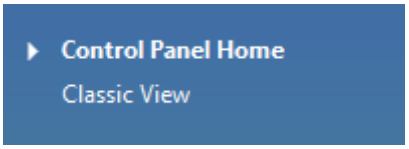
In this example, clicking the Replace button with the chevron reveals the **Replace with** text box. Once this is done, the Replace expander becomes the Replace command, so there is no way to restore the original state.

- **Use only the access keys appropriate for the progressive disclosure pattern**, as listed in the Usage patterns section. Don't use Enter to activate progressive disclosure.

Presentation

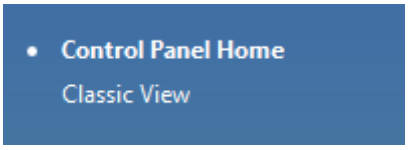
- **Don't use triangular-shaped arrowheads for a purpose other than progressive disclosure.**

  Incorrect:

  Although this example isn't a progressive disclosure pattern, using an arrow here suggests that commands will be shown in a pop-up window.

  Correct:

  In this example, a bullet is correctly used instead.

- **Remove (don't disable) progressive disclosure controls that don't apply in the current context.** Progressive disclosure controls should always deliver on their promise, so remove them when there isn't more information to give.

  Incorrect:

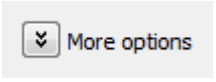
In this example, a progressive disclosure control that doesn't apply is incorrectly disabled.

Chevrons

- **Use single chevrons to show or hide in place. Use double chevrons to show or hide using a pop-up menu.** You should always use double chevrons for command buttons with internal labels, however.

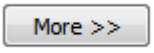
  Correct:

  Incorrect:

  In the incorrect example, a double chevron is used for in-place progressive disclosure.

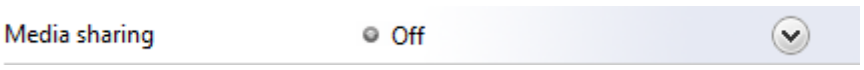
  Correct:

  In this example, a double chevron is used for in-place progressive disclosure because it is a command button with an internal label.

- **Provide a visual relationship between the chevron and its associated control.** Because in-place chevrons are placed to the right of their associated UI and right aligned, there can be quite a distance between a chevron and its associated control.

  Correct:

  In this example, there is a clear relationship between the in-place chevron and its associated UI.

  Incorrect:

In this example, there is no clear visual relationship between the in-place chevron and its associated UI, so it seems to be floating in space.

Arrows

- **Don't use arrow graphics that could be confused with Back, Forward, Go, or Play.** Use simple triangular-shaped arrowheads (arrows without stems) on neutral backgrounds.

  Correct:

  These arrows are clearly progressive disclosure controls.

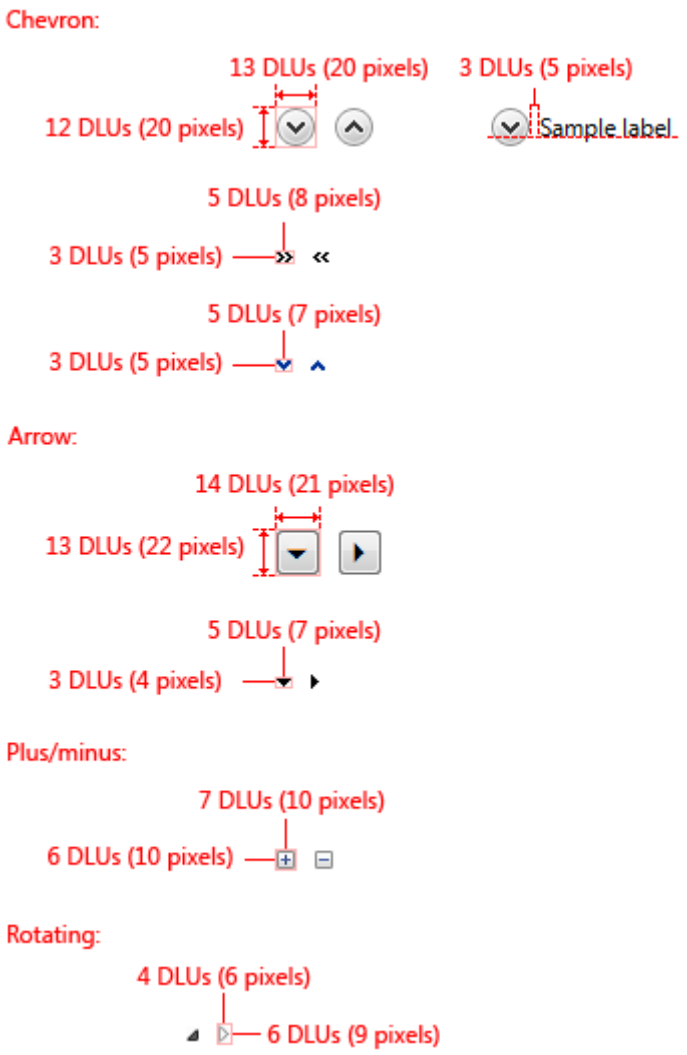
  Incorrect (for progressive disclosure):

  These arrows don't look like progressive disclosure controls.

  Incorrect (for Back, Forward):

  These arrows look like progressive disclosure controls, but they are not.

Recommended sizing and spacing



Recommended sizing and spacing for progressive disclosure controls.

Labels

- For chevrons on a command button with an external label:
  - Assign a unique access key. For guidelines, see Keyboard.
  - Use sentence-style capitalization.
  - Write the label as a phrase and use no ending punctuation.
  - Write the label so that it describes the effect of clicking the button, and change the label when the state changes.
  - If the surface always displays some options, commands, or details, use the following label pairs:
    - **More/Fewer options.** Use for options or a mixture of options, commands, and details.
    - **More/Fewer commands.** Use for commands only.
    - **More/Fewer details.** Use for information only.
    - **More/Fewer <object name>.** Use for other object types, such as folders.
  - Otherwise:

    - **Show/Hide options.** Use for options or a mixture of options, commands, and details.
    - **Show/Hide commands.** Use for commands only.
    - **Show/Hide details.** Use for information only.
    - **Show/Hide <object name>.** Use for other object types, such as folders.
- For chevrons on a command button with an internal label, follow the standard command button guidelines.

Documentation

When referring to progressive disclosure controls:

- If the control has a fixed label, refer to the control by its label only; don't try to describe the control. Use the exact label text, including its capitalization, but don't include the access key underscore.

- If the control has no label or it isn't fixed, refer to the control by its type: chevron, arrow, triangle, or plus/minus button. If necessary, also describe the control's location. If the control appears dynamically, such as the page space control, start the reference by describing how to make the control appear.

  Example: To display the files within a folder, move the pointer to the start of the folder name, and then click the triangle next to the folder.

- Don't refer to the control as a button, unless to contrast with other progressive disclosure controls that aren't buttons.

- To describe user interaction, use click. If needed for clarity, use click...to expand or collapse.

- When possible, format the label using bold text. Otherwise, put the label in quotation marks only if required to prevent confusion.

Examples:

- (For a chevron) To determine the file size, click **Details**.
- (For an arrow) To see all the options, click the arrow next to the **Search** box.
- (For plus/minus) To view your picture, click **Pictures**.

Feedback

Was this page helpful? Yes No

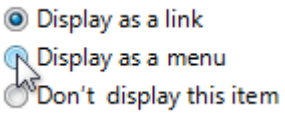
Provide product feedback | Get help at Microsoft Q&A

Radio Buttons

Article • 02/09/2021

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

With a radio button, users make a choice among a set of mutually exclusive, related options. Users can choose one and only one option. Radio buttons are so called because they function like the channel presets on radios.



A typical group of radio buttons.

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

> **Note**
>
> Guidelines related to layout and keyboard navigation are presented in a separate article.

Is this the right control?

To decide, consider these questions:

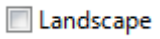
- **Is the control used to choose one option from a set of mutually exclusive choices?** If not, use another control. To choose multiple options, use check boxes, a multiple-selection list or a check box list instead.
- **Is the number of options between two and seven?** Since the screen space used is proportional to the number of options, keep the number of options in a group

between two and seven. For eight or more options, use a drop-down list or single-selection list.

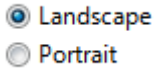
- **Would a check box be a better choice?** If there are only two options, you could use a single check box instead. However, check boxes are suitable only for turning a single option on or off, whereas radio buttons can be used for completely different alternatives. If both solutions are possible:

  - Use radio buttons if the meaning of the cleared check box isn't completely obvious.

    Incorrect:

    Correct:

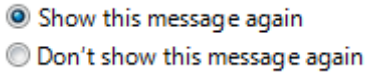
    

    In the correct example, the choices are not opposites so radio buttons are the better choice.

  - Use radio buttons on wizard pages to make the alternatives clear, even if a check box is otherwise acceptable.

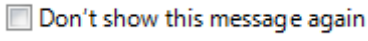
  - Use radio buttons if you have enough screen space and the options are important enough to be a good use of that screen space. Otherwise, use a check box or drop-down list.

    Incorrect:

    In this example, the options aren't important enough to use radio buttons.

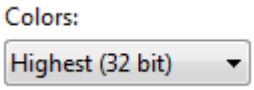
    Correct:

    In this example, a check box is an efficient use of screen space for this peripheral option.

  - Use a check box if there other check boxes on the page.

- **Would a drop-down list be a better choice?** If the default option is recommended for most users in most situations, radio buttons might draw more attention to the options than necessary.
    - Consider using a drop-down list if you don't want to draw attention to the options, or you don't want to encourage users to make changes. A drop-down list focuses on the current selection, whereas radio buttons emphasize all options equally.

      In this example, a drop-down list focuses on the current selection and discourages users from making changes.
    - Consider a drop-down list if there are other drop-down lists on the page.
- **Would a set of command buttons, command links, or a split button be a better choice?** If the radio buttons are used only to affect how a command is performed, it is often better to present the command variations instead. Doing so allows users to choose the right command with a single interaction.
- **Do the options present program options, rather than data?** The options' values shouldn't be based on context or other data. For data, use a drop-down list or single-selection list.
- If the control is used on a wizard page or control panel, **is the control a response to the main instruction and can users later change the choice?** If so, consider using command links instead of radio buttons to make the interaction more efficient.
- **Are the values non-numeric?** For numeric data, use text boxes, drop-down lists, or sliders.

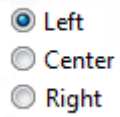
Guidelines

General

- **List the options in a logical order,** such as most likely to be selected to least, simplest operation to most complex, or least risk to most. Alphabetical ordering is not recommended because it is language dependent and therefore not localizable.

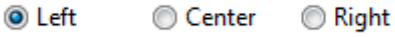
- **If none of the options is a valid choice, add another option to reflect this choice,** such as None or Does not apply.
- **Prefer to align radio buttons vertically instead of horizontally.** Horizontal alignment is harder to read and localize.

  Correct:

  In this example, the radio buttons are aligned vertically.

  Incorrect:

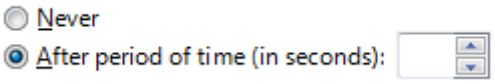
  

  In this example, the horizontal alignment is harder to read.

- **Reconsider using group boxes to organize groups of radio buttons**—this often results in unnecessary screen clutter.
- **Don't use radio button labels as group box labels.**
- **Don't use the selection of a radio button to:**
  - Perform commands.
  - Display other windows, such as a dialog box to gather more input.
  - Dynamically show or hide other controls related to the selected control (screen readers cannot detect such events). However, you can change text dynamically based on the selection.

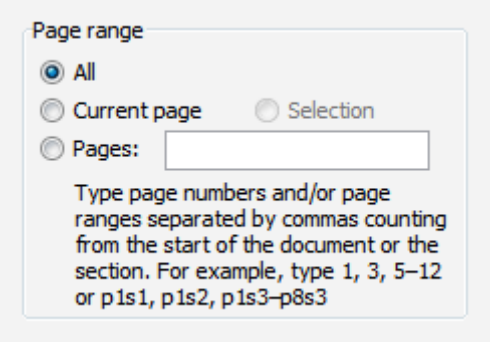
Subordinate controls

- Place subordinate controls to the right of or below (indented, flush with the radio button label) the radio button and its label. End the radio button label with a colon.

  In this example, the radio button and its subordinate control share the radio button label and its access key. In this case, the arrow keys move focus from the radio button to its subordinate text box.

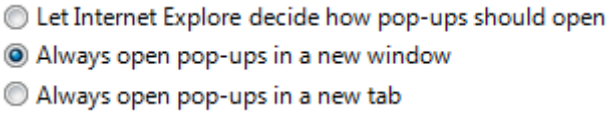
- **Leave dependent editable text boxes and drop-down lists enabled if they share the radio button's label.** When users type or paste anything into the box, select the corresponding option automatically. Doing so simplifies the interaction.



  In this example, entering a page number automatically selects Pages.

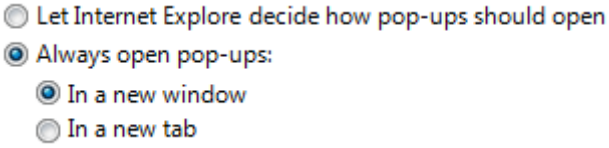
- **Avoid nesting radio buttons with other radio buttons or check boxes.** If possible, keep all the options at the same level.

  Correct:



  In this example, the options are at the same level.

  Incorrect:



  In this example, using nested options adds unnecessary complexity.

- If you do nest radio buttons with other radio buttons or check boxes, **disable these subordinate controls until the high-level option is selected.** Doing so avoids confusion about the meaning of the subordinate controls.

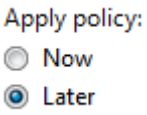
Default values

- Because a group of radio buttons represents a set of mutually exclusive choices, **always have one radio button selected by default. Select the safest (to prevent loss of data or system access) and most secure and private option.** If safety and security aren't factors, select the most likely or convenient option.

- **Exceptions:** Don't have a default selection if:
  - **There is no acceptable default option for safety, security, or legal reasons and therefore the user must make an explicit choice.** If the user doesn't make a selection, display an error message to force one.
  - **The user interface (UI) must reflect the current state and the option hasn't been set yet.** A default value would incorrectly imply that the user doesn't need to make a selection.
  - **The goal is to collect unbiased data.** Default values would bias data collection.
  - **The group of radio buttons represents a property in a mixed state**, which happens when displaying a property for multiple objects that don't have the same setting. Don't display an error message in this case since each object has a valid state.

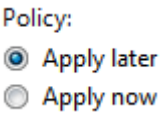
- **Make the first option the default option**, since users often expect that—unless that order isn't logical. To do this, you might need to change the option labels.

  Incorrect:

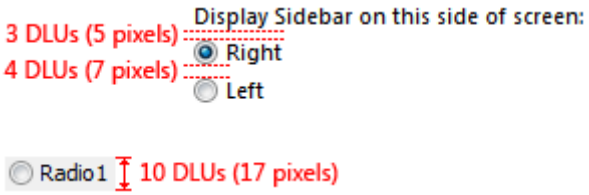
  In this example, the default option isn't the first option.

  Correct:

  In this example, the option labels are reworded to make the first option the default option.

Recommended sizing and spacing

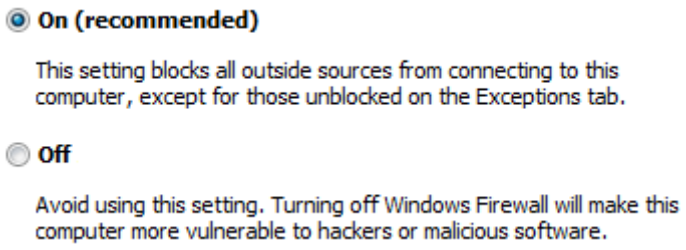


Recommended sizing and spacing for radio buttons.

Labels

Radio button labels

- Label every radio button.
- Assign a unique access key to each label. For guidelines, see Keyboard.
- Use sentence-style capitalization.
- Write the label as a phrase, not as a sentence, and use no ending punctuation.
  - **Exception:** If a radio button label also labels a subordinate control that follows it, end the label with a colon.
- Use parallel phrasing, and try to keep the length about the same for all labels.
- Focus the label text on the differences among the options. If all the options have the same introductory text, move that text to the group label.
- Use positive phrasing. For example, use do instead of do not, and print instead of do not print.
- Describe just the option with the label. Keep labels brief so it's easy to refer to them in messages and documentation. If the option requires further explanation, provide the explanation in a static text control using complete sentences and ending punctuation.



In this example, the options are explained using separate static text controls.

> **Note**
>
> Adding an explanation to one radio button doesn't mean that you have to provide explanations for all the radio buttons. Provide the relevant information in the label if you can, and use explanations only when necessary. Don't merely restate the label for consistency.

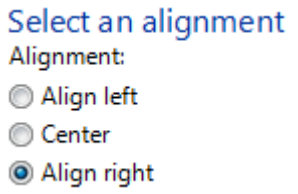
- **If an option is strongly recommended, add "(recommended)" to the label.** Be sure to add to the control label, not the supplemental notes.
- **If an option is intended only for advanced users, add "(advanced)" to the label.** Be sure to add to the control label, not the supplemental notes.
- If you must use multi-line labels, align the top of the label with the radio button.
- Don't use a subordinate control, the values it contains, or its units label to create a sentence or phrase. Such a design isn't localizable because sentence structure varies with language.

Radio button group labels

- Use the group label to explain the purpose of the group, not how to make the selection. Assume that users know how to use radio buttons. For example, don't say "Select one of the following choices".
- All radio button groups need labels. Write the label as a word or phrase, not as a sentence, ending with a colon using static text or a group box.

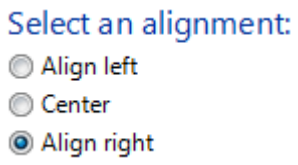
  Exception: Omit the label if it is merely a restatement of a dialog box's main instruction. In this case, the main instruction takes the colon (unless it's a question) and access key (if there is one).

  Acceptable:

  In this example, the radio button group label is just a restatement of the main instruction.

  Better:

  In this example, the redundant label is removed, so the main instruction takes the colon.

- Don't assign an access key to the label. Doing so isn't necessary and it makes the other access keys harder to assign.
  - **Exception:** If not all controls can have unique access keys, you can assign an access key to the label instead of the individual controls. For more information, see Keyboard.

Documentation

When referring to radio buttons:

- Use the exact label text, including its capitalization, but don't include the access key underscore or colon.
- In programming and other technical documentation, refer to radio buttons as radio buttons. Everywhere else use option buttons, especially in user documentation.
- To describe user interaction, use click.
- When possible, format the label using bold text. Otherwise, put the label in quotation marks only if required to prevent confusion.

Example: Click **Current page**, and then click **OK**.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Search Boxes

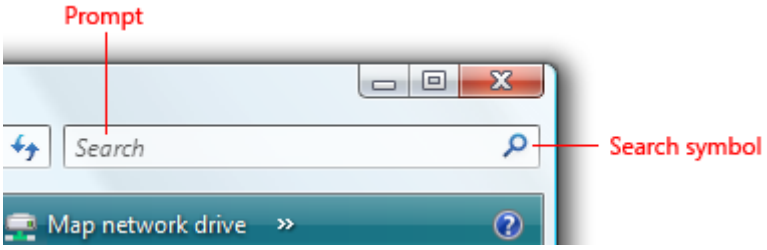
Article • 02/09/2021

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

With a Search box, users can quickly locate specific objects or text within a large set of data by filtering or highlighting matches. There is no standard search control, but you should strive to make your program's search features consistent with those of Windows.

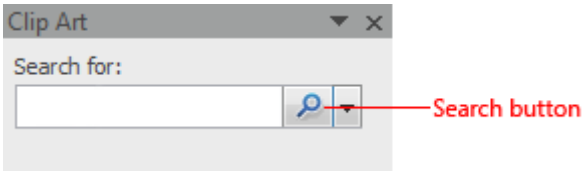
There are two types of searches:

- **Instant search**, where the results are displayed immediately as the user types. No button needs to be clicked, so the magnifying glass search symbol is shown as a graphic, not a button.

  A typical Search box using Instant search. Search is automatically executed on every keystroke.

- **Regular search**, where a search is performed when the user clicks the search button. The magnifying glass search symbol is shown on a button.

  A typical Search box using regular search. Users click a button to perform the search.

  You can provide either or both kinds of search options for your users.

Is this the right control?

To decide, consider these questions:

- **Are specific objects difficult to find?** This can happen when:
  - There are many objects.
  - The objects aren't located in a single location. Search is especially useful for finding objects in trees.
  - The search data is difficult to find (for example, metadata).
- **Do users need to find specific text within documents?**
- **Does your feature return relevant search results within five seconds?** If not, you can provide a search feature, but use an alternative design that gives visible feedback to accommodate long-running searches, such as a search dialog box.

Design concepts

Searching is a crucial first step in many scenarios: Users must find objects before they can use them. Users are saving more and more objects on increasingly larger hard disks, but browsing for objects doesn't scale well. Search must be a simple, consistent, reliable part of the user experience.

Search boxes in Windows:

- Are part of all Explorer windows, so they are easy to find and recognize.
- Have consistent appearance and behavior.
- Are efficient and fast, giving instant results in Instant search mode.

A Search box is used in Windows in these places:

- Explorers
- Experiences (Microsoft Windows Media Player, Windows Photo Gallery, Windows Internet Explorer)
- Start menu (to find programs and recent files)
- Control Panel home page (to find control panel items and tasks)
- Help (to find relevant Help topics)

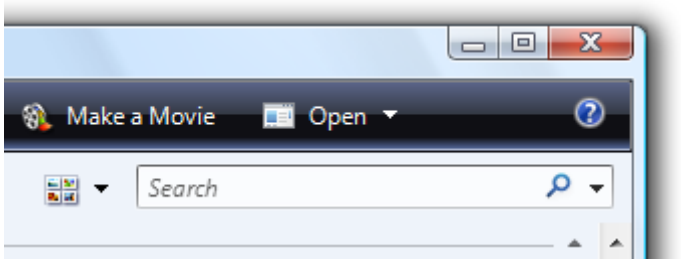
Look and feel

The feel of Search in Windows is significantly enhanced by supporting Instant search. Having instant results makes Windows feel more powerful and direct.

In Windows Explorer and application windows, Search is located in the upper-right corner if it is a supplemental entry point. In this case, users look for a search mechanism

when they don't find what they are looking for in the window. However, if Search is the primary entry point, it is centered at the top of the window.

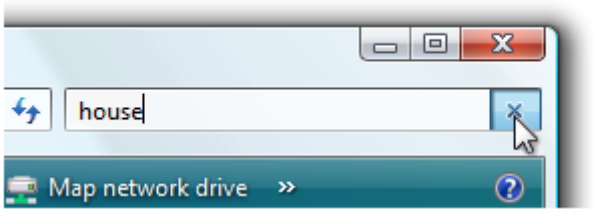
The Search button is visually connected to a Search box. To minimize space, an optional prompt is used inside a Search box instead of a label. The prompt may be an instruction (for example, Type to search) or indicate the scope of the search (for example, Search for pictures).



Without labels and separate buttons, Instant search in Windows has a lightweight look.

Performing a successful search creates a virtual page of the search results and adds it to the Back stack and Address bar. Users have several ways to restore the original page and clear a Search box, including clicking Back, clicking the original page in the Address bar, pressing Esc, or clearing the Search box.

Users can also simply clear the Search box without restoring a previous page of results. In Instant search mode, a clear button appears after the user starts typing; an "x" replaces the magnifying glass search symbol. On hover, the "x" gets a button look and can be clicked.



The user can clear the Search box by clicking "x" at the right end of the control.

In regular search mode, the clear button is optional. Users may find it useful, for example, if a search is taking a long time to complete. Users can click the "x" to stop the search in progress. If a search has already completed, users can click the "x" to clear the Search box.

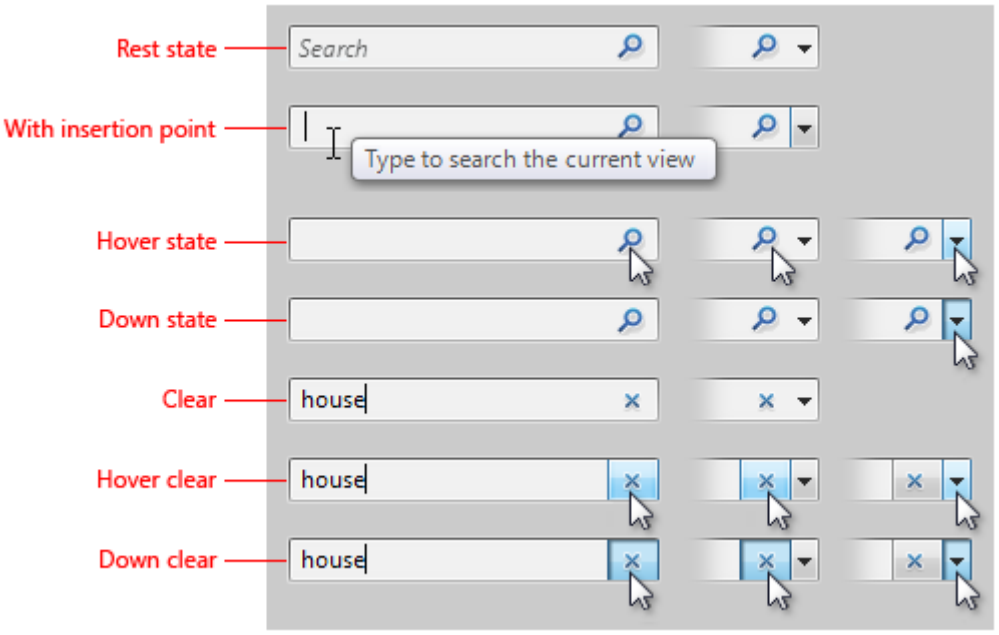
Guidelines

Location

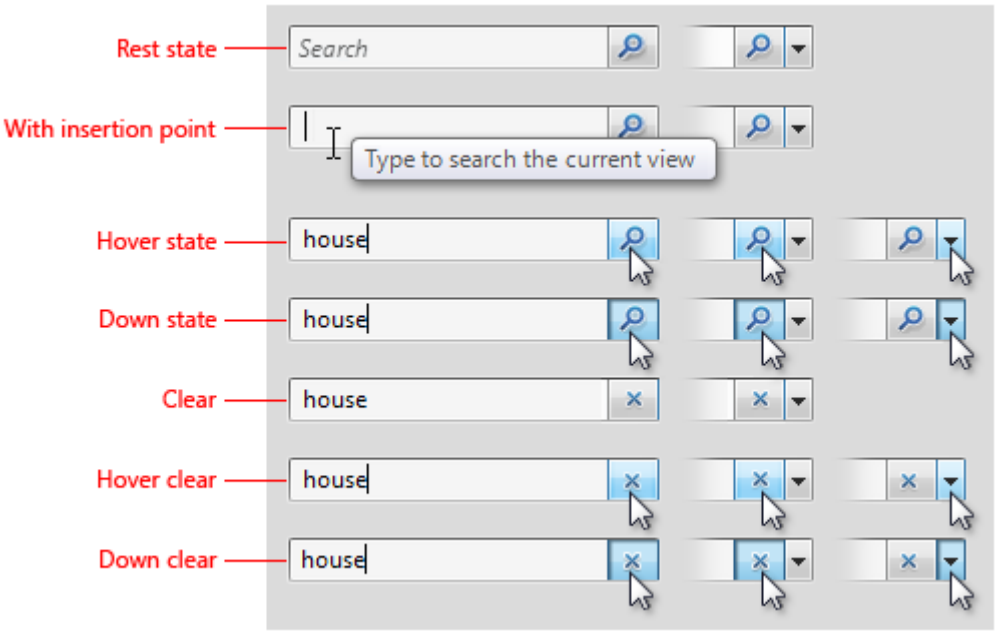
- For application windows, locate Search in the upper-right corner.
- For popup windows, locate Search wherever is most sensible and convenient.
- **Exception:** If Search is usually the first thing users do in a window (the primary entry point), center it at the top of the window.

Look

- Use the standard search button graphics. There are three versions:
  - **Magnifying glass search symbol only (no button on hover).** Use for Instant search.
  - **Magnifying glass search symbol with button.** Use when button needs to be clicked to start the search.
  - **Magnifying glass search symbol with button and drop-down arrow.** Add a drop-down arrow when users can change the scope or when other settings are available.
    - For Instant search, use a drop-down arrow only, and show a button on hover.
    - For regular search, show the drop-down arrow on a button.

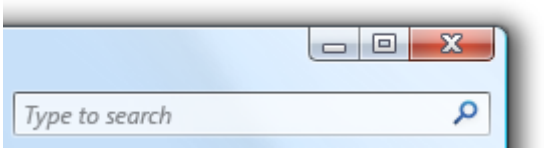


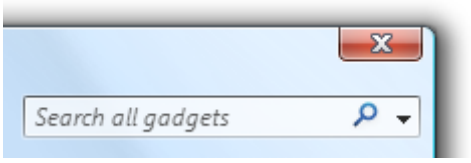
Visual specifications for Instant search.



Visual specifications for regular search.

- Don't use a label; use an optional prompt instead. If users tend to assume that the search is a generic file search, use the prompt to give the scope. Otherwise, use Type to search or a similar, concise phrase.

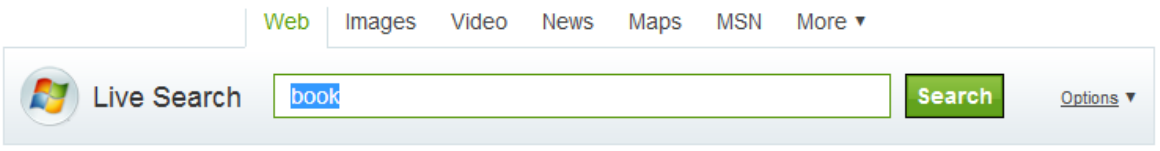




  In these examples, brief textual prompts help users work with Search.

Interaction

- **On input focus, automatically select any previously entered text.** Doing so allows users to enter a new search by typing, or to modify the previous search by positioning the caret using the arrow keys.



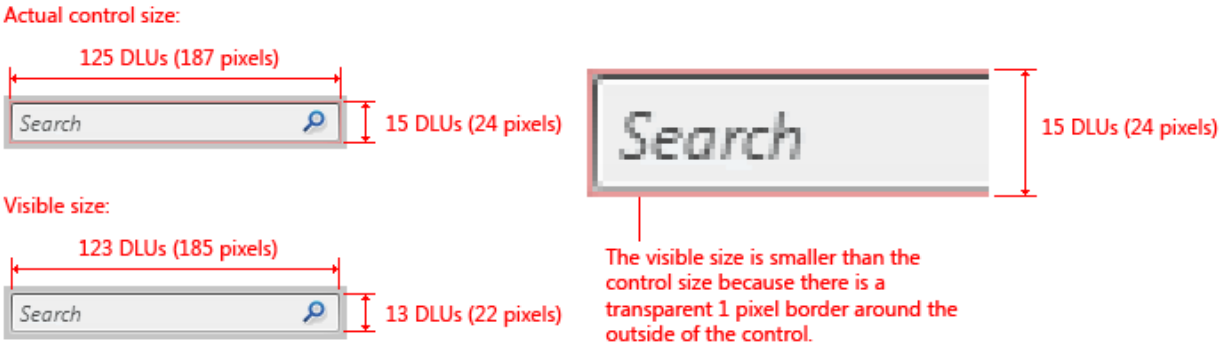
  In this example, previously entered text is selected on input focus.

- **Assign the keyboard shortcut for the Search box to be Ctrl+E.**

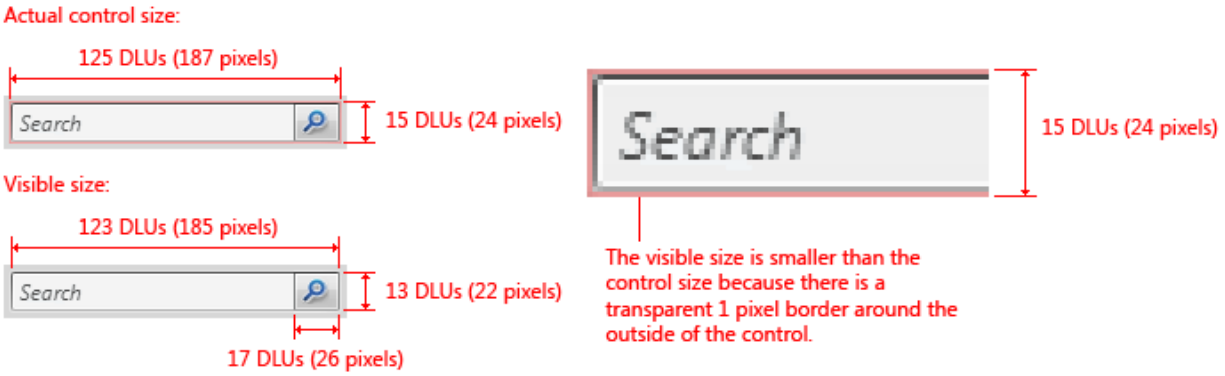
Functionality

- **Support Instant search whenever possible.** Provide both regular and Instant searches if there are scenarios where regular searching is worth the extra wait time.
- Regular searches must return relevant results within five seconds and Instant search must return results within two seconds. After this point, Search may continue to fill in less relevant results over time as long as the program is responsive and users can perform other tasks. You may have to index your search data to ensure this responsiveness.
- If you provide both regular and Instant search modes, the Instant search results must be a subset of the regular search results.
- All searching is prefix-based (no substring or suffix searching). The use of trailing wildcard characters is optional and doesn't affect the results. If multiple words are entered, use OR searching.
- A successful search adds a virtual page with the search results to the Back stack and Address bar. Multiple searches result in a single virtual page, so clicking Back always returns the original page.
- If necessary for scale, rank the search results by relevance.
- A blank search returns the original page.

Recommended sizing and spacing



Recommended sizing and spacing for Instant search.



Recommended sizing and spacing for regular search.

Text

- For the wording of the prompt in the Search box, either make it an instruction (for example, Type to search) or indicate the scope of the search (for example, Search for pictures).
- Prompt text should be brief. A single word or short phrase should suffice.
- Use sentence-style capitalization.
- Don't use ending punctuation or ellipsis.

Documentation

- Refer to this control as the Search box. Capitalize the initial letter of the first word; don't capitalize the initial letter of box.
- Refer to the two types of search as Instant search and regular search. Capitalize the initial letter of Instant search; don't capitalize the initial letter of regular search.

Related topics

Glossary

Feedback

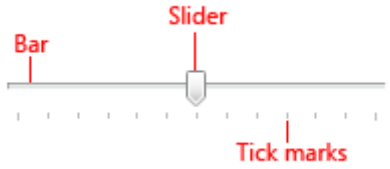
Was this page helpful? Yes No

Sliders (Design basics)

Article • 02/09/2021

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

With a slider, users can choose from a continuous range of values. A slider has a bar that shows the range and an indicator that shows the current value. Optional tick marks show values.



A typical slider.

> **Note**
>
> Guidelines related to layout are presented in a separate article.

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.

To decide, consider these questions:

- **Does the setting seem like a relative quantity?** If not, use radio buttons, or a drop-down or single-selection list.

- **Is the setting an exact, known numeric value?** If so, use a numeric text boxes.
- **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.
- **Can the user change the value?** Sliders are for user interaction. If a user can't ever change the value, use a read-only text box instead.

If a slider or a numeric text box is possible, use a numeric text box if:

- Screen space is tight.
- A user is likely to prefer using the keyboard.

Use a slider if:

- Users will benefit from instant feedback.

Guidelines

- **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.

- **Orient the slider to reflect the culture of your users.** For example, Western cultures read from left to right, so for horizontal sliders, put the low end of the range on the left and the high end on the right. For cultures that read from right to left, do the opposite.

- **Size the control so that a user can easily set the desired value.** For settings with discrete values, make sure the user can easily select any value using the mouse.

- **Consider using a non-linear scale if the range of values is large and users will likely select values at one end of the range.** For example, time value might be 1 minute, 1 hour, 1 day, or 1 month.

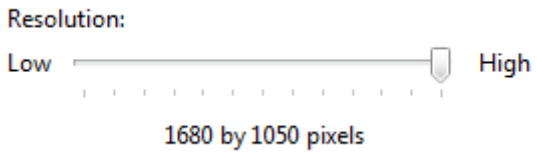
- **Whenever practical, give immediate feedback while or after a user makes a selection.** For example, the Microsoft Windows volume control beeps to indicate the resulting audio volume.

- **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 since the meaning is clear.

In this example, the slider's vertical orientation makes the range labels unnecessary.

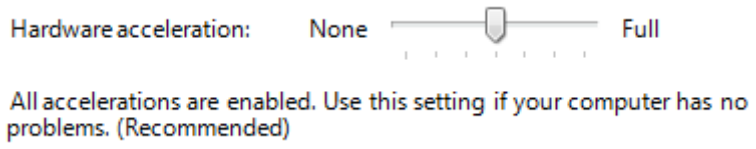
- **Use tick marks when users need to know the approximate value of the setting.**
- **Use tick marks and a value label when users need to know the exact value of the setting they choose.** Always use a value label if a user needs to know the units to make sense of the setting.



In this example, a label is used to clearly indicate the selected value.

- **For horizontally-oriented sliders, place tick marks under the slider.** For vertically-oriented sliders, place tick marks to the right for Western cultures; for cultures that read from right to left, do the opposite.
- **Place the value label completely under the slider control so that the relationship is clear.**

  Incorrect:



In this example, the value label isn't aligned under the slider.

- **When disabling a slider, also disable any associated labels.**
- **Don't use both a slider and a numeric text box for the same setting.** Use only the more appropriate control.

  Exception: Use both controls when the user needs both immediate feedback and the ability to set an exact numeric value.

- **Don't use a slider as a progress indicator.**
- **Don't change the size of the slider indicator from the default size.**

  Incorrect:

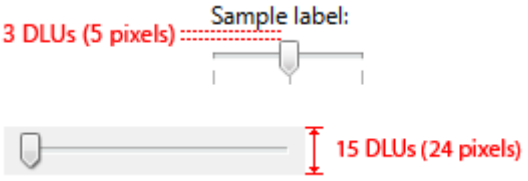
  In this example, a size smaller than the default is used.

  Correct:

  In this example, the default size is used.

- **Don't label every tick mark.**

Recommended sizing and spacing



Recommended sizing and spacing for sliders.

Labels

Slider labels

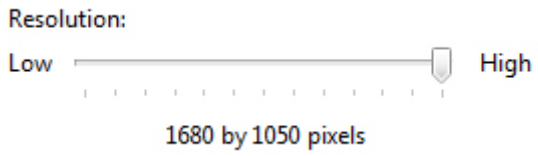
- Use a static text label ending with a colon, or a group box label with no ending punctuation.
- Assign a unique access key to each label. For assignment guidelines, see Keyboard.
- Use sentence-style capitalization.
- Position the slider label either to the left of the slider, or above and aligned with the left edge of the slider (or its left range identifier, if present).

Range labels

- Label the two ends of the slider range, unless a vertical orientation makes this unnecessary.
- Use only 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.
- Use sentence-style capitalization.
- Don't assign access keys.

Value labels

- 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).



In this example, the value label is centered under the slider and includes the units.

Documentation

When referring to sliders:

- Use the exact label text, including its capitalization, and include the word slider. Don't include the access key underscore or colon.
- To describe user interaction, use move.
- When possible, format the label using bold text. Otherwise, put the label in quotation marks only if required to prevent confusion.

Example: To increase your screen resolution, move the **Screen resolution** slider to the right.

Related topics

Glossary

Feedback

Was this page helpful? Yes No

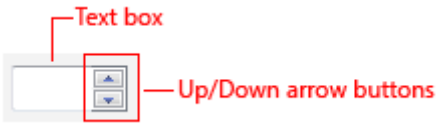
Provide product feedback | Get help at Microsoft Q&A

Spin Controls

Article • 02/09/2021

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

With a spin control, users can click arrow buttons to change incrementally the value within its associated numeric text box. The term spin box refers to the combination of a text box and its associated spin control.



A typical spin box.

Users often prefer spin controls because they can make changes without moving their hands from the mouse. When the spin control is paired with a text box, users can type or paste input directly in the text box, so use of the spin control is optional.

While spin controls are used for numeric input, the input doesn't have to be a pure whole number. The input can be decimal numbers and it can have negative signs, delimiters (such as colons or hyphens), and unit modifiers.

> **Note**
>
> Guidelines related to text boxes and layout are presented in separate articles.

Is this the right control?

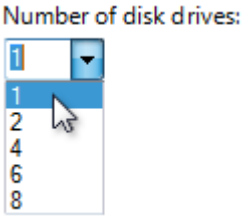
To decide, consider these questions:

- **Is the control used for numeric input?** If not, use another control, such as a drop-down list or slider, to select from a fixed set of values. Use scroll bars for scrolling.

- **Do users think of the value as a relative quantity, not a numeric value?** If so, use a slider instead. Use spin boxes only for exact, known numeric values. For example, users think about setting their audio volume to low or medium—not about setting the value to 2 or 5.
- **Is the control paired with a text box?** If not, don't use. Spin controls shouldn't be used alone or with other types of controls besides a text box.

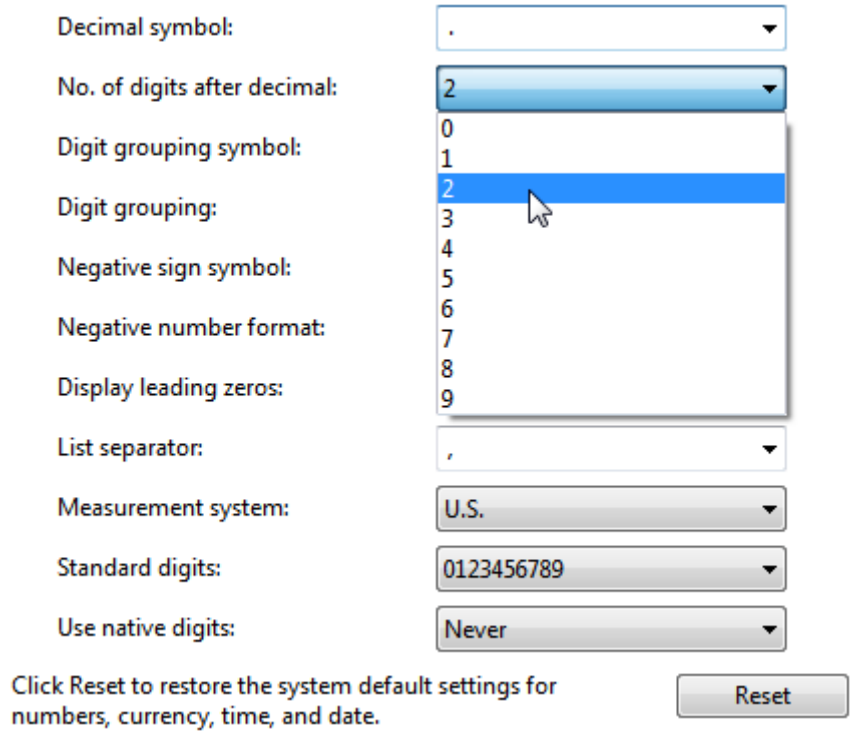
  Incorrect:

  In this example, a spin control is used to control a dynamic graphic.
- **Are contiguous value ranges valid?** If not, use a drop-down list of valid values instead.

  In this example, not all disk drive numbers are valid, so a drop-down list is a better choice.
- **Is using the spin control practical?** Using a spin control is practical for:
  - Entering a small number, typically under 100.
  - Making small changes to an existing or default value.

  While spin controls can be used for any numeric input, they are inefficient in situations other than these.
- **Is the spin control helpful?** Is the control used in a context where users are likely to be using their mouse? If not, consider a spin control optional.
- **Are the sibling controls drop-down lists?** If there are other drop-down lists, consider using a drop-down list for consistency.



In this example, a spin box could be used, but a drop-down list is used for consistency.

- **Are touch or pen users a primary target?** If so, consider using a drop-down list instead. The arrow buttons in a spin control are too small to be used efficiently with touch or a pen.

If a slider or a spin box is possible, use a spin box if:

- Screen space is tight.
- A user is likely to prefer using the keyboard.

Use a slider if:

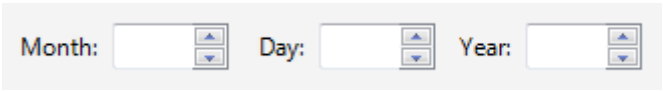
- Users will benefit from instant feedback.

Guidelines

General

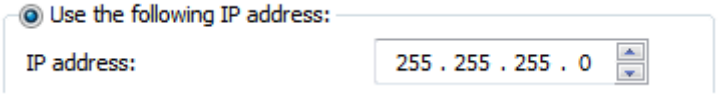
- **Use spin controls whenever they are practical and helpful.** See Is this the right control?
  - **Exception:** To be consistent with other text boxes on the same user interface (UI), use spin controls even if they aren't always practical.

  Correct:



In this example, a spin control is used with the year control for consistency, even though it isn't always practical.

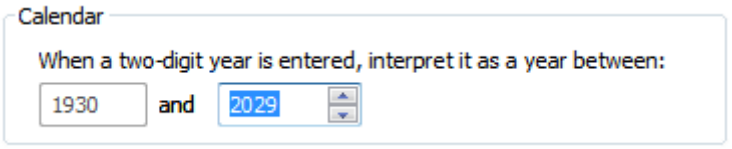
Incorrect:



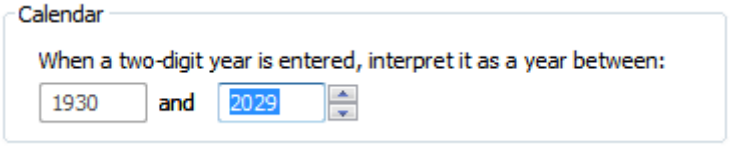
In this example, the spin control is unusable.

- **Always make a spin control the "buddy" of the text box.** Doing so places the spin control inside the text box.

  Correct:



  Incorrect:

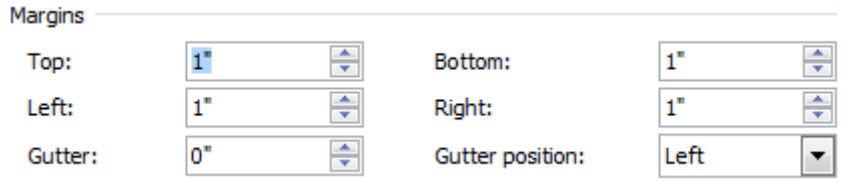


  In the correct example, the spin control is placed inside its associated text box.

- **Disable a spin control when its associated text box is disabled.** The spin control is a supplemental input method—never the only input method.

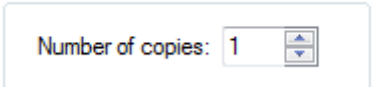
Values

- **Define the top button to increase the value by one unit and the bottom button to decrease by one unit.** Typically, the unit is one, but it should be the smallest common change in value. Ideally, the spin control should cover all valid values, and it should be more convenient than typing in the text.



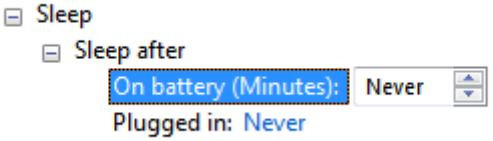
In this example, clicking a spin control changes the values by .1, which is the smallest common change in value. Using a smaller unit would cover the range of valid values but make the spin controls unusable.

- **Use the spin control to limit input to valid values.** Using a spin control should never result in an incorrect value.
- **At the end of a range of valid values, restart the range.** The spin control metaphor is that the user is spinning a wheel of values, hence this wheel-like behavior.
    - **Exception:** Don't restart the range if the resulting value is certain to be incorrect.



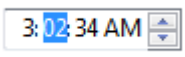
In this example, clicking the down arrow button doesn't restart the range (by going to the maximum value) because that value is certain to be incorrect.

- **Use text instead of special numeric values.** Allow users to spin to these special values instead of having to know them and type them in.



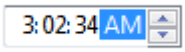
In this example, Never is a special value but users can spin to it.

- **If the value has delimiters, the associated text box should have multiple input focus points.** Doing so allows the numeric segments to be manipulated individually.



In this example, the spin control affects the values for hours, minutes, seconds, and A.M./P.M.—whichever has the focus.

- **If the value has units, use the spin control to change those units as well.**



In this example, the spin control can be used to change units.

Labels

- Apply the text box labeling guidelines to label the associated text box. Spin controls are never labeled directly.

Documentation

When referring to spin controls:

- Don't refer to spin controls in user documentation. Instead, refer to the label of the associated text box.
- Refer to spin controls and spin boxes only in programming and other technical documentation.

Example: In the **Date** box, type or select the part of the date you want to change.

Related topics

Glossary

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

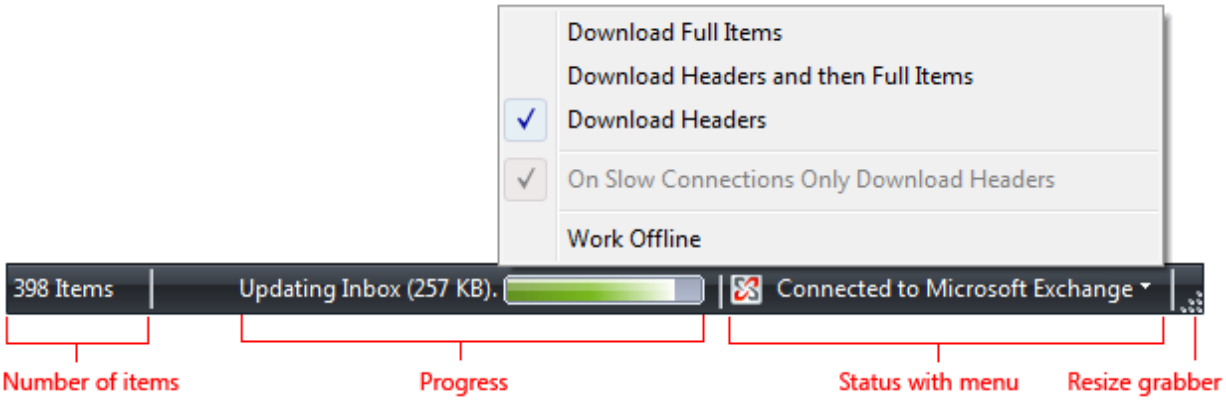
Status Bars (Design basics)

Article • 06/04/2021

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

A status bar is an area at the bottom of a primary window that displays information about the current window's state (such as what is being viewed and how), background tasks (such as printing, scanning, and formatting), or other contextual information (such as selection and keyboard state).

Status bars typically indicate status through text and icons, but they can also have progress indicators, as well as menus for commands and options related to status.



A typical status bar.

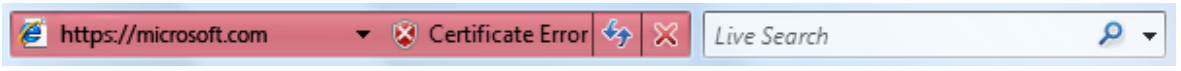
> **Note**
>
> Guidelines related to the notification area are presented in a separate article.

Is this the right user interface?

To decide, consider these questions:

- **Is the status relevant when users are actively using other programs?** If so, use a notification area icons.

- **Does the status item need to display notifications?** If so, you must use a notification area icon.
- **Is the window a primary window?** If not, don't use a status bar. Dialog boxes, wizards, control panels, and property sheets shouldn't have status bars.
- **Is the information primarily status?** If not, don't use a status bar. Status bars must not be used as a secondary menu bar or toolbar.
- **Does the information explain how to use the selected control?** If so, display the information next to the associated control using a supplemental explanation or instruction label instead.
- **Is the status useful and relevant? That is, are users likely to change their behavior as a result of this information?** If not, either don't display the status, or put it in a log file.
- **Is the status critical? Is immediate action required?** If so, display the information in a form that demands attention and cannot be easily ignored, such as a dialog box or within the primary window itself.



A red address bar in Windows Internet Explorer.

- **Is the program intended primarily for novice users?** Inexperienced users are generally unaware of status bars, so reconsider the use of status bars in this case.

Design concepts

Status bars are a great way to provide status information without interrupting users or breaking their flow. However, status bars are easy to overlook. So easy, in fact, that many users don't notice status bars at all.

The solution to this problem isn't to demand the user's attention by using garish icons, animation, or flashing, but to design for this limitation. You can do this by:

- **Making sure that the status information is useful and relevant.** If not, don't provide a status bar at all.
- **Not using status bars for crucial information.** Users should never have to know what is in the status bar. If users must see it, don't put it in a status bar.

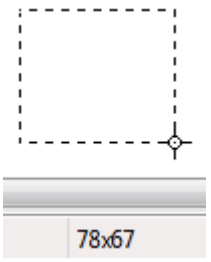
If you do only one thing...

Make sure that the status bar information is useful and relevant but not crucial.

Usage patterns

Status bars have several usage patterns:

Expand table

Usage	Example
Current window status Show the source of what is being displayed along with any view modes	 In this example, the status bar displays the path to the document.
Progress Show the progress of background tasks, either with a determinate progress bar or an animation.	 In this example, the status bar includes a progress bar to show the Web page loading into a Internet Explorer window.
Contextual information Show contextual information about what the user is currently doing.	 In this example, Microsoft Paint shows the selection size in pixels.

Guidelines

General

- Consider providing a View Status Bar command if only some users will need the status bar information. Hide the status bar by default if most users won't need it.
- Don't use the status bar to explain menu bar items. This help pattern isn't discoverable.

Presentation

- Disable modal status that doesn't apply. Modal status includes keyboard and document states.
- Remove non-modal status that doesn't apply.

- Present status information in the following order: current window status; progress; and contextual information.

Icons

- Choose easily recognizable status icon designs. Prefer icons with unique outlines over square or rectangular shaped icons.
- Use swaths of pure red, yellow, and green only to communicate status information. Otherwise, such icons are confusing.

  Correct:

  Incorrect:

  In the incorrect example, the red icon unintentionally suggests an error, creating confusion.

- Use icon variations or overlays to indicate status or status changes. Use icon variations to show changes in quantities or strengths. For other types of status, use these standard overlays:

  Expand table

Overlay	Status
	Warning
	Error
	Disabled/Disconnected
	Blocked/Offline

- Don't change status too frequently. Status bar icons shouldn't appear noisy, unstable, or demand attention. The eye is sensitive to changes in the peripheral field of vision, so status changes need to be subtle.
- For icons that provide important status information, prefer in-place labels.
- Unlabeled status bar icons should have tooltips.

For more information, see Icons.

Interaction

- Make a status bar area interactive to allow users direct access to related commands and options.
  - Use a control that looks and behaves like a menu button or a split button. These status bar areas must have a drop-down arrow to indicate that they are clickable.
  - Display the menu on left-click on mouse down, not mouse up.
  - Don't support right-clicking or double-clicking. Users don't expect such interactions in a status bar, so they aren't likely to attempt them.
- Display tooltips on hover.

Text

- Generally, use concise labels. Cut any text that can be eliminated.
- Prefer sentence fragments, without ending punctuation. Use full sentences (with ending punctuation) only when sentence fragments aren't significantly shorter.
- For optional progress labels, indicate what the operation is doing with a label that starts with a verb (gerund form) and ends with an ellipsis. For example: "Copying...". This label may change dynamically if the operation has multiple steps or is processing multiple objects.
- Don't use color, bold, or italic to emphasize status bar text.
- For tooltip phrasing guidelines, see Tooltips and Infotips.

Documentation

Refer to status bars as status bars, not status lines or other variations. Example: "The current page number is displayed on the status bar."

Feedback

Was this page helpful? Yes No

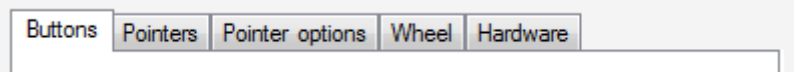
Provide product feedback | Get help at Microsoft Q&A

Tabs

Article • 06/04/2021

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

Tabs provide a way to present related information on separate labeled pages.



A typical set of tabs.

Tabs are usually associated with property windows (and vice versa), but tabs can be used in any type of window.

Tab controls represent the tabbed manila folders used to organize information in filing cabinets commonly found in the United States. (Manila folders aren't used worldwide.)

> **Note**
>
> Guidelines related to layout, tab menus, dialog boxes, and property windows are presented in separate articles.

Is this the right control?

To decide, consider these questions:

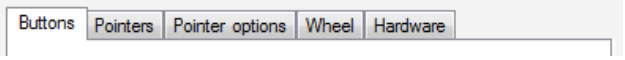
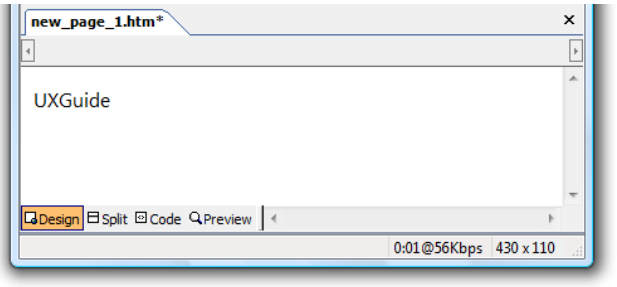
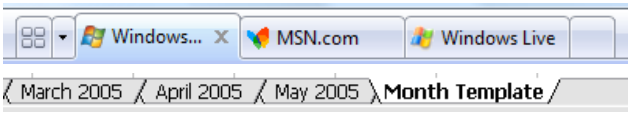
- **Can the controls comfortably fit on a single, reasonably sized page?** If so, use a single page.
- **Is there only one tab?** If so, use a single page.
- **Are the tabs related to each other in some obvious way?** If not, consider splitting the information into separate windows of related information.
- **If used for settings, are settings on different pages completely independent?** Will changing a setting on one page affect settings on other pages? If they're not independent, use task pages or a wizard instead.

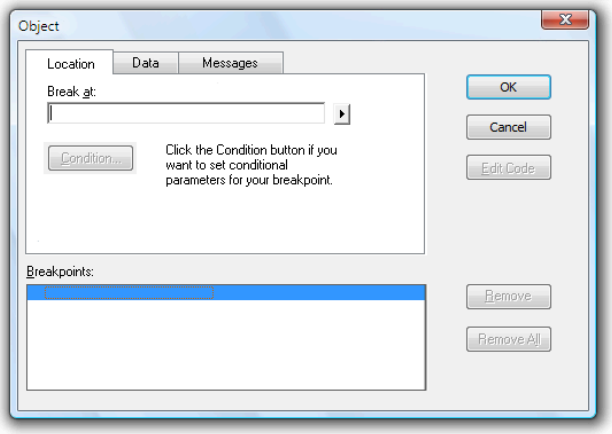
- **Are the tabs mostly peers of each other, or is there a hierarchical relationship?** If hierarchical, consider using progressive disclosure or child dialog boxes to show related information.
- **Are the tabs used to display steps within a task?** You can use "tabs" to display steps within a task only if they are presented to look like steps, and there is an obvious, alternative way to get to the text step, such as a Next button. Otherwise, if the steps are required, use pages in a page flow or a wizard. If the steps are optional, display the optional steps using modal dialog boxes instead.
- **Are the tabs different views of the same data?** If so, consider using a split button or drop-down list to change views. While tabs can be used effectively for changing views, the alternatives are more lightweight.

Usage patterns

Tabs have several usage patterns:

Expand table

Usage	Example
Dynamic window surface like scroll bars, tabs can be used to increase the window surface area to show related information.	With this pattern, using tabs is conceptually similar to placing all the information on the tabs linearly on a single scrollable surface, with the tab labels as headings.  In this example, tabs effectively increase the window surface area.
Multiple views like split buttons or drop-down lists, tabs can be used to show different views of the same or related information.	 In this example, tabs change views within a document.
Multiple documents like multiple windows, tabs can be used to show different documents in a single window.	 In these examples, tabs show different documents within a single application window.

Usage	Example
Exclusive options like radio buttons, tabs can be used to present multiple exclusive choices. in this pattern, only the selected tab applies and all other tabs are ignored.	 In this example, tabs are used (incorrectly) as a substitute for radio buttons. **This pattern is not recommended** because it uses a nonstandard behavior. The tabs behave as a setting instead of purely a way to navigate within the window.

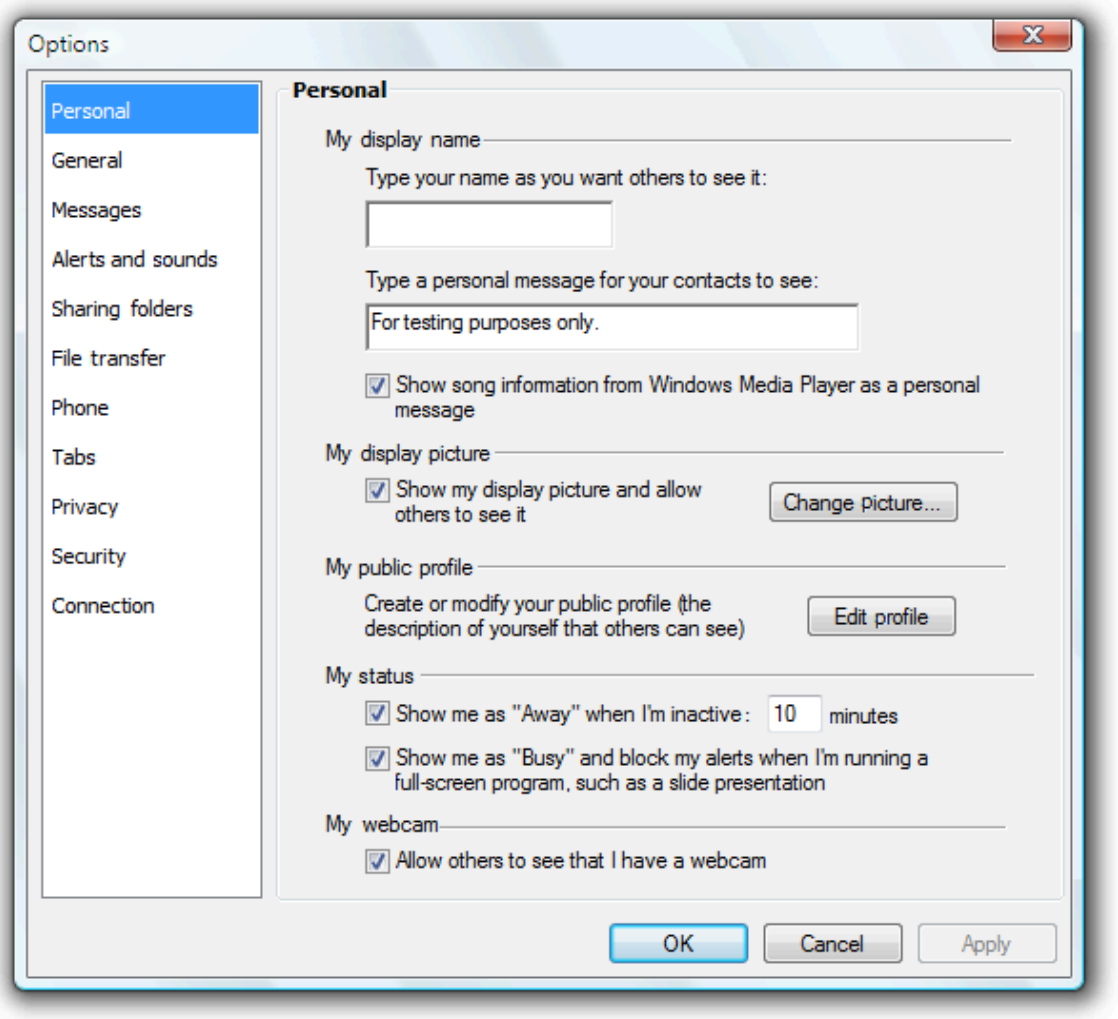
If you do only one thing...

Make sure the information on the tabs is related, yet settings on different pages are independent. The last tab selected should have no special meaning.

Guidelines

General

- **Use horizontal tabs if:**
  - The window has seven or fewer tabs.
  - **All the tabs fit on one row, even when the user interface (UI) is localized.**
- **Use vertical tabs if:**
  - The property window has eight or more tabs.
  - **Using horizontal tabs would require more than one row.**

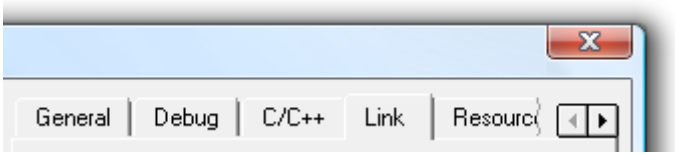


In this example, vertical tabs accommodate eight or more tabs.

- **Don't nest tabs or combine horizontal tabs with vertical tabs.** Instead, reduce the number of tabs, use only vertical tabs, or use another control such as a drop-down list.

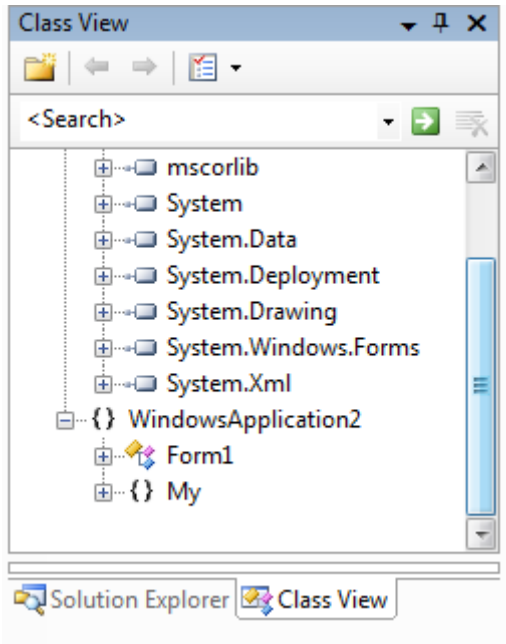
- **Don't scroll horizontal tabs.** Horizontal scrolling isn't readily discoverable. You may scroll vertical tabs, however.

  Incorrect:



  In this example, the horizontal tabs are scrolled.

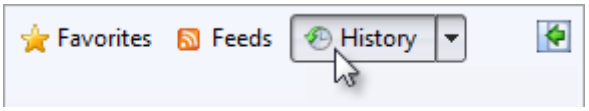
- For tabs on a resizable window or pane, put a scrollbar, when needed, on the page, not the window or pane. The tabs should always be visible and not scroll out of view.



In this example, the tab page has the scrollbar, not the pane.

- **Make sure the tabs look like tabs and not another type of control.**

  Incorrect:



  In this example, these tabs look like command buttons.

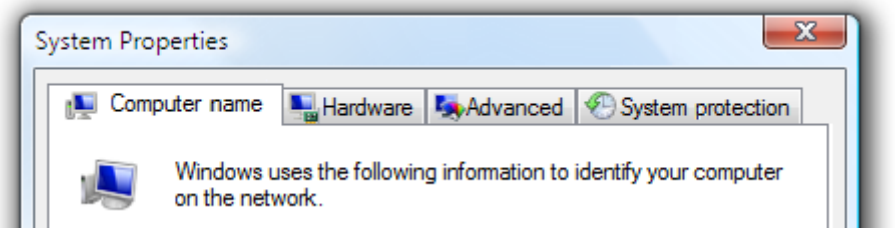
Interaction

- **When controls apply only to a page, place them within the border of the tabbed page.**
- **When controls apply to the entire window, place them outside the tabbed page.**
- **Don't assign effects to changing tabs.** Tabs must be accessible in any order. Changing the current tab should never have side effects, apply settings, or result in an error message.
- **Don't assign a special meaning to the last tab selected.** Tab selection is for navigation the user's last tab selection isn't a setting.
- **Don't make the settings on a page dependent on settings on other pages.** Put any dependent settings on the same page instead.
- **If users are likely to start with the last tab displayed, make the tab persist and select it by default.** Make the settings persist on a per-window, per-user basis. Otherwise, select the first page by default.

Icons

- **Don't put icons on tabs.** Icons usually add unnecessary visual clutter, consume screen space, and often don't improve user comprehension. Only add icons that aid in comprehension, such as standard symbols.

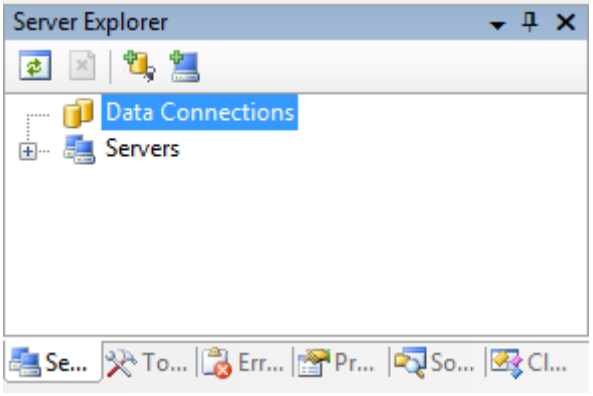
  Incorrect:

  In this example, the icons add visual clutter and do little to improve user comprehension.

  Exception: You can use clearly recognizable icons if there might be insufficient space to display meaningful labels:

  Correct:

  In this example, the window is so narrow that icons better communicate the tabs than the labels.

- **Don't use product logos for tab graphics.** Tabs aren't for branding.

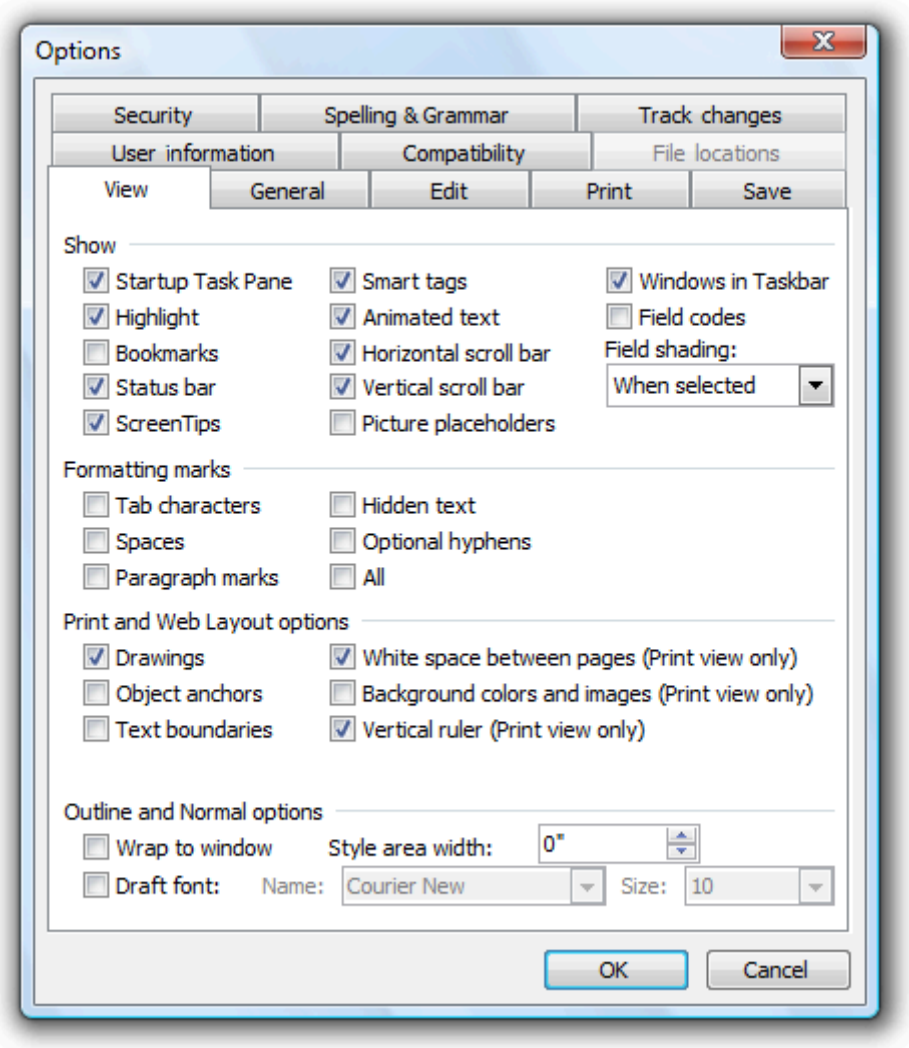
Dynamic window surface pattern

- **Don't use scroll bars on tab pages.** Tabs function similarly to scroll bars to increase the effective area of a window. One mechanism should be sufficient.

- **Use concise tab labels.** Use one or two words that clearly describe the content of the page. Longer labels consume screen space, especially when the labels are localized.

- **Use specific, meaningful tab labels.** Avoid generic tab labels that could apply to any tab, such as General, Advanced, or Settings.
- **If a tab doesn't apply to the current context and users don't expect it to, remove it.** Doing so simplifies the UI and users won't miss it.

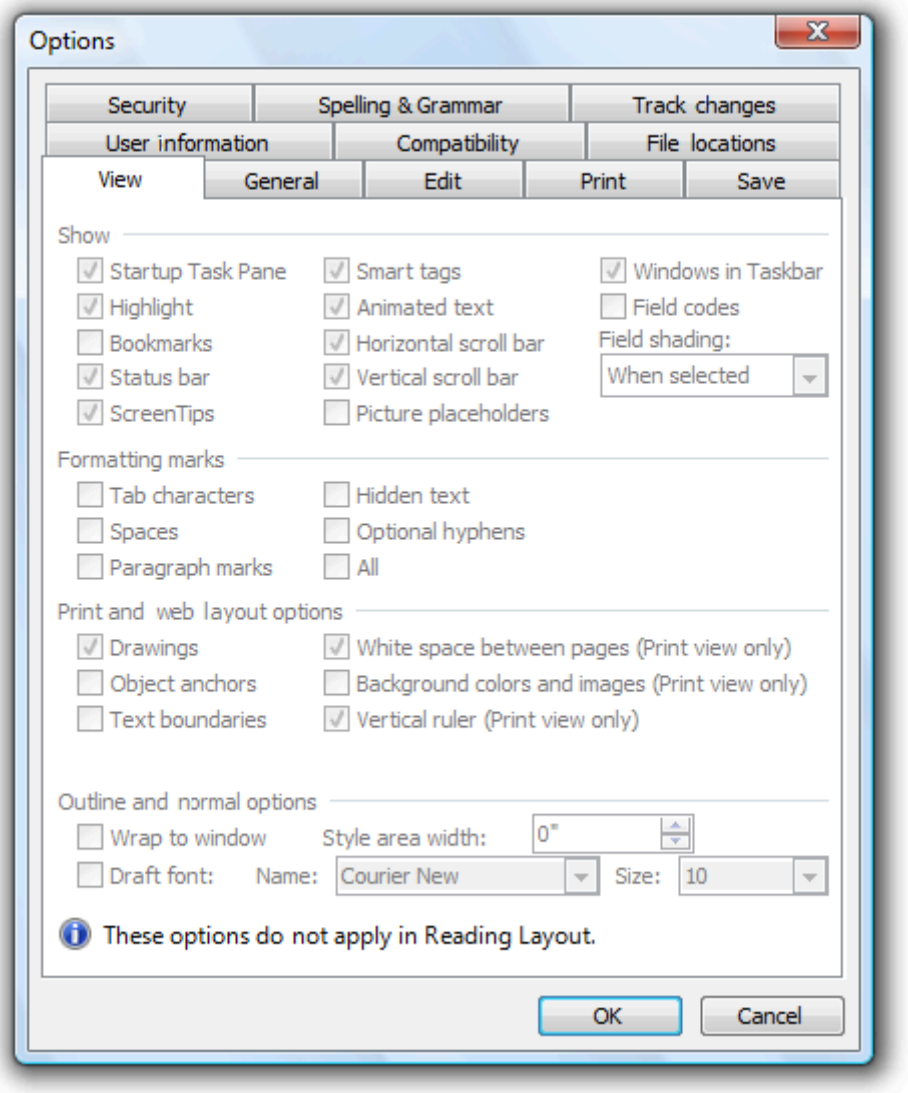
  Incorrect:

  In this example, the File Locations tab is incorrectly disabled when Microsoft Word is used as an e-mail editor. Rather than disabling this tab, it should be removed because users wouldn't expect to view or change file locations in this context.

- **If a tab doesn't apply to the current context and users might expect it to:**
  - Display the tab.
  - Disable the controls on the page.
  - Include text explaining why the controls are disabled.

  Don't disable the tab, because doing so isn't self-explanatory and prohibits exploration. Users looking for a specific value would be forced to look on all other tabs.



In this example, none of the View options apply in Reading Layout. However, users might expect them to apply based on the tab label, so the page is displayed but the options are disabled.

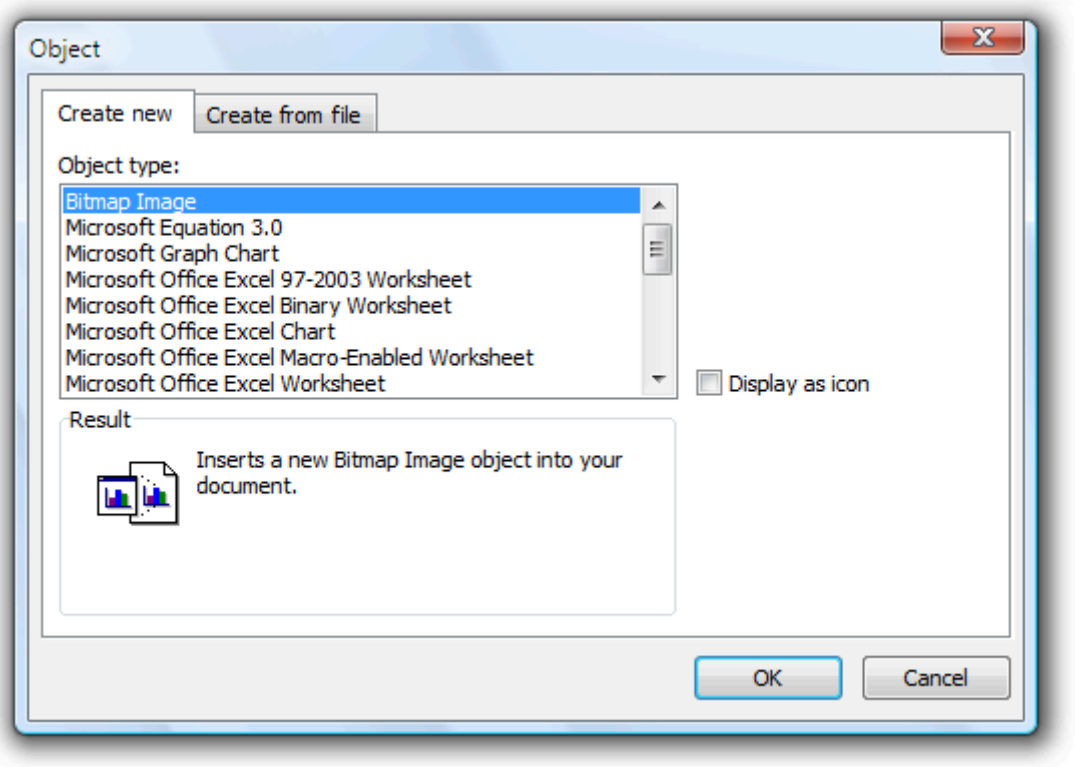
Multiple views and documents patterns

- **Use the view or document names on tab labels.**
- **Avoid excessively long tab names.** If necessary, either have a maximum name size or truncate the displayed tab label using ellipses. Longer labels consume screen space, especially when the labels are localized.
- **If a tab doesn't apply to the current context, remove the tab.**

Exclusive options pattern

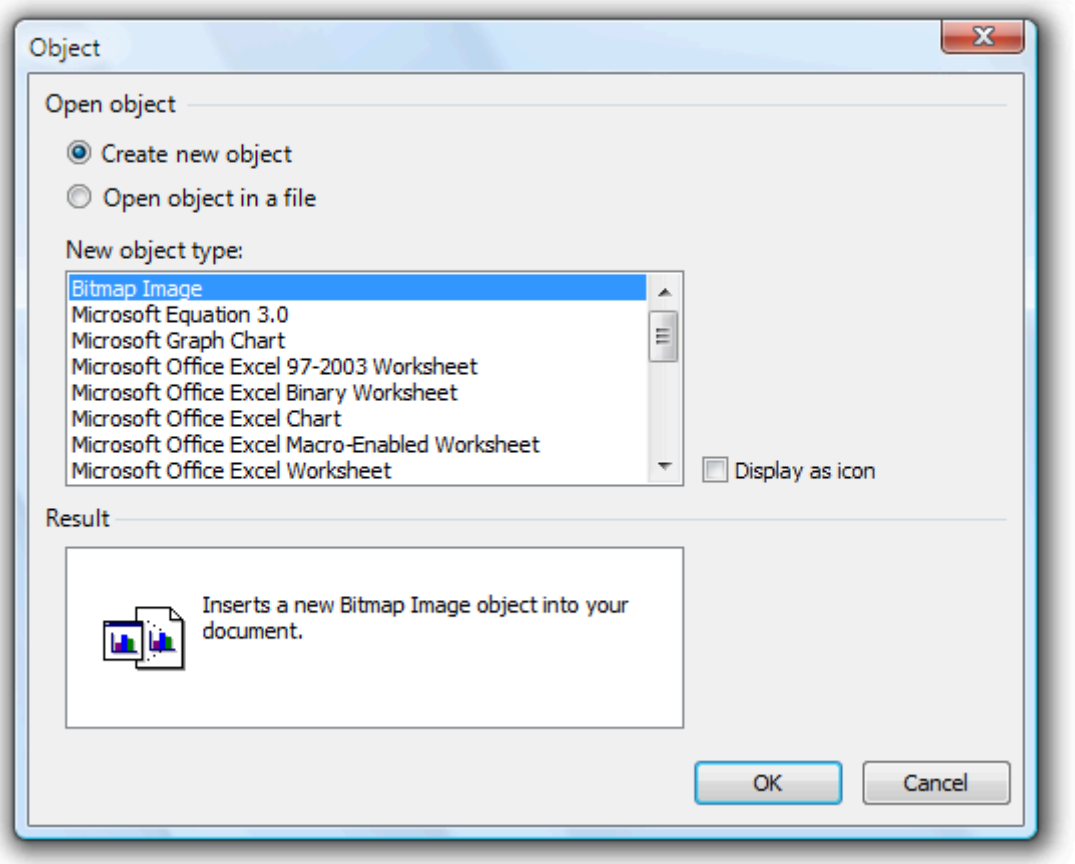
- **Don't use this pattern!** Use radio buttons or a drop-down list instead.

  Incorrect:



In this example, tabs are incorrectly used as a substitute for radio buttons.

Correct:



In this example, radio buttons are correctly used instead.

Labels

- Label tabs based on their pattern. Use nouns rather than verbs, without ending punctuation. See the preceding pattern guidelines for more information.
- Use sentence-style capitalization.
- Don't assign an access key. Tabs are accessible through their shortcut keys (Ctrl+Tab, Ctrl+Shift+Tab, Ctrl+PgUp, Ctrl+PgDn). There is a shortage of good access key choices, so not assigning access keys to tabs makes it easier to assign them to other controls.

Documentation

When referring to tabs:

- Use the exact label text, including its capitalization, and include the word tab.
- To describe user interaction, use click.
- When possible, format the label using bold text. Otherwise, put the label in quotation marks only if required to prevent confusion.
- Because multiple uses can be ambiguous, especially for a worldwide audience, use the noun tab alone to refer only to a tab control. For other uses, clarify the meaning with a descriptor: the Tab key, a tab stop, or a tab mark on the ruler.

Example: On the **Tools** menu, click **Options**, and then click the **View** tab.

Feedback

Was this page helpful? Yes No

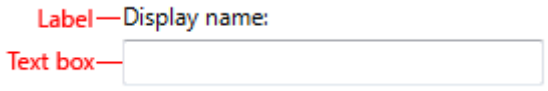
Provide product feedback | Get help at Microsoft Q&A

Text Boxes

Article • 08/26/2021

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

With a text box, users can display, enter, or edit a text or numeric value.



A typical text box.

> **Note**
>
> Guidelines related to layout, fonts, and balloons are presented in separate articles.

Is this the right control?

To decide, consider these questions:

- **Is it practical to enumerate all the valid values efficiently?** If so, consider a single-selection list, list view, drop-down list, editable drop-down list, or slider instead.
- **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 box.
- **Does the value represent a data type that has a specialized common control?** Examples include date, time, or IPv4 or IPv6 address. If so, use the appropriate control, such as a date control rather than a text box.
- If the data is numeric:
  - **Do users perceive the setting as a relative quantity?** 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 a text box. For example, users can easily choose a color using a slider because they can immediately see the effect of changes to hue, saturation, or luminosity values.

Design concepts

While text boxes have the benefit of being very flexible, they have the drawback of having minimal constraints. The only constraints on an editable text box are:

- You can optionally set the maximum number of characters.
- You can optionally restrict input to numeric characters (0 9) only.
- If you use a spin control, you can limit spin control choices to valid values.

Aside from their length and the optional presence of a spin control, text boxes don't have any visual clues that suggest the valid values or their format. This means relying on labels to convey this information to users. If users enter text that's not valid, you must handle the error with an error message.

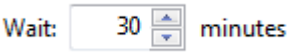
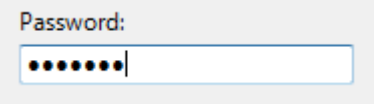
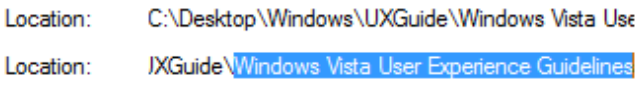
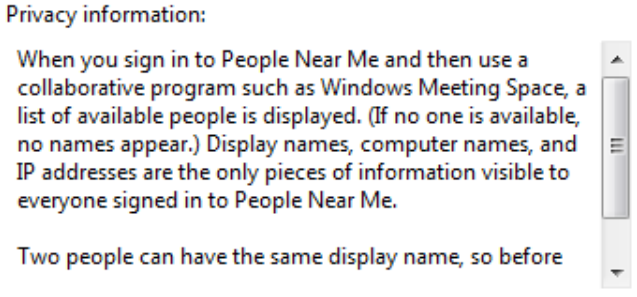
As a general rule, **you should use the most constrained control that you can**. Use unconstrained controls like text boxes as a last resort. That said, when you are considering constraints, bear in mind the needs of global users. For example, a control that is constrained to United States ZIP Codes isn't globalized, but an unconstrained text box that accepts any postal code format is.

Usage patterns

A text box is a flexible control with several possible uses.

Expand table

Label	Value
Data input A single-line, unconstrained text box used to enter or edit short strings.	 A single-line, unconstrained text box.
Formatted data input A set of short, fixed-sized, single-line text boxes used to enter data with a specific format.	 A text box used for formatted data input. **Note:** The auto-exit feature automatically advances the input focus from one text box to the next. One disadvantage to this approach is that the data can't be copied or pasted as a single unit.
Assisted data input A single-line, unconstrained text box used to enter or edit strings, combined	 In this example, the Browse command helps users

Label	Value
with a command button that helps users select valid values.	select valid values.
Textual input A multi-line, unconstrained text box used to enter or edit long strings.	 A multi-line, unconstrained text box.
Numeric input A single-line, numeric-only text box used to enter or edit numbers, with an optional spin control to facilitate mouse-based input.	 A text box used for numeric input. The combination of a text box and its associated spin control is called a spin box.
Password and PIN input A single-line, unconstrained text box used to enter passwords and PINs securely.	 A text box used to enter passwords.
Data output A single-line, read-only text box, always displayed without a border, used to display short strings.	Unlike static text, data displayed using a text box can be scrolled (useful if the data is wider than the control), selected, and copied.  A single-line, read-only text box used to display data.
Textual output A multi-line, read-only text box used to display long strings.	 A read-only text box used to display data.

Guidelines

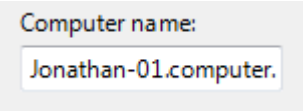
General

- **When disabling a text box, also disable any associated labels, instruction labels, spin controls, and command buttons.**

- **Use auto-complete to help users enter data that is likely to be used repeatedly**. Examples include user names, addresses, and file names. However, don't use auto-complete for text boxes that may contain sensitive information, such as passwords, PINs, credit card numbers, or medical information.

- **Don't make users scroll unnecessarily.** If you expect data to be larger than the text box and you can readily make the text box larger without harming the layout, size the box to eliminate the need for scrolling.

  Incorrect:

  In this example, the text box should be made much longer to handle its data.

- Scroll bars:
  - **Don't put horizontal scroll bars on multi-line text boxes.** Use vertical scrolling and line wrapping instead.
  - **Don't put any scroll bars on single-line text boxes.**

- **For numeric input, you may use a spin control.** For textual input, use a drop-down list or editable drop-down list instead.

- **Don't use the auto-exit feature except for formatted data input.** The automatic shift of focus can surprise users.

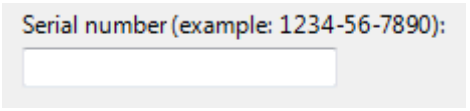
Editable text boxes

- **Limit the length of the input text when you can.** For example, if the valid input is a number between 0 and 999, use a numeric text box that is limited to three characters. All parts of text boxes that use formatted data input must have a short, fixed length.

- **Be flexible with data formats.** If users are likely to enter text using a wide variety of formats, try to handle all the most common ones. For example, many names, numbers, and identifiers can be entered with optional spaces and punctuation, and the capitalization often doesn't matter.

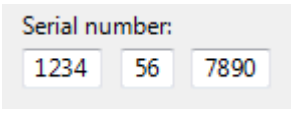
- If you can't handle the likely formats, require a specific format by using formatted data input or indicate the valid formats in the label.

  Acceptable:



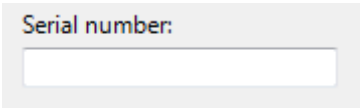
In this example, a text box requires input in a specific format.

Better:



In this example, the formatted data input pattern is used to require a specific format.

Best:

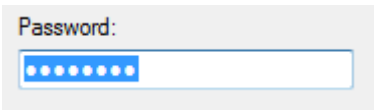


In this example, a text box handles all likely formats.

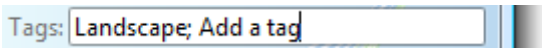
- Consider format flexibility when choosing the maximum input length. For example, a valid credit card number can use up to 19 characters so limiting the length to anything shorter would make it difficult to enter numbers using the longer formats.
- **Don't use the formatted data input pattern if users are more likely to paste in long, complex data.** Rather, reserve the formatted data input pattern for situations where users are more likely to type the data.

In this example, the formatted data input pattern isn't used, so that users can to paste IPv6 addresses.

- **If users are more likely going to reenter the entire value, select all the text on input focus.** If users are more likely to edit, place the caret at the end of the text.



In this example, users are more likely to replace than edit, so the entire value is selected on input focus.



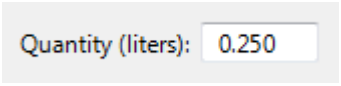
In this example, users are more likely to add keywords than replace the text, so the caret is placed at the end of the text.

- **Always use a multi-line text box if new-line characters are valid input.**
- **When the text box is for a file or path, always provide a Browse button.**

Numeric text boxes

- **Choose the most convenient unit and label the units.** For example, consider using milliliters instead of liters (or vice versa), percentages instead of direct values (or vice versa), and so on.

  Correct:

  In this example, the unit is labeled, but it requires users to enter decimal numbers.

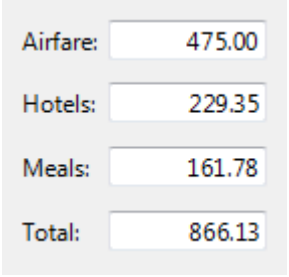
  Better:

  In this example, the text box uses a more convenient unit.

- **Use a spin control whenever it is helpful.** However, sometimes spin controls aren't practical, such as when users need to enter many large numbers. Use spin controls when:
  - The input is likely to be a small number, typically under 100.
  - Users are likely to make a small change to an existing number.
  - Users are more likely to be using the mouse than the keyboard.

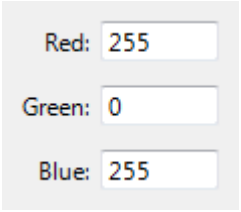
- **Right-align numeric text whenever:**
  - There is more than one numeric text box.
  - The text boxes are vertically aligned.
  - Users are likely to add or compare the values.

  Correct:



In this example, the numeric text is right-aligned to make it easy to compare values.

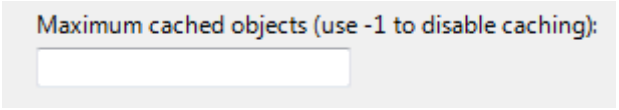
Incorrect:



In this example, the numeric text is incorrectly left-aligned.

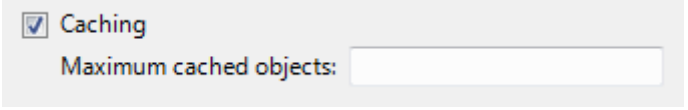
- **Always right-align monetary values.**
- **Don't assign special meanings to specific numeric values**, even if those special meanings are used internally by your application. Instead, use check boxes or radio buttons for an explicit user selection.

  Incorrect:

  In this example, the value -1 has a special meaning.

  Correct:

  In this example, a check box makes the option explicit.

Password and PIN input

- **Always use the password common control instead of creating your own.** Passwords and PINs require special treatment to be handled securely.

For more guidelines and examples, see Balloons.

Textual output

- **Consider using the white background system color for large, multi-line read-only text.** A white background makes the text easier to read. Lots of text on a gray background discourages reading.

For more information on background colors, see Fonts.

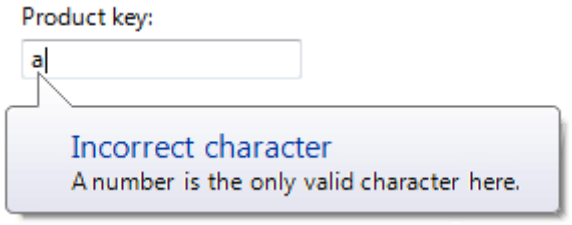
Data output

- **Don't use a border for single-line, read-only text boxes.** The border is a visual clue that the text is editable.
- **Don't disable single-line, read-only text boxes.** This prevents users from selecting and copying the text to the clipboard. It also prevents users from scrolling the data if it exceeds the size of its boundaries.
- **Don't set a tab stop on single-line, read-only text box unless the user is likely to need to scroll or copy the text.**

Input validation and error handling

Because text boxes are usually not constrained to accept only valid input, you may need to validate the input and handle any problems. Validate the various types of input problems as follows:

- If the user enters a character that isn't valid, ignore the character and display an input problem balloon that explains the valid characters.

  In this example, a balloon reports an incorrect input character.

- If the input data has a value or format that isn't valid, display an input problem balloon when the text box loses input focus.

- If the input data is inconsistent with other controls on the window, give an error message when the entire input is complete, such as when users click OK for a modal dialog box.

Don't clear invalid input data unless users aren't able to correct errors easily. Doing so allows users to correct mistakes without starting over. For example, you should clear incorrect passwords and PINs because users can't correct them easily.

For more guidelines and examples, see Error Messages and Balloons.

Prompts

A prompt is a label or short instruction placed inside a text box as its default value. Unlike static text, prompts disappear from the screen once users type something into the text box or it gets input focus.

A typical prompt.

Use a prompt when:

- Screen space is at such a premium that using a label or instruction is undesirable, such as on a toolbar.
- The prompt is primarily for identifying the purpose of the text box in a compact way. It must not be crucial information that the user needs to see while using the text box.

Don't use prompts just to direct users to type something or to click buttons. For example, don't write prompt text that says Enter a filename and then click Send.

When using prompts:

- Draw the prompt text in italic gray and the actual input text in normal black. The prompt text must not be confused with real text.
- Keep the prompt text concise. You can use fragments instead of full sentences.
- Use sentence-style capitalization.
- Don't use ending punctuation or ellipsis.
- The prompt text should not be editable and should disappear once users click in or tab into the text box.
  - **Exception:** If the text box has default input focus, the prompt is displayed, and it disappears once the user starts typing.
- The prompt text is restored if the text box is still empty when it loses input focus.

Recommended sizing and spacing

Recommended sizing and spacing for text boxes.

The width of a text box is a visual clue of the expected input size. When sizing text boxes:

- **Choose a width appropriate for the longest valid data.** In most situations, users shouldn't have to scroll the longest likely string they'll enter or view.
- **Include an additional 30 percent** (up to 200 percent for shorter text) for any text (but not numbers) that will be localized.
- **If the expected input has no particular size, choose a width that is consistent with the other text boxes or controls on the window.**
- **Size multi-line text boxes to display an integral number of lines of text.**

Labels

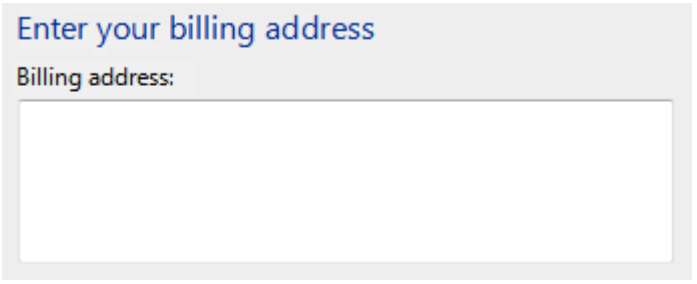
Text box labels

- All text boxes need labels. Write the label as a word or phrase, not as a sentence, ending with a colon, and using static text.

  Exceptions:

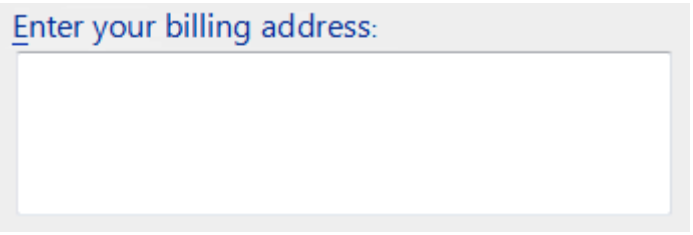
  - Text boxes with prompts located where space is at a premium.
  - For labeling, a group of text boxes used for **formatted data input** should be treated as a single text box.
  - If a text box is subordinate to a radio button or check box, and is introduced by its label ending with a colon, don't put an additional label on the text box.
  - **Omit control labels that restate the main instruction.** In this case, the main instruction takes the colon (unless it's a question) and access key.

    Acceptable:



In this example, the text box label is just a restatement of the main instruction.

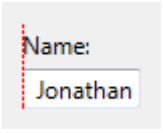
Better:

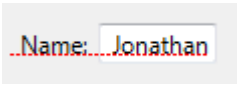


In this example, the redundant label is removed, so the main instruction takes the colon and access key.

- Assign a unique access key. For access key assignment guidelines, see Keyboard.
- Use sentence-style capitalization.
- Position the label either to the left of or above the text box, and align the label with the left edge of the text box. If the label is on the left, vertically align the label text with the text box text.

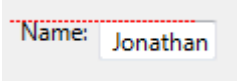
  Correct:

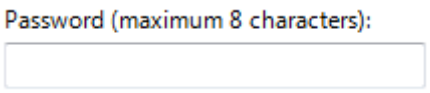
  In these examples, the label on top aligns with the left edge of the text box, and the label on the left aligns with the text in the text box.

  Incorrect:



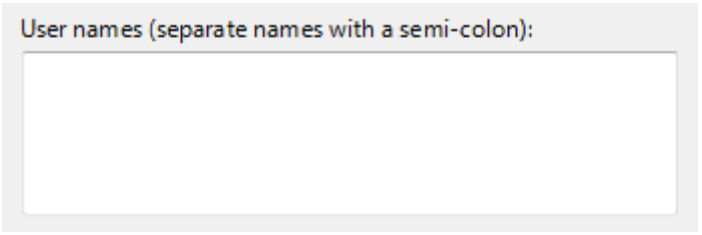
In these incorrect examples, the label on top aligns with the text in the text box, and the label on the left aligns with the top of the text box.

- You may specify units (for example, seconds or connections) in parentheses after the label.
- If a text box accepts an arbitrarily small maximum number of characters, you can state the maximum input in the label. The text box width should also suggest the maximum size.



In this example, the label gives the maximum number of characters.

- Don't make the content of the text box (or its units label) part of a sentence, because this is not localizable.
- If the text box can be used to enter several items, make it clear how to separate the items in the label.

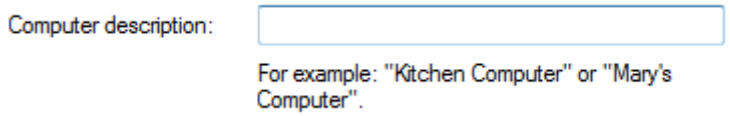


In this example, the item separator is given in the label.

- For guidelines on indicating required input, see Required input in Dialog Boxes.

Instruction labels

- If you need to add instructional text about a text box, add it above the label. Use complete sentences with ending punctuation.
- Use sentence-style capitalization.
- Additional information that is helpful but not necessary should be kept short. Place this information either in parentheses between the label and colon, or without parentheses below the text box.



In this example, additional information is placed below the text box.

Prompt labels

- Keep the prompt text concise. You can use fragments instead of full sentences.
- Use sentence-style capitalization.
- Don't use ending punctuation or ellipsis.
- If the prompt directs users to enter information that will be acted upon by a button next to the text box, simply place the button next to the text box. Don't use the prompt to direct users to click the button (for example, don't write prompt text that says, Drag a file and then click Send).

Documentation

When referring to text boxes:

- Use type to refer to user interactions that require typing or pasting; otherwise use enter if users can put information into the text box using other means, such as selecting the value from a list or using a Browse button.

- Use select to refer to an entry in a read-only text box.

- Use the exact label text, including its capitalization, and include the word box. Don't include the access key underscore or colon. Don't refer to a text box as a text box or a field.

- When possible, format the label using bold text. Otherwise, put the label in quotation marks only if required to prevent confusion.

  Example: Type your password into the **Password** box, and then click **OK**.

- If the text box requires a specific format, document only the most commonly used acceptable format. Let users discover any other formats on their own. You want to be flexible with data formats, but doing so should not result in complex documentation.

  Correct:

  Enter the part's serial number using the 1234-56-7890 format.

Incorrect:

Enter the part's serial number using any of the following formats:

1234567890

1234-56-7890

1234 56 7890

Feedback

Was this page helpful? Yes No

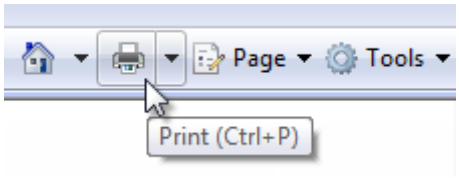
Provide product feedback | Get help at Microsoft Q&A

Tooltips and Infotips

Article • 06/04/2021

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

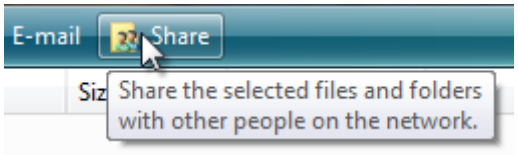
A tooltip is a small pop-up window that labels the unlabeled control being pointed to, such as unlabeled toolbar controls or command buttons.



A typical tooltip for a toolbar button.

Because tooltips have proved so useful, a related control called infotips exists, which provides more descriptive text than is possible with tooltips.

An infotip is a small pop-up window that concisely describes the object being pointed to, such as descriptions of toolbar controls, icons, graphics, links, Windows Explorer objects, Start menu items, and taskbar buttons. Infotips are a form of progressive disclosure controls, eliminating the need always to have descriptive text on screen.



A typical infotip.

For the purposes of this article, tooltips and infotips are referred to collectively as tips.

Tips help users understand unknown or unfamiliar objects that aren't described directly in the user interface (UI). They are displayed automatically when users hover the pointer over an object, and removed when users click the control or move the mouse, or when the tip times out.

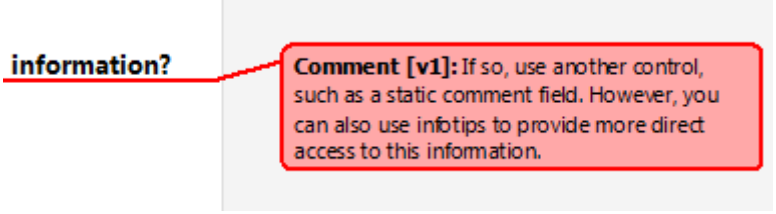
Developers: There is no infotip control; infotips are implemented with the tooltip control. The distinction is in usage, not implementation.

> **Note**
>
> Guidelines related to balloons, toolbars, and Help are presented in separate articles.

Is this the right control?

To decide, consider these questions:

- **Is the information 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. By contrast, balloons can display on their own (as they do with notifications), so they have a tail that identifies their source.
- **Does a control have a text label?** If not, use a tooltip to provide the label. Note that most controls should be labeled and therefore not have tooltips. Toolbar controls and command buttons with graphic labels should have tooltips.
- **Does an object benefit from a supplemental description or further information?** If so, use an infotip. However, 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 information an error, warning, or status?** If so, use another UI element, such as a balloon, error message, or status bar. Notification area icon infotips are an exception because they can be used to show status information.
- **Do users need to interact with the tip?** If so, use another control, such as a balloon. Users can't interact with tips because moving the mouse makes them disappear.
- **Do users need to print the supplemental information?** If so, use another control, such as a static comment field. However, you can also use infotips to provide more direct access to this information.

  In this example, a static comment field in Microsoft Word allows users to print comments.

- **Is the context such that users might find the tips annoying or distracting?** If so, consider using another solution including doing nothing at all. If you do use tips in such contexts, allow users to turn them off.

When used appropriately, tips improve communication with the user. **Never use tips as a substitute for good design.** If a graphic, button, or other object requires users to keep checking a tip to understand it, the design is bad. Fix the design instead.

Design concepts

Tips are a powerful way to simplify a user interface. They provide information users need when they need it, with minimal effort on their part. Tips can help you use screen space more effectively and reduce screen clutter. However, poorly designed tips can be annoying, distracting, unhelpful, overwhelming, or in the way. The following design concepts are intended to show the difference.

Discoverability

Tips display automatically when users hover the pointer over an object for a period of time. This time-delay mechanism makes tips very convenient, but it also reduces their discoverability.

Over time, users learn that certain standard objects like toolbar buttons, graphic buttons, Start menu items, and notification area icons have tips, so you can take their discoverability for granted.

It takes users longer to discover tips in nonstandard places. There is no visual clue, such as a hot spot or pointer change, that indicates that an object has a tip. Worse yet, some users move their mouse around a lot, especially when they are learning to navigate the UI. Users have to know that an object has a tip, either through past experience or by experimentation.

You can improve discoverability by using tips consistently, which in turn fosters predictability. If you provide tips for some objects, you should provide them for all similar objects for which users are likely to want supplemental information. Sometimes doing so can be challenging, because you must also make sure that the tips are helpful and not obvious.

If providing discoverable, consistently helpful tips proves to be a problem, consider alternative designs such as self-explanatory control labels or in-place supplemental text.

Appropriate information

Information appropriate for tips has the following characteristics:

- **Concise.** The pop-up windows used by tips are perfect for short sentences and sentence fragments, as well as formatted text. Large, unformatted blocks of text are difficult to read and overwhelming.
- **Helpful.** Tip text must be informative. It shouldn't be obvious or just repeat what is already on the screen.
- **Supplemental.** Because tip text isn't always visible, it should be supplemental information that users don't have to read. Important information should be communicated using self-explanatory control labels or in-place supplemental text.
- **Static.** Users don't expect tips to change from one instance to the next, so they are unlikely to notice changes in dynamic content, such as status information. Notification area icon tips are a notable exception: Users are more likely to discover changes in tip information there because these icons primarily communicate status.

Appropriate timeouts

The appropriate automatic display and removal of tips is crucial to the goal of users maintaining control of their UI environment. Tips have three timeout values:

- **Initial.** The time the pointer must remain stationary for the tip to appear. The default time is 0.5 seconds.
- **Reshow.** The time the pointer must remain stationary as the pointer moves from one target to another. The default time is 0.1 seconds.
- **Removal.** The time after which the tip is automatically removed. The default time is 5 seconds.

Having too short initial and reshow values results in an annoying, disruptive experience because they would often be shown inadvertently, whereas too long results in tips feeling unresponsive or not being discovered. The default removal time works well for short tip text, as used in tooltips. Infotips have longer text, so they need longer display times.

Appropriate placement

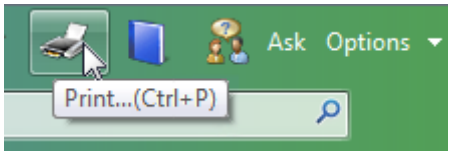
Tips should be placed near the object being hovered, usually at the pointer's tail or head if possible. However, they should never be placed in a way that interferes with what the user is doing by obscuring the object of interest. Preventing this problem might require you to move the tip away from the pointer but adjacent to the object. This isn't a

problem as long as the relationship between the object and its tip is clear. Make sure users don't move the pointer just to get your program's tips to go away.

Accessibility

Tips have an unusual effect on accessibility. While they are normally triggered by hovering the pointer over an object, tips are handled by screen readers for controls with keyboard access. When used appropriately for concise, helpful, static, supplemental information, tips can improve overall accessibility. In fact, the alt text tip pattern is the preferred way to make graphics accessible. However, when used inappropriately, they harm accessibility by making important or dynamic information harder to obtain.

Provide more than one way to access a control if that control requires a tip that doesn't have keyboard access.



In this example, users can print using the toolbar button (which isn't keyboard accessible) or the Print command keyboard shortcut.

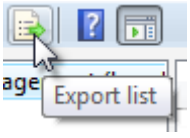
If you do only one thing...

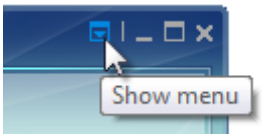
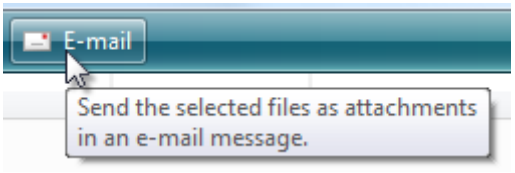
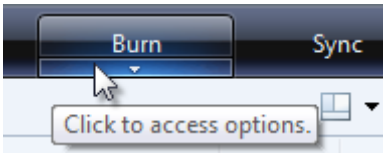
Design discoverable tips that display concise, helpful, static, supplemental information in the appropriate place at the appropriate time.

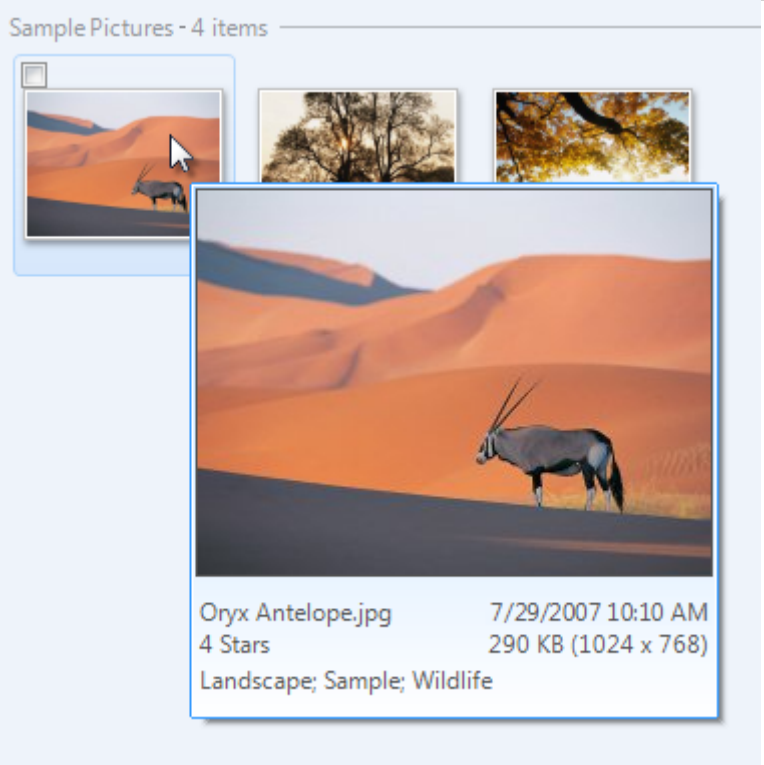
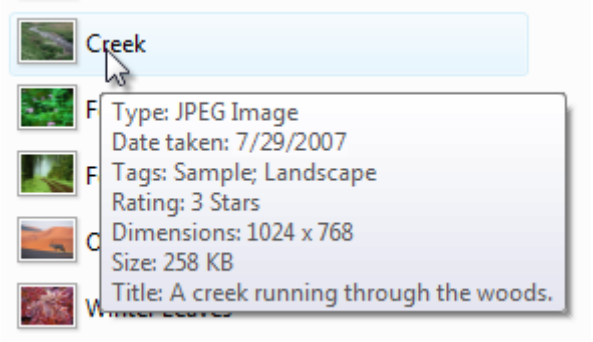
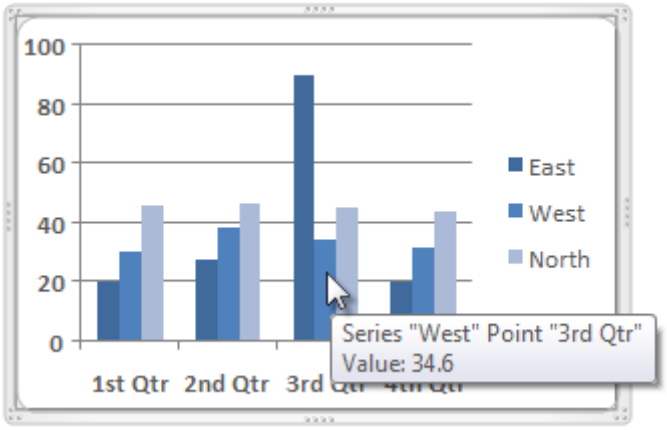
Usage patterns

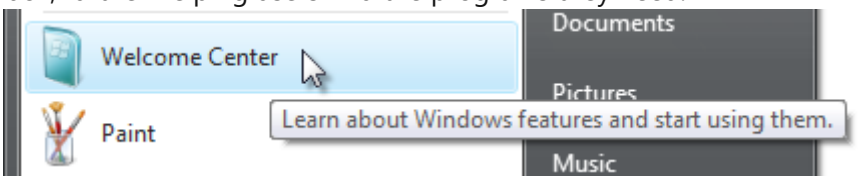
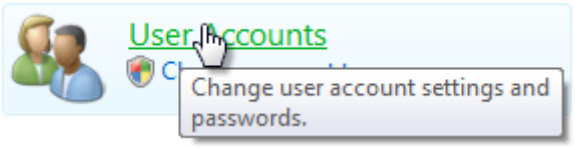
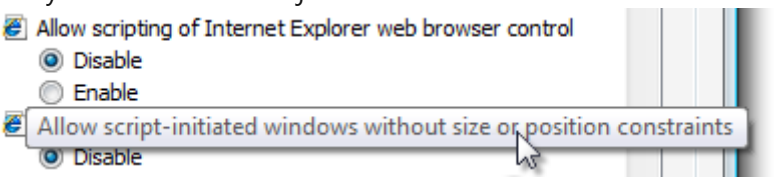
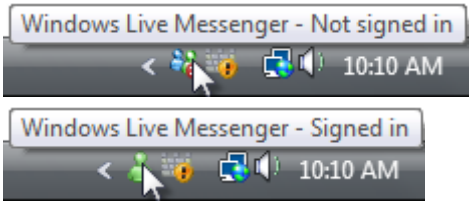
Tips have several usage patterns:

Expand table

Usage	Example
Tooltips display the label of an unlabeled control or glyph.	Because these tips serve as labels, their text follows the label guidelines for the underlying control.  in this example, the tooltip gives the command label.

Usage	Example
	 in these examples, tooltips label graphic buttons.  In this example, the tooltip labels a glyph.
Infotips provide a supplemental description or explanation of an object or control.	Use infotips to describe or explain objects and controls such as toolbar controls, icons (including icon overlays), links, tabs, progressive disclosure controls, and custom controls.   In these examples, infotips provide supplemental information about controls and objects.
Alt text infotips describe a graphic for accessibility.	This pattern is primarily for users who have some form of vision impairment, and may be using a screen reader. In this example, the infotip describes the Start menu graphic.
Thumbnails display a small image of an item.	Thumbnails give an easily recognizable graphic representation of a window or document. in this example, the windows taskbar gives thumbnail tips for its items.

Usage	Example
	 In this example, Windows Photo Gallery gives thumbnail tips for its items.
Detail infotips display detailed information about an object.	Infotips are an effective way to show detailed information about an object, or to provide data.   In these examples, infotips give detailed information about an object or data.

Usage	Example
Start Menu infotips describe an item on the start menu.	The start menu consists of program names and important windows destinations, such as documents, pictures, and control panel. these tips describe start menu items, typically by giving a brief description of the program or destination as well as the primary tasks users can perform with it. these descriptions are also indexed by the start menu search box, further helping users find the programs they need.  In this example, the infotip describes what users can do with a program in the Start menu.
Control Panel infotips describe a control panel category or task.	These tips provide supplemental information to help users choose the right control panel category and item.  In this example, the infotip describes the User Accounts Control Panel category.
Full name infotips display the full name of an item when the name is truncated or not fully visible.	These tips allow you to display items in a more compact space, while reducing the need for horizontal scrolling. this is especially important when the content length is unknown because it is dynamic. unlike the other patterns, when used in lists and trees these tips are displayed directly over the source object.  In this example, an infotip is used to display the full item name on hover.
Status infotips display status information for notification area icons.	Normally, tips should be static because users don't expect them to change from one instance to the next. **notification area icons are an exception** because these icons communicate status, and there is no other screen space available for status text.  In these examples, infotips give status information for notification area icons.

Guidelines

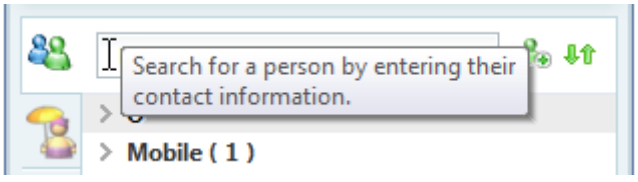
Timeouts

- **Use the default initial and reshow timeouts. Exception:**
  - Thumbnails that aren't redundant and displayed on the side of their associated object can be shown immediately (without any delay). However, use the default initial timeout for redundant thumbnails (such as a large thumbnail tip for a small graphic object) or thumbnails that cover their associated object.
- **For tooltips, use the default five-second tip removal timeout.**
- **For infotips, turn off the tip removal timeout. Developers:** Since you can't technically turn off the removal timeout, set it to its largest value.
- For accessibility, if you need to set the timeout values to something other than the maximum value, make them multiples of the SPI_GETMOUSEHOVERTIME and SPI_GETMESSAGEDURATION system parameters instead of using fixed times. Doing so adjusts the timeouts to the speed of the user.

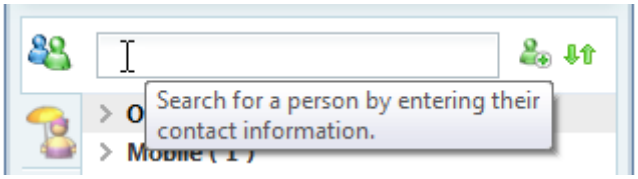
Placement

- **Avoid covering the object the user is about to view or interact with.** Always place the tip on the side of the object, even if that requires separation between the pointer and the tip. Some separation isn't a problem as long as the relationship between the object and its tip is clear.
  - **Exception:** Full name tooltips used in lists and trees.

  Incorrect:

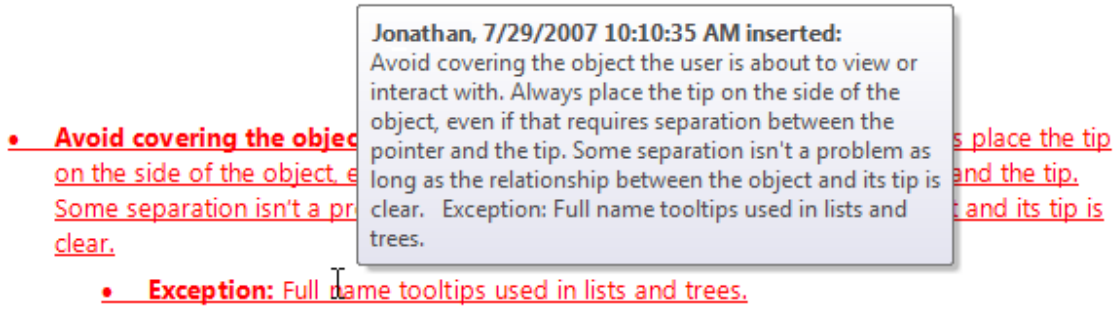
  

  Correct:

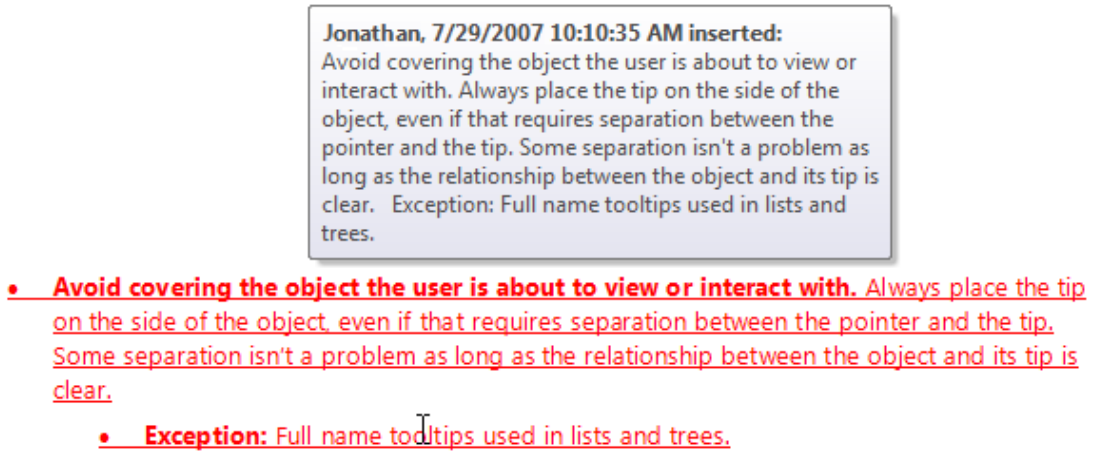
  

  In the correct example, the infotip is placed away from the Search box, even though that requires space between it and the caret.

  Incorrect:



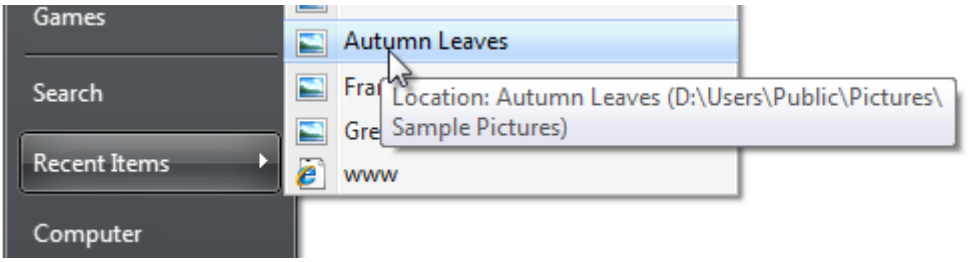
Correct:



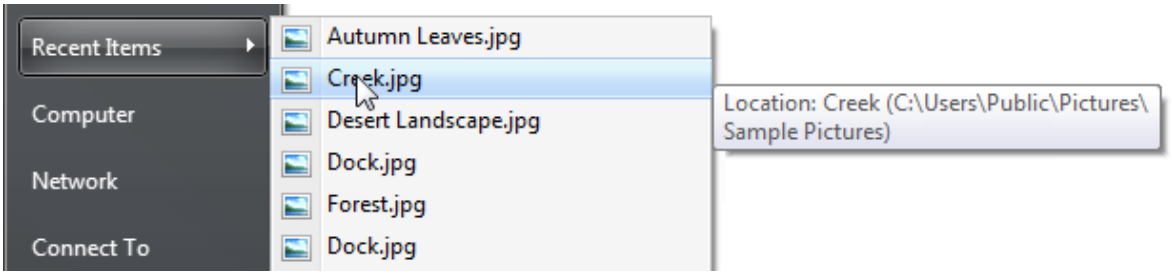
In the correct example, the underlying text is far more useful than the tip, so the infotip is placed well out of the way.

- **For collections of items, avoid covering the next object that the user is likely to view or interact with.** For horizontally arranged items, avoid placing tips to the right; for vertically arranged items, avoid placing tips below.

Incorrect:



Correct:



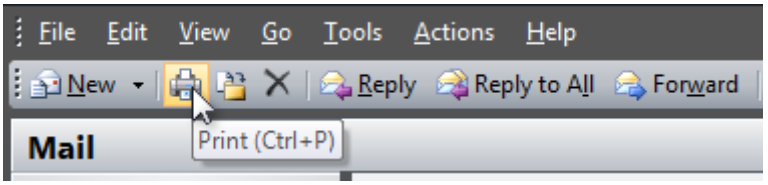
In the incorrect example, the tip covers the object the user is most likely to interact with next.

- **For potentially distracting (often large) tips, make sure that the information is helpful for most users.** If that's not the case, either make the distracting tips optional or even eliminate them. Otherwise, most users will have to move the pointer away from the target object to get rid of the tip.

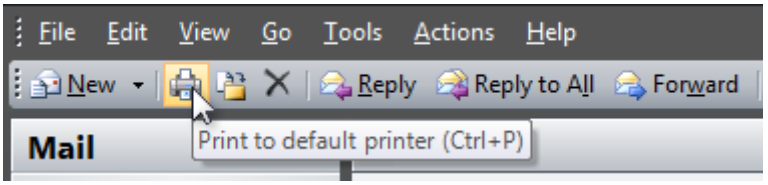
Tooltips

- **Use tooltips to provide labels for unlabeled controls.** Controls that commonly have tooltips are toolbar buttons, graphic buttons, and progressive disclosure controls. Controls with prompts are considered labeled, such as text boxes and combo boxes. All other controls should have explicit labels.
- Use sentence fragments without ending punctuation.
- Use sentence-style capitalization.
  - **Exception:** This guideline is new for Windows Vista. For legacy applications, you may use title-style capitalization if necessary to avoid mixing capitalization styles.
- Add an ellipsis if the label is for a command that needs additional information.
- As with normal labels, **keep tooltips brief** typically five words or less but prefer specific labels over vague ones.

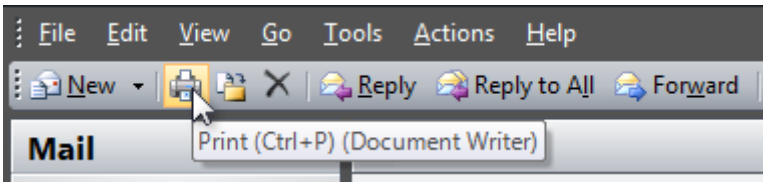
  Acceptable:



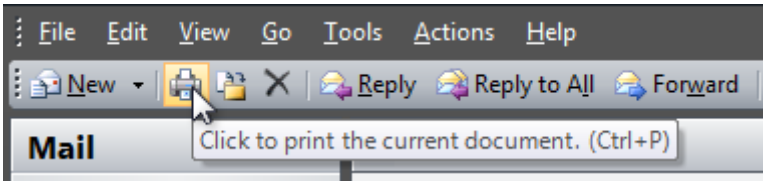
  Better:



  Best:



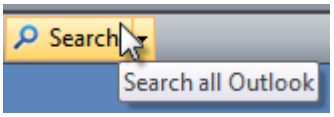
Incorrect:



In these examples, the best example is both concise and specific, whereas the incorrect example is unnecessarily verbose.

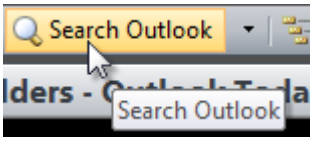
- **Tooltips may also provide more detail for labeled toolbar buttons if doing so is helpful.** Don't just repeat or give a wordy restatement of what is already in the label.

  Correct:

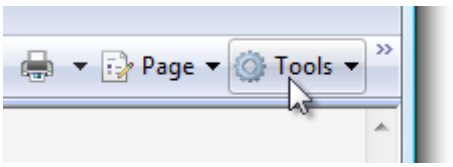
  In this example, the tooltip explains what Search does.

  Incorrect:

  In this example, the tooltip just repeats what is already in the label.

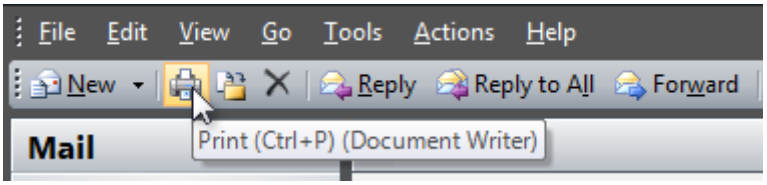
- **You don't have to give labeled controls tooltips simply for the sake of consistency.**

  In this example, the unlabeled toolbar buttons have tooltips, but the labeled ones don't.

- Whenever appropriate, **make tooltips more helpful by providing keyboard shortcuts and default values.** Put this additional information in parentheses. Doing

so makes tooltips helpful for labeled controls even when they otherwise just repeat the label. Don't consider this additional text when evaluating the conciseness of a tooltip.



In this example, Word displays default values and keyboard shortcuts in the toolbar tooltips.

Infotips

- **For infotips in nonstandard places, favor consistency over helpfulness to improve discoverability.** Provide tips for all objects for which users are likely to want supplemental information, even if a few infotips might be obvious. Doing so avoids having users wait for an infotip that will never come.
  - **Exception:** If only a few objects have helpful infotips, don't use infotips at all. Rather, use self-explanatory control labels or in-place supplemental text instead.

- Use full sentences with ending punctuation.
  - **Exception:** Notification area icon infotips don't use ending punctuation.

- Use sentence-style capitalization.

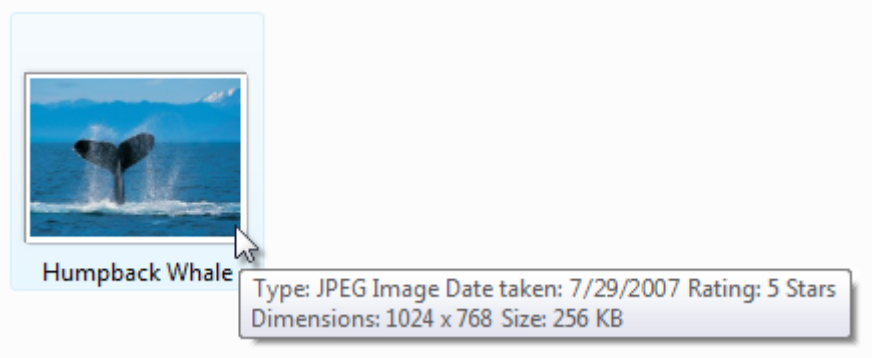
- Use present tense, not future.

- Use parallel grammatical constructions. Parallelism requires that words and phrases that have the same function have the same form.
  - **Exception:** For the full name infotip pattern, the infotip text exactly matches the phrasing, capitalization, and punctuation of the underlying control.

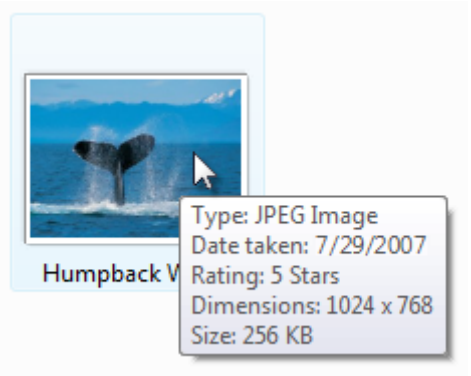
- **Avoid large infotips.** Large infotips are difficult to read, and difficult to position without interfering with the underlying object.

- **Format infotips to make their content easier to read and scan.** Large blocks of unformatted text can be difficult to read.

  Incorrect:



Correct:



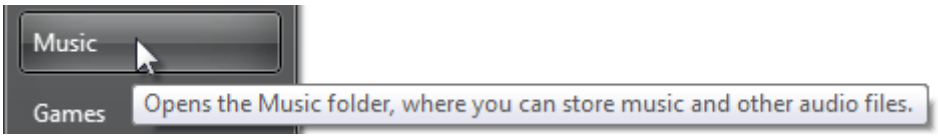
In the correct example, the formatted text is much easier to read and scan.

- On first use within an infotip, spell out the names of acronyms, followed by the acronym in parentheses. Example: "Dynamic Host Configuration Protocol (DHCP)."

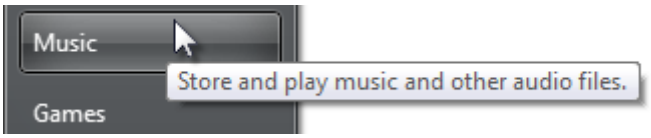
Start Menu infotips

- Use Start menu infotips to **describe the item concisely and list the primary tasks that users can perform with the item.**
- **Be helpful.** Focus on what users can do. Don't just repeat the item name or even use it in the description at all.
- **Be specific.** Avoid generic verbs and catch-all phrases like and other tasks. If the information is important, list it specifically; otherwise, assume that users understand that not everything is listed in the infotips.
- **Be concise.** Use 25 words or less. Longer infotips discourage reading.
- **Start with a present-tense, imperative verb** such as create, edit, show, and send. Prefer specific verbs over generic verbs such as manage and open, which really apply to most Start menu items. Get right to the point.

  Incorrect:



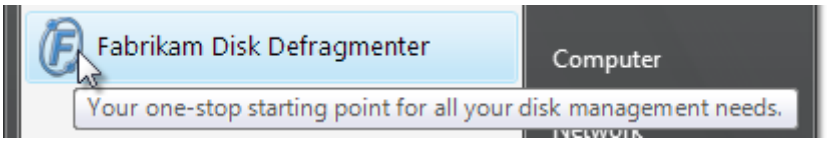
Better:



In the incorrect example, the infotip starts with a generic verb. The better example gets right to the point with specific verbs, but continues to use the unnecessary "and other" phrasing at the end of the tip.

- **Don't use language that sounds like marketing.**

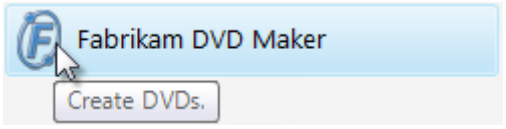
  Incorrect:



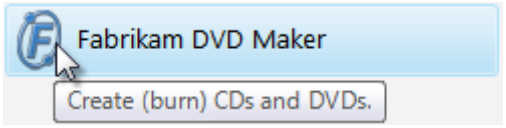
  In this example, the infotip sounds like marketing.

- Because these infotips are indexed for the Start menu search box, **describe your program's important tasks using terms for which users are most likely to search. Consider using keywords and common synonyms.**

  Incorrect:



  Correct:



  In the correct example, the infotip has common synonyms.

- Use sentence-style capitalization.

- **Developers:** The Start menu infotip text comes from the item's Comment field.

Quick Launch tooltips

- **Use a tooltip with the format:** Launch (full program name)
- Don't use ending punctuation.
- Don't use additional text to describe the program or what it does. Because users choose the programs displayed in the Quick Launch bar, they already know their purpose.

Control Panel infotips

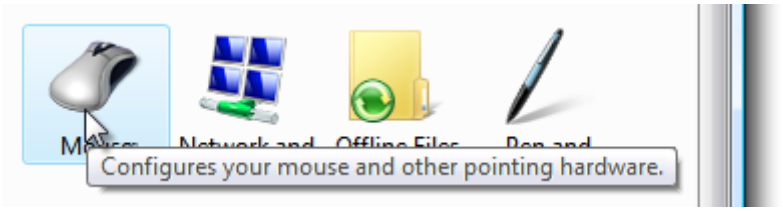
- Use Control Panel infotips to **concisely describe the Control Panel tasks and the hardware and software configured.**

- **Control Panel names and icons must have infotips.** Individual tasks don't have tooltips.

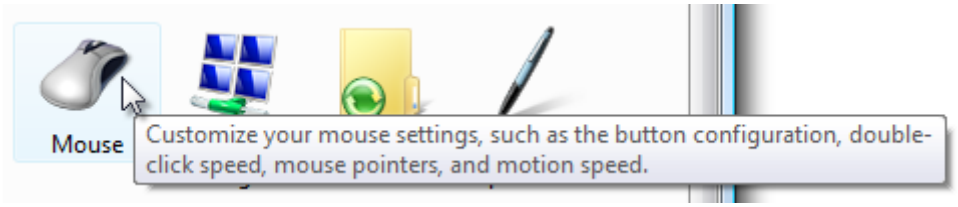
- **Be helpful.** Focus on what users can do. Don't just repeat the Control Panel item name or even use it in the description at all.

- **Be specific.** Avoid generic verbs and catch-all phrases like and other hardware. If the information is important, list it specifically; otherwise, assume that users understand that not everything is listed in the infotips.

  Incorrect:

  Correct:

  In the correct example, the types of hardware configured are specifically listed.

- **Be concise.** Use 25 words or less. Longer infotips discourage reading.

- **Start with a present-tense, imperative verb.**

  Correct:

  Configure Internet display and connection settings.

Adjust settings for vision, hearing, and mobility.

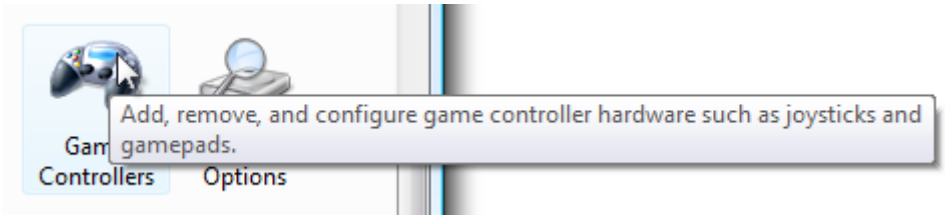
- **Get right to the point.** Don't use language that applies to any Control Panel, such as "Use to view and configure settings for the appearance and functionality of your..." or "Provides options for you to..."

- Don't use language that sounds like marketing.

  Incorrect:

  Your one-stop starting point for all your disk configuration needs.

- Because these infotips are indexed for the Control Panel search box, **describe items using terms for which users are most likely to search.** Consider using common synonyms for popular tasks and objects.



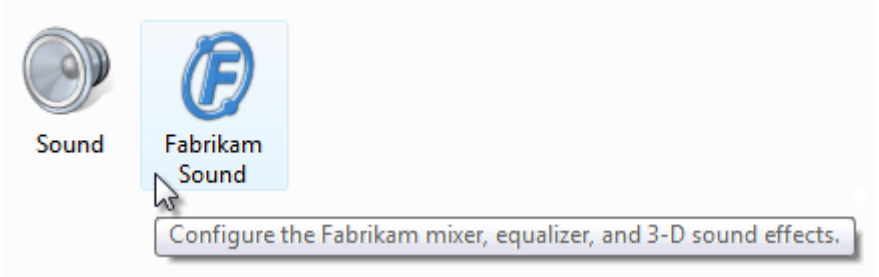
  In this example, the item is described using terms for which users are most likely to search.

- If a Control Panel item is likely to be confused with others, explain how it is different in the infotip.

  Incorrect:

  In this example, both Control Panel items configure sound, but the infotip doesn't clarify the difference.

  Correct:



In this example, the difference between the two items is more evident because of the tip.

Icons

Unlike previous versions of Windows, Windows Vista allows tips to have icons.

- For tooltips, don't use icons.
- For infotips, use icons only if they aid in recognition or comprehension, or provide context. Most infotips shouldn't have icons.

In this example, the infotip has an icon to help associate the icon with its meaning.

- The icon must use the Aero-style and have an unobtrusive appearance.

For general icon guidelines and examples, see Icons.

Documentation

When referring to tips:

- In programming and other technical documentation, refer to the type of tip (tooltip or infotip). Everywhere else, simply call it a tip.
- The following variations are incorrect: tool tip, Tooltip, and ToolTip.
- To describe user interaction, use hover.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

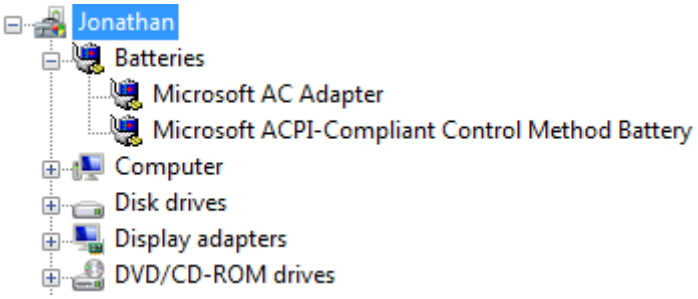
Tree Views

Article • 06/04/2021

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

With a tree view, users can view and interact with a hierarchically arranged collection of objects, using either single selection or multiple selection.

In a tree, objects that contain data are called leaf nodes and objects that contain other objects are called container nodes. A single, top-most container node is called the root node. Users can expand and collapse container nodes by clicking the plus and minus expander buttons.



A typical tree view.

> **Note**
>
> Guidelines related to layout and menus are presented in separate articles.

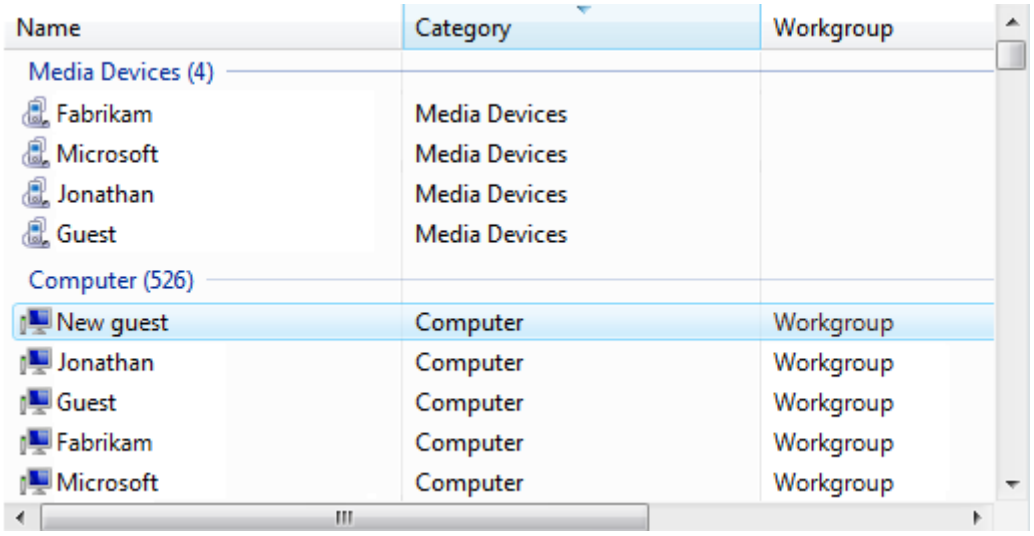
Is this the right control?

Having hierarchical data doesn't mean that you must use a tree view. Very often a list view is a simpler, yet more powerful choice. List views:

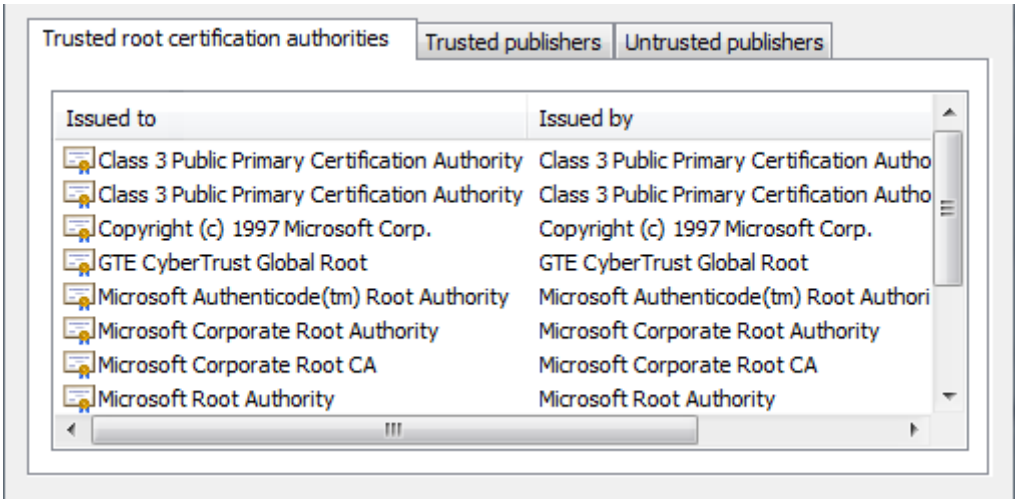
- Support several different views.
- Support sorting data by any of the columns in Details view.
- Support organizing data into groups, forming a two-level hierarchy.

To use a list view, you can flatten hierarchical information using the following techniques:

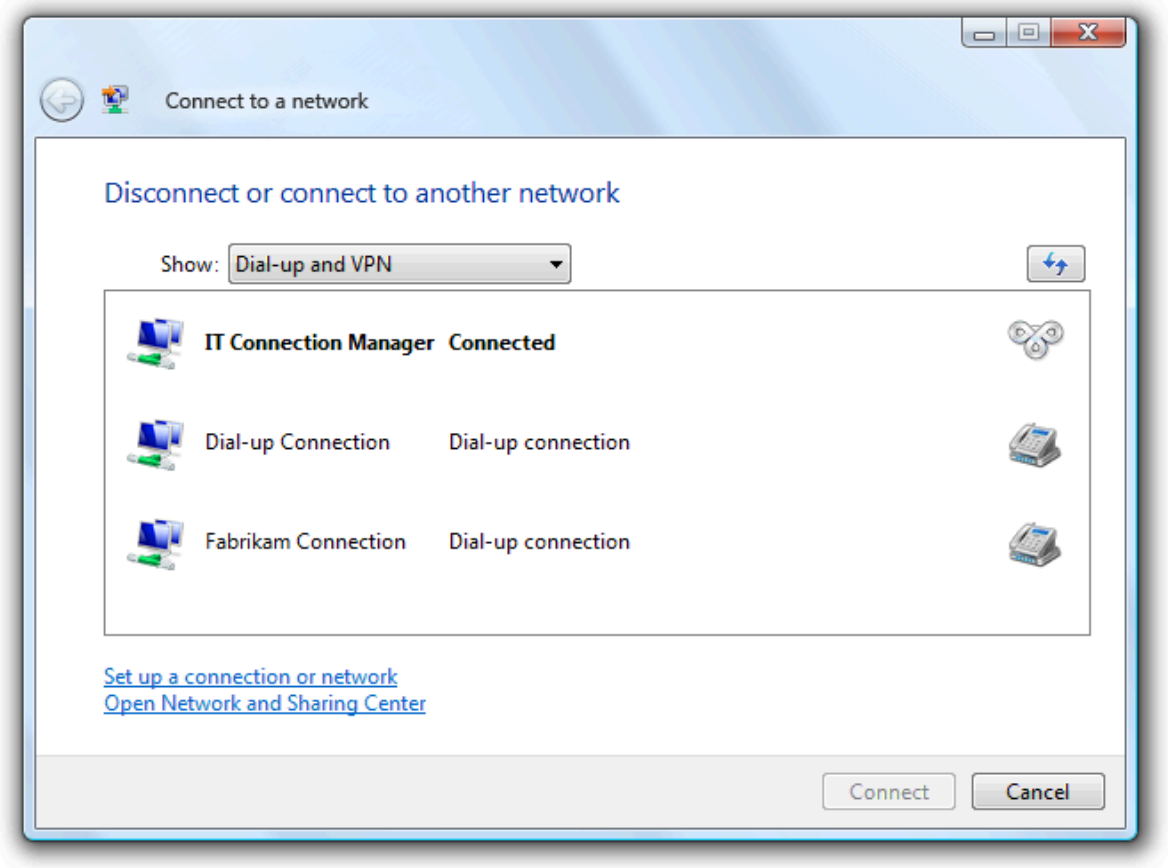
- Remove the root node if present, because it often isn't necessary.
- Use list view groups, tabs, drop-down lists, or expandable headings to replace the top-level containers.



In this example, list view groups are used for the top-level containers.

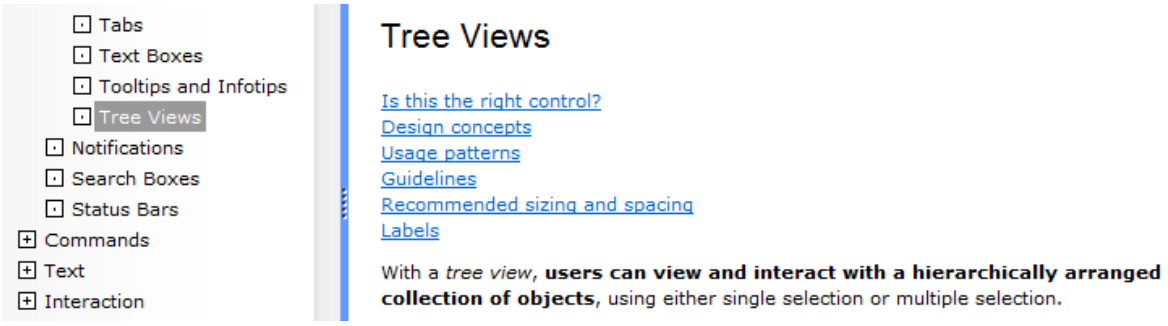


In this example, tabs are used for the top-level containers



In this example, a drop-down list is used for the top-level containers.

- If an associated control displays the content of the selected container, that control can display lower levels of the hierarchy.



In this example, low-level containers are displayed in the document window.

You must use a tree view if you need to display a hierarchy of more than two levels (not including the root node).

To decide if a tree view is the right control, consider these questions:

- **Is the data hierarchical?** If not, use another control.
- **Does the hierarchy have at least three levels (not including the root)?** If not, consider alternatives such as list view groups, tabs, drop-down lists, or expandable headings.

- **Do the items have auxiliary data?** If so, consider using a list view in the Details view mode to take full advantage of the auxiliary data.
- **Does the lower-level data relate to independent subtasks?** If so, consider displaying the information in an associated control or in a separate window (displayed using command buttons or links).
- **Are the target users advanced?** Advanced users are more proficient at using trees. If your application is aimed at novice users, avoid using tree views.
- **Do the items have a single, natural, hierarchical categorization that's familiar to most users?** If so, the data is ideal for a tree view. If there is a need for multiple views or sorting, use a list view instead.
- **Do users need to see the lower-level data in some but not all scenarios, or some but not all of the time?** If so, the data is ideal for a tree view.

> **Note**
>
> Sometimes a control that looks like a tree view is implemented using a list view. In such cases, apply the guidelines based on the usage, not on the implementation.

Design concepts

Trees are intended to organize data and make it easy to find, yet it's difficult to make data within a tree easily discoverable. Keep the following principles in mind when deciding about tree views and their organization.

Predictability and discoverability

A tree view is based on the relationships between objects. Trees work best when the objects form a clear, well known, mutually exclusive relationship in which every object maps to a single, easy-to-determine container.

A significant problem is that an object can appear in different nodes. For example, where would users expect to find a hardware device that plays music, has a large hard disk, and uses a USB port? Perhaps in any of several different container nodes, such as Multimedia, Storage, USB, and possibly in Hardware Resources. One solution is to place each object under the single most appropriate container regardless of the circumstances; another approach is to place each object under all containers that apply. The former promotes a simple, clean hierarchy and the latter promotes discoverability each has advantages and potential problems.

Users may not completely understand the layout of the tree, but they will form a mental model of the relationships after interacting with the tree for a while. If that mental model is incorrect, it leads to confusion. For example, suppose a music player can be found in the Multimedia, Storage, and USB containers. This arrangement improves discoverability. If a user first finds the device in Multimedia, the user may conclude that all devices like music players appear in the Multimedia container. The user will then expect similar devices, such as digital cameras, to appear in the Multimedia container and will become confused if that isn't the case.

The challenge when designing a tree is to balance discoverability with a predictable user model that minimizes confusion.

Breadth vs. depth

Usability studies have shown that **users are more successful at finding objects in a tree that is broad than in a tree that is deep**, so when designing a tree you should prefer breadth over depth. Ideally, a tree should have no more than four levels (not counting the root node) and the most commonly accessed objects should appear in the first two levels.

Other principles

- When users find what they are looking for, they stop looking. They don't look to see where else an object might be found because they don't need to. Those users may assume that the first path they find is the only path.
- Users have trouble finding objects in large, complex trees. Users will not perform an exhaustive, manual search to find objects in such trees; they stop once they think they've expended a reasonable effort. Consequently, large, complex trees need to be supplemented with other access methods, such as word search, an index, or filtering.
- Some programs allow users to create their own trees. While such self-designed trees might be aligned with a user's mental model, they are often created haphazardly and poorly maintained. For example, while a file system, e-mail program, and Favorites list typically store similar types of information, users rarely bother to organize them in the same way.

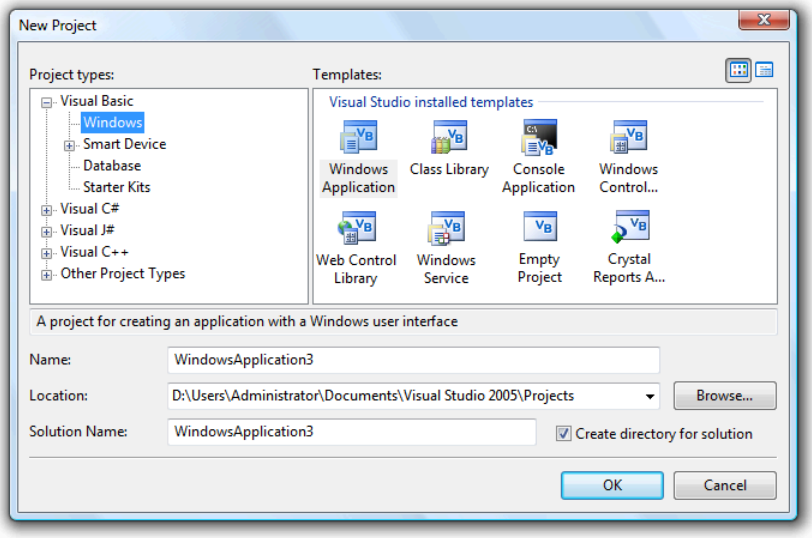
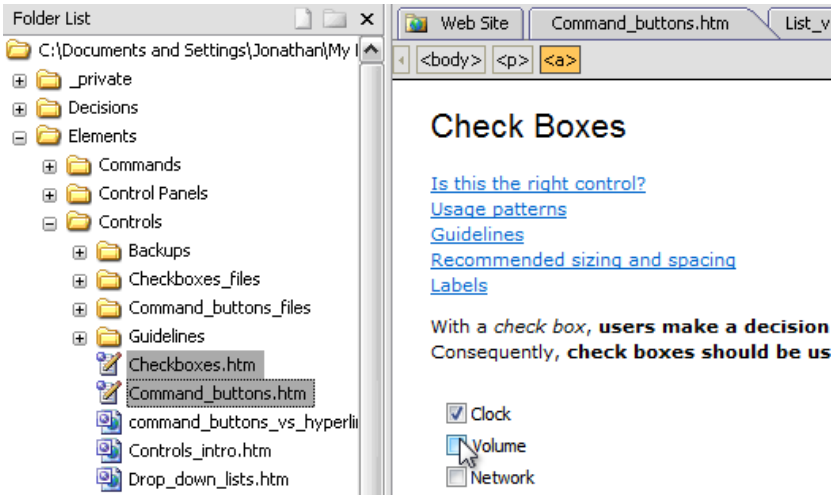
If you do only one thing...

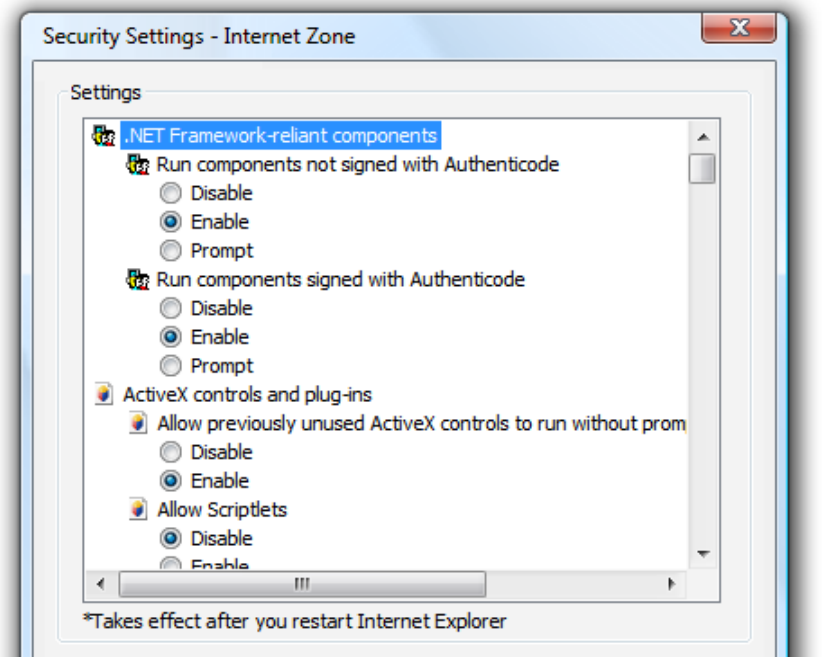
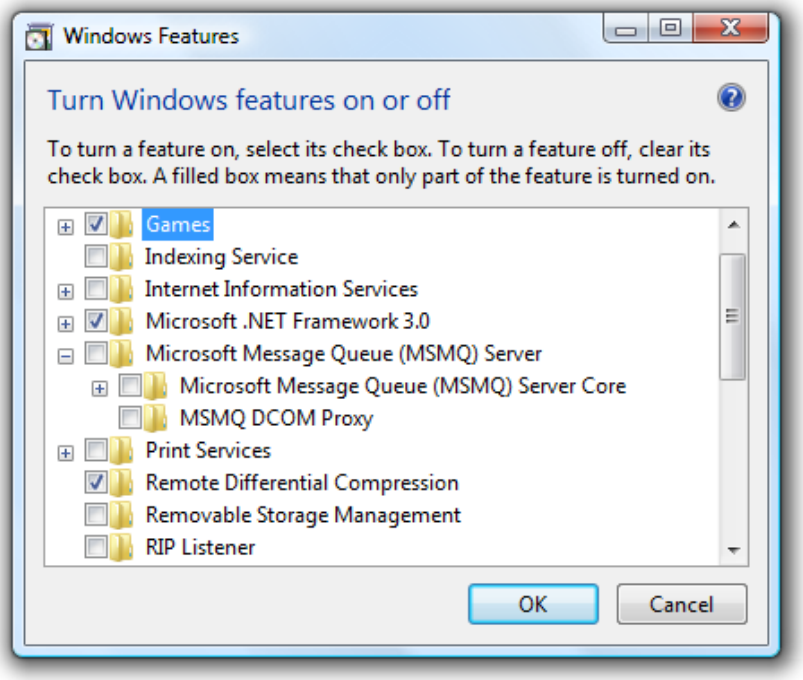
Carefully weigh the benefits and drawbacks of using tree views. Having hierarchically arranged data doesn't mean that you need to use a tree view.

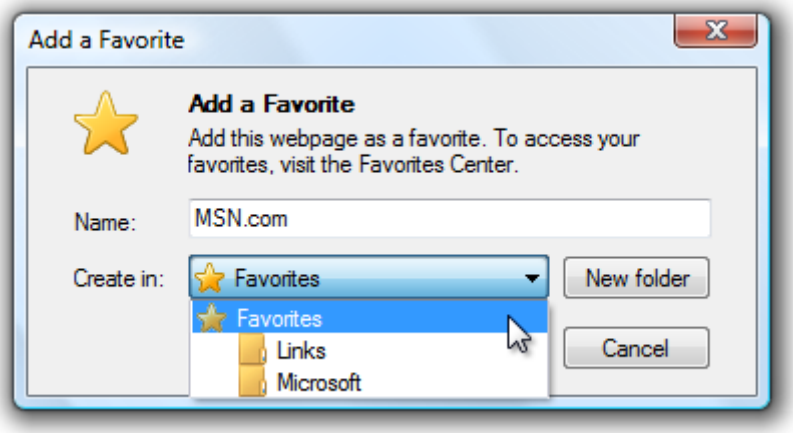
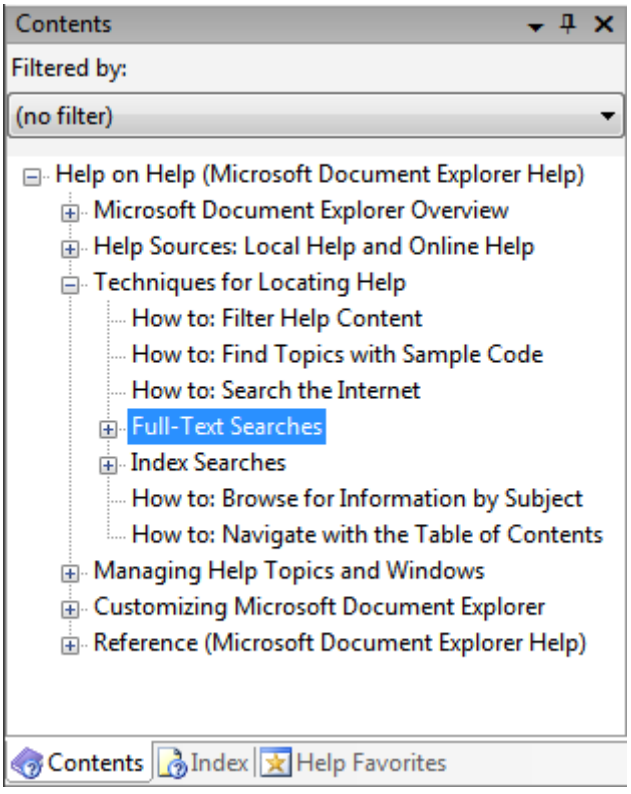
Usage patterns

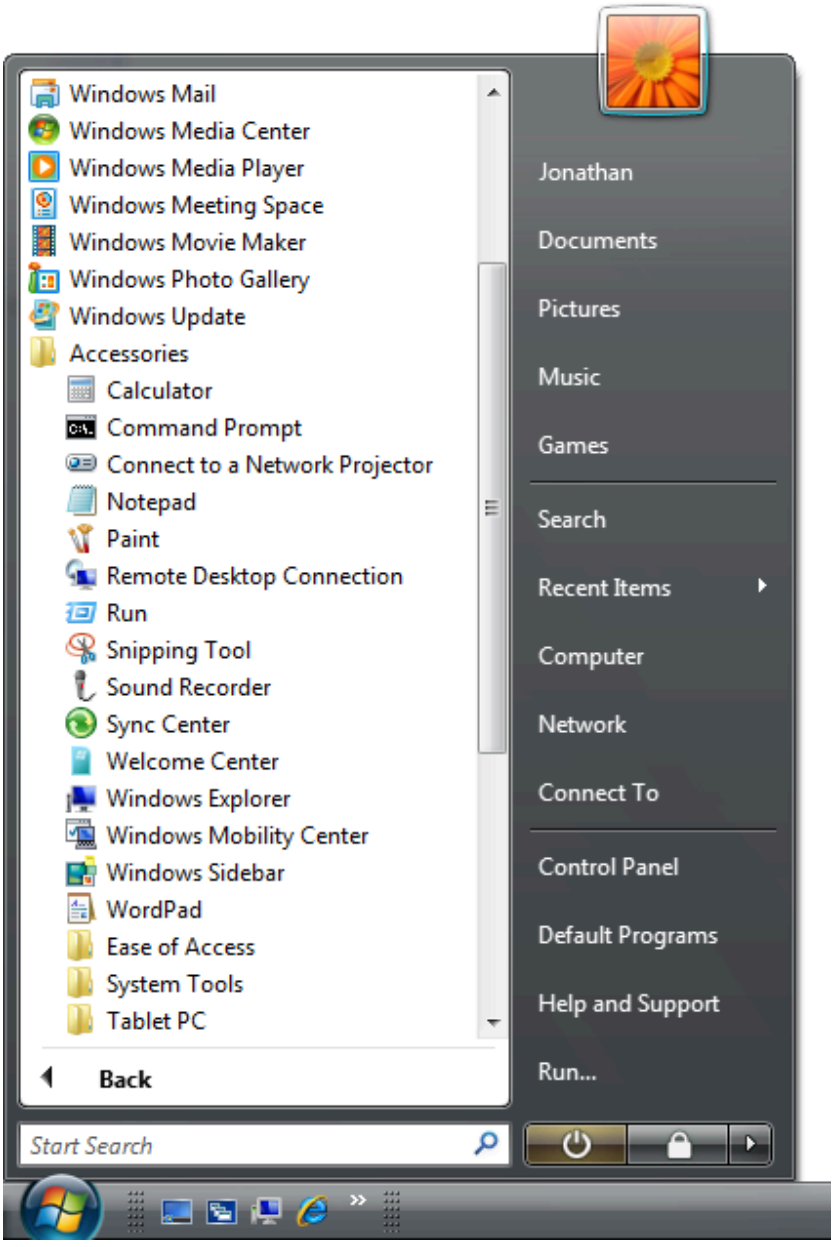
Tree views have several usage patterns:

Expand table

Usage	Example
Tree views with only container nodes users can view and interact with one container at a time.	Typically, these tree views have an associated control that displays the content of the selected container, so users can interact with only one container at a time.  In this example, the tree view has only container nodes. The contents of the selected node are displayed in the associated list view control.
Tree views with container and leaf nodes users can view and interact with containers and leaves.	Typically, these tree views have an associated control that displays the content of the selected container or leaf. allowing users to interact with leaves often makes it necessary to support multiple selection.  in this example, the tree view has both container and leaf nodes. since multiple selection is supported, the content of the opened items are displayed using tabs in the associated control. alternatively, the tree view can have an organized list, where the

Usage	Example
	containers are headings and the leaves are options.  In this example, the tree leaves are options and the containers are option categories.
Check box tree views users can select any number of items, including none.	The check boxes clearly indicate that multiple selection is possible. use this tree pattern when multiple selection is essential or commonly used.  In this example, a check box tree view allows selection of features to turn on or off.

Usage	Example
Tree view builders users can create a tree by adding one container or leaf at a time and optionally setting the order.	Many trees can be created or modified by users. some trees are built in place using context menus and drag-and-drop (such as the folders in windows explorer), whereas other trees are built using a specialized dialog box (such as the favorites list in windows internet explorer).  In this example from Internet Explorer, users can build their own list of favorites by using a dialog box.
Tree views with alternative access methods users can find objects in ways other than using a hierarchical tree.	As mentioned previously, users have trouble finding objects in large, complex trees, so such trees should be supplemented with other access methods, such as a word search, an index, or filtering.  in this example, users can also access information using a table of contents, an index, and favorites. for some users, the index and search tabs can be more useful than the contents tab.

Usage	Example
	 In this example, the Windows Start menu also lets users access programs, files, and Web pages by typing part of the name into the Search box.

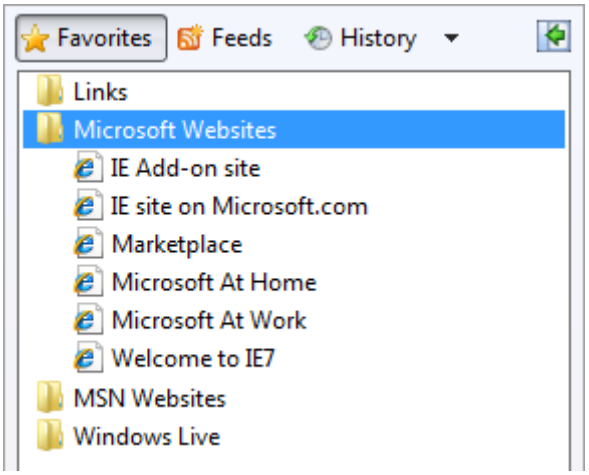
Guidelines

Presentation

- **Within a container, sort the items in a logical order.** Sort names in alphabetical order, numbers in numeric order, and dates in chronological order.
- **Use the Always Show Selection attribute** so that users can readily determine the selected item, even when the control doesn't have input focus.

- **If the tree is acting as a table of contents, use the Single Expand attribute to simplify the management of the tree.** This way, only the relevant portion of the tree is expanded.

- **Avoid presenting empty trees.** If a user creates a tree, initialize the tree with instructions or example items that users might need.



  In this example, the list is initially presented with examples.

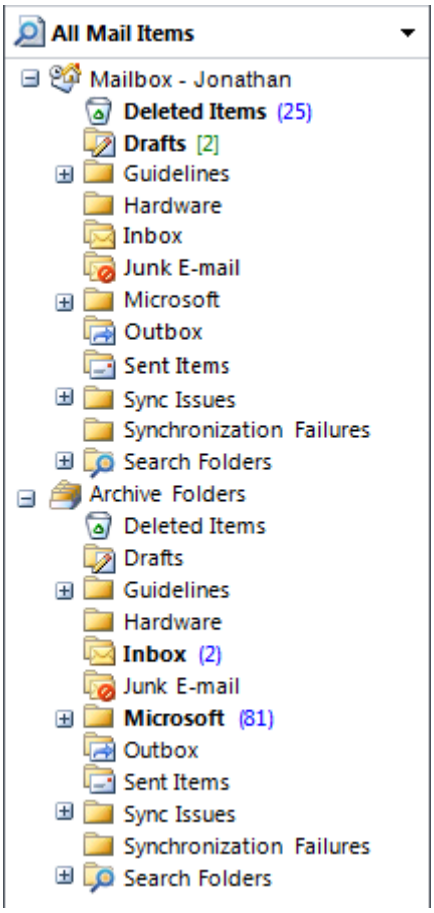
- **Don't make the container nodes collapsible if users have no reason to collapse them.** Doing so adds unnecessary complexity.

- **If load performance is a problem, display only the first and second level containers of the tree by default.** You can then load additional data on demand when a user expands branches in the tree.

- **If users expand or collapse a container, make that state persist so it takes effect the next time the tree view is displayed**, unless users are likely to prefer starting in the default state. Persistence should be on a per-tree view, per-user basis.

- **If high-level containers have similar contents, consider using visual clues to differentiate them.**

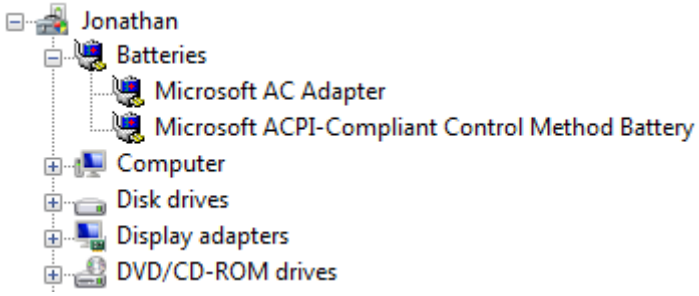
  Incorrect:



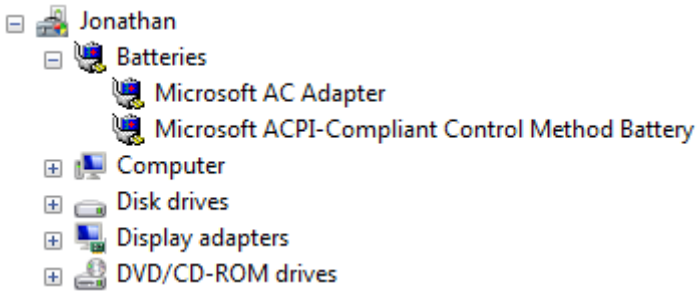
In this example, the Mailbox and Archive Folders have similar contents. Once the trees are further expanded, it is very difficult for users to know where they are in the tree, leading to confusion. Using slightly different icons in the different sections would address this problem.

- **Reconsider connecting lines.** Although these lines clearly show relationships between container and leaf nodes, they add visual clutter without aiding comprehension significantly. Specifically, they don't help when nodes are close together, nor do they help when nodes are so far apart that scrolling is required.

  Correct:



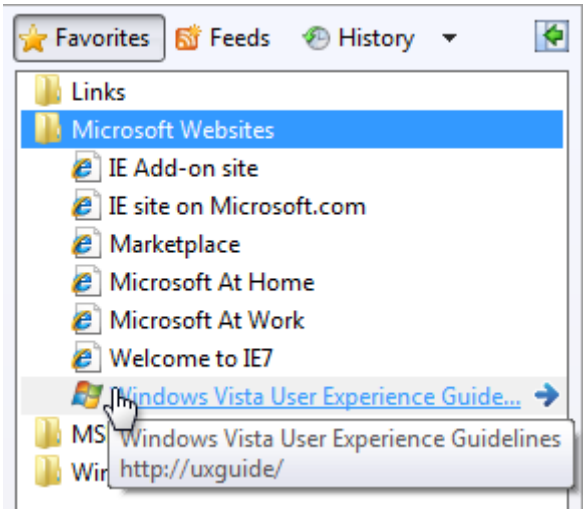
  Better:



The connecting lines do little to aid comprehension.

Interaction

- **Consider providing double-click behavior.** Double-clicking should have the same effect as selecting an item and performing its default command.
- **Make double-click behavior redundant.** There should always be a command button or context menu command that has the same effect.
- If an item requires further explanation, **provide the explanation in an infotip**.



In this example, an infotip provides further information.

- **Provide context menus of relevant commands.** Such commands include Cut, Copy, Paste, Remove or Delete, Rename, and Properties.
- **When disabling a tree view, also disable any associated labels and command buttons.**

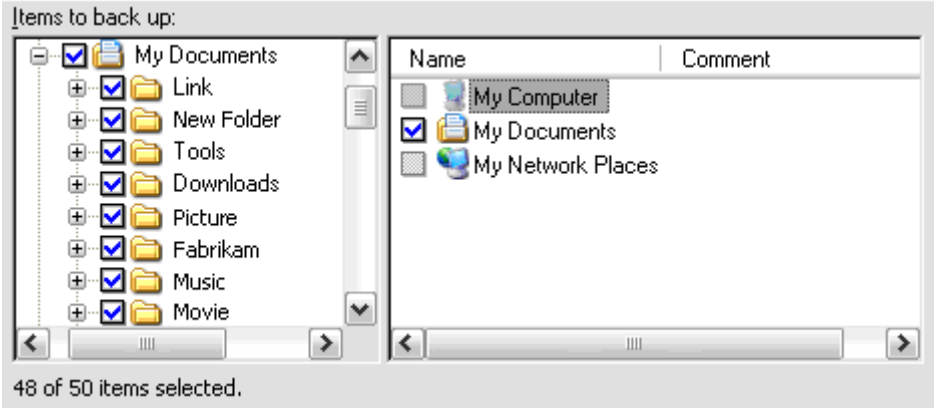
Tree organization

- **Use a natural hierarchical structure that's familiar to most users.**
- **If you can't use such a structure, try to balance discoverability with a predictable user model that minimizes confusion.**

- **To safely improve discoverability, place an item in multiple containers when:**
  - The item isn't related to any other similar items (so users don't become confused by incorrect associations).
  - There are only a few of such redundantly located items (so the tree doesn't become bloated).
- **Use the simplest hierarchical structure that works well.** To do so:
  - Place the most commonly accessed objects in the first two levels of the tree (not counting the root node), and place less commonly accessed objects farther down the hierarchy.
  - Eliminate unnecessary or combine redundant intermediate-level containers.
- **Prefer breadth over depth.** Ideally, a tree should have no more than four levels and the most commonly accessed objects should appear in the first two levels.
- **Determine if you really need a root node.** Provide a root node if users need the ability to perform commands on the entire tree (possibly using a context menu on the root node). Otherwise, the tree is simpler and easier to use without it.
- **If the tree has alternative access methods such as a word search or an index, optimize the tree for browsing by focusing on the most useful content.** With alternative access methods, the tree's content doesn't have to be comprehensive. Simplifying the tree makes it easier for users to find the most useful content.

Check box tree views

- **Display the number of selected items below the list**, especially if users are likely to select several items. This feedback helps users confirm that their selection is correct.

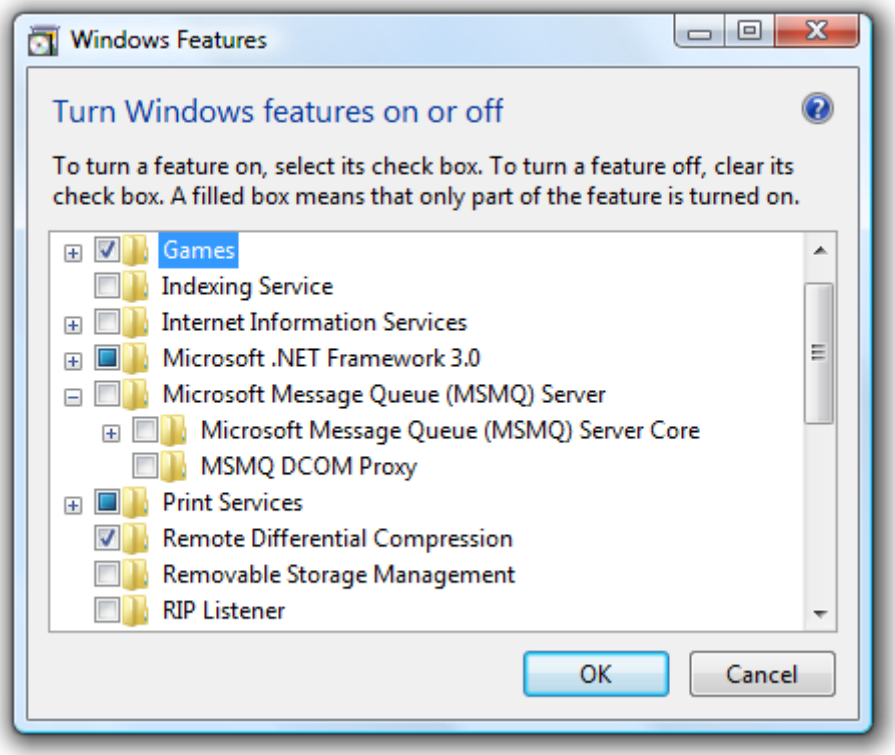


  In this example, the number of selected items is displayed below the tree. It is clear that two items are not selected.

- If there are potentially many items and selecting or clearing all of them is likely, add Select all and Clear all command buttons.

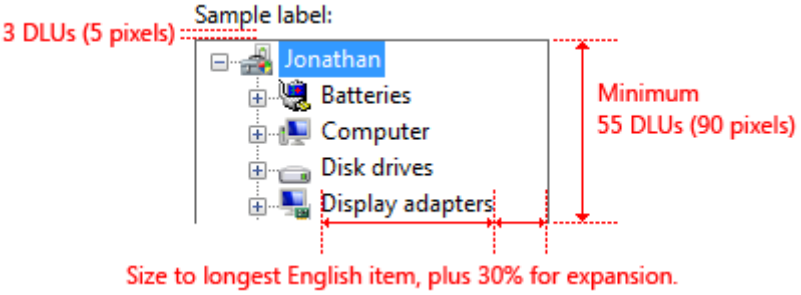
- **Use mixed-state check boxes to indicate partial selection of the items in a container.**

  Correct:



  In this example, the mixed-state check boxes are used to indicate partial selection.

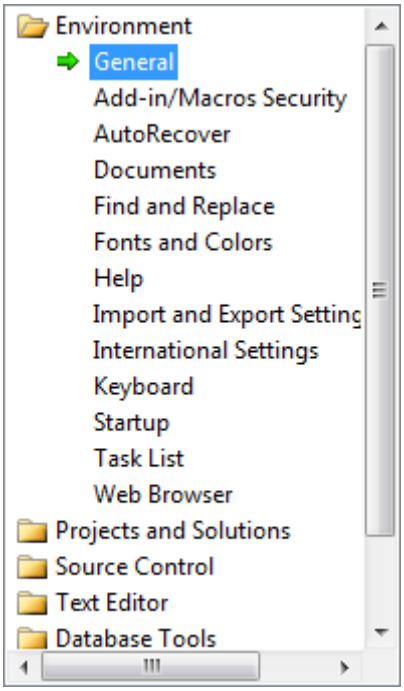
Recommended sizing and spacing



Recommended sizing and spacing for tree view controls.

- **Choose a tree view width that avoids the need for horizontal scrolling** for most items when the tree is fully expanded.
- **Include an additional 30 percent to accommodate localization.**
- **Choose a tree view height that eliminates unnecessary vertical scrolling.** Consider making a tree view slightly longer (or even more so if there is available space) if doing so reduces the need for a vertical scroll bar.

Incorrect:



In this example, making the tree view slightly wider and longer would eliminate the scroll bars in most cases. In this particular tree, only one container can be opened at a time.

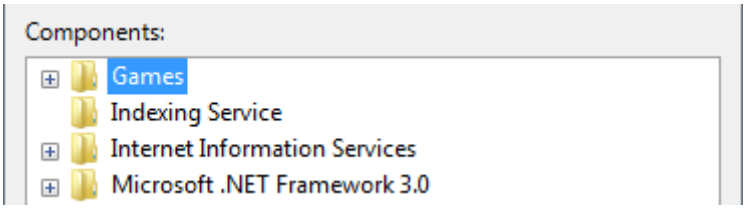
- **If users benefit from making the tree view larger, make the tree view and its parent window resizable.** Doing so allows users to adjust the tree view size as needed.

Labels

Control labels

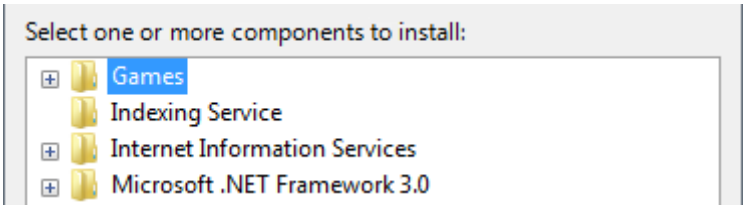
- All tree views need labels. Write the label as a word or phrase, not as a sentence, ending with a colon, and using static text.
- Assign a unique access key. For assignment guidelines, see Keyboard.
- Use sentence-style capitalization.
- Position the label above the control, and align the label with the left edge of the control.
- For multiple-selection tree views, write the label so that it's clear that multiple selection is possible. Check box tree view labels can be less explicit.

  Incorrect:



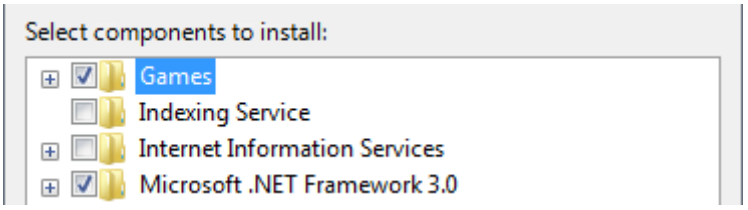
In this example, the label provides no information about multiple selection.

Better:



In this example, the label clearly indicates that multiple selection is possible.

Best:



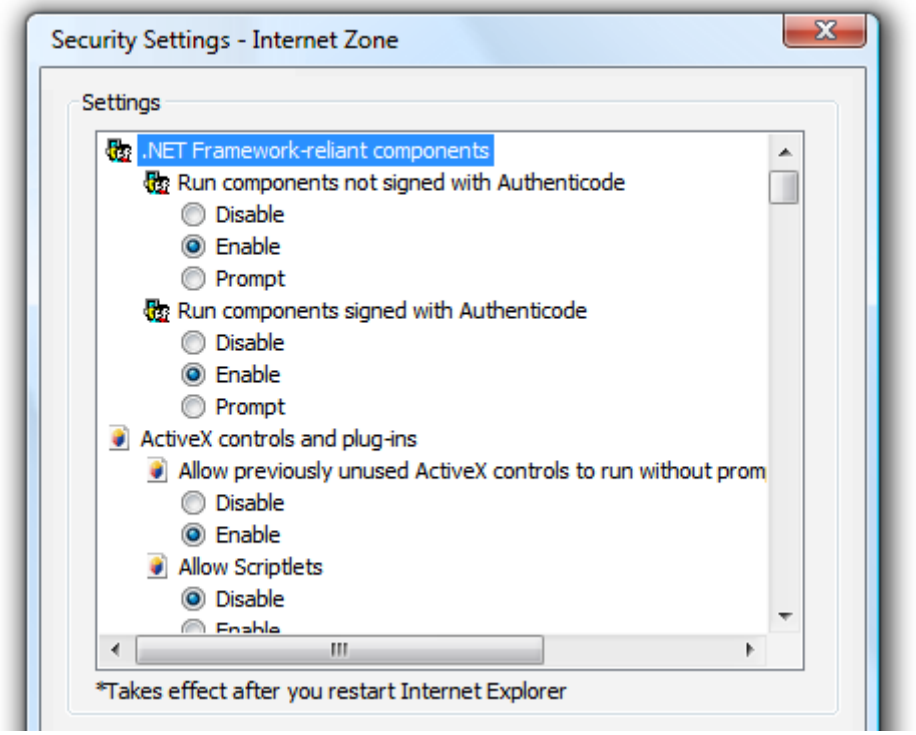
In this example, the check boxes clearly indicate that multiple selection is possible, so the label doesn't have to be explicit.

Data text

- Use sentence-style capitalization.

Instructional text

- If you need to add instructional text about a tree view, add it above the label. Use complete sentences with ending punctuation.
- Use sentence-style capitalization.
- Supplemental explanations that are helpful but not necessary should be kept short. Place this information either in parentheses between the label and colon, after the main instruction if used instead of a label, or below the control.



In this example, the supplemental explanation is below the control.

Documentation

When referring to tree views:

- Use the exact label text, including its capitalization, but don't include the access key underscore or colon. Include the word list or hierarchical list if the context requires making a distinction from regular lists.
- For tree items, use the exact item text, including its capitalization.
- Refer to tree views as tree views only in programming and other technical documentation. Everywhere else, use list or hierarchical list because the term tree is confusing to most users.
- To describe user interaction, use select for the data, and expand and collapse for the plus and minus buttons.
- When possible, format the label and tree items using bold text. Otherwise, put the label and items in quotation marks only if required to prevent confusion.

Example: In the **Contents** list, select **User Interface Design**.

When referring to check boxes in a tree view:

- Use the exact label text including its capitalization, and include the words check box. Don't include the access key underscore.
- To describe user interaction, use select and clear.

- When possible, format the label using bold text. Otherwise, put the label in quotation marks only if required to prevent confusion.

Example: In the **Items to back up** list, select the **My Documents** check box.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Commands (Design basics)

Article • 02/09/2021

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our **current design guidance**.

Commands are actions users can take while using your app. Learn the guidelines for adding commands to your app's menus, ribbons, and toolbars.

In this section

Topic	Description
Menus	Menus are hierarchical lists of commands or options available to users in the current context.
Ribbons	Ribbons are the modern way to help users find, understand, and use commands efficiently and directly with a minimum number of clicks, with less need to resort to trial-and-error, and without having to refer to Help.
Toolbars	Toolbars are a way to group commands for efficient access.

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

Windows 7 Menus (Design basics)

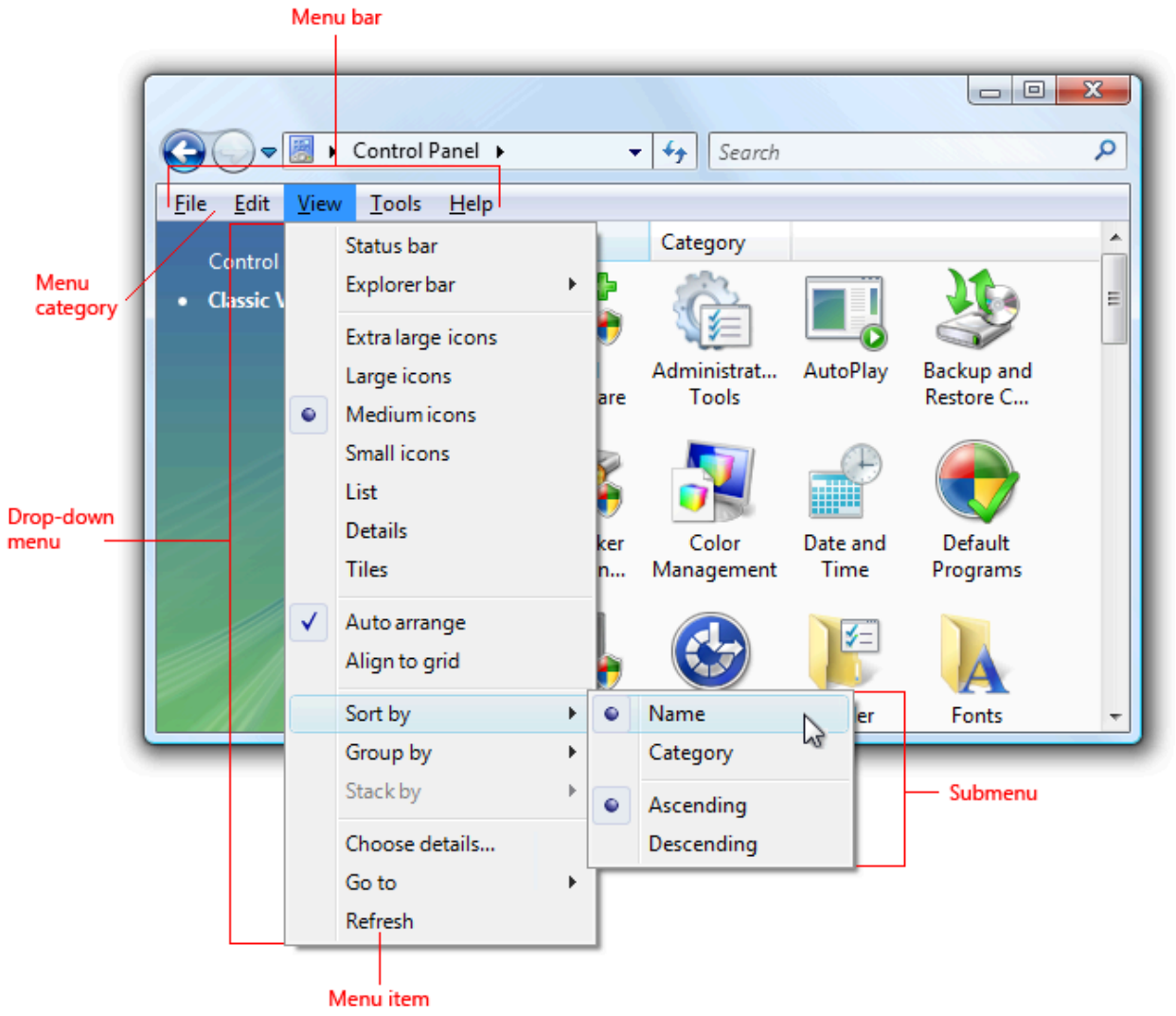
Article • 02/08/2022

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our **current design guidance**.

Menus are hierarchical lists of commands or options available to users in the current context.

Drop-down menus are menus displayed on demand on mouse click or hover. They are normally hidden from view and therefore are an efficient means of conserving screen space. A submenu or cascading menu is a secondary menu displayed on demand from within a menu. They are indicated by an arrow at the end of the submenu label. A menu item is an individual command or option within a menu.

Menus are often displayed from a menu bar, which is a list of labeled menu categories typically located near the top of a window. By contrast, a context menu drops down when users right-click on an object or window region that supports a context menu.



A typical menu bar displaying a drop-down menu and submenu.

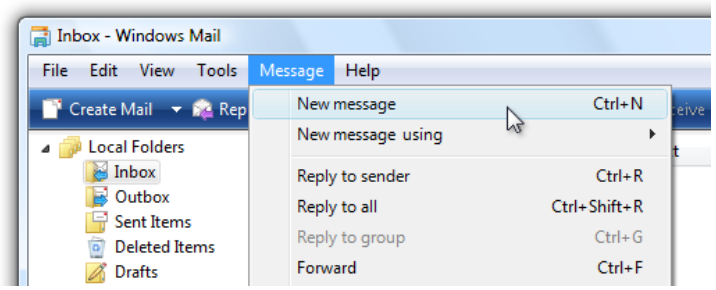
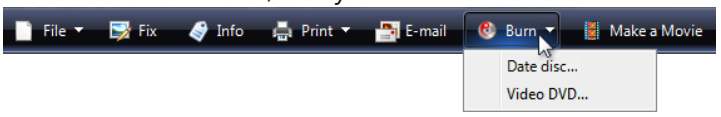
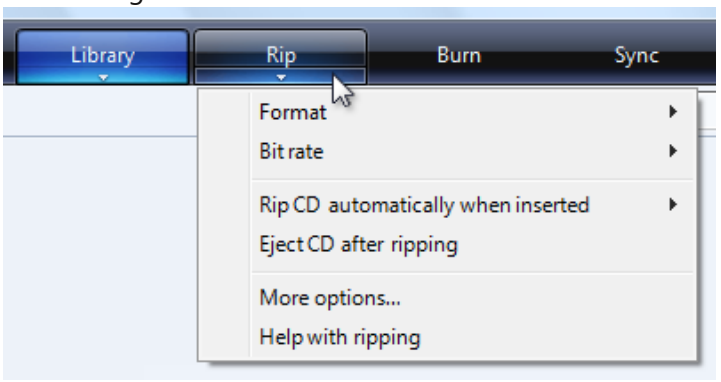
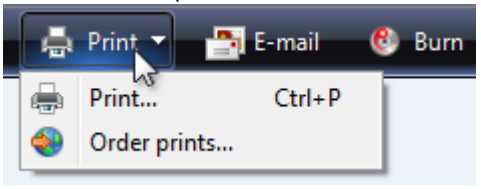
> **Note**
>
> Guidelines related to command buttons, Toolbars, and keyboard are presented in separate articles.

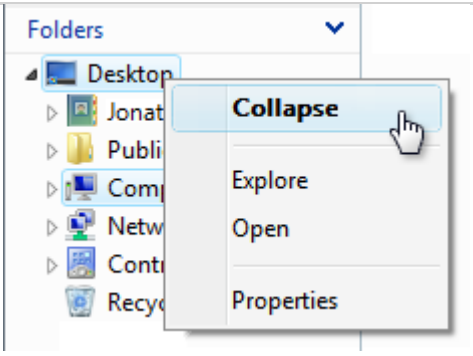
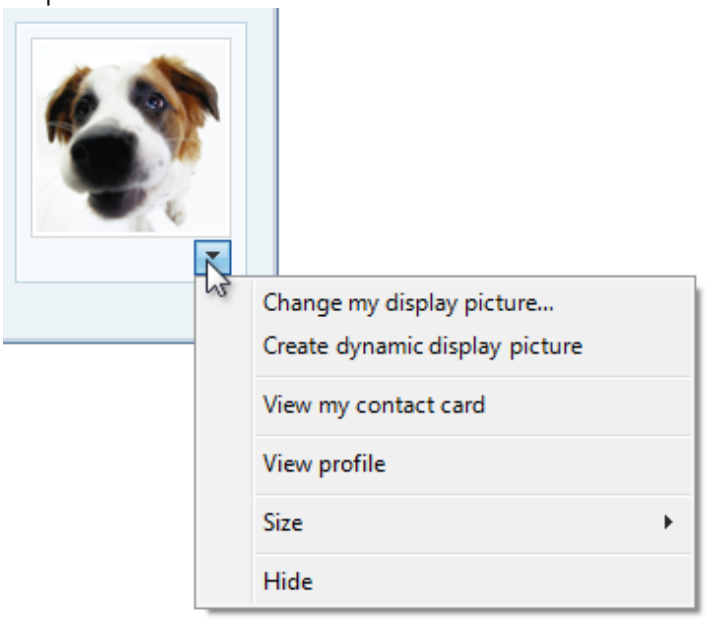
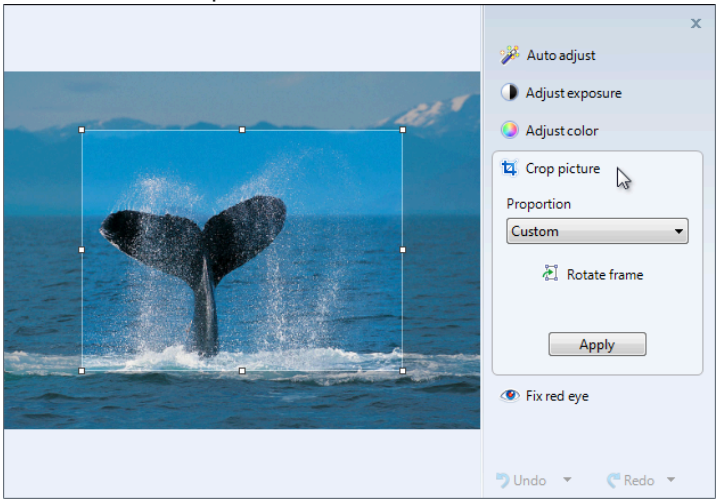
Usage patterns

Menus have several usage patterns:

Expand table

Usage	Example
Menu bars a menu bar displays commands and options in drop-down menus.	menu bars are very common and easy to find, as well as an efficient use of space.

Usage	Example
	 A menu bar from Windows Mail.
Toolbar menus a menu bar implemented as a toolbar.	toolbar menus are toolbars consisting primarily of commands in menu buttons and split buttons, with only a few direct commands, if any.  A toolbar menu in Windows Photo Gallery. For guidelines on this pattern, see Toolbars.
Tab menus buttons within tabs that display a small set of commands and options related to a tab in a drop-down menu.	tabs with menus look like ordinary tabs except their bottom portion has a button with drop-down arrow. clicking the button displays a drop-down menu instead of selecting the tab.  Tab menus are used in Windows Media Player.
Menu buttons command buttons that display a small set of related commands in a drop-down menu.	menu buttons look like ordinary command buttons except they have a drop-down arrow within them. clicking the button displays a drop-down menu instead of performing a command. split buttons are similar to menu buttons except that they are variations of a command, and clicking the left portion of the button performs the action on the label directly.  A menu button with a small set of related commands.
Context menus drop-down menus that display a	context menus drop-down when users right-click on an object or window region that supports a context menu.

Usage	Example
small set of commands and options related to the current context.	 a context menu from windows explorer. if context menus are the best menu choice but you need a solution suitable for all users, you can use a menu drop-down arrow button.  A context menu made visible with a menu drop-down button.
Task pane menus a small set of commands related to the selected object or program mode.	unlike context menus, they are displayed automatically within a window pane, instead of on demand.  A task pane menu from the Windows Photo Gallery viewer.

Is this the right user interface?

To decide, consider these questions:

Menu bars

Do the following conditions apply:

- Is the window a primary window?
- Are there many menu items?
- Are there many menu categories?
- Do the majority of the menu items apply to the entire program and primary window?
- Does the menu need to work for all users?

If so, consider using a menu bar.

Toolbar menus

Do the following conditions apply:

- Is the window a primary window?
- Does the window have a toolbar?
- Are there only a few menu categories?
- Does the menu need to work for all users?

If so, consider using a toolbar menu instead of or in addition to a menu bar.

Tab menus

Do the following conditions apply:

- Is the window a primary window?
- Does the window have tabs, where each tab is used for a dedicated set of tasks (as opposed to using tabs to show different views)?
- Is there one menu category that applies to each tab?
- Are there many commands and options, but only a small set for each tab?

If so, consider using a tab menu instead of a menu bar.

Context menu

Do the following conditions apply:

- Is there a small set of contextual commands and options that apply to the selected object or window region?
- Are these menu items redundant?
- Are the target users familiar with context menus?

If so, consider providing context menus for the objects and window regions that need them.

For browser-based programs, task pane menus are a more common solution for contextual commands. Currently, users expect context menus in browser-based programs to be generic and unhelpful.

Task pane menu

Do the following conditions apply:

- Is the window a primary window?
- Is there a small set of contextual commands and options that apply to the selected object or program mode?
- Are there a few menu categories?
- Does the menu need to work for all users?

If so, consider using a task pane menu instead of a context menu.

Design concepts

Effective menus that promote a good user experience:

- Use a command presentation that matches your program type, window types, command usage, and target users.
- Are well organized, using standard menu organization when appropriate.
- Use menu bars, toolbars, and context menus effectively.
- Use icons effectively.
- Use access keys and shortcut keys effectively.

If you do only one thing...

Choose a command presentation that matches your program type, window types, command usage, and target users.

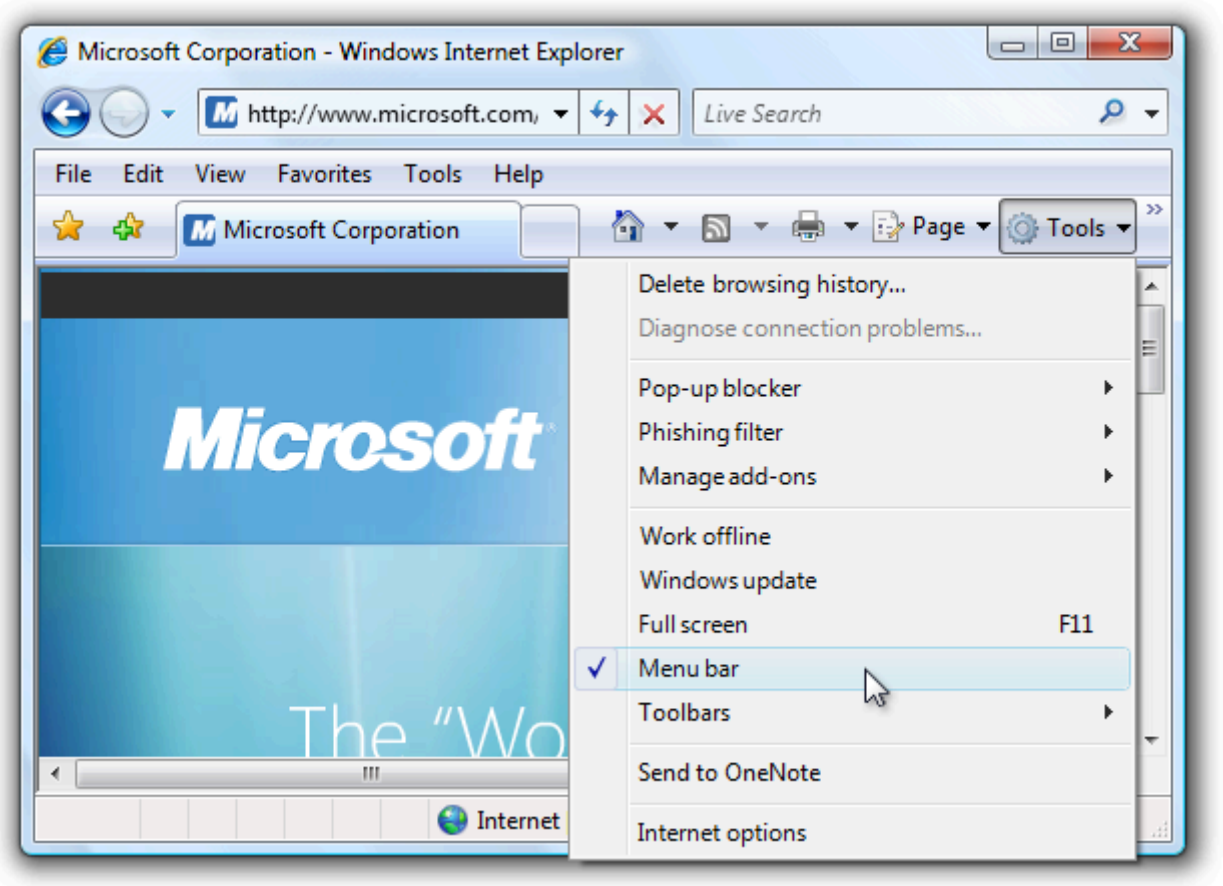
Guidelines

General

- **All menu patterns except menu bars need a drop-down arrow to indicate the presence of a pull-down menu.** The presence of menus goes without saying in a menu bar, but not in the other patterns.
- **Don't change menu item names dynamically.** Doing so is confusing and unexpected. For example, don't change a Portrait mode option to Landscape mode upon selection. For modes, use bullets and checkmarks instead.
  - **Exception:** You can change menu item names that are based on object names dynamically. For example, lists of recently used files or window names can be dynamic.

Menu bars

- **Consider eliminating menu bars with three or fewer menu categories.** If there are only a few commands, prefer lighter alternatives such as toolbar menus, or more direct alternatives such as command buttons and links.
- **Don't have more than 10 menu categories.** Too many menu categories is overwhelming and makes the menu bar difficult to use.
- **Consider hiding the menu bar** if the toolbar or direct commands provide almost all of the commands needed by most users. Allow users to show or hide with a Menu bar check mark option in a toolbar menu.



In this example, Windows Internet Explorer provides a menu bar option.

For more information, see hiding menu bars.

Hiding menu bars

Generally, toolbars work great together with menu bars because having both allows each to focus on their strengths without compromise.

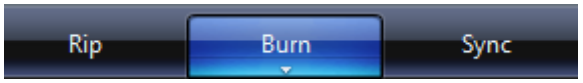
- Hide the menu bar by default if your toolbar design makes having a menu bar redundant.
- Hide the menu bar instead of removing it completely, because menu bars are more accessible for keyboard users.
- To restore the menu bar, provide a Menu bar checkmark option in the View (for primary toolbars) or Tools (for secondary toolbars) menu category. For more information, see Standard menu and split buttons.

Menu categories

- **Choose single word names for menu categories.** Using multiple words makes the separation between categories confusing.
- **For programs that create or view documents, use the standard menu categories** such as File, Edit, View, Tools, and Help. Doing so makes common menu items

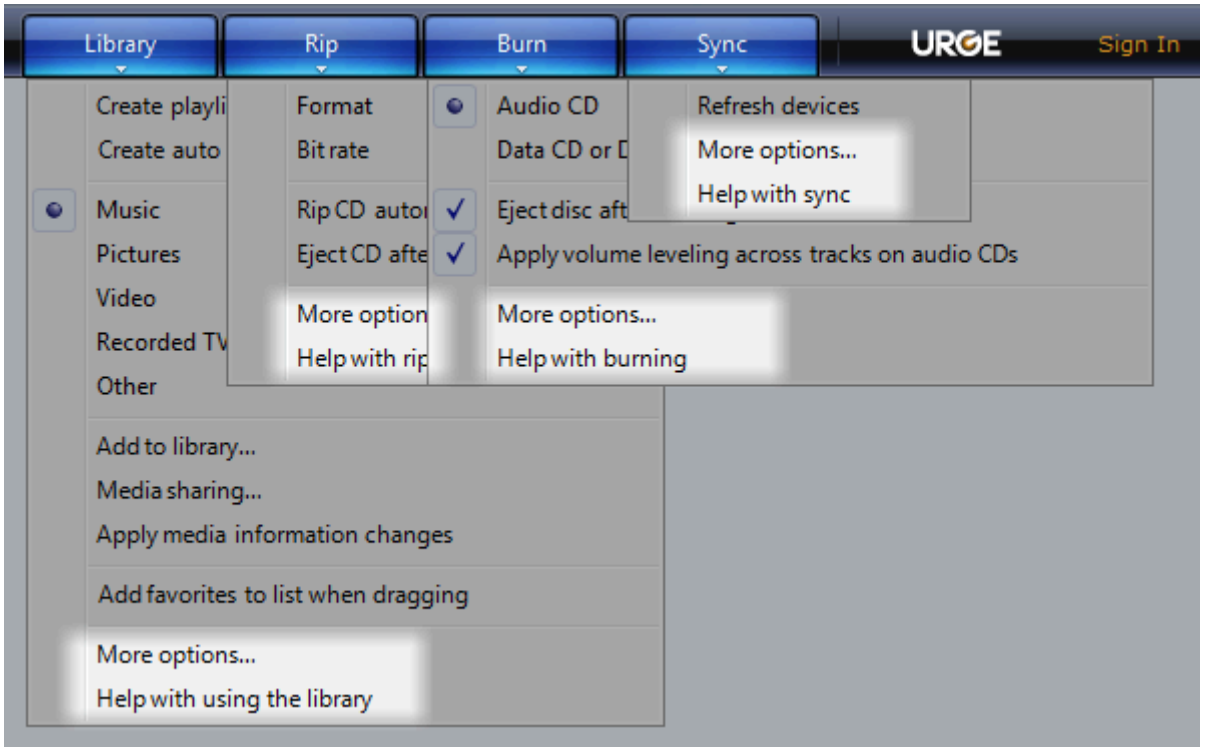
predictable and easier to find.

- **For other types of programs, consider organizing your commands and options into more useful, natural categories** based on your program's purpose and the way users think about their tasks and goals. Don't feel obligated to use the standard menu organization if it isn't suitable for your program.
- **If you choose to use non-standard menu categories, you must choose good category names.** For more information, see the Labels section.
- **Prefer task-oriented menu categories over generic categories.** Task-oriented categories make menu items easier to find.



In this example, Windows Media Player uses task-oriented menu categories.

- **Avoid menu categories with only one or two menu items.** If sensible, consolidate with other menu categories, perhaps using a submenu.
- **Consider putting the same menu item in multiple categories only if:**
  - The menu item logically belongs in multiple menu categories.
  - You have data showing that users have trouble finding the item in a single menu category.
  - You have only one or two hard-to-find menu items in multiple categories.
- **Don't put different menu items that use the same name in multiple categories.** For example, don't have different Options menu items in multiple categories.
  - **Exception:** The tab menu pattern may have different Options and Help menu items in each tab menu.



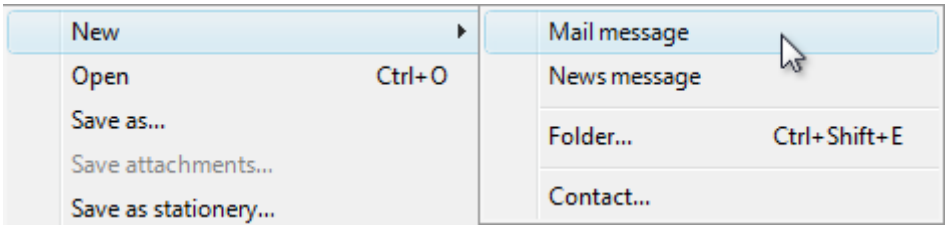
In this example, Windows Media Player has Options and Help menu items in each tab menu.

Menu item organization and order

- **Organize the menu items into groups of seven or fewer strongly related items.** For this, submenus count as a single menu item in the parent menu.
- **Don't put more than 25 items within a single level of a menu** (not counting submenus).
- **Put separators between the groups within a menu.** A separator is a single line that spans the width of the menu.
- **Within a menu, put the groups in their logical order.** If there is no logical order, place the most commonly used groups first.
- **Within a group, put the items in their logical order.** If there is no logical order, place the most commonly used items first. Put numeric items (such as zoom percentages) in numeric order.

Submenus

- **Avoid using submenus unnecessarily.** Submenus require more physical effort to use and generally make the menu items more difficult to locate.
- **Don't put frequently used menu items in a submenu.** Doing so would make using these commands inefficient. However, you can put frequently used commands in a submenu if they are normally accessed more directly, such as with a toolbar.
- **Consider using a submenu if:**
  - Doing so simplifies the parent menu because it has many items (20 or more), or the submenu is part of a group of more than seven items.
  - The items in the submenu are used less frequently than those in the parent menu.
  - The submenu would have three or more items.
  - There are three or more commands that begin with the same word. In this case, use that word as the submenu label.



In this example, the New submenu replaces separate commands for New mail message, New news message, New folder, and New contact.

- **Use at most three levels of menus.** That is, you can have a primary menu and at most two levels of submenus. Two levels of submenus should be rare.

Presentation

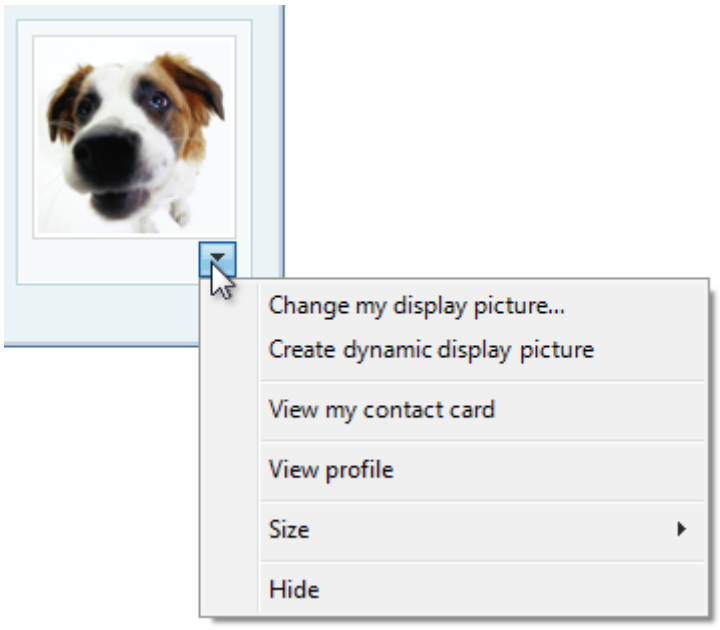
- **Disable menu items that don't apply to the current context,** instead of removing them. Doing so makes menu bar contents stable and easier to find. **Exceptions:**
  - For contextual menu categories, **remove rather than disable context menu items that don't apply to the current context.** A menu category is contextual when it is displayed only for specific modes, such as when a certain object type is selected. For details, see the remove vs. disable guidelines for context menus.
  - If determining when a menu item should be disabled causes noticeable performance problems, leave the menu item active and if necessary have its selection result in an error message.

Tab menus

- **Each tab menu may have context specific Options and Help menu items.** This is in contrast to all other menu patterns. Each tab is used for a dedicated set of tasks, so any redundancy across tab menus isn't confusing.

Context menus

- **Use context menus only for contextual commands and options.** The menu items should apply only to the selected (or clicked upon) object or window region, not the entire program.
- **Don't make commands only available through context menus.** Like shortcut keys, context menus are alternative means of performing commands and choosing options. For example, a Properties command is also available through the menu bar or the Alt+Enter access key.
- **Provide context menus for all objects and window regions** that benefit from a small set of contextual commands and options. Many users right-click regularly and expect to find context menus anywhere.
- **Consider using a menu drop-down arrow button for context menus targeted at all users.** Normally context menus are suitable for commands and options targeted at advanced users. However, you can use a menu drop-down button in cases where context menus are the best menu choice and you need to target all users.



In this example, a menu drop-down button is used to make a context menu visible.

Menu item organization and order

- **Organize the menu items into groups of seven or fewer strongly related items.**
- **Avoid using submenus** to keep context menus simple, direct, and efficient.
- **Don't put more than 15 items within a context menu.**
- **Put separators between the groups within a menu.** A separator is a single line that spans the width of the menu.
- **Present menu items using the following order:**

Primary (most frequently used) commands
Open Run Play Print <separator>
Secondary commands supported by the object
<separator>
Transfer commands
Cut Copy Paste <separator>
Object settings
<separator>
Object commands
Delete Rename <separator> Properties

Presentation

- **Display the default command using bold.** When practical, also make it the first menu item. The default command is invoked when users double-click or select an object and press Enter.
- **Remove rather than disable context menu items that don't apply to the current context.** Doing so makes context menus contextual and efficient.

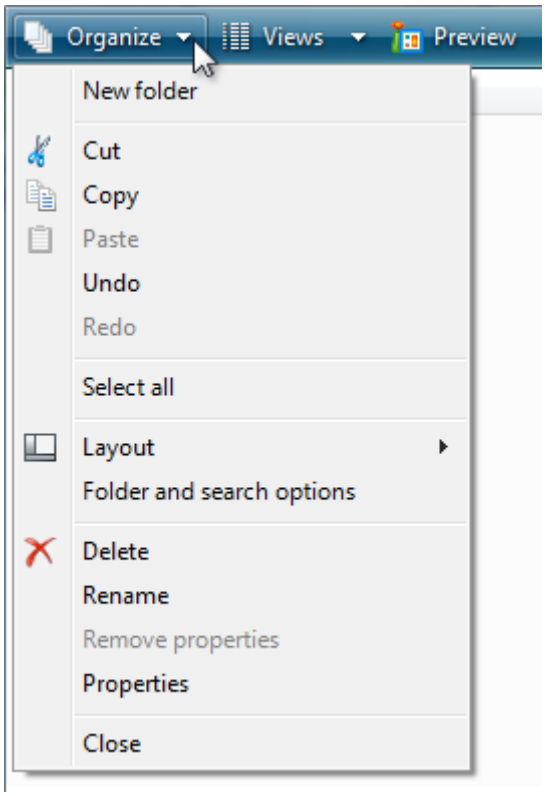
    - **Exception:** Disable menu items that don't apply if there is a reasonable expectation for them to be available:
        - Always have the relevant standard context menu commands, such as Cut, Copy, Paste, Delete, and Rename.
        - Always have the commands that complete related sets. For example, if there is a Back, there should also be a Forward. If there's a Cut, always have a Copy and Paste.

Bullets and checkmarks

- **Menu items that are options may use bullets and checkmarks.** Commands may not.
- **Use a bullet to choose one option from a small set of mutually exclusive choices.** There should always be at least two bullets in a group. For more information, see Radio buttons.
- **Use a checkmark to toggle an independent setting on or off.** If the selected and cleared states aren't clear and unambiguous opposites, use a set of bullets instead. For more information, see Check boxes.
- **For a mixed checkmark state, display a menu item without a checkmark.** The mixed state is used for multiple selection to indicate that the option is set for some, but not all, objects, so each individual object has either the selected or cleared state. The mixed state is not used as a third state for an individual item.
- **Put separators between the related sets of checkmarks or bullets.** A separator is a single line that spans the width of the menu.

Icons

- **Consider providing menu item icons for:**
    - The most commonly used menu items.
    - Menu items whose icon is standard and well known.
    - Menu items whose icon well illustrates what the command does.
- **If you use icons, don't feel obligated to provide them for all menu items.** Cryptic icons aren't helpful, create visual clutter, and prevent users from focusing on the important menu items.



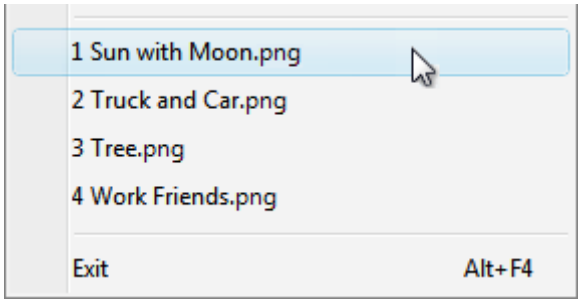
In this example, the Organize menu has icons only for the most commonly used menu items.

- **Make sure menu icons conform to the Aero-style icon guidelines.**

For more information and examples, see Icons.

Access keys

- **Assign access keys to all menu items.** No exceptions.
- **Whenever possible, assign access keys for commonly used commands according to the Standard Access Key Assignments.** While consistent access key assignments aren't always possible, they are certainly preferred - especially for frequently used commands.
- **For dynamic menu items (such as recently used files), assign access keys numerically.**



In this example, the Paint program in Windows assigns numeric access keys to recently used files.

- **Assign unique access keys within a menu level.** You can reuse access keys across different menu levels.
- **Make access keys easy to find:**
  - For the most frequently used menu items, choose characters at the beginning of the first or second word of the label, preferably the first character.
  - For less frequently used menu items, choose letters that are a distinctive consonant or a vowel in the label.
- **Prefer characters with wide widths,** such as w, m, and capital letters.
- **Prefer a distinctive consonant or a vowel,** such as "x" in "Exit."
- **Avoid using characters that make the underline difficult to see,** such as (from most problematic to least problematic):
  - Letters that are only one pixel wide, such as i and l.
  - Letters with descenders, such as g, j, p, q, and y.
  - Letters next to a letter with a descender.

For more guidelines and examples, see Keyboard.

Shortcut keys

- **Assign shortcut keys to the most frequently used menu items.** Infrequently used menu items don't need shortcut keys because users can use access keys instead.
- **Don't make a shortcut key the only way to perform a task.** Users should also be able to use the mouse or the keyboard with Tab, arrow, and access keys.
- **For well-known shortcut keys, use the standard assignments.**
- **Don't assign different meanings to well-known shortcut keys.** Because they are memorized, inconsistent meanings for well-known shortcuts are frustrating and error prone. See Windows Keyboard Shortcut Keys for the well-know shortcut keys used by Windows programs.
- **Don't try to assign system-wide program shortcut keys.** Your program's shortcut keys will have effect only when your program has input focus.
- **Document all shortcut keys.** Doing so helps users learn the shortcut key assignments.
  - **Exception:** Don't display shortcut key assignments within context menus. Context menus don't display the shortcut key assignments because they are optimized for efficiency.
- **For non-standard key assignments:**
  - **Choose shortcut keys that don't have standard assignments.** Never reassign standard shortcut keys.
  - **Use non-standard key assignments consistently throughout your program.** Don't assign different meanings in different windows.

    - **If possible, choose mnemonic key assignments,** especially for frequently used commands.
    - **Use function keys for commands that have a small-scale effect,** such as commands that apply to the selected object. For example, F2 renames the selected item.
    - **Use Ctrl key combinations for commands that have a large-scale effect,** such as commands that apply to an entire document. For example, Ctrl+S saves the current document.
    - **Use Shift key combinations for commands that extend or complement the actions of the standard shortcut key.** For example, the Alt+Tab shortcut key cycles through open primary windows, whereas Alt+Shift+Tab cycles in the reverse order. Similarly, F1 displays Help, whereas Shift+F1 display context-sensitive Help.
    - **Don't use the following characters for shortcut keys:** @ $ {} [] \ ~ | ^ ' < >. These characters require different key combinations across languages or are locale specific.
    - **Don't use Ctrl+Alt combinations,** because Windows interprets this combination in some language versions as an AltGR key, which generates alphanumeric characters.
- **If your program assigns many shortcut keys, provide the ability to customize the assignments.** Doing so allows users to reassign conflicting shortcut keys and migrate from other products. Most programs don't assign enough shortcut keys to need this feature.

For more guidelines and standard shortcut key assignments, see Keyboard.

Standard menus

- **Use the standard menu organization for programs that create or view documents.** The standard menu organization makes common menu items predictable and easier to find.
- **For other types of programs, use the standard menu organization only when it makes sense to.** Consider organizing your commands and options into more useful, natural categories based on your program's purpose and the way users think about their tasks and goals.

Standard menu bars

The standard menu bar structure is as follows. This list shows the menu category and item labels, their order with separators, their access and shortcut keys, and their ellipses.

File

New Ctrl+N
Open... Ctrl+O
Close Save Ctrl+S
Save as... Send to Print... Ctrl+P
Print preview
Page setup

1 `<filename>`
2 `<filename>`
3 `<filename>`
...

Exit Alt+F4 (shortcut usually not given)

Edit

Undo Ctrl+Z
Redo Ctrl+Y
<separator>
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
<separator>
Select all Ctrl+A
<separator>
Delete Del (shortcut usually not given)
<separator>
Find... Ctrl+F
Find next F3 (command usually not given)
Replace... Ctrl+H
Go to... Ctrl+G

View

Toolbars
Status bar
<separator>
Zoom Zoom in Ctrl++
Zoom out Ctrl+-
<separator>
Full screen F11
Refresh F5

Tools

<separator>
Options Help `<program name>` help F1
<separator>
About `<program name>`

Standard toolbar menu buttons

The standard toolbar menu buttons are as follows. This list shows the menu category and item labels, their order with separators, their shortcut keys, and their ellipses.
Tools
Full screenF11(Reassign access key if Find is also used.) Toolbars(Note that the Menu bar command goes here.) <separator> Print... Find... <separator> Zoom Text size <separator> Options
Organize
New folderCtrl+N <separator> CutCtrl+X CopyCtrl+C PasteCtrl+V <separator> Select allCtrl+A <separator> DeleteDel(shortcut usually not given) Rename <separator> Options
Page
New windowCtrl+N <separator> Zoom Text size

Standard context menus

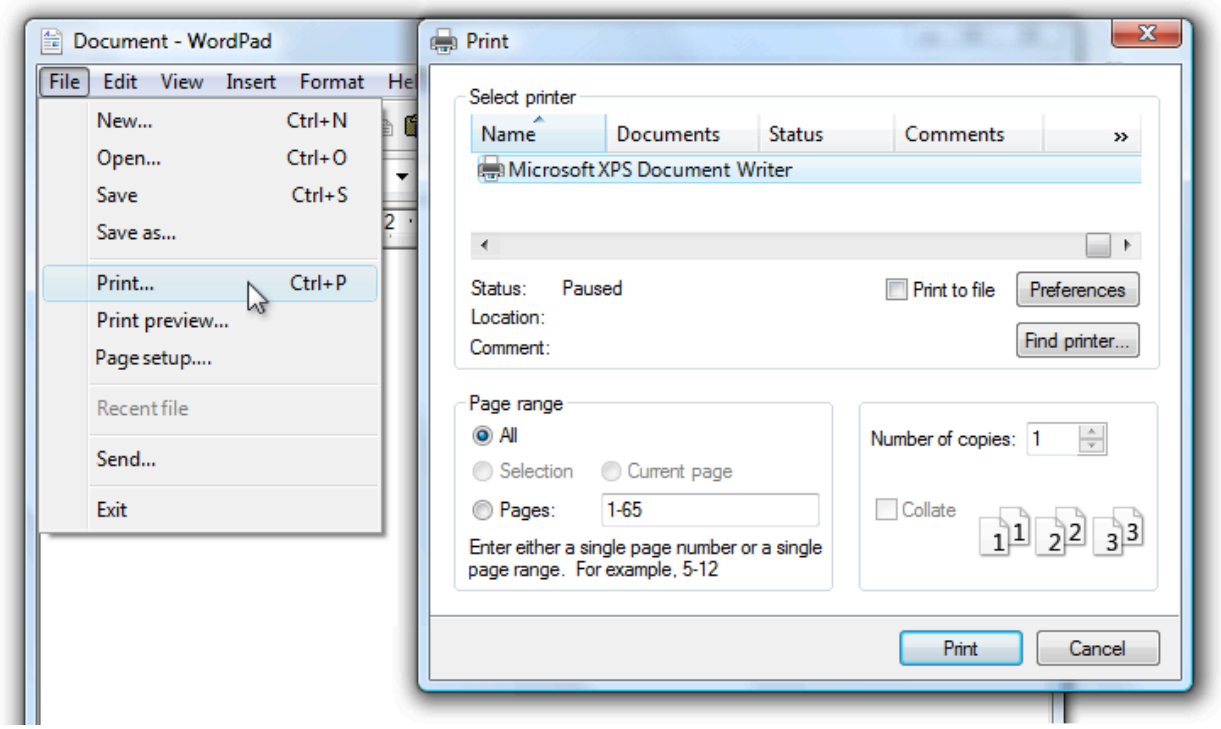
The standard context menu contents are as follows. This list shows the menu item labels, their order with separators, their access keys, and their ellipses. Context menus don't show shortcut keys.
Open Run Play Edit Print... <separator> Cut Copy Paste <separator> Delete Rename <separator>

Lock the `<object name>(checkmark)`

Properties

Using ellipses

While menu commands are used for immediate actions, more information might be needed to perform the action. **Indicate a command that needs additional information (including a confirmation) by adding an ellipsis at the end of the label.**



In this example, the Print... command displays a Print dialog box to gather more information.

Proper use of ellipses is important to indicate that users can make further choices before performing the action, or even cancel the action entirely. The visual cue offered by an ellipsis allows users to explore your software without fear.

This doesn't mean you should use an ellipsis whenever an action displays another window only when additional information is required to perform the action. For example, the commands About, Advanced, Help, Options, Properties, and Settings must display another window when clicked, but don't require additional information from the user. Therefore they don't need ellipses.

In case of ambiguity (for example, the command label lacks a verb), decide based on the most likely user action. If simply viewing the window is a common action, don't use an ellipsis.

Correct:

More colors...

Version information

In the first example, users are most likely going to choose a color, so using an ellipses is correct. In the second example, users are most likely going to view the version information, making ellipses unnecessary.

> ⓘ Note

When determining if a menu command needs an ellipsis, don't use the need to elevate privileges as a factor.

Elevation isn't information needed to perform a command (rather, it's for permission) and the need to elevate is indicated with the security shield.

Labels

- **Use sentence-style capitalization.**
  - **Exception:** For legacy applications, you may use title-style capitalization if necessary to avoid mixing capitalization styles.

Menu category names

- **Use menu category names that are single word verbs or nouns.** A multiple-word label might be confused for two one-word labels.
- **Prefer verb-based menu names.** However, omit the verb if it is Create, Show, View, or Manage. For example, the following menu categories don't have verbs:
  - Table
  - Tools
  - Window
- For non-standard category names, **use a single, specific word that clearly and accurately describes the menu contents.** While the names don't have to be so general that they describe everything in the menu, they should be predictable enough so that users aren't surprised by what they find in the menu.

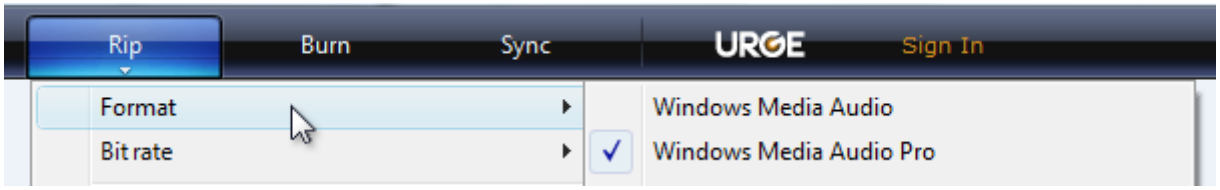
Menu item names

- **Use menu item names that start with a verb, noun, or noun phrase.**
- **Prefer verb-based menu names.** However, omit the verb if:
  - **The verb is Create, Show, View, or Manage.** For example, the following commands don't have verbs:
    - About
    - Advanced
    - Full screen
    - New
    - Options
    - Properties
  - **The verb is the same as the menu category name to avoid repetition.** For example, in the Insert menu category, use Text, Table, and Picture instead of

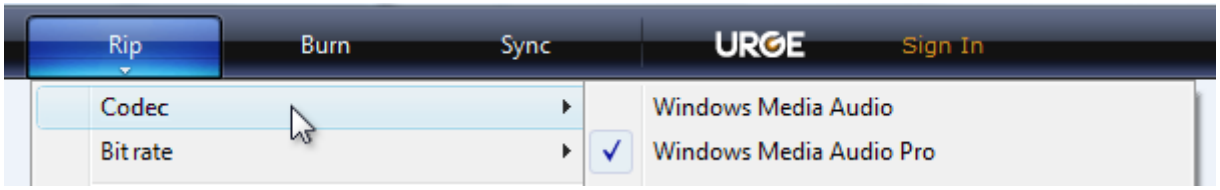
Insert text, Insert table, and Insert picture.

- **Use specific verbs.** Avoid generic, unhelpful verbs, such as Change and Manage.
- **Use singular nouns for commands that apply to a single object**, otherwise use plural nouns.
- **Use modifiers as necessary to distinguish between similar commands.** Examples: Insert row above, Insert row below.
- **For pairs of complementary commands, choose clearly complementary names.** Examples: Add, Remove; Show, Hide; Insert, Delete.
- **Choose menu item names based on user goals and tasks, not on technology.**

Correct:



Incorrect:



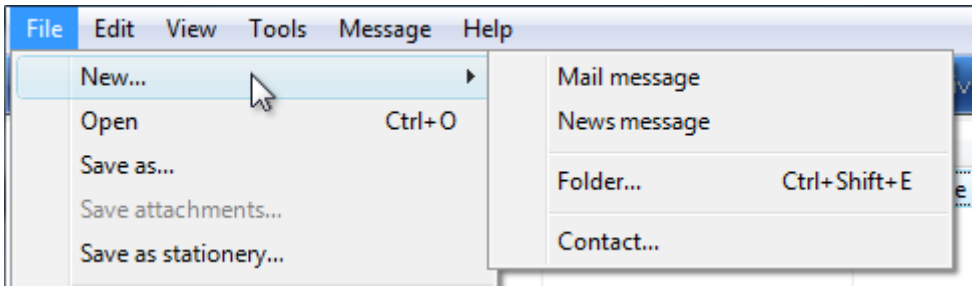
In the incorrect example, the menu item is based on its technology.

- Use the following menu item names for the stated purpose:
  - **Options** To display program options.
  - **Customize** To display the program options specifically related to mechanical UI configuration.
  - **Personalize** To display a summary of commonly used personalization settings.
  - **Preferences** Don't use. Use Options instead.
  - **Properties** To display an object's property window.
  - **Settings** Don't use as a menu label. Use Options instead.

Submenu names

- **Menu items that display submenus never have an ellipsis on their label.** The submenu arrow indicates that another selection is required.

Incorrect:



In this example, the New menu item incorrectly has an ellipsis.

Documentation

When referring to menus:

- In commands that show or hide menus, refer to menu bars. Don't refer to them as classic menus.
- Refer to menus by their labels. Use the exact label text, including its capitalization, but don't include the access key underscore or ellipsis.
- To refer to menu categories, use "On the `<category name>` menu." If the location of a menu item is clear from the context, you don't need to mention the menu category.
- To describe user interaction of menu items, use click, without the word menu or command. Don't use choose, select, or pick. Don't refer to a menu item as a menu item except in technical documentation.
- To describe removing a check mark from a menu option, use click to remove the check mark. Don't use clear.
- Refer to context menus as context menus, not shortcut menus.
- Don't use cascading, pull-down, drop-down, or pop-up to describe menus, except in programming documentation.
- Refer to unavailable menu items as unavailable, not as dimmed, disabled, or grayed. Use disabled in programming documentation.
- When possible, format the labels using bold text. Otherwise, put the labels in quotation marks only if required to prevent confusion.

Examples:

- On the **File** menu, click **Print** to print the document.
- On the **View** menu, point to **Toolbars**, and then click **Formatting**.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

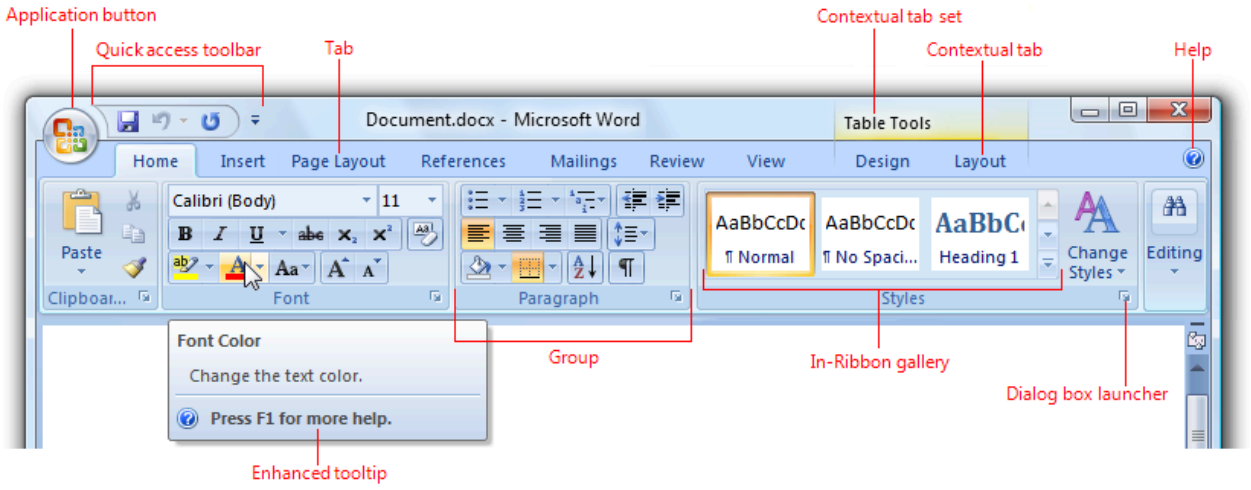
Windows 7 Ribbons

Article • 02/08/2022

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

Ribbons are the modern way to help users find, understand, and use commands efficiently and directly with a minimum number of clicks, with less need to resort to trial-and-error, and without having to refer to Help.

A ribbon is a command bar that organizes a program's features into a series of tabs at the top of a window. Using a ribbon increases discoverability of features and functions, enables quicker learning of the program as a whole, and makes users feel more in control of their experience with the program. A ribbon can replace both the traditional menu bar and toolbars.



A typical ribbon.

Ribbon tabs are composed of groups, which are a labeled set of closely related commands. In addition to tabs and groups, ribbons consist of:

- An Application button, which presents a menu of commands that involve doing something to or with a document or workspace, such as file-related commands.
- A Quick Access Toolbar, which is a small, customizable toolbar that displays frequently used commands.
- Core tabs are the tabs that are always displayed.

- Contextual tabs, which are displayed only when a particular object type is selected. Tabs that are always displayed are called core tabs.
- A tab set is a collection of contextual tabs for a single object type. Because objects can have multiple types (for example, a header in a table that has a picture is three types), there can be multiple contextual tab sets displayed at a time.
- Modal tabs, which are core tabs displayed with a particular temporary mode, such as print preview.
- Galleries, which are lists of commands or options presented graphically. A results-based gallery illustrates the effect of the commands or options instead of the commands themselves. An in-ribbon gallery is displayed within a ribbon, as opposed to a pop-up window.
- Enhanced tooltips, which concisely explain their associated commands and give the shortcut keys. They may also include graphics and references to Help. Enhanced tooltips reduce the need for command-related Help.
- Dialog box launchers, which are buttons at the bottom of some groups that open dialog boxes containing features related to the group.

Ribbons were originally introduced with Microsoft Office 2007. To learn why Office needs to use ribbons and the many problems using a ribbon solves, see The Story of the Ribbon.

> **Note**
>
> Guidelines related to menus, toolbars, command buttons, and icons are presented in separate articles.

Is this the right user interface?

To decide to use a ribbon, consider these questions:

Program type

- **What type of program are you designing?** The program type is a good indicator of the appropriateness of a ribbon. Ribbons work well for document creation and authoring programs, as well as document viewers and browsers. Ribbons might work for other types of programs, but other forms of command presentation may be more appropriate. Generally, lightweight programs should have a lightweight command presentation.

Discoverability and learning issues

- **Do users have trouble finding commands? Are users requesting features that are already in the program?** If so, using a ribbon will make commands easier to find by having self-explanatory labels and grouping of related commands. Using a ribbon also scales better than menu bars and toolbars for future growth.
- **Do users have trouble understanding the program's commands? Do they often resort to "trial and error" to select the right command or determine how commands work?** If so, using a ribbon with results-oriented commands based on galleries and live previews makes commands easier to understand.

Command characteristics

- **Are the commands presented in several locations? If your program already exists, are commands presented in menu bars, toolbars, task panes, and within the work area itself?** If so, using a ribbon will unify the commands into a single location, making them easier to find.
- **Do the commands apply to the entire window or only to specific panes?** Ribbons work best for commands that apply to the entire window or to specific objects. In-place commands work better for individual window panes.
- **Can most of the commands be presented directly? That is, can users interact with them using a single click? If commonly used commands are accessed from menus and dialog boxes, can they be refactored to be direct?** While some commands can be presented using menus and dialog boxes, presenting most commands this way undermines the efficiency of a ribbon, possibly making a menu bar a better choice.

Command scale

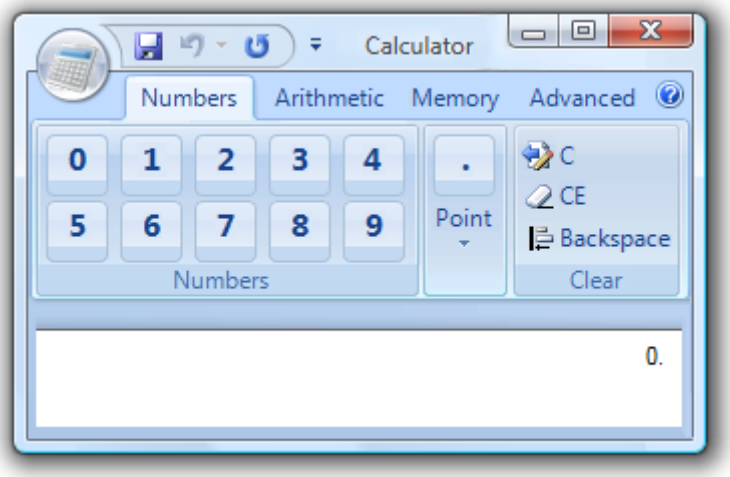
- **Is there a small number of commands? Can the most frequently used commands be presented easily on a single, simple toolbar?** Using a ribbon is worthwhile if adding core and contextual tabs results in a simple Home tab that can be used alone to perform the most common tasks. If not, the benefit of using a ribbon might not justify its extra weight for a small number of commands.
- **Is there a large number of commands? Would using a ribbon require more than seven core tabs? Would users constantly have to change tabs to perform common tasks?** If so, using toolbars (which don't require changing tabs) and palette windows (which may require changing tabs, but there can be several open at a time) might be a more efficient choice.
- **Do users tend to use a small number of commands most of the time?** If so, they can use a ribbon efficiently by putting such commands on the Home tab. Constantly changing tabs would make a ribbon too inefficient.

- **Does the program benefit from making the content area of the program as large as possible?** If so, using a menu bar and a single toolbar is more space efficient than a ribbon. However, if your program requires three or more rows of toolbars or uses task panes, using a ribbon is more space efficient.
- **Do users tend to work in a specific area within a large window in the program for long periods of time?** If so, they would benefit from the close proximity of mini-toolbars, palette windows, and direct commands. Making the round trip from the work area to the ribbon would be too inefficient.
- **For efficiency and flexibility, do users need to make significant changes to the command presentation contents, location, or size?** If so, customizable and extensible toolbars and palette windows are a better choice. Note that some types of toolbars can be undocked to become palette windows, and palette windows can be moved, resized, and customized.

Finally, consider this ultimate question: **Is the improvement in discoverability, ease of learning, efficiency, and productivity worth the cost of the extra space and the need for tabs to organize commands?** If so, using a ribbon is an excellent choice. If you're not sure, consider usability testing a ribbon-based design and comparing it to the best alternative.

Ribbons are a new and engaging form of command presentation, and a great way to modernize a program. But as compelling as they are, they aren't the right choice for every program.

Incorrect:



Please don't do this!

Seven most important things

1. Choose a command solution that is suitable for your program type. Using a ribbon should make a program feel simpler, more efficient, and easier to use never the opposite. If using a ribbon isn't appropriate, consider using rich commands instead.
2. Don't underestimate the challenge of creating an effective ribbon. Don't expect it to be a simple port of your existing menu bars and toolbars. And don't take for granted that using a ribbon automatically makes your program better. Being willing to commit the time and effort required for a command redesign is an important factor in deciding to use a ribbon.
3. Make the commands discoverable. Choose a tab design that has a clear, obvious, unique mapping between your commands and the descriptively labeled tabs where they reside. Users should be able to determine quickly and confidently which tab has the command they are looking for, and rarely choose the wrong tab.
4. Make the commands self-explanatory. Users should understand the effect of a command from its label, icon, tooltip, and preview. They shouldn't have to experiment or read a Help topic to see how a command works.
5. Make using the commands efficient:

    - Users should spend most of their time on the Home tab.
    - Users should rarely have to change tabs during common tasks.
    - When the window is maximized and users are on the correct tab, the most frequently used commands have the most visual emphasis and users can invoke them with a single click. Users can perform all other commands on the tab with at most four clicks.
    - Users shouldn't have to open dialog boxes to give commands and change attributes in common tasks.

6. Help users choose commands and options confidently and minimize the need for trial-and-error. Use results-oriented commands whenever appropriate, often in the form of galleries and live previews.
7. Make sure the ribbon scales well from the largest window sizes to the smallest.

Design concepts

Adapting a ribbon in an existing program

While you might simply refactor a traditional menu bar and toolbar design of an existing program to a ribbon format, doing so misses most of the value of using a ribbon. Ribbons have the most value when used to present immediate, results-oriented commands, often in the form of galleries and live previews. Results-oriented commands

make commands easier to understand and users much more efficient and productive. Instead of refactoring your existing commands, it's better to redesign completely how commands are performed in your program.

Don't underestimate the challenge of creating an effective ribbon. And don't take for granted that using a ribbon automatically makes your program better. Creating an effective ribbon takes a lot of time and effort. Being willing to commit the time and effort required for such a command redesign is an important factor in deciding to use a ribbon.

The nature of ribbons

Compared to traditional menu bars and toolbars, ribbons have the following characteristics:

- **A single user interface (UI) for all commands.** Menu bars are comprehensive and easy to learn, and toolbars are efficient and direct, but why not use a bit more screen space to create a single commands UI that accomplishes all of these? With only one UI, ribbons don't require users to figure out which UI has the command they are looking for.
- **Visible and self-explanatory.** Menu bar commands are self-explanatory through their labels, but are hidden from view most of the time. To save screen space, toolbar buttons are primarily represented by icons instead of labels (although some toolbar buttons use both), and depend on tooltips when the icon isn't self-explanatory. However, users generally know the icons only for the most commonly used commands.
- By presenting most commands with labeled icons, ribbon commands are both visible and self-explanatory, and use tooltips only to provide supplemental information. There's little need to go elsewhere (such as Help) to understand a command.
- **Labeled grouping.** While menu categories are labeled, groups within a drop-down menu are not and are indicated only with an unlabeled separator. Groups within toolbars are also indicated with unlabeled separators.
- By organizing commands in labeled groups, ribbons make it easier to find commands and determine their purpose.
- **Modal but not hierarchical.** Menu bars scale by creating a hierarchy of commands. Menus with many items can use one or more levels of submenus to provide more commands.
- Ribbon commands require more space than toolbar commands, so they use tabs to scale. This use of tabs makes ribbons modal, requiring users to change modes

occasionally to find commands. However, within a tab most commands are either direct or use a single split button or menu button, not a hierarchy.

- **Direct and immediate.** A command is direct if invoked with a single click (that is, without navigating through menus) and immediate if it takes effect immediately (that is, without dialog boxes to gather additional input). Menu bar commands are always indirect and often not immediate. Like toolbars, most ribbon commands are designed to be direct and immediate, with the most frequently used commands invoked with a single click, and without requiring a dialog to gather additional input.
- **Spacious.** Menu bars and toolbars are primarily designed to be space efficient. To provide their benefits, ribbons may consume more vertical space, being roughly the equivalent of a menu bar plus three rows of toolbars. Being that few programs have three or more rows of toolbars, ribbons usually consume more space than traditional UIs for commands.
- **Has an Application button and Quick Access Toolbar.** A ribbon is always presented with an Application button and Quick Access Toolbar. Doing so allows users to access file-related and frequently used commands without changing tabs, and promotes consistency across programs.
- **Minimal customization.** While menu bars have a fixed presentation, many toolbars are quite customizable, allowing users to set locations, sizes, and contents. A ribbon itself is not customizable, but the Quick Access Toolbar provides limited customization.
- **Improved keyboard accessibility.** Menu bars have excellent keyboard accessibility because pressing the Alt key directly gives the menu bar input focus. However, there is no such mechanism for toolbars because they share keyboard navigation with the window's content. Consequently, users must navigate to the toolbar using the Tab key (which is given the last tab stop), and then navigate to a specific command using the arrow keys.

By contrast, ribbons provide enhanced keyboard accessibility through keytips, usually with a three-step process:

- Press Alt to enter keytip mode.
- Press a character to choose a tab, the Application button, or a command in the Quick Access Toolbar.
- Within a tab, press a one or two letters to choose a command.

  This approach is highly visual. It is also more flexible, allowing it to scale better and to have more mnemonic access key assignments.

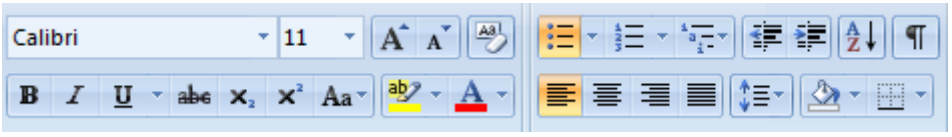
Don't confuse access keys with shortcut keys. While both access keys and shortcut keys provide keyboard access to UI, they have different purposes and guidelines. For more information, see Keyboard.

The nature of rich commands

Rich commands refer to the presentation and interaction of commands used by ribbons, without necessarily using a ribbon container. Rich commands have these characteristics:

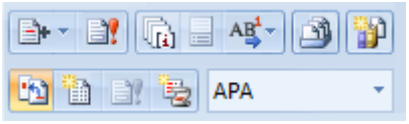
- **Labeling.** All commands are given self-explanatory labels, with exceptions only when the icons are extremely well known and space is at a premium.

  Correct:

  These commands are extremely well known so they don't need labels.

  Incorrect:

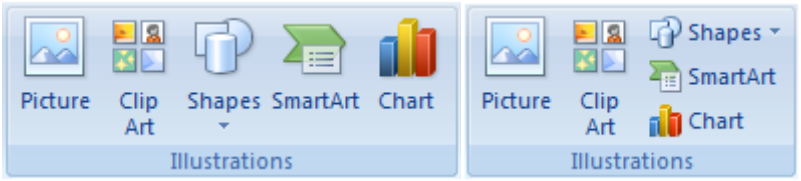
  

  These unintelligible icons require labels for rich commands.

- **Sizing.** Instead of uniform sizing, commands are sized relative to their frequency of use and importance. In addition to making the most frequently used commands easier to find and click, it also makes them more touchable.

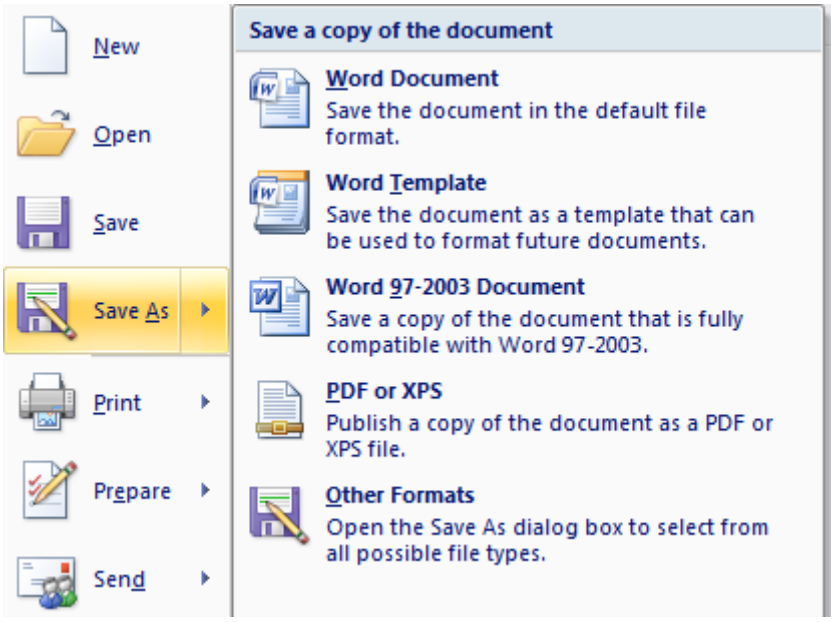
  In this example, the most frequently used button is larger than the others.

- **Dynamic sizing.** Rich command controls resize themselves to take full advantage of the available space, as opposed to using a fixed size and either truncating or using overflow when the size is too small.



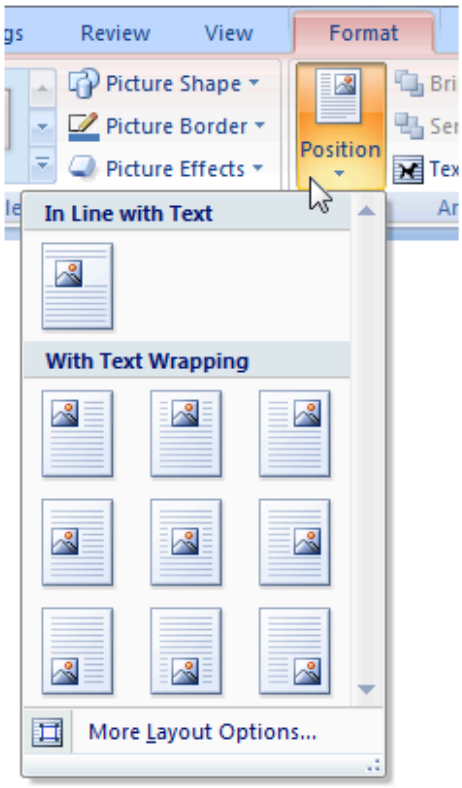
In this example, the command buttons resize to work well in the available space.

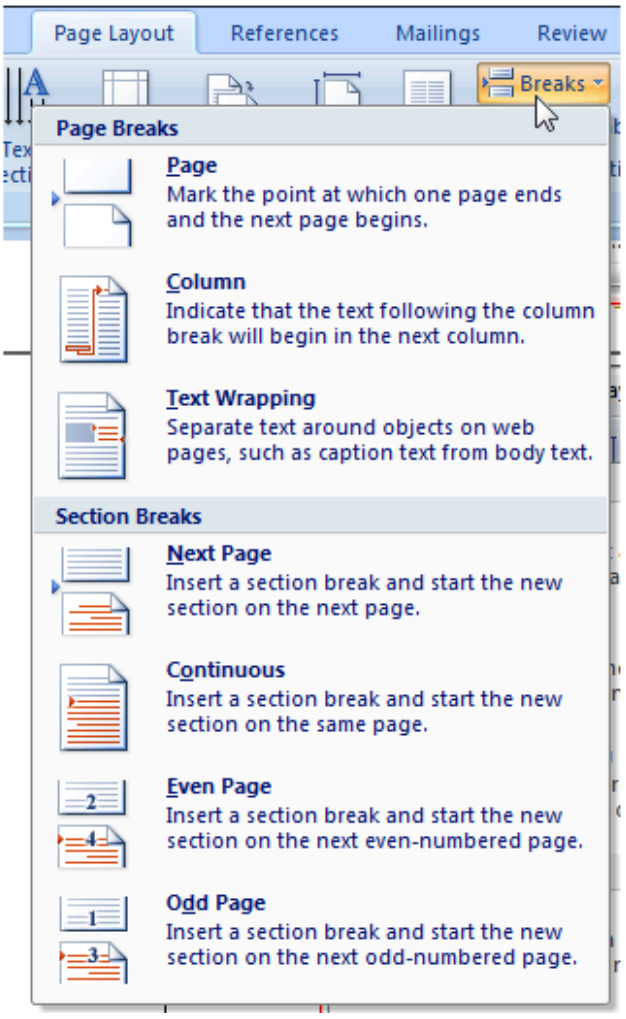
- **Split buttons.** Split buttons are a good way to consolidate a set of variations of a command when needed, while maintaining directness for the most frequently used command.



In this example, the Save As command uses a split button, where the main button performs the most common variation and the menu portion reveals a menu with variations of the command.

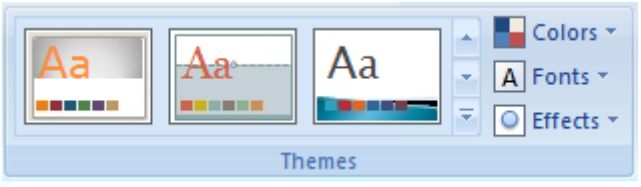
- **Rich drop-down menus and galleries.** Drop-down menus, drop-down lists, and galleries take the space they need to communicate and differentiate the effect of the choices, often using graphics and text descriptions. Categories are used to organize large sets of options.





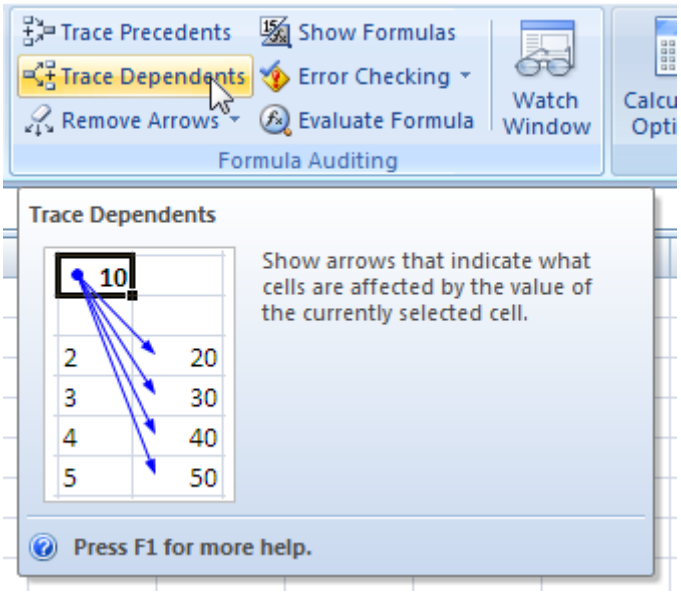
In these examples, clicking a menu button displays a list of choices that show their effect.

- **Live previews.** Whenever the user hovers over a formatting option, the program shows what the results would look like with that formatting using the actual content.



Live previews show the results of applying a formatting option on hover.

- **Enhanced tooltips.** These concisely explain their associated commands and give the shortcut keys. They may also include graphics and references to Help (although they largely eliminate the need for command-related Help).



Enhanced tooltips concisely explain their associated commands.

While ribbons might not be suitable for all programs, all programs can potentially benefit from rich commands.

Ribbons always have an Application button and Quick Access Toolbar

The Application button and Quick Access Toolbar provide commands that are useful in any context, thus reducing the need to change tabs. While these three components are logically independent, a ribbon must always have an Application button and Quick Access Toolbar. Given that commands can go in either the ribbon or the Application button, you might be wondering where to place commands. The choice is not arbitrary.

The Application button is used to present a menu of commands that involve doing something to or with a file, such as commands that traditionally go in the File menu to create, open, and save files, print, and send and publish documents.

By contrast, the ribbon itself is for commands that affect the content of the window. Examples include commands used to read, modify, or use the content, or change the view.

If you use a ribbon, you must also use an Application button even if your program doesn't involve documents or files. In such cases, use the Application menu to present commands for printing, program options, and exiting the program. While arguably the Application button isn't necessary for such programs, using it provides consistency across programs. Users don't have to hunt for save and undo commands or program options because they are always in the same place.

The Quick Access Toolbar is required even if the ribbon only uses one tab. Again, while arguably such programs don't need a Quick Access Toolbar (because all the commands are already present on the single tab), having a customizable Quick Access Toolbar provides consistency across programs. For example, if users are in the habit of clicking the Print command, they should be able to do so in any program that uses a ribbon.

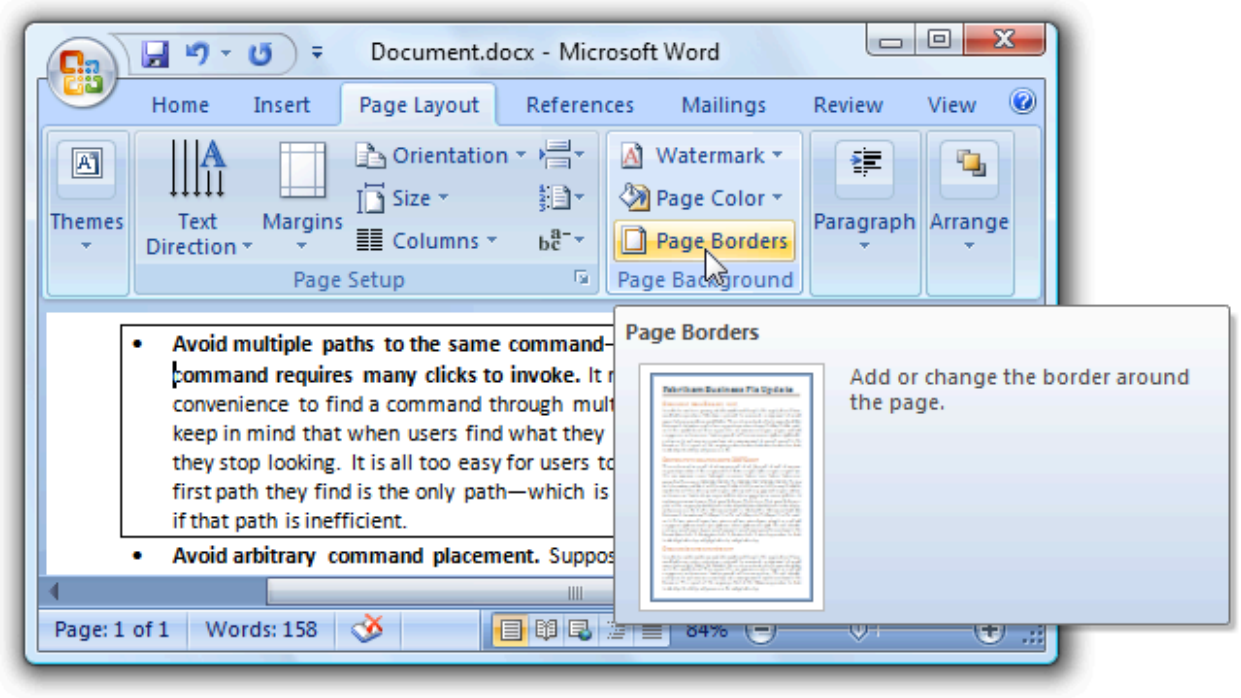
Organization and discoverability

By providing tabs and groups, ribbons allow you to organize your commands to aid discoverability. The challenge is that if the organization is done poorly, it can greatly harm discoverability instead. There should be a clear, obvious, and unique mapping between your commands and the descriptively labeled tabs and groups where they reside.

Users will form a mental model of the ribbon after using it for a while. If that mental model doesn't make sense to users, is inefficient, or is incorrect, it will lead to confusion and frustration. **Your most important goal in designing a ribbon is to facilitate finding commands quickly and confidently. If you do not accomplish this, your ribbon design will fail.** Achieving this goal requires careful design, user testing, and iteration. Don't assume that it will be easy.

Here are some common pitfalls to avoid:

- **Avoid generic tab and group names.** A good tab or group name must accurately describe its specific contents, preferably using task- and goal-based language. Avoid generic tab and group names, as well as technology-based names. For example, almost any command in a document creating and authoring program could belong in tabs labeled Edit, Format, Tools, Options, Advanced, and More. Rely on specific, descriptive labels, not memorization.
- **Avoid overly specific tab and group names.** While we want tab and group names to be specific, they shouldn't be so specific that users are surprised by their content. Users often look for things using the process of elimination, so prevent them from overlooking your tabs or groups because the name is misleading.
- **Avoid multiple paths to the same command especially if the path is unexpected or the command requires many clicks to invoke.** It may seem like a convenience to find a command through multiple paths. But keep in mind that when users find what they are looking for, they stop looking. It is all too easy for users to assume that the first path they find is the only path which is a serious problem if that path is inefficient or unexpected. Furthermore, having duplicate commands makes it harder for users to find other commands they are scanning for.



In this example, you can change paragraph borders through the Page Borders command, even though there is a more direct path on the Home tab. If users looking for paragraph borders were to stumble across this unexpected path, they might easily assume that it's the only path.

- **Avoid arbitrary command placement.** Suppose that you think you have a good tab and group design, but discover that several commands just don't fit in. Chances are, your tab and group design isn't as good as you think it is, and you need to continue to refine it. Don't solve this problem by putting those commands where they don't belong. If you do, users likely will have to inspect every tab to find them then promptly forget where they are.
- **Avoid marketing-based placement.** Suppose that you have a new version of your program and your marketing team really wants to promote its new features. It might be tempting to put them on the Home tab, but doing so is a costly mistake if it harms overall discoverability. Consider future versions of your product and how much frustration a constantly changing organization will cause.

Tabs

The best first step is to review the standard ribbon tabs. If your program's commands map naturally into the standard tabs, base your tab organization on these standards. On the other hand, if you program's commands don't map naturally, don't try to force it. Determine a more natural structure, and be sure to perform a lot of user testing to make sure that you've got it right.

For non-standard tabs, consider these issues:

- **Each tab name should describe its content.** Choose meaningful names that are specific, but not too specific. Users should never be surprised by their content.
- **Each tab name should reflect its purpose.** Consider the goal or task associated with the commands.
- **Each tab name should be clearly distinct from all the other tab names.**

The Home tab is an exception to these considerations. While you don't have to have a Home tab, most programs should. The Home tab is the first tab, and contains the most frequently used commands. If you have frequently used commands that don't fit into the other tabs, the Home tab is the right place for them.

If you can't determine a meaningful, descriptive tab name, it is probably because the tab isn't well designed. If your ribbon organization just isn't working, reconsider your tab design.

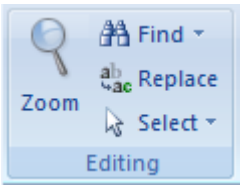
Groups

Dividing commands into groups structures the commands into related sets. The group label explains the common purpose of its commands.

There are a variety of factors to consider when determining groups and their presentation:

- **Standard grouping.** While there are significant differences in commands across programs, there are standard groups that are common across many programs. Having these commands appear with the same names and similar locations greatly improves discoverability.

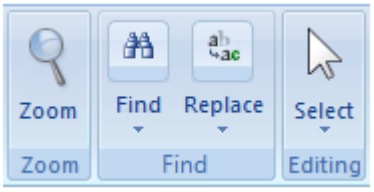
Expand table

Correct	Incorrect
Editing commands group includes all editing commands, but does not include the Zoom command.	 The Zoom command is not an editing command, but is in the editing group.

- **Granularity.** Some structure is good, but too much structure makes commands harder to find. If the group names are generic, you might not have enough granularity. If there are groups with only one or two commands each, you probably

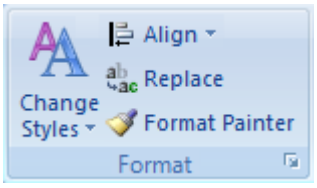
have too much (although having an in-ribbon gallery without any other commands within a group is acceptable).

Expand table

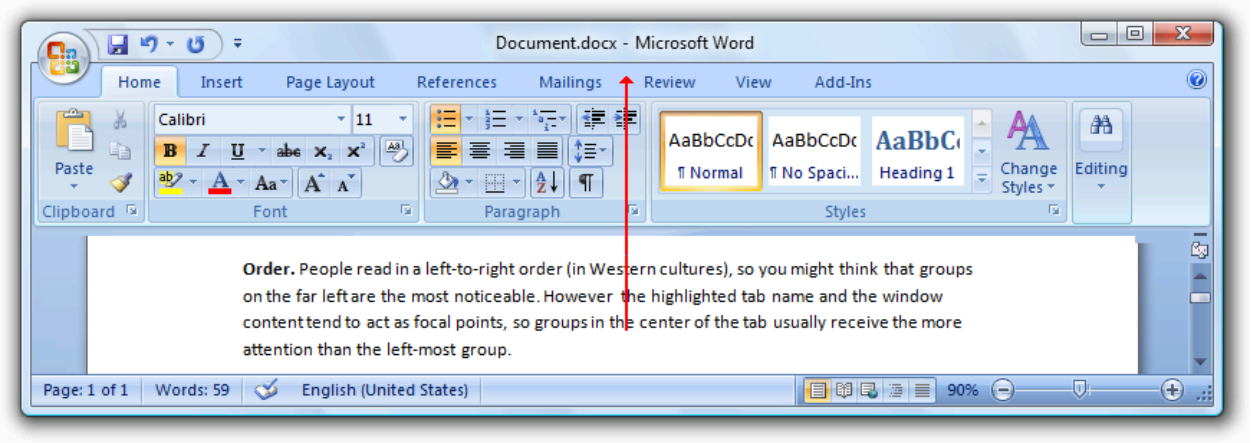
Correct	Incorrect
 Editing commands group includes all editing commands	 Editing commands group has been split into sections that are too granular. Avoid groups with only one or two commands.

- **Names.** Good group names explain the purpose of their commands. If your group names don't, reconsider the name or the grouping. If you can't determine a meaningful, descriptive name, it is probably because the group isn't well designed.

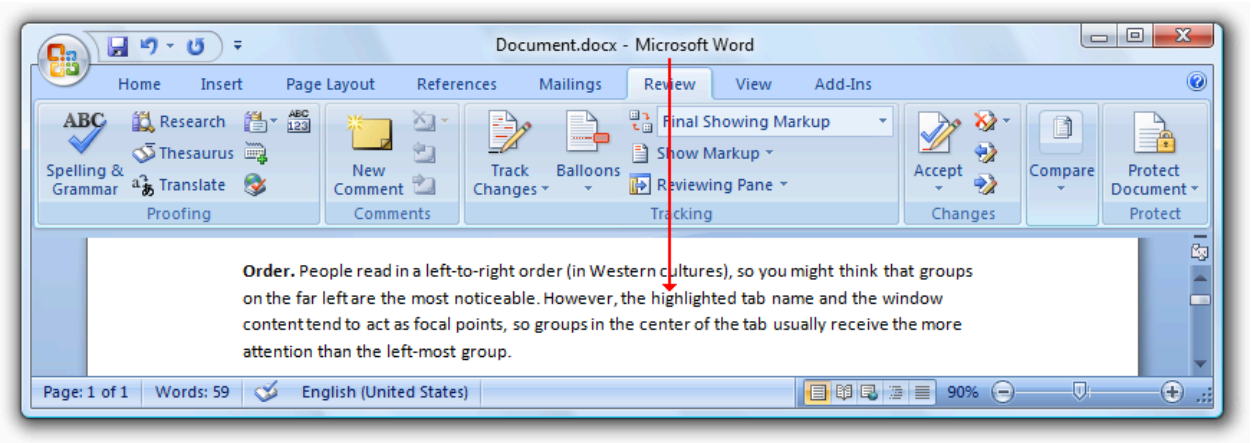
Expand table

Correct	Incorrect
 Use group names that are specific enough to describe the commands contained in the group.	 This group name is too vague to be helpful. A better approach would be to reorganize these commands into more specific groups.

- **Order.** People read in a left-to-right order (in Western cultures), so you might think that groups on the far left are the most noticeable. However, the highlighted tab name and the window content tend to act as focal points, so groups in the center of the tab usually receive more attention than the left-most group. Place the most commonly used groups in the most prominent locations, and make sure there is a logical flow for the groups across the tab.



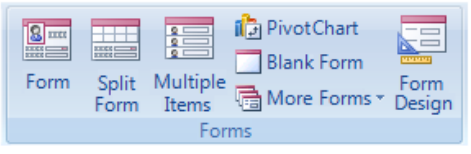
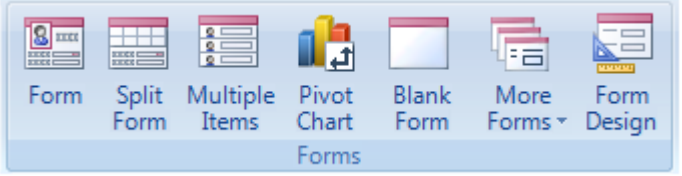
In this example, the Font and Paragraph groups are more noticeable than the Clipboard group, because they are what the eye sees first when moving up from the document.



In this example, the Tracking group receives the most attention, in part because the highlighted Review tab acts as a focal point.

- **Uniformity.** It can be hard to recognize commands when the command presentation all looks the same. Using icons with different shapes and colors, groups with varying formats, and commands with different sizes makes it easier for users to recognize command groups. Commands should have uniform sizing only when the ribbon is scaled down to its smaller sizes.

Expand table

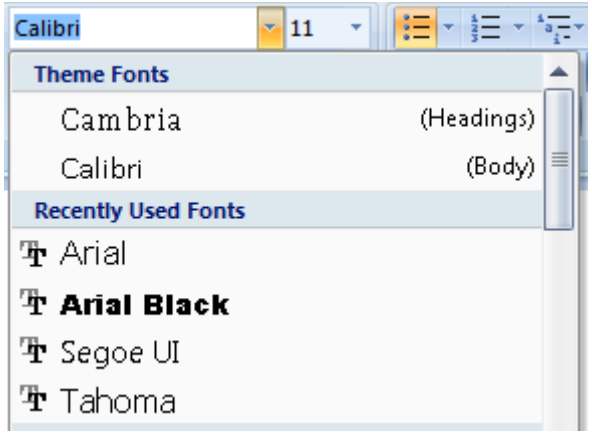
Correct	Incorrect
 Use a variety of icon sizes to improve recognizability	 These commands look too similar to each other because they are all the same size.

Previews

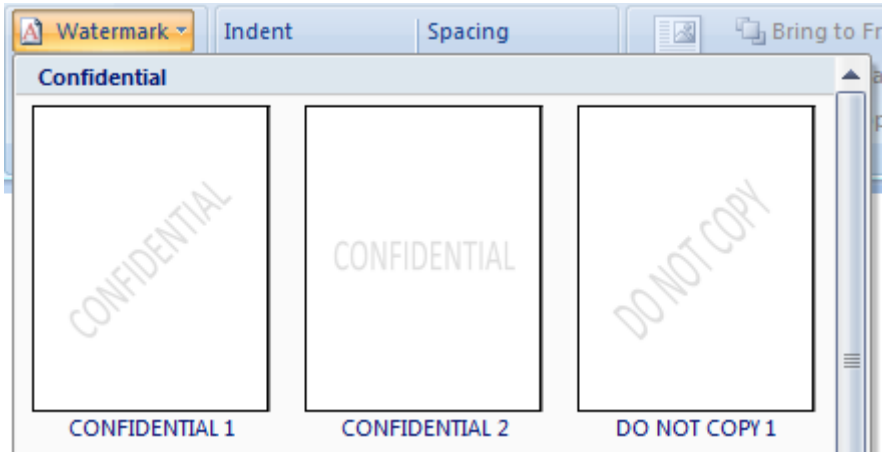
You can use various types of previews to show what will result from a command. By using helpful previews, you can improve the efficiency of your program and reduce the need for the trial-and-error learning approach. Live previews also invite experimentation and encourage creativity.

Here are some of the different types of previews that you can use:

- **Realistic static icons and graphics.** Static images that give a realistic indication of a command's effect. These can be used in galleries, drop-down menus, and enhanced tooltips.



In this example, the Font drop-down list shows each font name using the font itself.



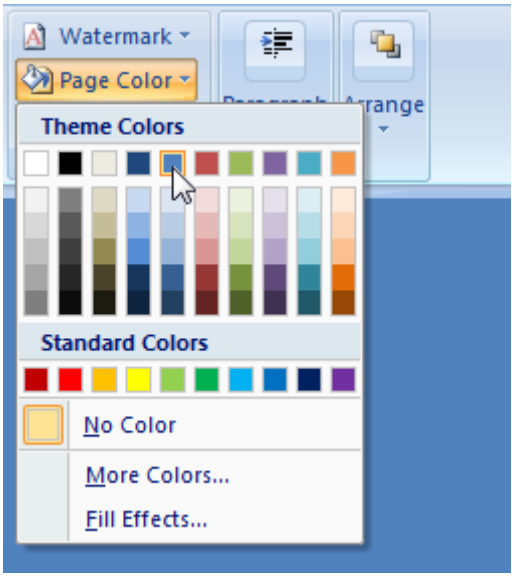
In this example, realistic thumbnails are used to show the different watermarks.

- **Dynamic icons and graphics.** Icons and graphics that are modified to reflect the current state. Such icons are especially useful for galleries, as well as split buttons that change their default effect to be the same as the last action.

In this example, Microsoft Word changes the Styles gallery to reflect the current styles.

In this example, Word changes the Text highlight color and Font color commands to indicate their current effect.

- **Live previews.** When users hover over a formatting option, live preview shows what the results would look like with that formatting. Live previews help users make selections more efficiently and confidently based on the user's actual context.

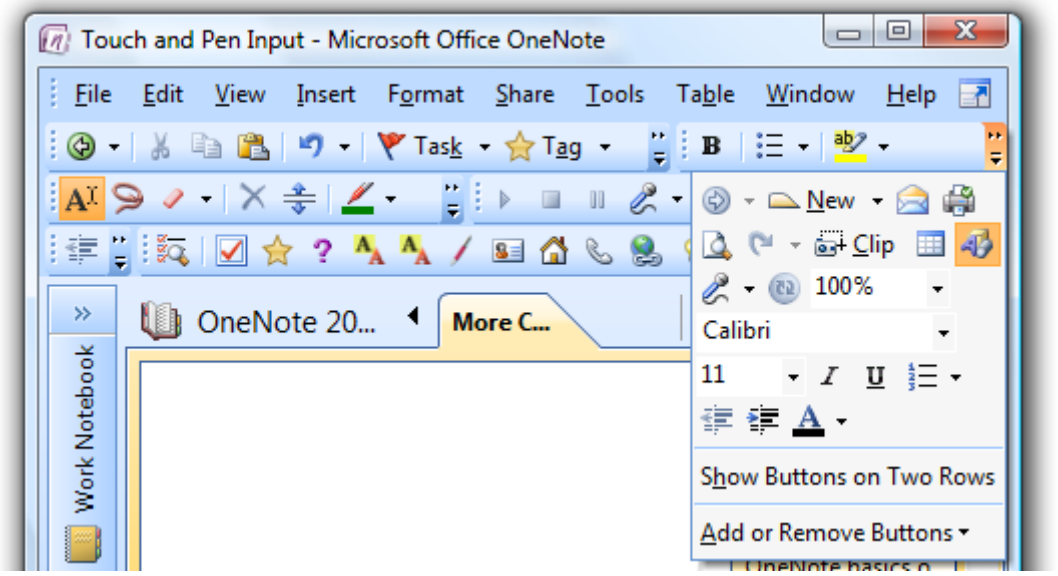


In this example, the Page Color command performs a live preview by showing the effect of the color options on hover.

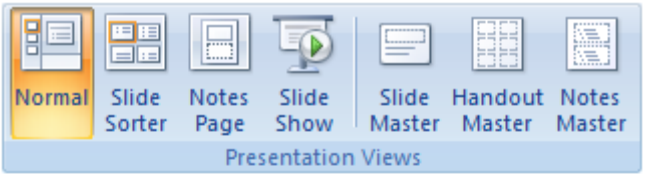
Live previews are a powerful feature that can really improve your users' productivity, but even simple static previews can be a big help.

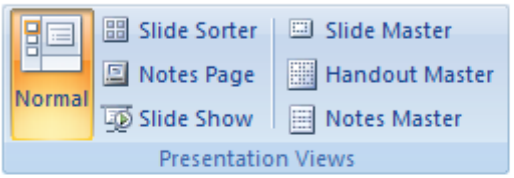
Scaling the ribbon

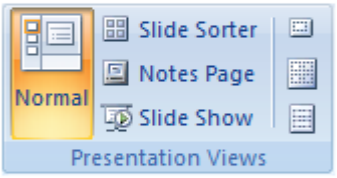
Scaling a toolbar is simple: if a window is too narrow to display a toolbar, the toolbar displays what fits and makes everything else accessible through an overflow button. A goal of rich commands is to take full advantage of the available space, so scaling a ribbon requires more design work. There is no default ribbon size, so you should not design a ribbon with a particular width in mind. You have to design layouts with a wide range of widths and realize that any one of them could be the one most of your users will see. Scaling is a fundamental part of ribbon design, not the last step. When designing a tab, specify the different layouts for each group (up to three) as well as the combinations that can be used together. The ribbon will show the largest valid combination that fits the current window size.

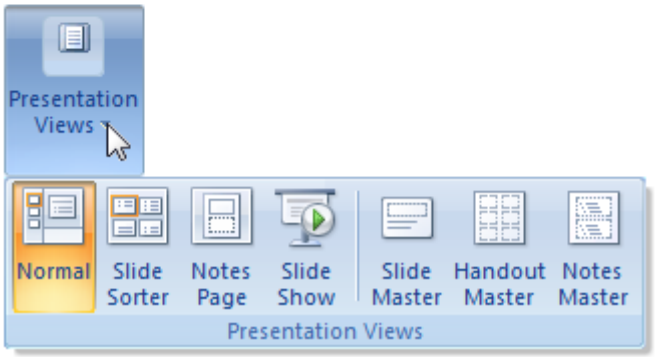


Toolbars scale using an overflow button.









There is no default ribbon size. The smallest size is a single pop-up group icon.

Guidelines

General

- **Don't combine ribbons with menu bars and toolbars within a window.** Ribbons must be used in place of menu bars and toolbars. However, a ribbon may be combined with palette windows and navigation elements, such as Back and Forward buttons and an Address bar.
- **Always combine a ribbon with an Application button and Quick Access Toolbar.**
- **Select the left-most tab (usually Home) when a program is started.** Don't make the last selected tab persist across program instances.
- **Show the ribbon in its normal state (not minimized) when a program is started for the first time.** Users often leave default settings unchanged, so minimizing the ribbon at program start will likely cause all commands to be less efficient. Also, showing the ribbon initially minimized can be disorienting.
- **Make the ribbon state persist across program instances.** For example, if a user minimizes the ribbon, it should be shown minimized the next time the program is run. But again, don't make the last selected tab persist in this way.

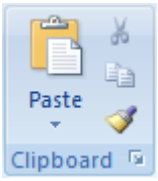
Using tabs

Generally, having fewer tabs is better, so remove tabs that don't help achieve these goals.

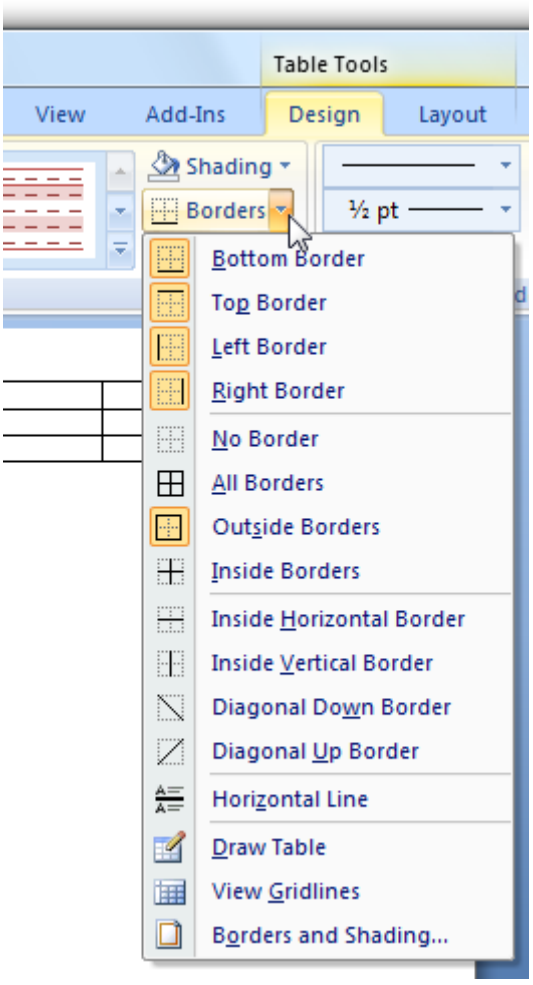
- **Whenever practical, use standard tabs.** Using standard tabs greatly improves discoverability, especially across programs. See the standard ribbon tabs later in this article.
- **Label the first tab Home, if appropriate.** The Home tab should contain the most frequently used commands. If you have frequently used commands that don't fit into the other tabs, the Home tab is the right place for them.
- Add a new tab if:
  - **Its commands are strongly related to specific tasks, and can be accurately described by the tab label.** Adding the tab should help make its commands easy to find, not harder.
  - **Its commands are mostly unrelated to tasks on other tabs.** Adding the tab shouldn't require more tab switching during commonly performed tasks.
  - **The tab has enough commands to justify having an extra place to look.** Don't have tabs with only a few commands. **Exception:** Consider adding a tab with a few commands if they are strongly related to a specific task and adding the tab greatly simplifies an overly complex Home tab.
- **For the remaining tabs, place the most frequently used tabs first, while maintaining a logical order across the tabs.**
- **Optimize the tab design so that users find commands quickly and confidently.** All other considerations are secondary.

- **Don't provide a Help tab.** Instead, provide assistance using program-wide Help and enhanced tooltips.
- **Use a maximum of seven core tabs.** If there are more than seven, it becomes difficult to determine which tab has a command. While seven core tabs is acceptable for applications with many commands, most programs should aim for four or fewer tabs.

Contextual tabs

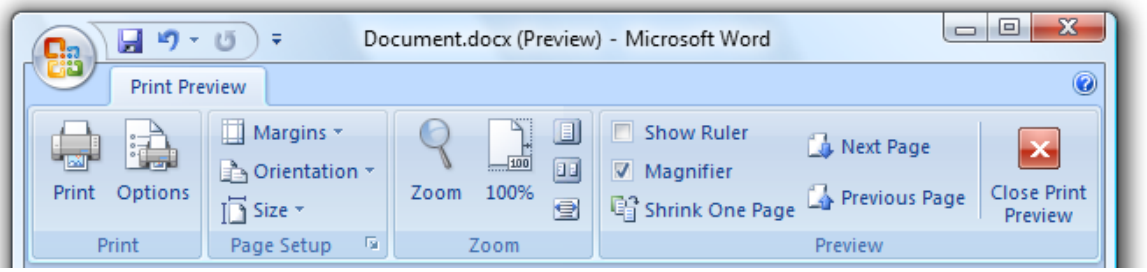
- **Use a contextual tab to display a collection of commands that are relevant only when users select a particular object type.** If there are only a few, frequently used commands, it may be more convenient and more stable to use a regular tab, and simply disable commands when they don't apply.
- 

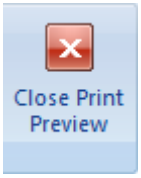
  It's better to disable common commands like Cut and Copy than to use a contextual tab.
- **Include only the commands that are specific to a particular object type.** Don't put commands only on a contextual tab if users might need them without first selecting an object.
- **Include the commands frequently used when working with a particular object type.** Put frequently used general contextual commands on context menus and mini-toolbars to avoid tab switching during commonly performed tasks. Alternatively, consider putting general commands redundantly on a contextual tab if doing so avoids frequent tab switching. But don't overdo this - don't try to include every command that a user might need while working with the object.

- 

  In this example, the Borders command is included on the Design tab to avoid frequent tab switching during commonly performed tasks.\
- **Choose a contextual tab color that is different from the currently displayed contextual tabs.** The same tab set can appear at a later time using a different color in order to achieve this, but try to use consistent color assignments across invocations whenever possible.
- **Selecting a contextual tab automatically** aids discoverability, improves the perception of stability, and reduces the need to switch tabs. Select a contextual tab automatically when:
  - **The user inserts an object.** In this case, select the first contextual tab in the set.
  - **The user double-clicks an object.** In this case, select the first contextual tab in the set.
  - **The user selected a contextual tab, clicked off the object, then immediately clicked an object of the same type.** In this case, return to the previously selected contextual tab.
- **When removing a contextual tab that is the active tab, make the Home tab or first tab the active tab.** Doing so appears the most stable.

Modal tabs

- **Use a modal tab to display a collection of commands that apply with a particular temporary mode, and none of the core tabs apply.** If some of the core tabs apply, use a contextual tab instead, and disable the commands that don't apply. Because modal tabs are very limiting, they should be used only when there isn't a better alternative.
- 

  Print preview is a commonly used modal tab.
- **To close a modal tab, put the Close <mode> command as the last command on the tab.** Use the Close icon to make the command easy to find. Give the mode in the command to prevent confusion about what is being closed.
- 

  In this example, explicitly labeling the Close command with the mode removes any doubt about what is being closed.
- **To close a modal tab, also redefine the Close button on the window's title bar to close the mode instead of the program.** User testing has shown that many users expect this behavior.

Standard ribbon tabs

Whenever practical, map your program's commands to these standard tabs, given in their standard order of appearance.

Regular tabs

- **Home.** Contains the most frequently used commands. If used, it is always the first tab.
- **Insert.** Contains commands to insert content and objects into a document. If used, it is always the second tab.
- **Page layout.** Contains commands that affect the page layout, including themes, page setup, page backgrounds, indenting, spacing, and positioning. (Note that the indenting and spacing groups can be on the Home tab instead, if there is enough room there.) If used, it is always the third tab.

- **Review.** Contains commands to add comments, track changes, and compare versions.
- **View.** Contains commands that affect the document view, including view mode, show/hide options, zooming, window management, and macros the commands traditionally found in the Windows menu category. If used, it is the last regular tab unless the Developer tab is showing.
- **Developer.** Contains commands used only by developers. If used, it is hidden by default and the last regular tab when displayed.

Most programs don't need the Review and Developer tabs.

Standard Contextual tabs

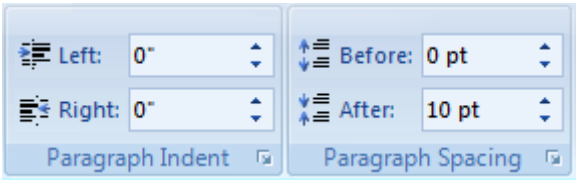
- **Format.** Contains commands related to changing the format of the selected object type. Usually applies to part of an object.
- **Design.** Contains commands, often in galleries, to apply styles to the selected object type. Usually applies to the entire object.
- **Layout.** Contains commands to change the structure of a complicated object, such as a table or chart.

If you have contextual commands related to format, design, and layout, but not enough for multiple tabs, just provide a Format tab.

Standard groups

- **Whenever practical, use standard groups.** Having common commands appear with the same names and similar locations greatly improves discoverability. See the standard ribbon groups later in this article.
- **Add a new group** if:
  - **Its commands are strongly related and can be accurately described by the group label.** Adding the group should help make its commands easy to find, not harder.
  - **Its commands have a weaker relationship to the commands in other groups.** While all the commands on a tab should be strongly related, some command relationships are stronger than others.
  - **The group has enough commands to justify having an extra place to look.** Aim for 3-5 commands for most groups. Avoid having groups with only 1-2 commands, although having an in-ribbon gallery without any other commands within a group is acceptable. Having many groups with a single command suggests too much structure or lack of command cohesion.
- **Don't over-organize** by adding groups where they aren't needed.

- **Consider splitting a group** if:
  -

    The group has many commands of different sizes and needs organization.
  - 

    The group has commands that greatly benefit from having extra labels.
- **Place the most commonly used groups in the most prominent locations, and make sure there is a logical order for the groups across the tab.**
- **Optimize the group design so that users find commands quickly and confidently.** All other considerations are secondary.
- **Don't scale groups containing a single button to a pop-up group icon.** When scaling down, leave them as a single button.
- **Use a maximum of seven groups.** If there are more than seven groups, it becomes more difficult to determine which group has a command.

Standard ribbon groups

Whenever practical, map your program's commands to these standard groups, which are given within their associated tabs in their standard order of appearance.

Main tab

- Clipboard
- Font
- Paragraph
- Editing

Insert tab

- Tables
- Illustrations

Page layout tab

- Themes
- Page setup

- Arrange

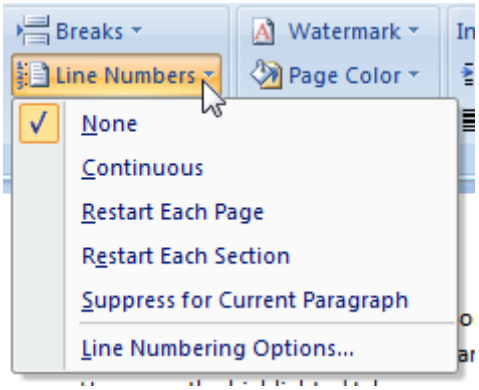
Review tab

- Proofing
- Comments

View tab

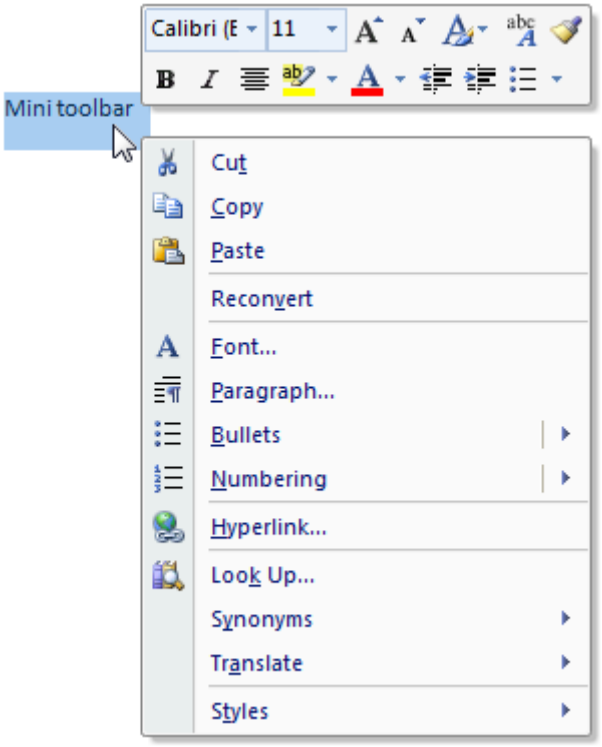
- Document views
- Show/hide
- Zoom
- Window

Commands

- 

  Take advantage of the discoverability and scalability of ribbons by exposing all the commonly used commands. When appropriate, move frequently used commands from dialog boxes to the ribbon, especially those that are known to be hard to find. Ideally, users should be able to perform common tasks without using any dialog boxes.

- **Don't use the scalability of ribbons to justify adding unnecessary complexity.** Continue to exercise restraint don't add commands to a ribbon just because you can. Keep the overall command experience simple. The following are ways to simplify the presentation:



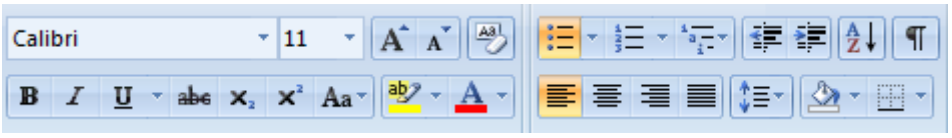
  - **Use context menus and mini-toolbars for in-place, contextual commands.**
  - **Move (or keep) rarely used commands in dialog boxes.** Use dialog box launchers to access these commands. You can still use dialog boxes with ribbons! Just try to reduce the need for using them during common tasks.
  - **Eliminate redundant, seldom used features.**

Presentation

- **Present each command on only one tab. Avoid multiple paths to the same command especially if the command requires many clicks to invoke.** It may seem like a convenience to find a command through multiple paths. But keep in mind that when users find what they are looking for, they stop looking. It is all too easy for users to assume that the first path they find is the only path which is a serious problem if that path is inefficient. **Exception:** Contextual tabs may duplicate a few commands from the Home and Insert tabs if doing so prevents changing tabs for common contextual tasks.

- **Within a group, put the commands in their logical order, while giving preference to the most frequently used commands.** Overall, the commands should have a logical flow to make them easy to find, while still having the most frequently used commands appear first. Generally, commands with 32x32 pixel icons appear before commands with 16x16 pixel icons to aid scanning across groups.

- **Avoid placing destructive commands next to frequently used commands.** A command is considered destructive if its effect is widespread and either it cannot be easily undone or the effect isn't immediately noticeable.

- **Use separators to indicate strongly related commands, such as a set of mutually exclusive options.**
- 

  Consider using toolbar-style groups for sets of strongly related, well-known commands that don't need labels. Doing so allows you to present many commands in a compact space without affecting discoverability and ease of learning. To be so well known, such commands are frequently used, instantly recognized, and therefore tend to be on the Home tab.

- **Use 32x32 pixel icons for the most frequently used and important labeled commands.** When scaling a group down, make these commands the last to convert to 16x16 pixel icons.

- **Avoid arbitrary command placement.** Think carefully about your tab and group design to ensure that users aren't wasting time inspecting every tab to find the command they want.

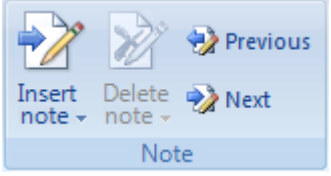
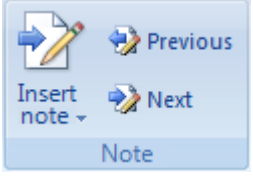
- **Avoid marketing-based placement.** Marketing objectives around the promotion of new features tend to change over time. Consider future versions of your product and how much frustration a constantly changing organization will cause.

Interaction

- **Disable commands that don't apply to the current context, or that would directly result in an error.** If helpful, use the enhanced tooltip to explain why the command is disabled. Don't hide such commands because doing so can cause the ribbon layout to change, making the ribbon presentation unstable.

- **Don't update command labels dynamically.** Again, doing so might cause the tab layout to change, resulting in an unstable appearance. Instead, design commands so that they work with constant labels.

Expand table

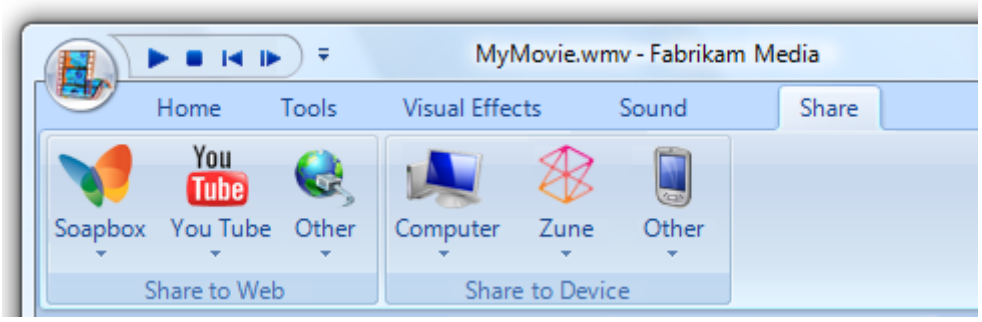
Correct	Incorrect
 Disable commands when they are	 Do not hide commands, even when they are not

Correct	Incorrect
unavailable	available

- **Prefer direct controls.** A command is direct if invoked with a single click (that is, without navigating through menus). However, with the exception of in-ribbon galleries, direct controls don't support Live preview, so the need for Live preview is also a factor.

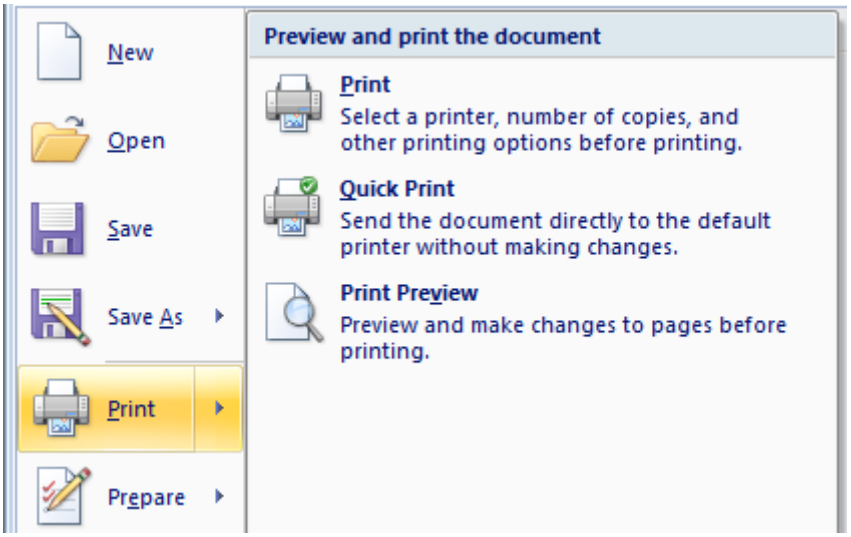
- **Use Live preview** to indicate the effect of the options when a command is among a related set of formatting options, and Live preview is important and practical, especially if users are likely to choose the wrong option otherwise.
  - If the command is used frequently, use an in-ribbon gallery for directness.
  - If the command is used infrequently, use a drop-down gallery.

- **Expose direct commands** using the following controls in the following order of preference
  - **Command buttons, check boxes, radio buttons, and in-place galleries.** These are always direct.
  - **Split buttons.** Direct for the most common command, but indirect for the command variations.
  - **Menu buttons.** These are indirect, but present many commands that are easy to find.
  - **Text boxes (with spin controls).** Text input generally requires more effort than the other control types.

- 

  If your ribbon consists mostly of menu buttons when displayed at full size, you might as well use a menu bar.

- **Prefer immediate commands.**



  A command is immediate if it takes effect immediately (that is, without dialog boxes to gather additional input). If a command might require input, consider using a split button, with the immediate command in the button portion, and the commands that require input in the submenu.

Galleries

Use a gallery if:

- **There is a well-defined, related set of choices from which users typically choose.** There may be an unbounded number of variations, but the likely selections should be well contained. If the choices aren't strongly related, consider using separate galleries.
- **The choices are best expressed visually, such as formatting features.** Using thumbnails makes it easier to browse, understand, and make choices. While the choices can be labeled, the selection is made visually and text labels shouldn't be required to understand the choices.
- **The choices show the result that is achieved immediately with a single click.** There shouldn't be any follow-up dialog box to further clarify the user's intention, or a set of steps to achieve the indicated result. If users might want to adjust the choice, let them do so afterwards.

Use an in-ribbon gallery if:

- **The choices are used frequently.** The choices need the space and are worth the space potentially being taken from other commands.
- **For typical usage, there is no need to group or filter the presented choices.**
- **The choices can be displayed effectively within the height of a ribbon (which is 48 pixels).**

Thumbnails in galleries

Choose the smallest standard gallery thumbnail size that does the job well.

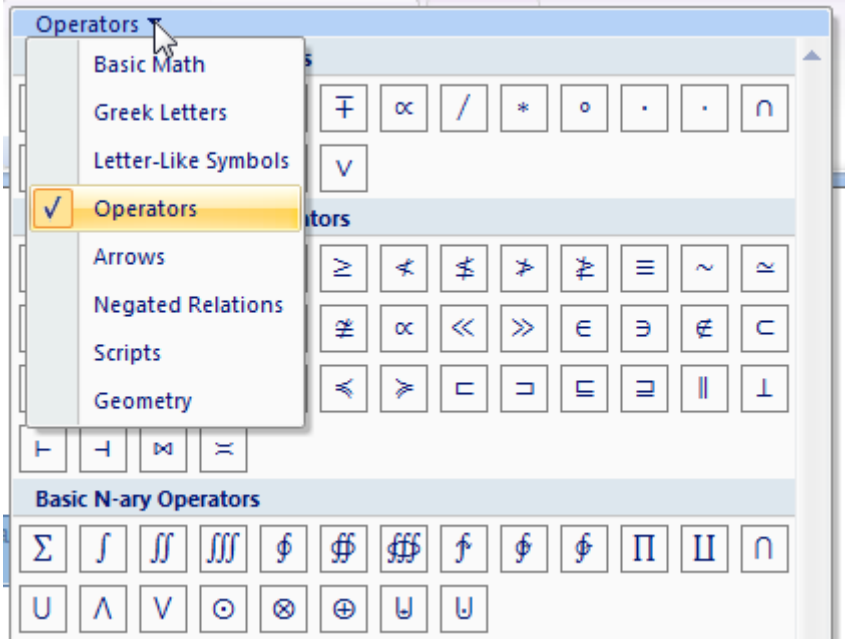
- For in-ribbon galleries, use thumbnails of 16x16, 48x48, or 64x48 pixels.
- For drop-down galleries, use thumbnails of 16x16, 32x32, 48x48, 64x48, 72x96, 96x72, 96x96, or 128x128 pixels.
- All gallery items should have the same thumbnail size.

For in-ribbon galleries:

- **Display a minimum of three choices; more if there is room.** If there isn't sufficient space to display at least three choices in the typical window size, use a drop-down gallery instead.
- **Expand in-ribbon galleries to take advantage of available space.** Use the additional space to show more items and make them easier to choose with a single click.

For drop-down galleries:

- **Display the gallery from either a combo box, drop-down list, split button, or menu button.**
- **If the user clicks the main window to dismiss the drop-down gallery, just dismiss the gallery without selecting or modifying the contents of the main window.**
- **If a gallery has many choices and some choices are rarely used, simplify the default gallery by focusing on the commonly used choices.** For the remaining commands, provide an appropriate command at the bottom of the gallery drop-down.
  - If the command shows a list of more variations, name it "More `singular feature name` options..."
  - If the command presents a dialog box that allows users to create their own custom options, name it "Custom `feature name`..."
- **Organize the choices into groups, if doing so makes browsing more efficient.**

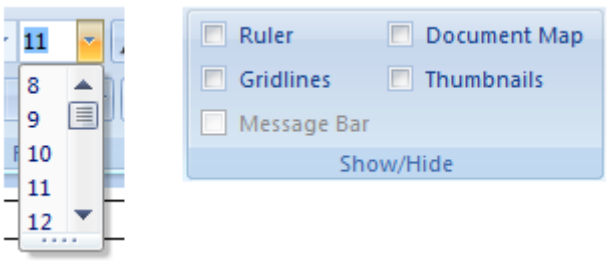
- 

  If a gallery has many items, consider adding a filter to help users find choices more efficiently. To avoid confusion, initially display the gallery unfiltered. However, most galleries shouldn't require a filter because they shouldn't have so many choices, and using groups should be sufficient.

Command previews

- **Use previews to show the effect of a command without users having to perform it first.** By using helpful previews, you can improve the efficiency and ease of learning of your program, and reduce the need for trial-and-error. For the different types of command previews, see Previews in the Design Concepts section of this article.
- **For live previews, make sure that the preview can be applied and the current state restored within 500 milliseconds.** Doing so requires the ability to apply formatting changes quickly and in a way that is interruptible. Users must be able to evaluate different options rapidly for live previews to have their full benefit.
- **Avoid using text in previews.** Otherwise, the preview images will have to be localized.

Icons

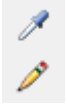
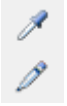
- 

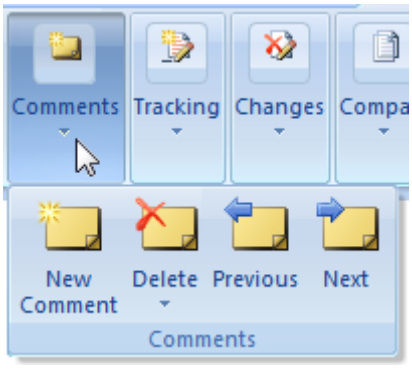
  Provide icons for all ribbon controls except drop-down lists, check boxes, and

radio buttons. Most commands will require both 32x32 and 16x16 pixel icons (only 16x16 pixel icons are used by the Quick Access Toolbar). Galleries typically use 16x16, 48x48, or 64x48 pixel icons.

- **Provide unique icons.** Don't use the same icon for different commands.
- **Make sure ribbon icons are clearly visible against the ribbon background color.** Always evaluate ribbon icons in context and in high-contrast mode.
- **Choose icon designs that clearly communicate their effect,** especially for the most frequently used commands. Well-designed ribbons have self-explanatory icons to help users find and understand commands efficiently.
- **Choose icons that are recognizable and distinguishable,** especially for the most frequently used commands. Make sure the icons have distinctive shapes and colors. Doing so helps users find the commands quickly, even if they don't remember the icon symbol.

Expand table

Correct	Incorrect
 Use shape and color to make icons that are easy to distinguish.	 Icons that are the same color are hard to distinguish

- 

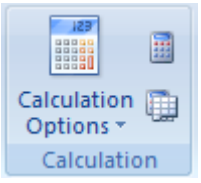
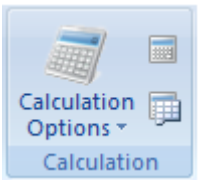
  Consider creating pop-up group icons by placing the 16x16 pixel icon of the most prominent command in the group inside a 32x32 pixel visual container. You don't have to create different icons for pop-up groups.

-

  If useful, change the icon to reflect the current state. Doing so is especially useful for split buttons whose default effect can change.

- **Make sure ribbon icons conform to the Aero-style icon guidelines.** However, ribbon icons are shown straight on instead of being shown in perspective.

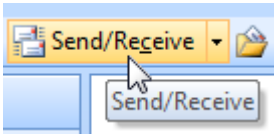
Expand table

Correct	Incorrect
 Use two-dimensional icons.	 Do not use three-dimensional icons.

Enhanced tooltips

- **All ribbon commands should have enhanced tooltips** to give the command name, shortcut key, description, and optional supplemental information. Avoid tooltips that simply restate the label.

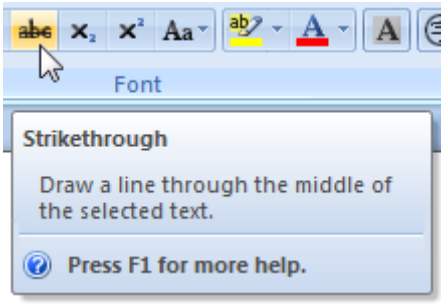
  Incorrect:

  In this example, the tooltip simply restates the command label.

- **When practical, completely describe the command using a concise description.** Link to Help only if further explanation is really necessary.

  Incorrect:

  In this example, the command doesn't need Help.

- **When helpful, illustrate the effect of the command using a preview.**

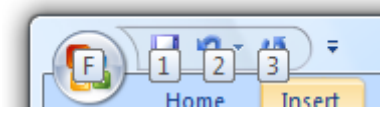
In this example, the tooltip image illustrates the effect of the command.

For labeling guidelines, see Enhanced tooltip labels.

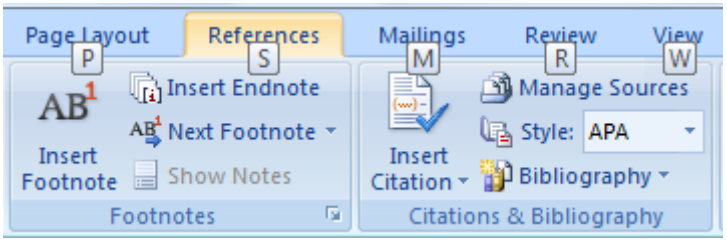
Access keys and keytips

Keytips are the mechanism used to display access keys for commands displayed directly on a ribbon.

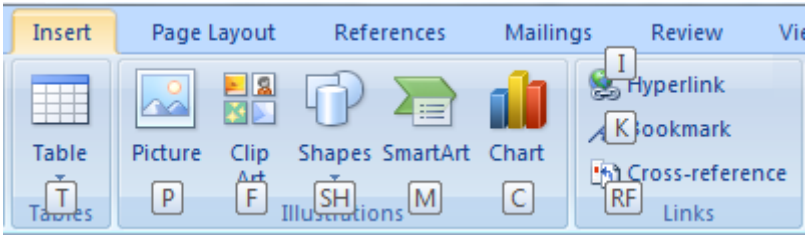
Access keys for drop-down menu commands are indicated with an underlined character. They differ from menu access keys in the following ways:

- Two character access keys can be used. For example, FP can be used to access the Format painter command.
- The access key assignments are shown using tips instead of underlines, so the character width and descenders aren't a factor in making assignments.
- **Assign access keys to all ribbon tabs and commands.** The only possible exception is for commands coming from legacy add-ins.
- **For the Application button and Quick Access Toolbar:**
  - Assign F to the Application button. This assignment is used because of the Application button's similarity to the traditional File menu.
  - 

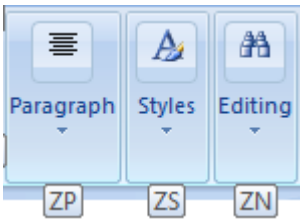
    For the Quick Access Toolbar and recently used file lists, assign access keys numerically.

- 

  For tabs:
  - Assign H to Home.
  - Starting with the most frequently used tabs, assign the first letter of the label.
  - For any tabs that cannot be assigned to the first letter, choose a distinctive consonant or a vowel in the label.
  - For programs that used to support menu bars, strive to maintain access key compatibility to the best extent practical. Avoid assigning different meanings to access keys from legacy menu categories. For example, if the legacy menu bar version of a program had an Edit menu, strive to use an E access key to the equivalent tab. If there is no equivalent tab, don't assign an E access key to any tab to prevent confusion.

- 

  For ribbon commands, menus, and submenus:
  - Assign unique access key combinations within a tab. You can reuse access key combinations within different tabs.
  - Whenever possible, assign the standard access keys for commonly used commands. See the standard access key table.
  - For other commands:
    - For the most frequently used commands, choose letters at the beginning of the first or second word of the label, preferably the first letter.
    - For less frequently used commands, choose letters that are a distinctive consonant or a vowel in the label, such as "x" in "Exit."
    - For the least frequently used commands and dialog box launchers, use two letters as necessary.
    - For menus and submenus, use a single letter to reduce the number of keystrokes required for the complete command.
    - Don't use access keys starting with J, Y, or Z because they are used for contextual tabs, unassigned keytips, and popup groups.

- 

  For pop-up groups:
  - Use a two-letter access key that starts with Z.
  - Starting with the most frequently used groups, assign the second access key letter to the first letter of the label.
  - For any remaining groups, choose a distinctive consonant or a vowel in the label.

For shortcut key guidelines, see Keyboard.

Application buttons

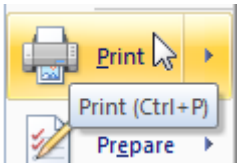
- **Use an Application button to present a menu of commands that involve doing something to or with a file.** Examples include commands that traditionally go in the File menu to create, open, and save files, print, and send and publish documents.

- **Always provide an Application button when using a ribbon.** If the program doesn't use files, use the Application button to access the program options and the Exit command. Application buttons always display a command menu they are never just decorative.

- **Use the following standard Application menu commands when appropriate:**
  - New
  - Open
  - Save
  - Save as...
  - Print...
  - Quick print
  - Print preview
  - Close
  - Options
  - Exit

- **Reserve commands that belong in the Application menu only for that menu.** Don't place them redundantly in any of the tabs.

- For each menu item, provide:
  - **A label with the command name.**

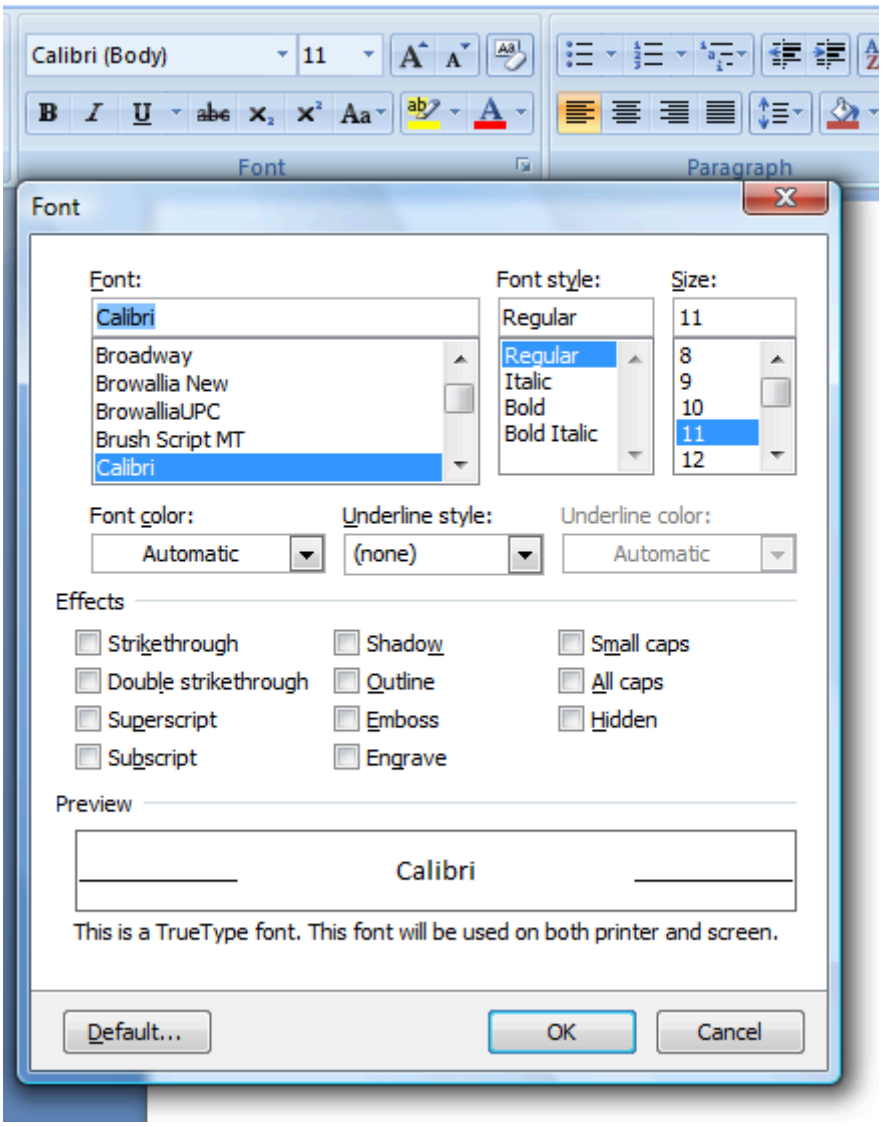
    - **A 32x32 pixel icon.**
    - **A brief description.** Make sure the description can be displayed using at most two lines of text.
- 

  Use tooltips to document the shortcut keys. Unlike normal menus, Application menus don't document the shortcut keys using labels.

Quick Access Toolbars

- **Use the Quick Access Toolbar to provide access to frequently used commands.** The commands can be from the Application button or the ribbon.
- **Always provide a Quick Access Toolbar when using a ribbon.** Do so even if the ribbon has a single tab; this provides consistency across programs.
- **Prepopulate the Quick Access Toolbar with the frequently used commands in the Application menu.** Provide Save and Undo if your program supports them, and Open and Print if supported and frequently used.
- **For the Customize Quick Access Toolbar menu, provide up to 12 of the most frequently used immediate commands.** Immediate commands don't require additional input before they take effect, and are therefore well-suited for the Quick Access Toolbar. While these can be any immediate commands, prefer those commands that aren't on the Home tab, because users are more likely to choose those.
- **For the Customize Quick Access Toolbar menu, if there is a pair of related commands, provide both, regardless of frequency.** Common pairs are Open/Close, Back/Forward, and Undo/Redo.
- **For the Customize Quick Access Toolbar dialog, provide a way to add any command.** Provide a Popular commands filter that displays the most frequently used commands, and select this filter by default.

Dialog box launchers

- 

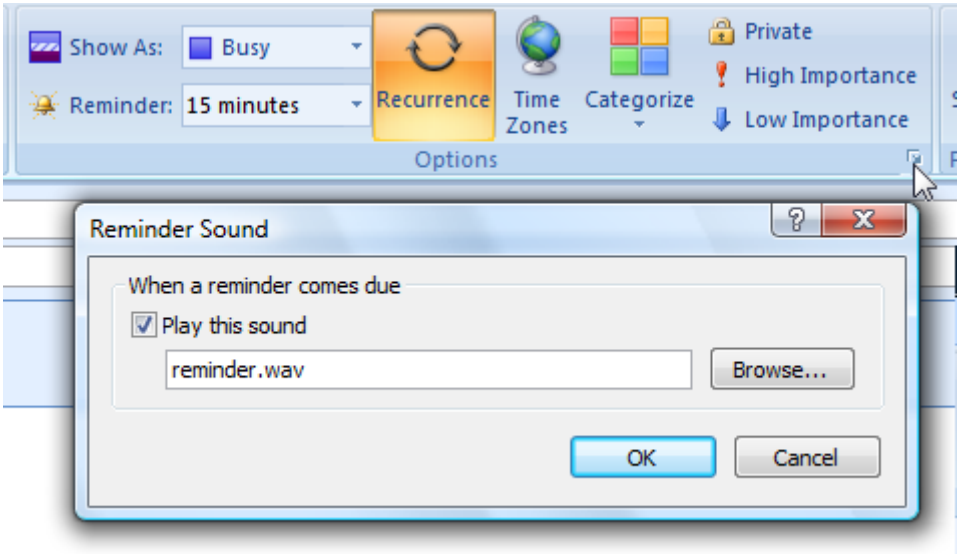
  Provide a group with a dialog box launcher if there is a related dialog box with infrequently used commands and settings. The dialog box should contain all the commands in the group, plus others not a completely different set of commands or the same commands as the group.

- **Don't use a dialog box launcher to perform commands directly.** A dialog box launcher must display a dialog box.

- **Don't use a dialog box launcher to access frequently used commands and settings.** Compared to commands directly on the ribbon, the dialog box commands and settings are relatively undiscoverable.

- **Match the name of the dialog box with the name of the group.** It doesn't have to be an exact match, but the names should be similar enough so that users aren't surprised by the results.

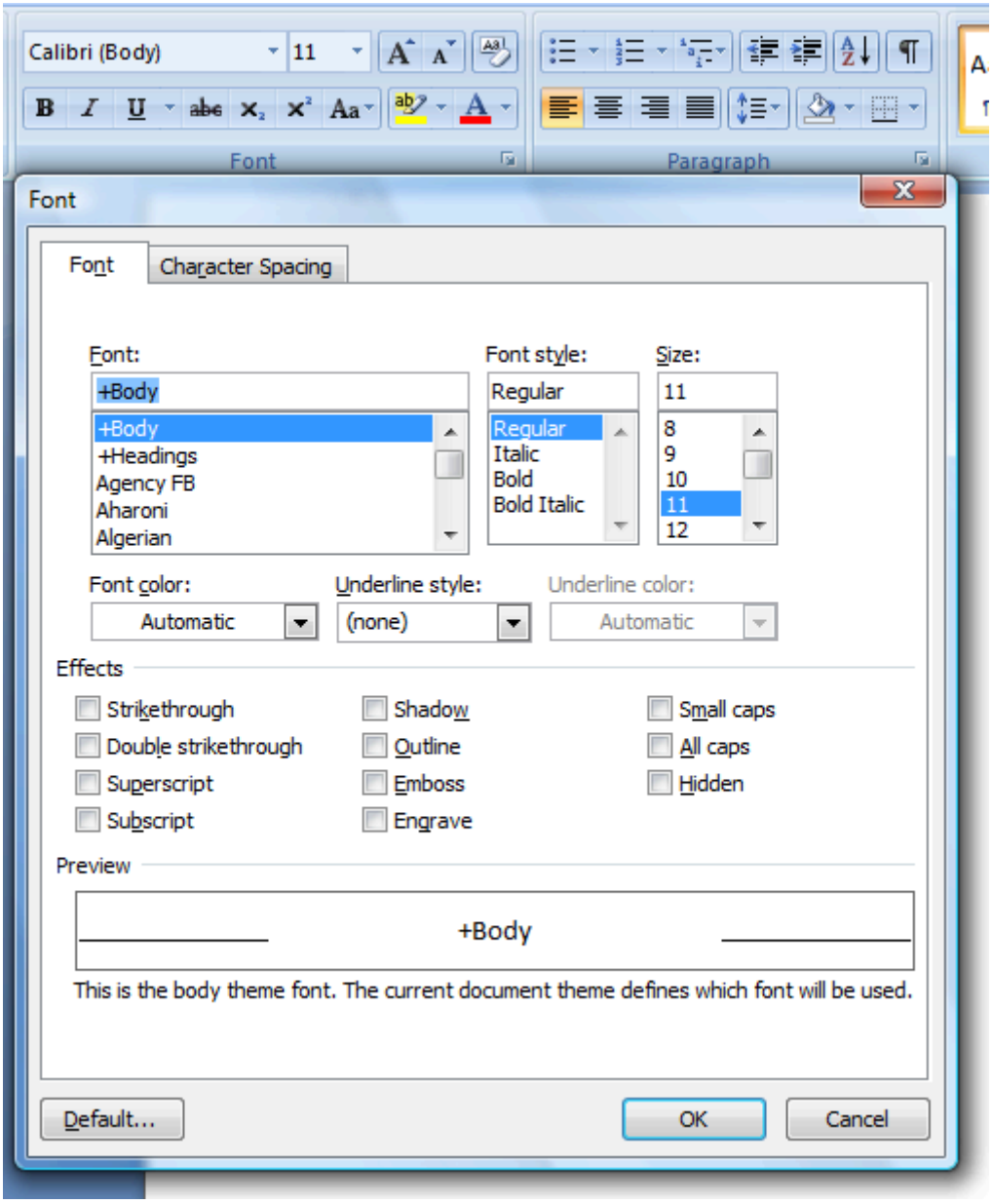
  Incorrect:



While a reminder sound is a reminder option, using the dialog box launcher to set the reminder sound is unexpected.

- **Display only the commands and settings that relate to the group.** If the dialog box displays other things, users may conclude that this path to these other commands and settings is the only path.

  Incorrect:



In this example, the Font dialog box displays Character Spacing settings, which are unrelated to the associated tab.

Labels

Tab labels

- **Label all tabs.**
- Whenever practical, **use the standard ribbon tabs**.
- **Prefer concise, single word labels.** While multi-word labels are acceptable, they take more space and are harder to localize.
- **Choose meaningful tab names that clearly and accurately describe their content.** The names should be specific, but not overly specific. Tab names should be predictable enough so that users aren't surprised by their content. Note that the

Home tab is generically named because it is used for the most frequently used commands.

- **Do not** use group names such as "Basic" and "Advanced." They require users to determine if the command they are looking for is basic or advanced.
- **Choose tab names that reflect their purpose.** Consider the goals or tasks associated with the tab.
- **Choose tab names that are clearly distinct from all the other tab names.**
- **Use either nouns or verbs for tabs.** Tab names don't require parallel phrasing, so choose the best label regardless of whether it's a noun or verb.
- **Don't use gerunds** (names that end in "-ing"). Use the verb from which the gerund is derived instead. (e.g. use "Draw" instead of "Drawing".)
- **Avoid tab names with the same initial letters, especially adjacent tabs.** When the ribbon is scaled down, these tab names will have the same truncated text.
- **Prefer singular names.** However, you can use a pural name if the singular name is awkward.
- **Use title-style capitalization.**
- **Don't use ending punctuation.**

Contextual tab and tab set labels

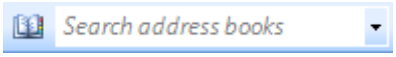
- **End contextual tab set labels with "Tools".** Doing so helps identify the purpose of contextual tabs.
- Use title-style capitalization.
- Don't use ending punctuation.

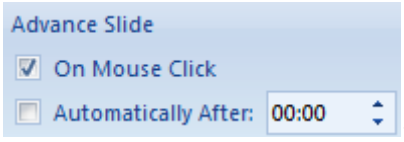
Group labels

- **Label all groups** unless the group has a single command and the group and command labels would be the same.
- **Use the standard ribbon groupsWhenever practical.**
- **Prefer concise, single word labels.** While multi-word labels are acceptable, they take more space and are harder to localize.
- **Choose meaningful group names that clearly and accurately describe their content.** The names should be specific, not generic.
- **Choose group names that reflect their purpose.** Consider the goals or tasks associated with the commands in the group.

- **Avoid using gerunds** (names that end in "-ing"). You can use gerunds, however, if using the verb from which the gerund is derived would be confusing. For example, use "Editing" and "Proofing" instead of "Edit" and "Proof."
- **Don't use group names that are the same as tab names.** Using the tab name that the group is on provides no information, and using the name of a different tab is confusing.
- **Prefer singular names.** However, you can use a pural name if the singular name is awkward.
- **Use sentence-style capitalization.**
- **Don't use ending punctuation.**

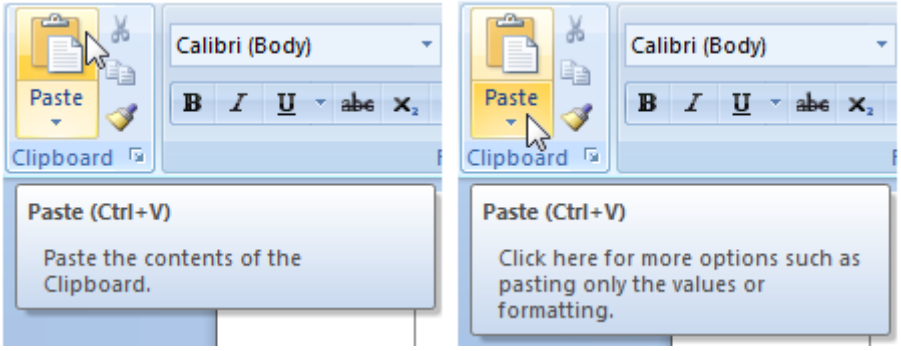
Command labels

- **Label all commands.** Having explicit text labels helps users find and understand commands. **Exception:** A command can be unlabeled if its icon is extremely well known and space is at a premium. Most likely, unlabeled commands will be on the Home tab. In this case, assign its Name property to an appropriate text label. This enables assistive technology products such as screen readers to provide users with alternative information about the graphic.
- **For command buttons, use a concise, self-explanatory label.** Use a single word if possible; four words maximum.
- **For drop-down lists, if the list always has a value, use the current value as the label.**
- 

If an editable drop-down list doesn't have a value, use a prompt.
- **Drop-down lists that aren't self-explanatory or are infrequently used need an explicit label.** Put a colon at the end of the label.
- 

<br.>**For text boxes, use an explicit label.** Put a colon at the end of the label.
- **Use sentence-style capitalization.** Doing so is more appropriate for the Windows tone.
- **Start the label with an imperative verb.** unless it's the same as the tab or group name or a common verb like Show, Create, Insert, or Format.
- **Don't use ending punctuation.**
- **To conserve space, don't put ellipses on ribbon command labels.** However, ellipses are used by commands in the Application button and drop-down menus.

Enhanced tooltip labels

- **Use the title to give the command name and its shortcut key, if applicable.**
- **For the title, don't use ending punctuation.**
- **Start the description with a verb.** Use the description to help users determine whether a specific feature is the one they are looking for. The description should be phrased to complete the sentence "This is the right feature to use if you want to...".
- **Keep the description short.** Get right to the point. Lengthy text discourages reading.
- 

  For split buttons, use a different tooltip to explain the split button menu.
- **Use an optional supplemental description to explain how to use the control.** This text can include information about the state of the control (including why it is disabled) if the control itself doesn't indicate state. Keep this text short, and use a Help topic for more detailed explanations.
-

  For the description and supplemental description, use complete sentences with ending punctuation.
- Use sentence-style capitalization.

Application button labels

- screen shot of quick print option selected
  Use "Quick" to indicate an immediate version of a command.

- **Use an ellipsis to indicate that a command requires more information.**
- **Use sentence-style capitalization.**

Documentation

When referring to ribbons:

- Refer to the ribbon and its components as ribbon, tabs, groups, and controls. These terms are not capitalized.
- Refer to the round button as the Application button, and the menu it contains as the Application menu.
- Refer to the toolbar as the Quick Access Toolbar.
- Refer to tabs by their labels and the word tab. Use the exact label text, including its capitalization.
- Refer to commands by their labels. Refer to unlabeled commands by their tooltip names. Use the exact label text, including its capitalization, but don't include the ellipsis. Don't include the word button or command.
- To describe user interaction, use click for tabs and controls. Use enter for editable drop-down lists. Don't use choose, select, or pick.
- Refer to unavailable items as unavailable, not as dimmed, disabled, or grayed. In programming documentation, use disabled.
- When possible, format the labels using bold text. Otherwise, put the labels in quotation marks only if required to prevent confusion.

Examples:

- On the **Home** tab, click **Paste special**.
- On the **Home** tab, in the **Font** box, enter "Segoe UI".
- On the **Review** tab, click **Show markup**, and then click **Reviewers**.
- On the **Format** tab, in **Picture tools**, click **Compress pictures**.

Feedback

Was this page helpful? Yes No

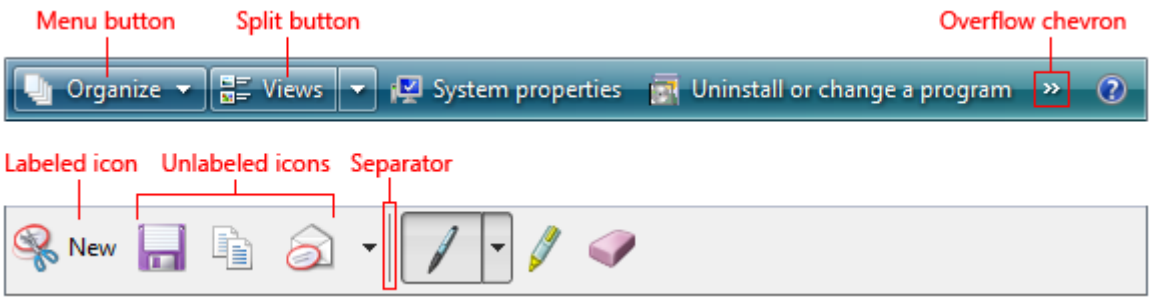
Provide product feedback | Get help at Microsoft Q&A

Windows 7 Toolbars

Article • 02/08/2022

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

Toolbars are a way to group commands for efficient access.



Some typical toolbars.

Use toolbars in addition to or in place of menu bars. Toolbars can be more efficient than menu bars because they are direct (always displayed instead of being displayed on mouse click), immediate (instead of requiring additional input) and contain the most frequently used commands (instead of a comprehensive list). In contrast to menu bars, toolbars don't have to be comprehensive or self-explanatory just quick, convenient, and efficient.

Some toolbars are customizable, allowing users to add or remove toolbars, change their size and location, and even change their contents. Some types of toolbars can be undocked, resulting in a palette window. For more information about toolbar varieties, see Usage patterns in this article.

> **Note**
>
> Guidelines related to menus, command buttons, and icons are presented in separate articles.

Is this the right user interface?

To decide, consider these questions:

- **Is the window a primary window?** Toolbars work well for primary windows, but are usually overwhelming for secondary windows. For secondary windows, use command buttons, menu buttons, and links instead.
- **Are there a small number of frequently used commands?** Toolbars can't handle as many commands as menu bars, so they work best as a way to efficiently access a small number of frequently used commands.
- **Are most of the commands immediate?** That is, are they mostly commands that don't require additional input? To be efficient, toolbars need to have a direct and immediate feel. If not, menu bars are better suited for commands that require additional input.
- **Can most of the commands be presented directly?** That is, users interact with them using a single click? While some commands can be presented using menu buttons, presenting most commands this way undermines the efficiency of the toolbar, making a menu bar a better choice.
- **Are the commands well represented by icons?** Toolbar buttons are usually represented by icons instead of text labels (although some toolbar buttons use both), whereas menu commands are represented by their text. If the command icons aren't high quality and aren't self-explanatory, a menu bar may be a better choice.

If your program has a toolbar without a menu bar, and most of the commands are accessible indirectly through menu buttons and split buttons, this toolbar is essentially a menu bar. Apply the toolbar menus pattern in the Menus guidelines instead.

Design concepts

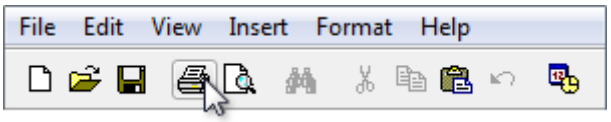
A good menu bar is a comprehensive catalog of all the available top-level commands, whereas a good toolbar gives quick, convenient access to frequently used commands. A toolbar doesn't attempt to train users just make them productive. Once users learn how to access a command on a toolbar, they rarely continue to access the command from the menu bar. For these reasons, a program's menu bar and its toolbar don't need to correspond directly.

Toolbars vs. menu bars

Traditionally, toolbars are different from menu bars in the following ways:

- **Frequency.** Toolbars present only the most frequently used commands, whereas menu bars catalog all the available top-level commands within a program.

- **Immediacy.** Clicking a toolbar command takes effect immediately, whereas a menu command might require additional input. For example, a Print command in a menu bar first displays the Print dialog, whereas a Print toolbar button immediately prints a single copy of a document to the default printer.

  In this example, clicking the Print toolbar button immediately prints a single copy of a document to the default printer.

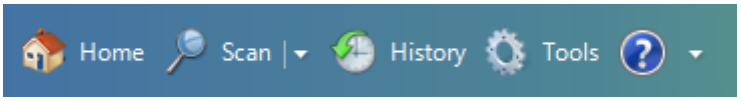
- **Directness.** Toolbar commands are invoked with a single click, whereas menu bar commands require navigating through the menu.

- **Number and density.** The screen space required by a toolbar is proportional to the number of its commands and that space is always used, even if the commands are not. Consequently, toolbars must use their space efficiently. By contrast, menu bar commands are normally hidden from view and their hierarchical structure allows for any number of commands.

- **Size and presentation.** To pack many commands in a small space, toolbars usually use icon-based commands (with tooltip-based labels), whereas menu bars use text-based commands (with optional icons). While toolbar buttons can have standard text labels, these do use significantly more space.

  Labeled toolbar buttons take at least three times as much space as unlabeled ones.

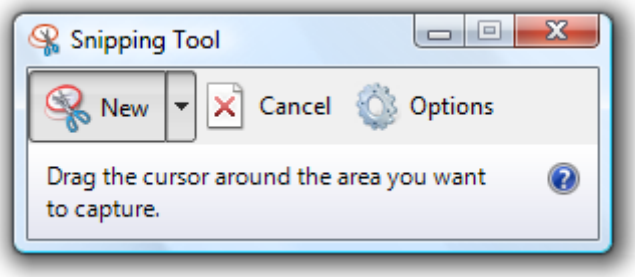
- **Self-explanatory.** Well-designed toolbars need icons that are mostly self-explanatory because users can't find commands efficiently just using tooltips. However, toolbars still work well if a few less frequently used commands aren't self-explanatory.

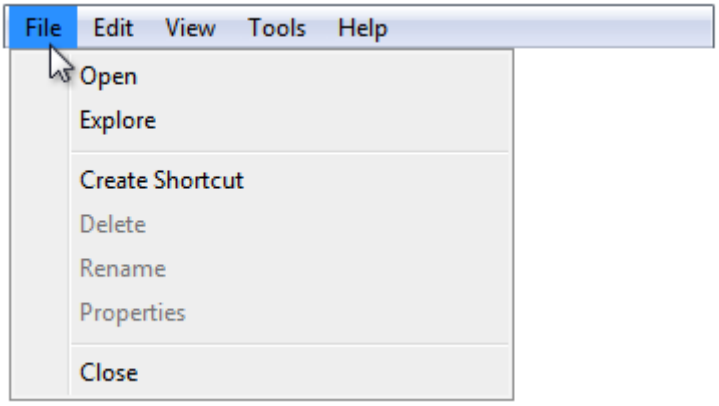
  In this example, the most frequently used icons are self-explanatory.

- **Recognizable and distinguishable.** For frequently used commands, users remember toolbar button attributes like location, shape, and color. With well-designed toolbars, users can find the commands quickly even if they don't remember the exact icon symbol. By contrast, users remember frequently used

menu bar command locations, but rely on the command labels for making selections.



For toolbar commands, distinctive location, shape, and color help make icons recognizable and distinguishable.



For menu bar commands, users ultimately depend upon their labels.

Efficiency

Given their characteristics, toolbars must be designed primarily for efficiency. An inefficient toolbar just doesn't make any sense.

If you do only one thing...

Make sure your toolbars are designed primarily for efficiency. Focus toolbars on commands that are frequently used, immediate, direct, and quickly recognizable.

Hiding menu bars

Generally, toolbars work great together with menu bars: good toolbars provide efficiency and good menu bars provide comprehensiveness. **Having both menu bars and toolbars allows each to focus on its strengths without compromise.**

Surprisingly, this model breaks down with simple programs. For programs with only a few commands, having both a menu bar and a toolbar doesn't make sense because the

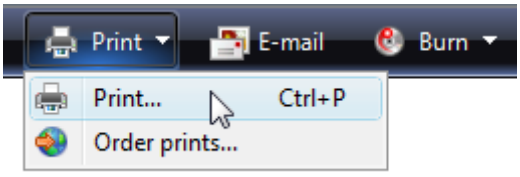
menu bar ends up being a redundant, inefficient version of the toolbar.

To eliminate this redundancy, many simple programs in Windows Vista focus on providing commands solely through the toolbar, and hiding the menu bar by default. Such programs include Windows Explorer, Windows Internet Explorer, Windows Media Player, and Windows Photo Gallery.

This is no small change. Removing the menu bar fundamentally changes the nature of toolbars because such toolbars need to be comprehensive and change in the following ways:

- **Frequency.** Removing the menu bar means that all commands not available directly from a window or its context menus must be accessible from the toolbar, regardless of their frequency of use.

- **Immediacy.** Removing the menu bar makes the toolbar the only visible access point for commands, requiring the toolbar to have the fully functional versions. For example, if there is no menu bar, a Print command on a toolbar must display the Print dialog box instead of printing immediately. (Although using a split button is an excellent compromise in this case. See Standard menu and split buttons for the standard Print split button.)

  In this example, the Print toolbar button in Windows Photo Gallery has a Print command that displays the Print dialog box.

- **Directness.** To save space and prevent clutter, less frequently used commands may be moved to menu buttons, making them less direct.

Toolbars used to supplement a menu bar are designed differently than toolbars designed for use with a removed or hidden menu bar. And because you can't assume that users will display a hidden menu bar to perform a single command, hiding a menu bar should be treated the same as removing it completely when making design decisions. (If you hide the menu bar by default, don't assume that users will think of displaying the menu bar to find a command or even figure out how to display it.)

Designing a toolbar to work without a menu bar often involves some compromises. But for efficiency, don't compromise too much. If hiding the menu bar results in an inefficient toolbar, don't hide the menu bar!

Keyboard accessibility

From the keyboard, accessing toolbars is quite different from accessing menu bars. Menu bars receive input focus when users press the Alt key and they lose input focus with the Esc key. Once a menu bar has input focus, it is navigated independently of the remainder of the window, handling all arrow keys, Home, End, and Tab keys. By contrast, toolbars receive input focus when users press the Tab key through the entire contents of the window. Because toolbars are last in tab order, they might take some significant effort to activate on a busy page (unless users know to use Shift+Tab to move backwards).

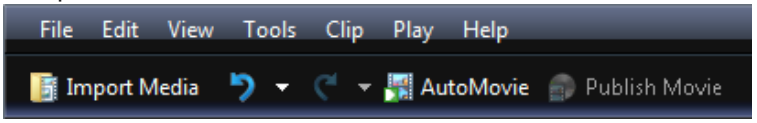
Accessibility presents a dilemma here: while toolbars are easier for mouse users, they are less accessible for keyboard users. This isn't a problem if there is both a menu bar and a toolbar, but it is if the menu bar is removed or hidden.

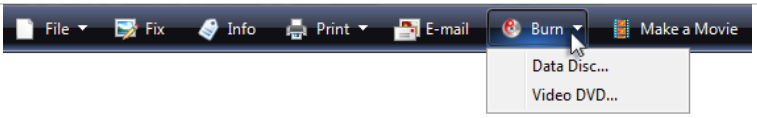
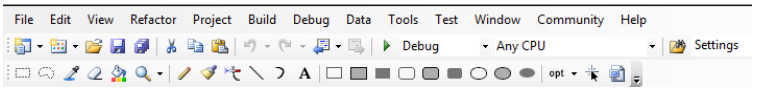
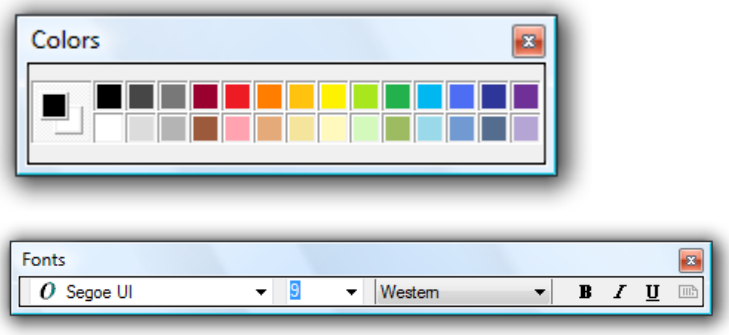
For accessibility reasons, then, prefer to retain the menu bar rather than remove it completely in favor of a toolbar. If you must choose between removing the menu bar and simply hiding it, prefer to hide it.

Usage patterns

Toolbars have several usage patterns:

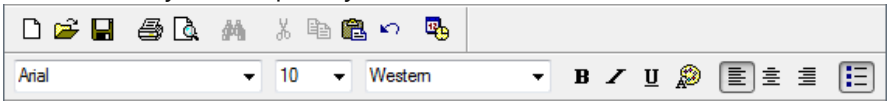
Expand table

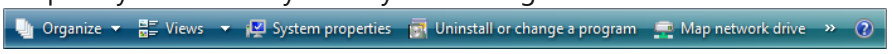
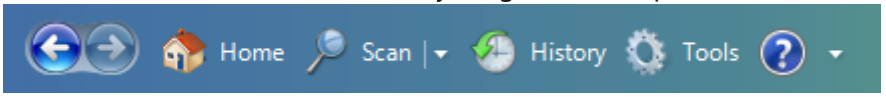
Usage	Example
Primary toolbars a toolbar designed to work without a menu bar, either hidden or removed.	primary toolbars must balance the need for efficiency with comprehensiveness, so they work best for simple programs. A primary toolbar from Windows Explorer.
Supplemental toolbars a toolbar designed to work with a menu bar.	supplemental toolbars can focus on efficiency without compromise.  A supplemental toolbar from Windows Movie Maker.
Toolbar menus a menu bar implemented as a toolbar.	toolbar menus are toolbars consisting primarily of commands in menu buttons and split buttons, with only a few direct commands, if any.

Usage	Example
	 A toolbar menu in Windows Photo Gallery.
Customizable toolbars a toolbar that can be customized by users.	customizable toolbars allow users to add or remove toolbars, change their size and location, and even change their contents.  A customizable toolbar from Microsoft Visual Studio.
Palette windows a modeless dialog box that presents an array of commands.	palette windows are undocked toolbars.  Palette windows from Windows Paint.

Toolbars have these styles:

Expand table

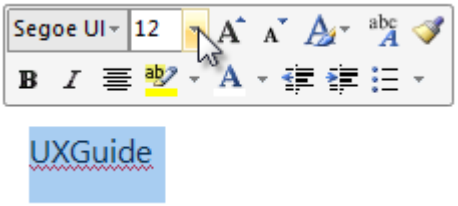
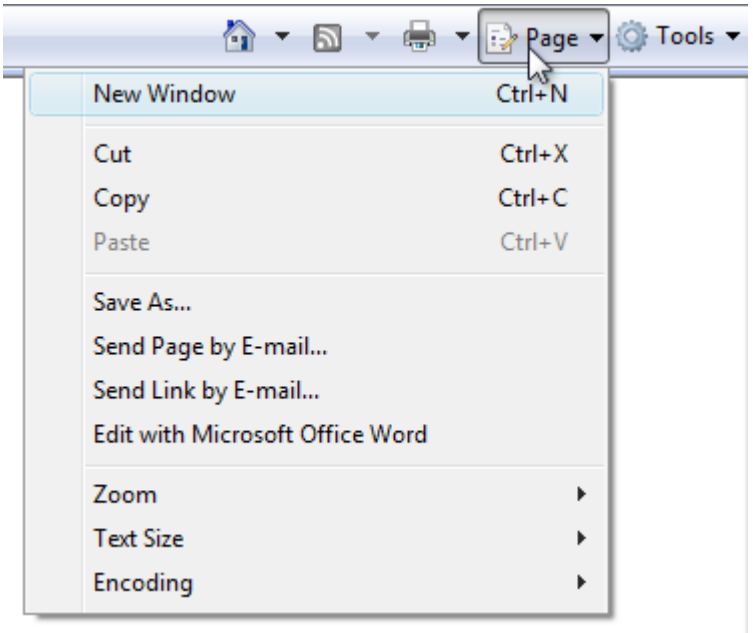
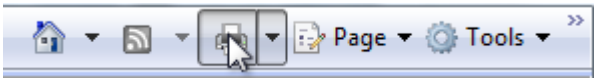
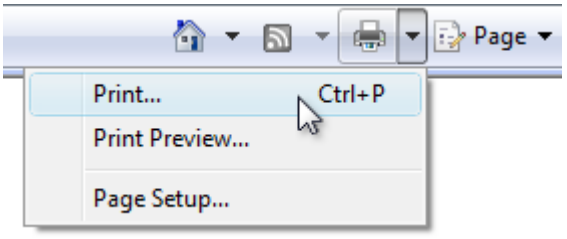
Style	Example
Unlabeled icons one or more rows of small unlabeled icon buttons.	use this style if there are too many buttons to label or the program is frequently used. with this style, programs with complex functionality can have multiple rows, and therefore, this is the only style that needs to be customizable. with this style, some command buttons can be labeled if they are frequently used.  An unlabeled icons toolbar from WordPad.
Large unlabeled icons a single row of large unlabeled icon buttons.	use this style for simple utilities that have easily recognizable icons and are usually run in small windows.

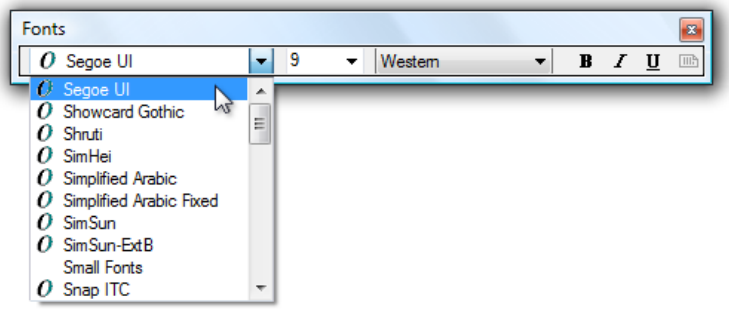
Style	Example
	Large unlabeled icons toolbars from Windows Live Messenger and the Windows Snipping Tool.
Labeled icons a single row of small labeled icons.	use this style if there are few commands or the program isn't frequently used. this style always has a single row.  A labeled icons toolbar from Windows Explorer.
Partial toolbars a partial row of small icons used to save space when a full toolbar isn't necessary.	use this style for windows with navigation buttons, a search box, or tabs to eliminate unnecessary weight at the top of the window. Partial toolbars can be combined with navigation buttons, a search box, or tabs.
Large partial toolbars a partial row of large icons used to save space when a full toolbar isn't necessary.	use this style for simple utilities that have navigation buttons or a search box to eliminate unnecessary weight at the top of the window.  A large partial toolbar from Windows Defender.

Finally, toolbar controls have several usage patterns:

Expand table

Usage	Example
Command icon buttons clicking a command button initiates an immediate action.	 Examples of icon command buttons from Windows Fax and Scan.
Mode icon buttons clicking a mode button enters the selected mode.	 Examples of mode buttons from Windows Paint.

Usage	Example
Property icon buttons a property button's state reflects the state of the currently selected objects, if any. clicking the button applies the change to the selected objects.	 Examples of property buttons from Microsoft Word.
Labeled icon buttons a command button or property button labeled with an icon and a text label.	these buttons are used for frequently used toolbar buttons whose icon isn't sufficiently self-explanatory. they are also used in toolbars that have so few buttons that each button can have a text label. A toolbar with its most frequently used buttons labeled.
Menu buttons a command button used to present a small set of related commands.	a single downward-pointing triangle indicates that clicking the button shows a menu.  A menu button with a small set of related commands.
Split buttons a command button used to consolidate variations of a command, especially when one of the commands is used most of the time.	 a split button in its normal state. like a menu button, a single downward-pointing triangle indicates that clicking the rightmost portion of the button shows a menu. 

Usage	Example
	a dropped down split button. in this example, a split button is used to consolidate all the print-related commands. the immediate print command is used most of the time, so users normally don't need to see the other commands. unlike a menu button, clicking the left portion of the button performs the action on the label directly. split buttons are effective in situations where the next command is likely to be the same as the last command. in this case, the label is changed to the last command, as with a color picker: In this example, the label is changed to the last command.
Drop-down lists a drop-down list (editable or read-only) used to view or change a property.	 In this example, drop-down lists are used to view and set font attributes. A drop-down list in a toolbar reflects the state of the currently selected object, if any. Changing the list changes the selected object's state.

Guidelines

Presentation

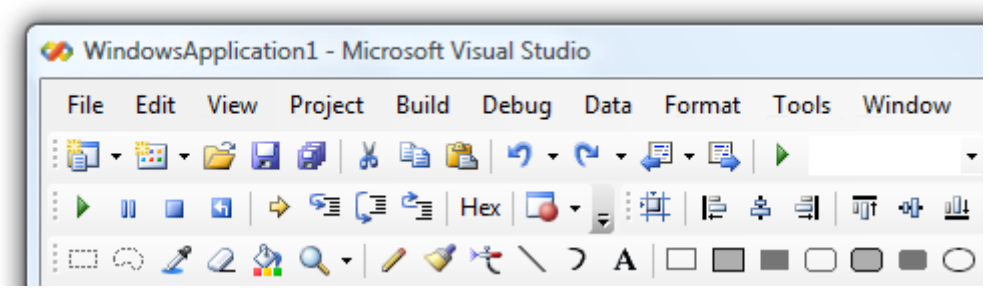
- **Choose a suitable toolbar style based on the number of commands and their usage.** See the previous toolbar style table for guidance on how to choose. Avoid using a toolbar configuration that takes too much space from the program work area.
- **Place toolbars just above the content area,** below the menu bar and address bar, if present.
- **If space is at a premium, save space by:**
  - Omitting the labels of well-known icons and less frequently used commands.
  - Using only a partial toolbar instead of the entire window width.
  - Consolidating related commands with a menu button or split button.

- Using an overflow chevron to reveal less frequently used commands.
- Displaying commands only when they apply to the current context.

The Windows Internet Explorer toolbar saves space by omitting labels of well-known icons, using a partial toolbar, and using an overflow chevron for less frequently used commands.

- **For the unlabeled icons toolbar pattern, use a default configuration with no more than two rows of toolbars.** If more than two rows might be useful, make the toolbars customizable. Starting with more than two rows can overwhelm users and take too much space from the program work area.

  Incorrect:

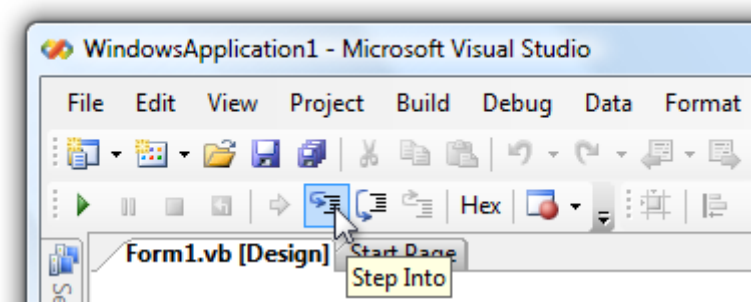


A default configuration with more than two rows of toolbars results in too much visual clutter.

- **Disable individual toolbar buttons that don't apply to the current context,** instead of removing them. Doing so makes toolbar contents stable and easier to find.

- **Disable individual toolbar buttons if clicking on them would directly result in an error.** Doing so is necessary to maintain a direct feel.

- **For the unlabeled icons toolbar pattern, remove entire toolbars if they don't apply to the current context.** Display them only in the applicable modes.



In this example, the Debug toolbar is shown only when the program is being run.

- **Display toolbar buttons left aligned.** The Help icon, if present, is right aligned.

  All toolbar buttons are left aligned except for Help.

  Exception: Windows 7-style toolbars left align program specific commands, but right align standard, well-known commands such as Options, View, and Help.

- **Don't change toolbar button labels dynamically.** Doing so is confusing and unexpected. However, you can change the icon to reflect the current state.

  In this example, the icon is changed to indicate the default command.

Controls and commands

- **Prefer the most frequently used commands.**
  - **For primary toolbars, provide comprehensive commands.** Primary toolbars don't have to be as comprehensive as menu bars, but they have to provide all the commands that aren't readily discoverable elsewhere. Primary toolbars don't need to have commands for:
    - Commands that are directly on the UI itself.
    - Commands typically accessed through context menus.
    - Standard, well-known commands like Cut, Copy, and Paste.
  - **For supplemental toolbars, provide commands that are used the most frequently.** Menu bar commands are a superset of the toolbar commands, so you don't have to provide everything. Focus on quick, convenient command access and skip the rest.

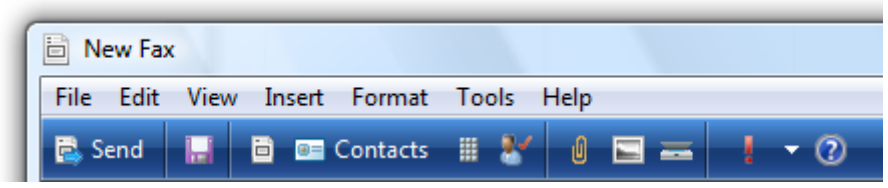
- **Prefer direct controls.** Use toolbar buttons in the following order of preference:
  - **Icon button.** Direct and takes minimal space.
  - **Labeled icon button.** Direct, but takes more space.
  - **Split button.** Direct for the most common command, but handles command variations.
  - **Menu button.** Indirect, but presents many commands.

- **Prefer immediate commands.** For commands that can either be immediate or have additional input for flexibility:
  - For primary toolbars, use the flexible versions of commands, (such as Print...).
  - For supplemental toolbars, use the immediate versions in the toolbar (such as Print) and use flexible versions in the menu bar (such as Print...).

- **Provide labels for frequently used commands,** especially if their icons aren't well-known icons.

  Acceptable:

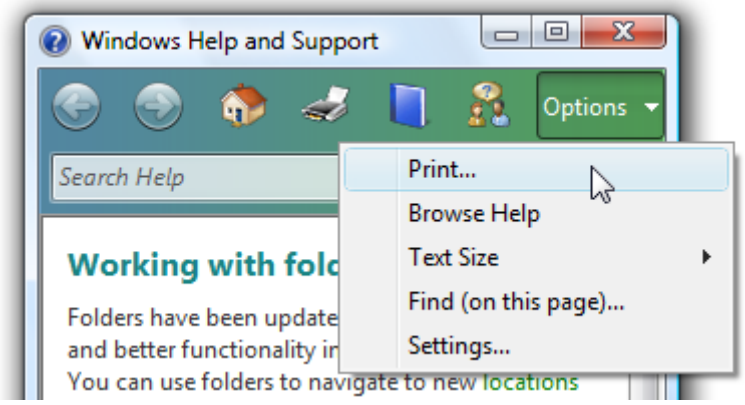
  Better:

  The Windows Fax and Scan toolbar has few commands, so the better version labels the most important ones.

- Don't put commands in toolbar menus that are also directly on the toolbar.

  Incorrect:

  In this example, Print is directly on the toolbar, so it doesn't need to be in the menu.

Organization and order

- **Organize the commands within a toolbar into related groups.**
- **Place the most frequently used groups first. Within a group, put the commands in their logical order.** Overall, the commands should have a logical flow to make them easy to find, while still having the most frequently used commands appear first. Doing so is most efficient, especially if there is overflow.

- **Use group dividers only if the commands across groups are weakly coupled.** Doing so makes the groupings obvious and the commands easier to find.

Examples of grouped toolbars from Windows Mail.

- **Avoid placing destructive commands next to frequently used commands.** Use either order or grouping to get separation. Also, consider not placing destructive commands in the toolbar, but only in the menu bar or context menus instead.

Acceptable:

Better:

In the better example, the Delete command is physically separated from Print.

- **Use the overflow chevron to indicate that not all commands can be displayed.** But use overflow only if there isn't sufficient room to display all the commands.

Incorrect:

The overflow chevron indicates that not all commands are displayed, but more of them could be with a better layout.

- **Make sure that the most frequently used commands are directly accessible from the toolbar (that is, not in overflow) in small window sizes.** If necessary, reorder the commands, move less frequently used commands to menu buttons or split buttons, or even remove them completely from the toolbar. If this remains a problem, reconsider your choice of toolbar style.

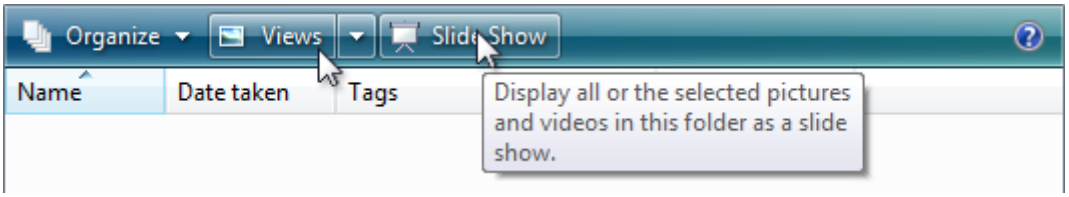
Hiding menu bars

Generally, toolbars work great together with menu bars because having both allows each to focus on their strengths without compromise.

- Hide the menu bar by default if your toolbar design makes having a menu bar redundant.
- Hide the menu bar instead of removing it completely, because menu bars are more accessible for keyboard users.
- To restore the menu bar, provide a Menu bar checkmark option in the View (for primary toolbars) or Tools (for secondary toolbars) menu category. For more information, see Standard menu and split buttons.
- Display the menu bar when users press the Alt key, and set input focus on the first menu category.

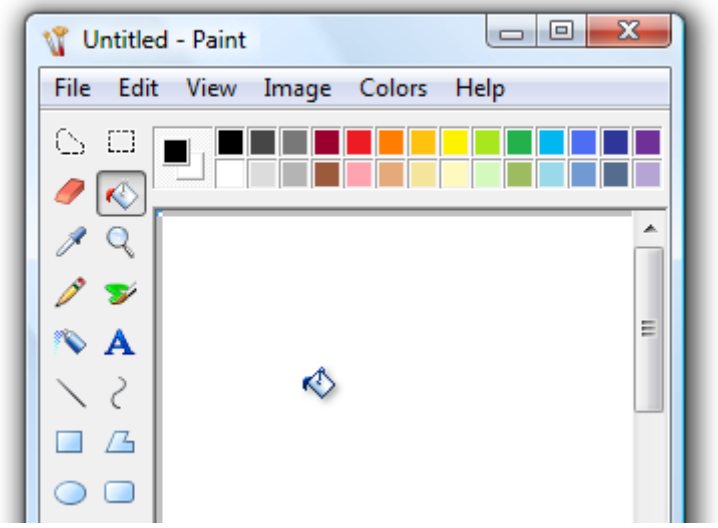
Interaction

- On hover, display the button affordance to indicate that the icon is clickable. After the tooltip timeout, display the tooltip or infotip.



  This example shows the various display states.

- On left single-click:
  - For command buttons, interact with the control as normal.
  - For mode buttons, display the control to reflect the currently selected mode. If the mode affects the behavior of mouse interaction, also change the pointer.

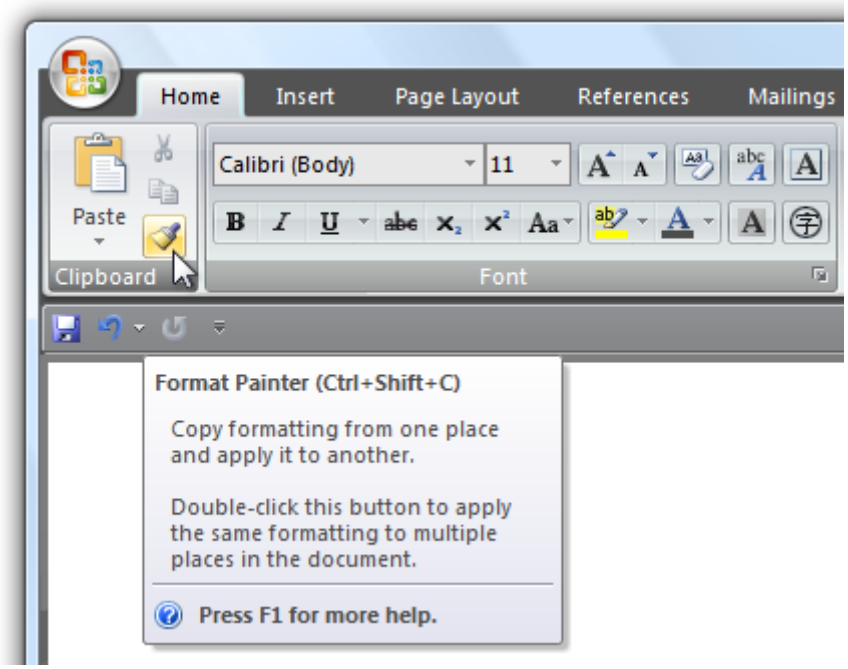


In this example, the pointer is changed to show the mouse interaction mode.

  - For property buttons and drop-down lists, display the control to reflect the state of the currently selected objects, if any. On interaction, update the control's state and apply the change to the selected objects. If nothing is selected, do nothing.

- On left double-click, perform the same action as a left single-click.

  - **Exception:** On rare occasions, a toolbar command can be used more efficiently modally. In such cases, use double-click to toggle the mode.

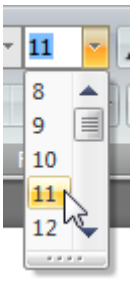


In this example, double-clicking the Format painter command enters a mode where all subsequent clicks apply the format. Users can leave the mode by left single-clicking.

- On right-click:
  - For customizable toolbars, display the context menu for customizing the toolbar. Display the menu on right-click on mouse down, not mouse up.
  - For other toolbars, do nothing.

Icons

- **Provide icons for all toolbar controls except drop-down lists.**



Drop-down lists don't need icons, but all other toolbar controls do.

Exception: Windows 7-style toolbars use icons only for commands whose icons are well known; otherwise they use text labels without icons. Doing so improves the clarity of the labels, but requires more space.

- **Make sure toolbar icons are clearly visible against the toolbar background color.** Always evaluate toolbar icons in context and in high-contrast mode.
- **Choose icon designs that clearly communicate their purpose, especially for the most frequently used commands.** Well-designed toolbars need icons that are self-explanatory because users can't find commands efficiently using their tooltips. However, toolbars still work well if icons for a few less frequently used commands aren't self-explanatory.
- **Choose icons that are recognizable and distinguishable, especially for the most frequently used commands.** Make sure the icons have distinctive shapes and colors. Doing so helps users find the commands quickly even if they don't remember the icon symbol.
- **Make sure toolbar icons conform to the Aero-style icon guidelines.**

For more information and examples, see Icons.

Standard menu and split buttons

If you are using menu buttons and split buttons in a toolbar, try to use the following standard menu structures and their relevant commands whenever possible. Unlike menu bars, toolbar commands don't take access keys.

Primary toolbars

These commands mirror the commands found in standard menu bars, so they should be used only for primary toolbars. This list shows the button labels (and type) with their order and separators, shortcut keys, and ellipses. **Note that the command for displaying and hiding the menu bar is in the View menu.**

File NewCtrl+N
Open...Ctrl+O
Close
<separator>
SaveCtrl+S
Save as...
<separator>
Send to
<separator>
Print...Ctrl+P
Print preview
Page setup
<separator>
ExitAlt+F4(shortcut usually not given)

Edit(menu button) UndoCtrl+Z
RedoCtrl+Y
<separator>
CutCtrl+X
CopyCtrl+C
PasteCtrl+V
<separator>
Select allCtrl+A
<separator>
DeleteDel(shortcut usually not given)
Rename...
<separator>
Find...Ctrl+F
Find nextF3(command usually not given)
Replace...Ctrl+H
Go to...Ctrl+G

Print(split button) Print...Ctrl+P
Print preview
Page setup

View(menu button) Menu bar(check if visible)
Details pane(check if visible)
Preview pane(check if visible)
Status bar(check if visible)

Zoom
Zoom inCtrl++
Zoom outCtrl+-

Text size(selected setting has bullet)

Largest
Larger
Medium
Smaller
Smallest

Full screenF11
RefreshF5

Tools(menu button) ...

Options Help(split button, use the Help icon) `<program name>` helpF1

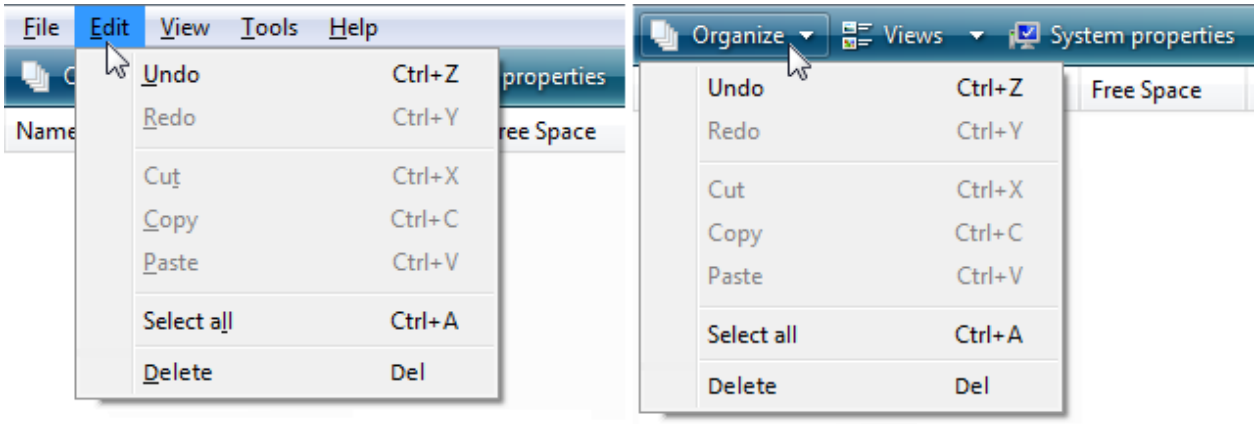
About `<program name>`

Supplemental toolbars

These commands supplement standard menu bars. This list shows the button labels (and type) with their order and separators, shortcut keys, and ellipses. **Note that the command for displaying and hiding the menu bar is in the Tools menu.**

The supplemental toolbar category names differ from the standard menu category names because they need to be more encompassing. For example, the Organize category is used instead of Edit because it contains commands that aren't related to editing. **To maintain consistency between menu bars and toolbars, use the standard menu category names if doing so wouldn't be misleading.**

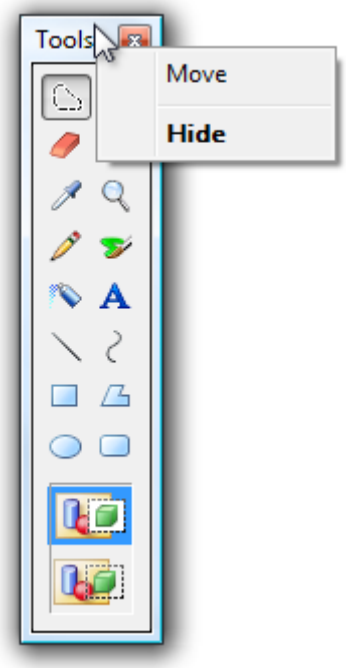
Incorrect:



In this example, the toolbar should use Edit instead of Organize for consistency because it has the standard Edit menu commands.

Palette windows

- **Palette windows use shorter title bars to minimize their screen space.** Put a Close button on the title bar.
- **Set the title bar text to the command that displayed the palette window.**
- **Use sentence-style capitalization without ending punctuation.**
- **Provide a context menu for window management commands.** Display this context menu when users right-click on the title bar.

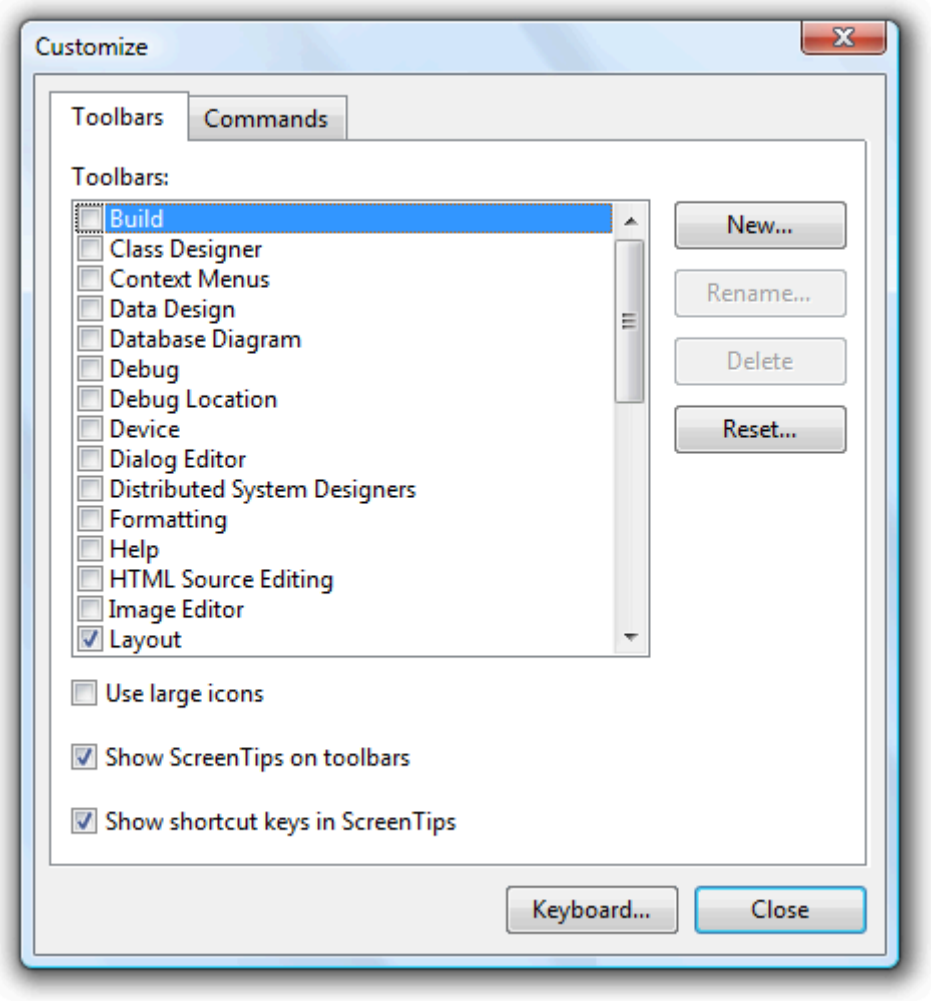


  In this example, users can right-click on the title bar to display the context menu.

- **When possible and useful, make palette windows resizable.** Indicate that the window is resizable, using resize pointers when over the window frame.
- **When a palette window is redisplayed, display it using the same state as last accessed.** When closing, save the window size and location. When redisplaying, restore the saved window size and location. Also, consider making these attributes persistent across program instances on a per user basis.

Customization

- **Provide customization for toolbars consisting of two or more rows.** Only the unlabeled icons style needs customization. Simple toolbars with few commands don't need customization.
- **Provide a good default configuration.** Users shouldn't have to customize their toolbars for common scenarios. Don't depend upon users customizing their way out of a bad initial configuration. Assume that most users won't customize their toolbars.
- **Provide a context menu with the following commands:**
  - A check box list to display the available toolbars
  - Lock/Unlock toolbars
  - Customize...
- **Lock customizable toolbars by default**, to prevent accidental changes.
- **For the Customize command, display an options dialog box** that provides the ability to choose which toolbars are displayed and the commands on each toolbar.



In this example, Visual Studio provides an options dialog box to customize its toolbars.

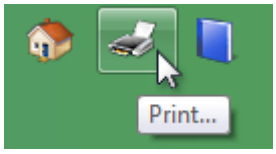
- Provide a Reset command to return to the original toolbar configuration in the Customize options dialog box.
- **Provide the ability to customize the toolbars using drag-and-drop in the following ways:**
  - Set toolbar order and positions.
  - Set toolbar lengths, displaying any toolbars that are too small to display their contents with an overflow chevron.
  - If supported, undock toolbars to become palette windows and vice versa.

  When the Customize options dialog box is displayed:
  - Set the toolbar contents.
  - Set the order of the toolbar contents.

  Doing so allows users to make changes more directly and efficiently.
- **Save all toolbar customizations,** on a per-user basis.

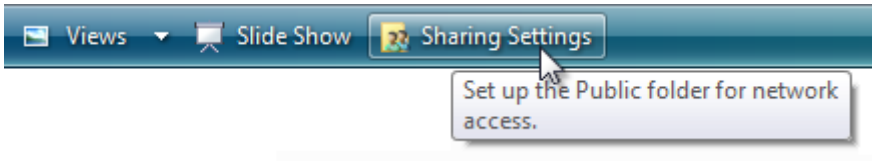
Using ellipses

While toolbar commands are used for immediate actions, sometimes more information is needed to perform the action. Use an ellipsis to indicate that a command requires more information before it can take effect. Put the ellipsis at the end of the tooltip and label, if there is one.



In this example, the Print... command displays a Print dialog box to gather more information.

If a command cannot take effect immediately, however, no ellipsis is required. So, for example, sharing settings doesn't have an ellipsis even though it needs additional information, because the command can't possibly take effect immediately.

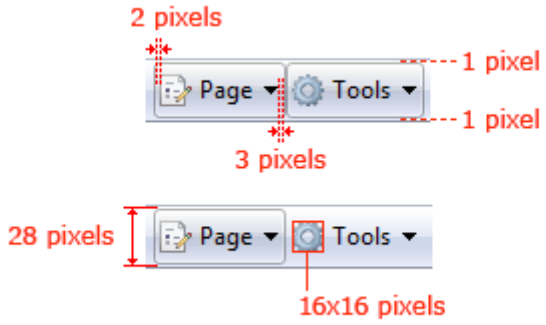


The Sharing Settings command doesn't have an ellipsis because it can't take effect immediately.

Because toolbars are constantly displayed, and space is at a premium, **ellipses should be used infrequently**.

> ⓘ Note
>
> For menus displayed by a toolbar, apply the menu ellipses guidelines.

Recommended sizing and spacing



Recommended sizing and spacing for standard toolbars.

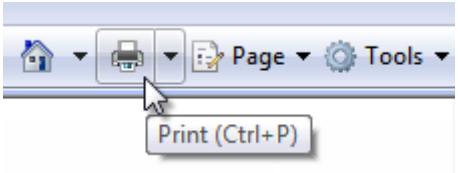
Labels

General

- **Use sentence-style capitalization.**
  - **Exception:** For legacy applications, you may use title-style capitalization if necessary to avoid mixing capitalization styles.

Unlabeled icon buttons

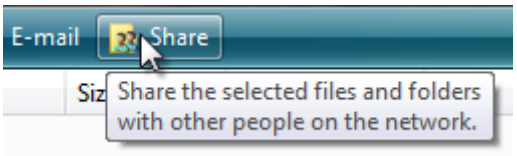
- **Use a tooltip to label the command.** For the tooltip text, use what the label would be if the button were labeled, but include the shortcut key if there is one.



An example of an icon button tooltip.

Labeled icon buttons

- **Use a concise label.** Use a single word if possible, four words maximum.
- **Place the label to the right of the icon.**
- **Use an infotip to describe the command.** Because the buttons are labeled, using a tooltip instead of an infotip would be redundant.



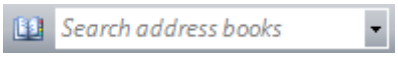
An example of a labeled icon button infotip.

Drop-down lists

- **If the list always has a value, use the current value as the label.**

In this example, the currently selected font name acts as the label.

- If an editable drop-down list doesn't have a value, use a prompt.



In this example, a prompt is used for the drop-down list's label.

Menu buttons and split buttons

- **Prefer verb-based menu button names.** However, omit the verb if it is Create, Show, View, or Manage. For example, **Tools** and **Page** menu buttons don't have verbs.
- **Use a single, specific word that clearly and accurately describes the menu contents.** While the names don't have to be so general that they describe everything in the menu, they should be predictable enough so that users aren't surprised by what they find in the menu.
- **While not required, provide infotip descriptions if they are helpful.**

Menu items

- **Use menu item names that start with a verb, noun, or noun phrase.**

- **Prefer verb-based menu names.** However, omit the verb if it is Create, Show, View, or Manage. For example, the following commands don't use verbs:
  - About
  - Advanced
  - Full screen
  - New
  - Options
  - Properties
- **Use specific verbs.** Avoid generic, unhelpful verbs, such as Change and Manage.
- **Use singular nouns for commands that apply to a single object,** otherwise use plural nouns.
- **For pairs of complementary commands, choose clearly complementary names.** Examples: Add, Remove; Show, Hide; Insert, Delete.
- **Choose menu item names based on user goals and tasks, not on technology.**
- Use the following menu item names for the stated purpose:
  - **Options:** To display program options.
  - **Customize:** To display the program options specifically related to mechanical UI configuration.
  - **Personalize:** To display a summary of commonly used personalization settings.
  - **Preferences:** Don't use. Use Options instead.
  - **Properties:** To display an object's property window.
  - **Settings:** Don't use as a menu label. Use Options instead.
- **Menu items that display submenus never have an ellipsis on their label.** The submenu arrow indicates that another selection is required.

Documentation

When referring to toolbars:

- If there is only one toolbar, refer to it as the toolbar.
- If there are multiple toolbars, refer to them by name, followed by the word toolbar. Refer to the main toolbar that is on by default and contains buttons for basic tasks, such as opening and printing a file, as the standard toolbar.
- Toolbar is a single, uncapitalized word. (By contrast, menu bar is two words.)
- Refer to toolbar buttons by their tooltip labels. Use the exact label text, including its capitalization, but don't include any ellipsis.
- Refer to toolbar menu buttons by their labels and the word menu. Use the exact label text, including its capitalization.
- Refer to toolbar controls generally as toolbar buttons.
- To describe user interaction, use click for toolbar buttons and read-only drop-down lists, and enter for editable drop-down lists. Don't use choose, select, or pick.

- Don't use cascading, pull-down, drop-down, or pop-up to describe menu buttons, except in programming documentation.
- Refer to unavailable items as unavailable, not as dimmed, disabled, or grayed. Use disabled in programming documentation.
- When possible, format the labels using bold text. Otherwise, put the labels in quotation marks only if required to prevent confusion.

Examples:

- On the **Page** menu on the toolbar, click **Send page by e-mail**.
- In the **Fonts** box on the toolbar, enter "Segoe UI."
- On the **Formatting** toolbar, point to **Show**, and then click **Comments**.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Text (Design basics)

Article • 02/09/2021

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

Text includes any text users can see in your app. Review these guidelines on the use of UI text, style, and tone for your apps.

In this section

Expand table

Topic	Description
User Interface Text	User interface text appears on UI surfaces.
Style and Tone	Tone in writing is the attitude that the writer conveys to the reader. It's not necessarily what you say, but how you say it. Tone is intended to create a specific response or emotion in the reader; it creates a personality that can either engage or repel users.

You can also find specific text guidelines in the Text or Labels sections for Controls and Windows.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

User Interface Text

Article • 10/18/2022

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

User interface text appears on UI surfaces. This text includes control labels and static text:

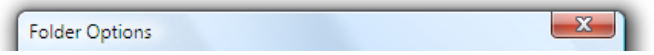
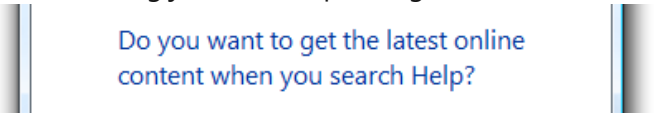
- Control labels identify controls and are placed directly on or next to the controls.
- Static text, which is so called because it is not part of an interactive control, provides users with detailed instructions or explanations so they can make informed decisions.

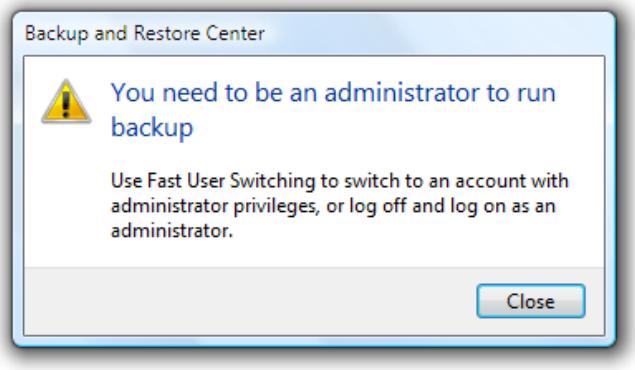
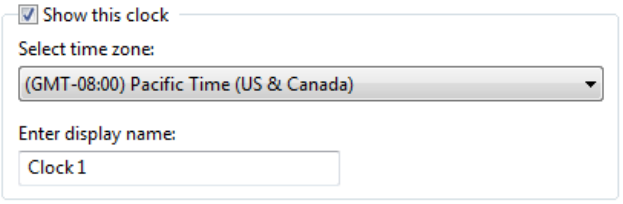
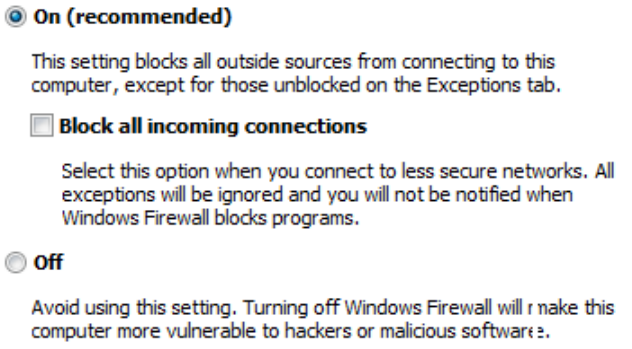
Note: Guidelines related to style and tone, fonts, and comon control labels are presented in separate articles.

Usage patterns

UI text has several usage patterns:

Expand table

Usage	Description
Title bar text use title bar text to identify a window or the source of a dialog box.	 In this example, the title bar text identifies a window.
Main instructions use the prominent main instruction to explain concisely what to do in the window or page.	The instruction should be a specific statement, imperative direction, or question. Good main instructions communicate the user's objective rather than focusing just on manipulating the UI.  In this example, the main instruction text directly engages the user with a question in terms of the user's own benefit or interest.

Usage	Description
Supplemental instructions when necessary, use a supplemental instruction to present additional information helpful to understanding or using the window or page.	You can provide more detailed information, provide context, and define terminology. Supplemental instructions elaborate on the main instruction without simply re-wording it.  In this example, the supplemental instructions provide two possible courses of action to take in response to the information presented in the main instruction.
Control labels labels directly on or next to controls.	 In this example, control labels identify desktop clock settings that users can select or modify.
Supplemental explanations an elaboration of the control labels (typically for command links, radio buttons, and check boxes).	 In this example, the supplemental explanations clarify the choices.

Design concepts

Software developers often think of text as relegated to product documentation and technical support. "First we'll write the code, and then we'll hire someone to help us explain what we have developed." Yet in reality, important text is written earlier in the

process, as the UI is conceived and coded. This text is, after all, seen more frequently and by more people than perhaps any other type of technical writing.

Comprehensible text is crucial to effective UI. Professional writers and editors should work with software developers on UI text as an integral part of the design process. Have them work on text early because text problems often reveal design problems. If your team has trouble explaining a design, quite often it is the design, not the explanation, that needs improving.

A design model for UI text

As you think about UI text and its placement on your UI surfaces, consider these facts:

- During focused, immersive reading, people read in a left-to-right, top-to-bottom order (in Western cultures).
- When using software, users aren't immersed in the UI itself but in their work. Consequently, users don't read UI text they scan it.
- When scanning a window, users may appear to be reading text when in reality they are filtering it. They often don't truly comprehend the UI text unless they perceive the need to.
- Within a window, different UI elements receive different levels of attention. Users tend to read control labels first, especially those that appear relevant to completing the task at hand. By contrast, users tend to read static text only when they think they need to.

For a general design model, don't assume that users carefully read the text in a left-to-right, top-to-bottom order. Rather, assume that users start by quickly scanning the whole window, then read UI text in roughly the following order:

- Interactive controls in the center
- The commit buttons
- Interactive controls found elsewhere
- Main instruction
- Supplemental explanations
- Window title
- Other static text in main body
- Footnotes

You should also assume that once users have decided what to do, they will immediately stop reading and do it.

Eliminate redundancy

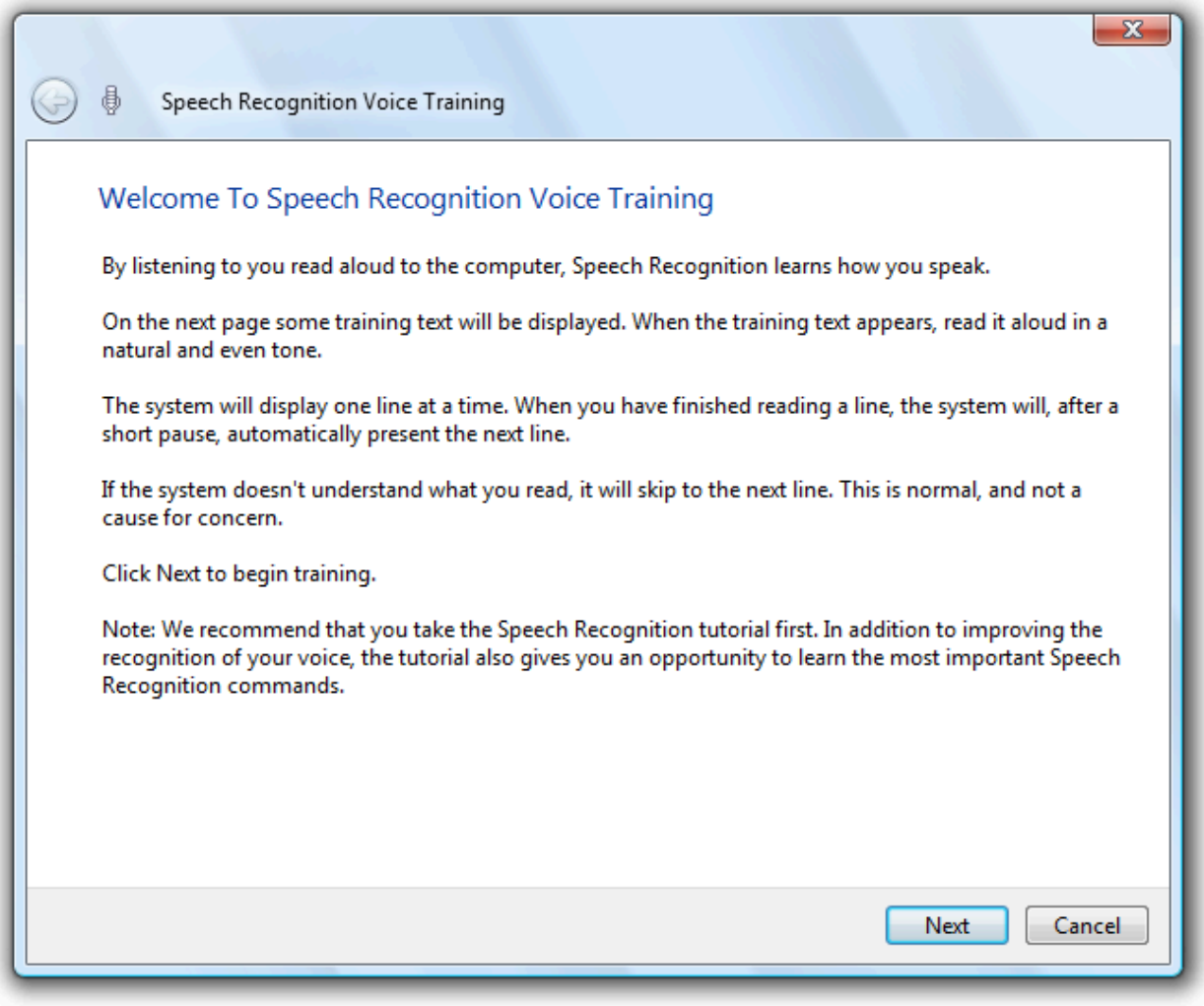
Redundant text not only takes valuable screen space, but weakens the effectiveness of the important ideas or actions that you are trying to convey. It is also a waste of the reader's time, and all the more so in a context where scanning is the norm. **Windows strives to explain what users need to do once well and concisely.**

Review each window and eliminate duplicate words and statements, both within and across controls. Don't avoid important text; be explicit wherever necessary, but don't be redundant and don't explain the obvious.

Avoid over-communication

Even if text isn't redundant, it can simply be too wordy in an effort to explain every detail. **Too much text discourages reading; the eye tends to skip right over it ironically resulting in less communication rather than more.** In UI text, concisely communicate the essential information. If more information is necessary for some users or some scenarios, provide a link to more detailed Help content, or perhaps to a glossary entry for clarification of a term.

Incorrect:



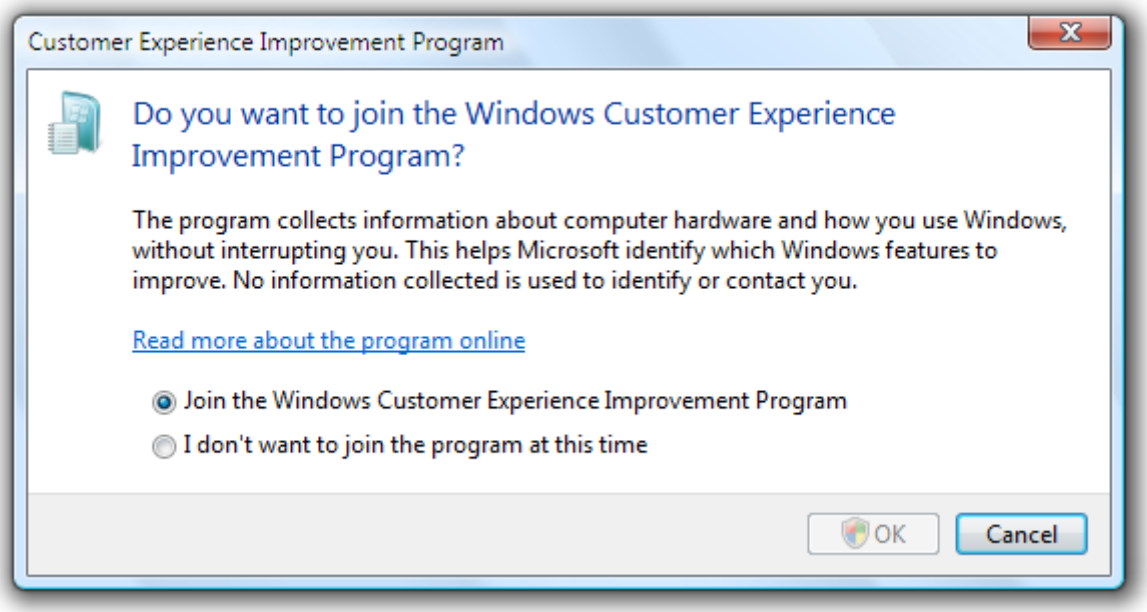
In this example, there is too much text to scan easily. Although not intended by the designer, there is so much text that users will most likely click Next without reading anything.

To avoid text that discourages reading, craft your text to make every word count. What doesn't add subtracts, so use simple, concise text.

Use the inverted pyramid

Academic writing typically uses a "pyramid" structural style that lays down a foundation of facts, works with those facts, and builds up to a conclusion forming a pyramid-like structure. By contrast, journalists use an "inverted pyramid" style that starts with the conclusion as the fundamental "takeaway" that readers must have. It then fills in progressively more detail that readers may be interested in perhaps just to scan. The advantage of this style is that it gets right to the point, and allows readers to stop reading at any point they choose and still understand the essential information.

You should apply the inverted pyramid structure to UI text. Get right to the point with the essential information, let users stop reading at any time they choose, and use a Help link to present the remainder of the pyramid.



In this example, the essential information is in the query of the main instruction text, additional helpful information is in the supplemental instructions, and details are available by clicking a Help link.

If you do only five things...

1. Work on text early because text problems often reveal design problems.
2. Design your text for scanning.

3. Eliminate redundant text.
4. Use easy-to-understand text; don't over-communicate.
5. When necessary, provide links to Help content for more detailed information.

Guidelines

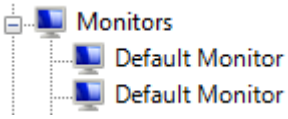
General

- **Remove redundant text.** Look for redundant text in window titles, main instructions, supplemental instructions, content areas, command links, and commit buttons. Generally, leave full text in main instructions and interactive controls, and remove any redundancy from the other places.

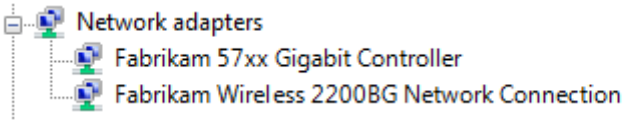
- **Avoid large blocks of UI text.** Ways of doing this include:
  - Chunking text into shorter sentences and paragraphs.
  - When necessary, providing Help links to useful, but not essential, information.

- **Choose object names and labels that clearly communicate and differentiate what the object does.** Users shouldn't have to figure out what the object really means or how it differs from other objects.

  Incorrect:

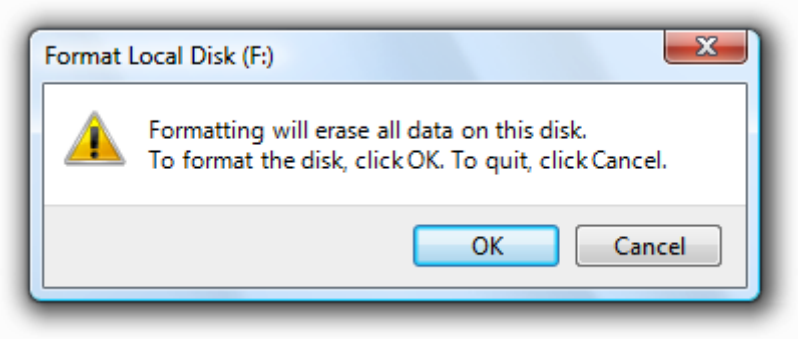
  

  Better:

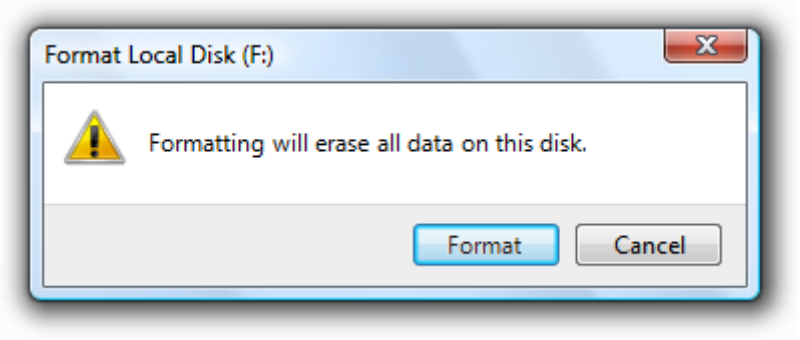
  

  In the incorrect example, the object names are not differentiated at all; the better example shows strong differentiation by product name.

- **If you want to make sure that users read specific text related to an action, place it on an interactive control.**
  - **Acceptable:**



- In this example, there's a chance that users won't read the text that explains what they're confirming.
- **Better:**



- In this example, you can be sure that at least users understand that they are about to format a disk.

- **Use one space between sentences.** Not two.

Text fonts, sizes, and colors

- **Use blue text only for links and main instructions.**
- **Use green text only for URLs in search results.**

The following fonts and colors are defaults for Windows.

Expand table

Pattern	Theme symbol	Font, Color
Title bar text	CaptionFont	9 pt. black (#000000) Segoe UI
Main instructions	MainInstruction	12 pt. blue (#003399) Segoe UI
Secondary instructions	Instruction	9 pt. black (#000000) Segoe UI
Normal text	BodyText	9 pt. black (#000000) Segoe UI
Emphasized text	BodyText	9 pt. black (#000000) Segoe UI, bold or

Pattern	Theme symbol	Font, Color
		italic
Editable text	BodyText	9 pt. black (#000000) Segoe UI, in a box
Disabled text	Disabled	9 pt. dark gray (#323232) Segoe UI
Link	HyperLinkText	9 pt. blue (#0066CC) Segoe UI
Links (Hover)	Hot	9 pt. light blue (#3399FF) Segoe UI
Group header		11 pt. blue (#003399) Segoe UI
File name (in content view)		11 pt. black (#000000) Segoe UI
Document text	(none)	9 pt. black (#000000) Calibri
Document headings	(none)	17 pt. black (#000000) Calibri

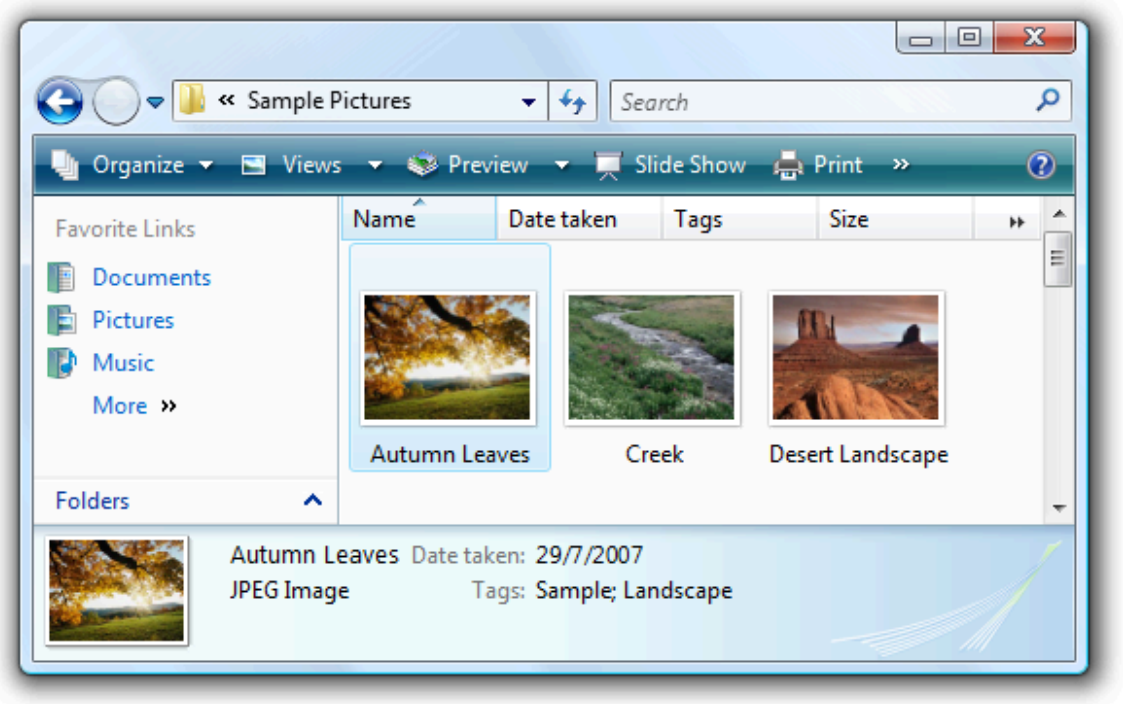
For more information and examples, see Fonts and Color.

Other text characteristics

Bold

- **Use bold sparingly to draw attention to text users must read.** For example, users scanning down a list of radio button options may appreciate seeing the labels in bold, to stand out from text that adds supplemental information about each option. Be aware that using too much bold lessens its impact.
- **With labeled data, use bold to emphasize whichever is more important for the data as a whole.**
  - For mostly generic data (where the data has little meaning without its labels, as with numerals or dates), use bold labels and plain data so that users can more easily scan and understand the types of data.
  - For mostly self-explanatory data, use plain labels and bold data so that users can focus on the data itself.
  - Alternatively, you can use dark gray text to de-emphasize less important information instead of using bold to emphasize the more important

information.



In this example, instead of emphasizing the data using bold, the labels are de-emphasized by using dark gray.

- **Not all fonts support bold, so it should never be crucial to understanding the text.**

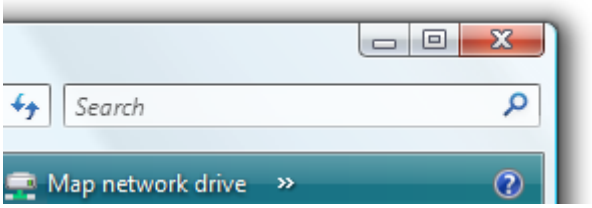
Italic

- Use to refer to text literally. Don't use quotation marks for this purpose.

  Correct:

  The terms document and file are often used interchangeably.

- Use for prompts in text boxes and editable drop-down lists.



In this example, the prompt in the Search box is formatted as italic text.

- Use sparingly to emphasize specific words to aid in comprehension.
- **Not all fonts support italic, so it should never be crucial to understanding the text.**

Bold italic

- Don't use in UI text.

Underline

- Don't use, except for links.
- Don't use for emphasis. Use italic instead.

Punctuation

Periods

- **Don't place at the end of control labels, main instructions, or Help links.**
- Place at the end of supplemental instructions, supplemental explanations, or any other static text that forms a complete sentence.

Question marks

- **Place at the end of all questions.** Unlike periods, question marks are used for all types of text.

Exclamation points

- In business applications, avoid.
  - **Exceptions:** Exclamation points are sometimes used in the context of download completion ("Done!") and to call attention to Web content ("New!").

Commas

- In a list of three or more items, always put a comma after the next-to-last item in the list.

Colons

- **Use colons at the end of external control labels.** This is particularly important for accessibility because some assistive technologies look for colons to identify control labels.
- Use a colon to introduce a list of items.

Ellipses

- **Ellipses mean incompleteness.** Use ellipses in UI text as follows:
  - **Commands:** Indicate that a command needs additional information. Don't use an ellipsis whenever an action displays another window only when additional

information is required. For more information, see Command Buttons.

  - **Data:** Indicate that text is truncated.
  - **Labels:** Indicate that a task is in progress (for example, "Searching...").

    Tip: Truncated text in a window or page with unused space indicates poor layout or a default window size that is too small. Strive for layouts and default window sizes that eliminate or reduce the amount of truncated text. For more information, see Layout.

- **Don't make ellipses interactive.** To show truncated text, let users resize the control to see more text or use a progressive disclosure control instead.

Quotation marks and apostrophes

- To refer to text literally, use italic formatting rather than quotation marks.
- Put window titles and control labels in quotation marks only if required to prevent confusion and you can't format using bold instead.
- For quotation marks, prefer double-quotation marks (" "); avoid single-quotation marks.

  Correct:

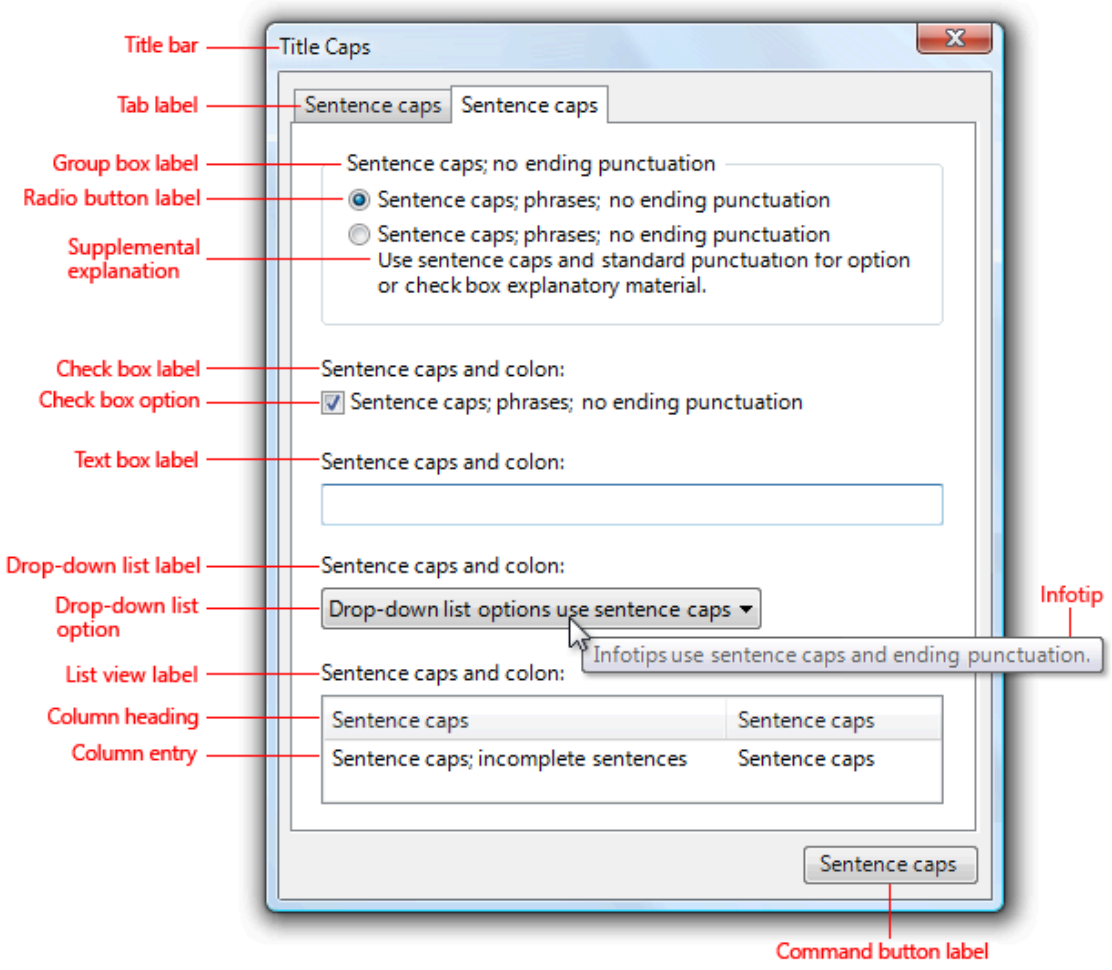
  Are you sure you want to delete "Sparky's cat folder"?

  Incorrect:

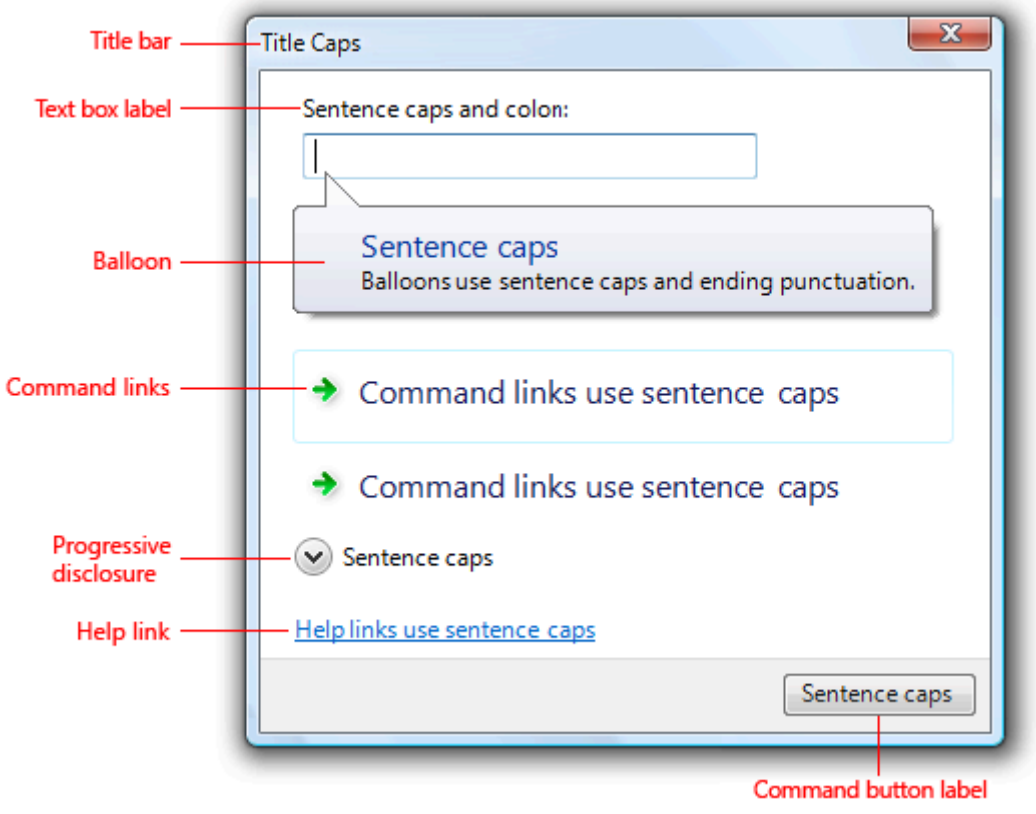
  Are you sure you want to delete 'Sparky's cat folder'?

Capitalization

- **Use title-style capitalization for titles, sentence-style capitalization for all other UI elements.** Doing so is more appropriate for the Windows tone.
  - **Exception:** For legacy applications, you may use title-style capitalization for command buttons, menus, and column headings if necessary to avoid mixing capitalization styles.



This generic example shows correct capitalization and punctuation for property sheets.



This generic example shows correct capitalization and punctuation for dialogs.

- **For feature and technology names, be conservative in capitalizing.** Typically, only major components should be capitalized (using title-style capitalization).

  Correct:

  Analysis Services, cubes, dimensions

  Analysis Services is a major component of SQL Server, so title-style capitalization is appropriate; cubes and dimensions are common elements of database analysis software, so it is unnecessary to capitalize them.

- **For feature and technology names, be consistent in capitalizing.** If the name appears more than once on a UI screen, it should always appear the same way. Likewise, across all UI screens in the program, the name should be consistently presented.

- Don't capitalize the names of generic user interface elements, such as toolbar, menu, scroll bar, button, and icon.
  - **Exceptions:** Address bar, Links bar.

- Don't use all capital letters for keyboard keys. Instead, follow the capitalization used by standard keyboards, or lowercase if the key is not labeled on the keyboard.

  Correct:

  spacebar, Tab, Enter, Page Up, Ctrl+Alt+Del

  Incorrect:

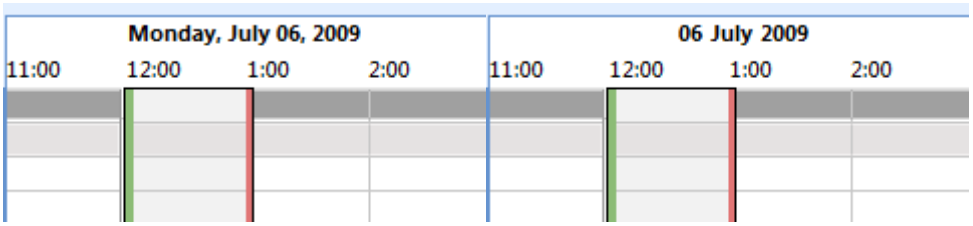
  SPACEBAR, TAB, ENTER, PG UP, CTRL+ALT+DEL

- **Don't use all capital letters for emphasis.** Studies have shown that this is hard to read, and users tend to regard it as "screaming." For warnings, use a warning icon and a clearly-worded explanation of the situation. There is no need to add, for example, the term WARNING in all capital letters.

For more information, see the "Text" or "Labels" section in the specific UI component guidelines.

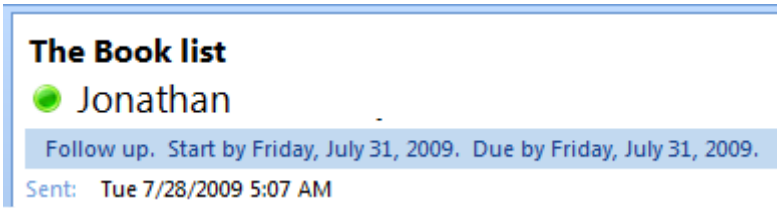
Dates and times

- **Don't hard-code the format of dates and times.** Respect the user's choice of locale and customization options for the date and time formats. The user selects these in the Region and Language control panel item.



In these examples from Microsoft Outlook, both formats for the long date are correct. They reflect different choices users have made in the Region and Language control panel item.

- **Use the long date format for scenarios that benefit from having additional information.** Use the short date format for contexts that don't have sufficient space for the long format. While users choose what information they would like to include in the long and short formats, designers choose which format to display in their programs based on the scenario and the context.



In this example, the long date format helps users organize tasks and deadlines.

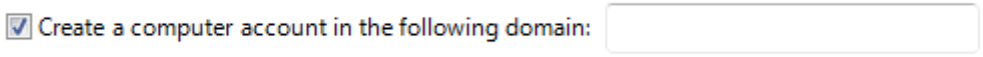
Globalization and localization

Globalization means to create documents or products that are usable in any country, region, or culture. Localization means to adapt documents or products for use in a locale other than the country/region of origin. Consider globalization and localization when writing UI text. Your program may be translated into other languages and used in cultures very different from your own.

- For controls with variable contents (such as list views and tree views), **choose a width appropriate for the longest valid data.**
- **Include space enough in the UI surface for an additional 30 percent** (up to 200 percent for shorter text) for any text (but not numbers) that will be localized. Translation from one language to another often changes line length of text.
- Don't compose strings from substrings at run time. Instead, use complete sentences so that there is no ambiguity for the translator.
- **Don't use a subordinate control, the values it contains, or its units label to create a sentence or phrase.** Such a design is not localizable because sentence structure varies with language.

Incorrect:

Correct:



In the incorrect example, the text box is placed inside the check box label.

- Don't make only part of a sentence a link, because when translated, that text might not remain together. Link text should therefore form a complete sentence by itself.
  - **Exception:** Glossary links can be inserted inline, as part of a sentence.

For more information, see the Go Global Developer Center.

Title bar text

- Choose the title bar text based on the type of window:
  - **Top-level, document-centric program windows:** Use a "document name program name" format. Document names are displayed first to give a document-centric feel.
  - **Top-level program windows that are not document-centric:** Display the program name only.
  - **Dialog boxes:** Display the command, feature, or program from which the dialog box came. Don't use the title to explain the dialog box's purpose that's the purpose of the main instructions. For more guidelines, see Dialog Boxes.
  - **Wizards:** Display the wizard name. Note that the word "wizard" should not be included in wizard names. For more guidelines, see Wizards.
- **For top-level program windows, if the title bar caption and icon are displayed prominently near the top of the window, you can hide the title bar caption and icon to avoid redundancy.** However, you still have to set a suitable title internally for use by Windows.
- **For dialog boxes, don't include the words "dialog" or "progress" in the titles.** These concepts are implied and leaving these words off makes the titles easier for users to scan.

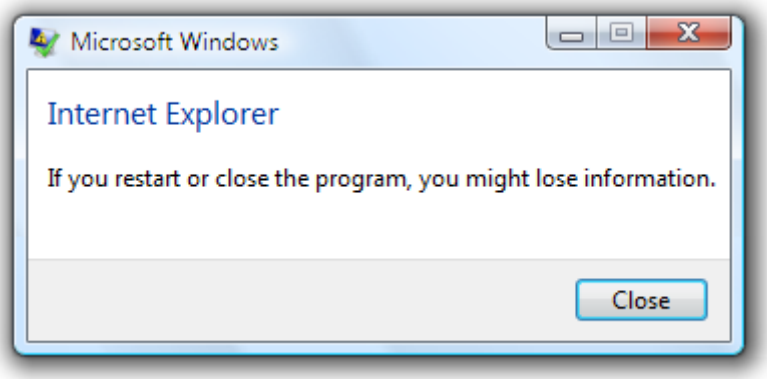
Main instructions

- **Use the main instruction to explain concisely what users should do in a given window or page.** Good main instructions communicate the user's objective rather

than focusing just on manipulating the UI.

- **Express the main instruction in the form of an imperative direction or specific question.**

  Incorrect:

  In this example, the main instruction simply states the name of the program; it doesn't explicitly invite a course of action for the user to take.

  Exceptions: Error messages, warning messages, and confirmations may use different sentence structures in their main instructions.

- **Use specific verbs whenever possible.** Specific verbs (examples: connect, save, install) are more meaningful to users than generic ones (examples: configure, manage, set).

  - For control panel pages and wizard pages, if you can't use a specific verb, you may prefer to omit the verb completely.

    Acceptable:

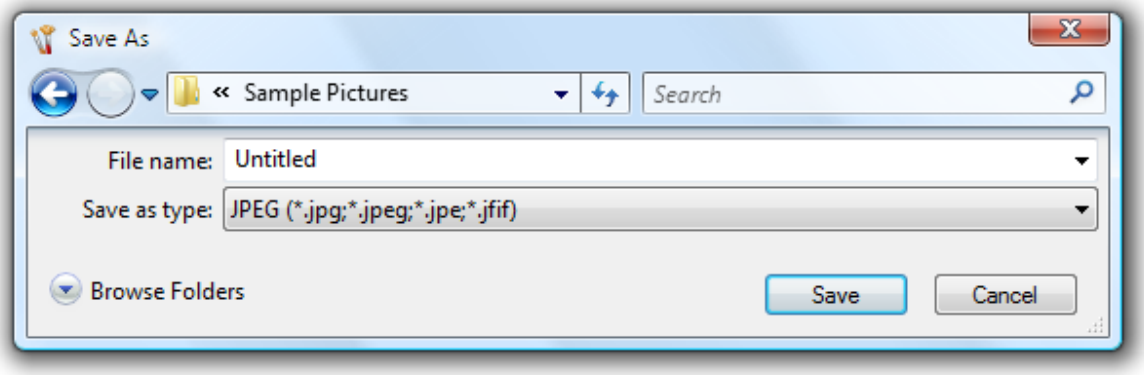
    Enter your locale, region, and language

    Better:

    Locale, region, and language

  - For dialogs, such as error messages and warnings, don't omit the verb.

- Don't feel obliged to use main instruction text if adding it would only be redundant or obvious from the context of the UI.



In this example, the context of the UI is already very clear; there is no need to add main instruction text.

- **Be concise use only a single, complete sentence.** Pare the main instruction down to the essential information. If you must explain anything more, consider using a supplemental instruction.
- Use sentence-style capitalization.
- **Don't include final periods if the instruction is a statement.** If the instruction is a question, include a final question mark.
- **For progress dialogs, use a gerund phrase briefly explaining the operation in progress,** ending with an ellipsis. Example: "Printing your pictures..."
- **Tip:** You can evaluate a main instruction by imagining what you would say to a friend when explaining what to do with the window or page. If responding with the main instruction would be unnatural, unhelpful, or awkward, rework the instruction.

For more information, see the "Main instruction" section in the specific UI component guidelines.

Supplemental instructions

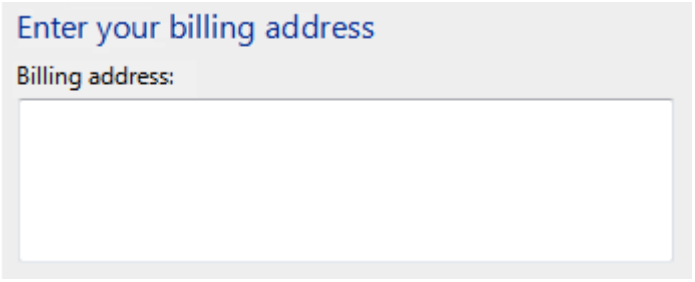
- When necessary, **use a supplemental instruction to present additional information helpful to understanding or using the window or page,** such as:
  - Providing context to explain why the window is being displayed if it is program or system initiated.
  - Qualifying information that helps users decide how to act on the main instruction.
  - Defining important terminology.
- **Don't use a supplemental instruction if one isn't necessary.** Prefer to communicate everything with the main instruction if you can do so concisely.

- **Don't repeat the main instruction with slightly different wording.** Instead, omit the supplemental instruction if there is nothing more to add.
- Use complete sentences and sentence-style capitalization.

Control labels

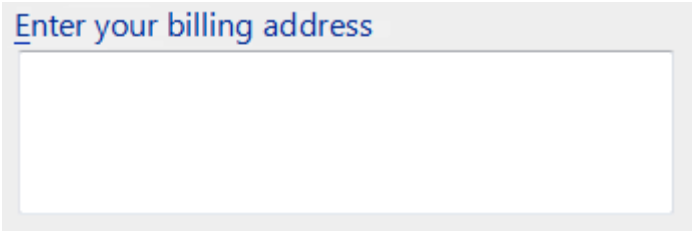
- **Label every control or group of controls. Exceptions:**
  - Text boxes and drop-down lists can be labeled using prompts.
  - Progressive disclosure controls are generally unlabeled.
  - **Subordinate controls use the label of their associated control.** Spin controls are always subordinate controls.
  - **Omit control labels that restate the main instruction.** In this case, the main instruction takes the access key.

    Acceptable:

    In this example, the text box label is just a restatement of the main instruction.

    Better:

    In this example, the redundant label is removed, so the main instruction takes the access key.

- Label placement:
  - Balloons, check boxes, command buttons, group boxes, links, tabs, and tips are labeled directly by the control itself.
  - Drop-down lists, list boxes, list views, progress bars, sliders, text boxes, and tree views are labeled above, flush left, or to the left.

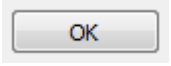
    - Progressive disclosure controls are usually unlabeled. Chevron buttons are labeled to the right.

- **Assign a unique access key for each interactive control** except for links. For more information, see Keyboard.

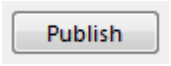
- **Keep labels brief.** Note, however, that adding a word or two to a label can help clarity, and sometimes eliminates the need for supplemental explanations.

- **Prefer specific labels over generic ones.** Ideally users shouldn't have to read anything else to understand the label.

  Incorrect:

  Correct:

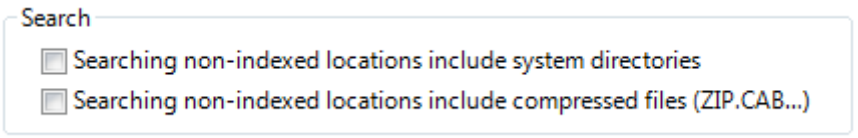
  

  In the correct example, a specific label is used for the commit button.

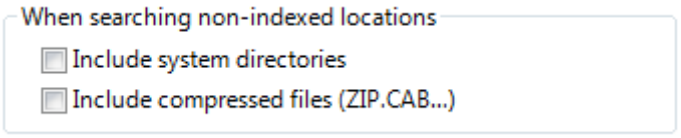
- **For lists of labels, such as radio buttons, use parallel phrasing,** and try to keep the length about the same for all labels.

- **For lists of labels, focus the label text on the differences among the options.** If all the options have the same introductory text, move that text to the group label.

  Incorrect:

  Correct:

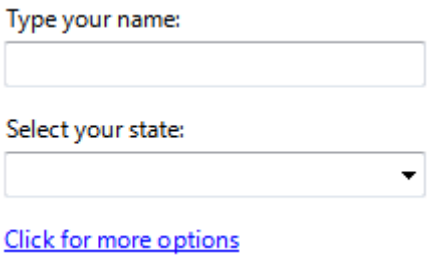
  The correct example moves the identical introductory phrasing to the label, so the two options are more cleanly differentiated.

- **In general, prefer positive phrasing.** For example, use do instead of do not, and notify instead of do not notify.

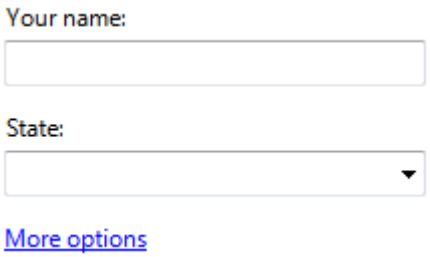
- **Exception:** The check box label, "Don't show this message again," is widely used.

- **Omit instructional verbs that apply to all controls of the given type.** Rather, focus labels on what is unique about the controls. For example, it goes without saying that users need to type into a text box control or that users need to click a link.

  Incorrect:

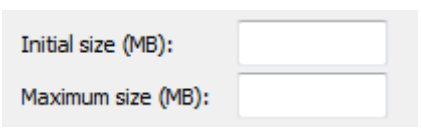
  Correct:

  In the incorrect examples, the control labels have instructional verbs that apply to all controls of their type.

- In some cases, the following parenthetical annotations to control labels may be helpful:
  - **If an option is optional, consider adding "(optional)" to the label.**
  - **If an option is strongly recommended, add "(recommended)" to the label.** Doing so means the setting is optional, but should be set anyway.
  - **If an option is intended only for advanced users, consider adding "(advanced)" to the label.**

- You may specify units (seconds, connections, and so on) in parentheses after the label.

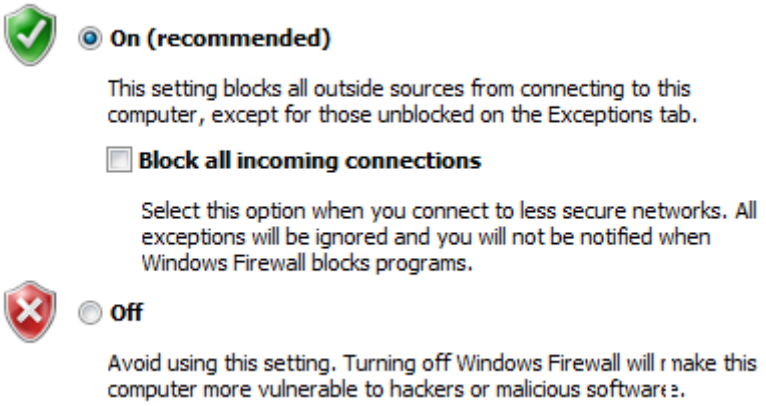
  This example shows that the unit of measurement is megabytes (MB).

For more information, see the "Text" or "Labels" section in the specific UI component guidelines.

Supplemental explanations

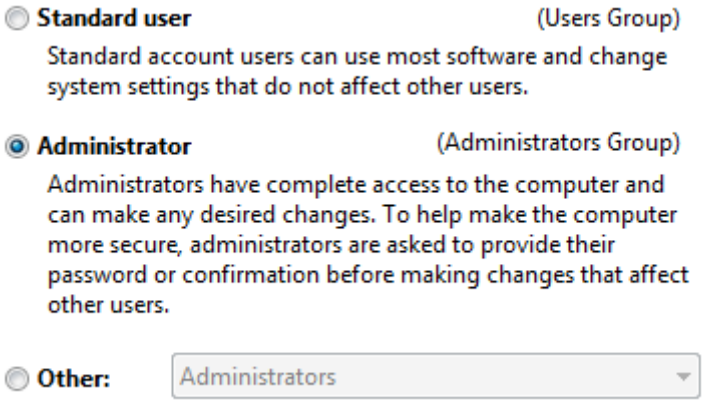
- **Use supplemental explanations when controls require more information than can be conveyed by their label.** But don't use a supplemental explanation if one isn't necessary prefer to communicate everything with the control label if you can do so concisely. Typically, supplemental explanations are used with command links, radio buttons, and check boxes.

- When necessary, **use bold in the control labels to make the text easier to scan** when there are supplemental explanations.



  In this example, the radio button labels are bold to make them easier to scan.

- **Adding a supplemental explanation to one control in a group doesn't mean that you have to provide explanations for all the other controls in the group.** Provide the relevant information in the label if you can and use explanations only when necessary. Don't have supplemental explanations that merely restate the label for consistency.



  In this example, two controls in the group include supplemental explanations, but the third does not.

- If a supplemental explanation follows a command link, write the supplemental text in second person.

  Example: Command link: Create wireless network settings and save to USB flash drive

  Supplemental explanation: This will create settings that you can transfer to the router with a USB flash drive. Do this only if you have a wireless router that supports USB flash drive configuration.

- Use complete sentences and ending punctuation.

Commit button labels

The following table shows the most common commit button labels and their usage.

Expand table

Button label	When to use
OK	• In dialog boxes: apply the changes or commit to the task and close the window. • In owner property windows: apply the pending changes (made since the window was opened or the last Apply) and close the window. • In owned property windows: keep the changes, close the window, and apply the changes when the owner window's changes are applied. • Use with windows that aren't task specific, such as property sheets. • For windows used to perform one specific task, use a specific label instead that starts with a verb (example: Print). • For windows in which users can't make changes, use Close. • Enter
Yes/No	Yes is the affirmative response to a yes or no question, whereas No is the negative response. • Use Yes and No buttons only to respond to yes or no questions. Never use OK and Cancel for yes or no questions. • Prefer specific responses over Yes and No buttons. While there's nothing wrong with using Yes and No, specific responses can be understood more quickly, resulting in efficient decision making. • However, consider using Yes and No responses if the phrasing of specific responses turns out to be long or awkward. • Don't use Yes and No buttons if the meaning of the No response is unclear. If so, use specific responses instead. • Yes and No must always be used as a pair. • Y and N

Button label	When to use
Cancel	• In dialog boxes: discard all changes or work in progress, revert to the previous state (leaving no noticeable side effect), and close the window. • In property sheets: discard all pending changes (made since the window was opened or the last Apply) and close the window. • In control panel items: discard all changes or work in progress, revert to the previous state, and return to the hub page from which the task was launched. If there is no such hub page, close the control panel item window instead. • Use when all pending changes or actions can be discarded and any side effects can be undone. • For changes that can't be discarded, use Close. For actions in progress that can be stopped, use Stop. If initially changes or actions can be discarded, you can use Cancel initially then change to Close or Stop once it can't be undone. • Esc
Close	Close the window. Any changes or side effects are not discarded. • Use when changes or side effects can't be discarded. Use Close instead of Cancel for primary windows. • Use for windows in which users can't make changes. • Alt+F4, Ctrl+F4
Stop	Stop a currently running task and close the window. Any work in progress or side effects are not discarded. • Use when work in progress and any side effects can't or won't be discarded, typically with progress bars or animations. • Esc
Apply	In owner property sheets: apply the pending changes (made since the window was opened or the last Apply), but leave the window open. Doing so allows users to evaluate the changes before closing the property sheet. In owned property sheets: don't use. • Use only in property sheets. • Provide an Apply button only if the property sheet has settings (at least one) with effects that users can evaluate in a meaningful way. Typically, Apply buttons are used when settings make visible changes. Users should be able to apply a change, evaluate the change, and make further changes based on that evaluation. If not, remove the Apply button instead of disabling it. • A
Next	In wizards and multi-step tasks: advance to the next step without committing to the task. • Use only in wizards and multi-step tasks to advance to the next step without commitment. • The effect of a Next button can always be undone by clicking Back. • N
Finish	In wizards and multi-step tasks: close the window. If the task hasn't been performed yet, perform the task. If that task has already been performed, any changes or side

Button label	When to use

	effects are not discarded.
	• Use only in wizards and multi-step tasks. However, the use of Finish is discouraged because there is usually a better, more specific commit button:
	  • If clicking the button commits to the task (so the task hasn't already been performed), use a specific label that starts with a verb (examples: Print, Connect, Start) that is a response to the main instruction.
	  • If the task has already been performed within the wizard, use Close instead.
	• However, you can use Finish when:
	  • The specific label is still generic, such as Save, Select, Choose, or Get.
	  • The task involves changing a setting or collection of settings.
	• Enter

Done	Not applicable. • Don't use. Done as a command is grammatically incorrect. • Not applicable.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Style and Tone

Article • 11/17/2021

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

Tone in writing is the attitude that the writer conveys to the reader. It's not necessarily what you say, but how you say it. Tone is intended to create a specific response or emotion in the reader; it creates a personality that can either engage or repel users.

Note: Guidelines related to user interface text are presented in a separate article.

Design concepts

The Windows tone

Use the Windows tone to inspire confidence by communicating to users on a personal level by being accurate, encouraging, insightful, objective, and user-focused. The Windows tone also strives to be light, inspiring, straightforward, and trustworthy.

Don't use a distracting, condescending, or arrogant tone. Avoid the extremes of the "machine" voice (where the speaker is removed from the language) and the "sales rep" voice (where the writing tries to sell us something, to cajole us, to cheer us up, to gloss over everything as "simple.")

Research from Microsoft shows that in software, there is often:

- **Overuse of computer terminology and jargon,** which many users don't understand or misinterpret.
- **Inconsistent use of terminology.**
- **Impenetrable, often unintentionally patronizing messages,** which users struggle to decipher.

These problems make users feel confused, unintelligent, disheartened, and ultimately disengaged from their experience with software.

Renewed attention to message quality, including the tone of the text, implies that language itself is an important ingredient in overall customer satisfaction. Support users with different levels of technical knowledge, especially novice users, to take advantage of all the things your program can do.

If you do only one thing...

Make sure that your text is accurate, encouraging, insightful, objective, and user-focused. Be straightforward and trustworthy, and where appropriate use a light and inspiring tone.

Guidelines

Use the Windows tone

Tone in your program should be:

- **Accurate.** Users should feel reassured that the information is technically accurate. If the information isn't accurate, users' experience with that specific task is spoiled, and they lose faith in any other assistance they read from that source.
- **Encouraging.** Use language that conveys that the software empowers users to do things, rather than allowing them to do things. For example, use "you can" rather than "Windows lets you" or "this feature allows you." (Exception: it's okay to use "allow" when referring to features—such as security features—that permit or deny an action.)
- **Insightful.** Users should believe that you (and by extension your application) know when a certain task is complicated and that you will guide them through it. At the same time, treat users as intelligent people who happen to need help with a particular problem.
- **Objective.** Sometimes users want a richer explanation; often though they just want to know what they need to move on. This requires objectivity—to recognize that the goal (productivity, curiosity, enjoyment) is the user's goal, not the writer's. It also requires that you shed any predisposed notions about the user.
- **User-focused.** Write from the user's perspective and preferably from the perspective of what you can do for the user. Users should feel that they will find information that is relevant and accessible to them.

You can further develop communication with your users by being:

- **Light.** A light voice feels easy to deal with and quick to comprehend. It isn't difficult or burdensome, and it avoids straining to be profound or serious.

- **Inspiring.** An inspiring voice motivates users, stimulating them to take action. A certain enthusiasm marks this kind of voice, giving it a more engaging personality than the machine-like tone users have often come to expect from their computers.
- **Straightforward.** A straightforward voice is candid and open, free from pretense or deceit.
- **Trustworthy.** A trustworthy voice is worthy of confidence, reliable, accurate, and honest. It takes only a few inaccuracies to lose the user's trust, but it may take a long time to earn that trust back.

By contrast, be sure to avoid the following tones, which are much more likely to receive a negative reaction:

- **Machine tone.** Feels like having an impersonal, inflexible exchange with a computer or robot.
- **Corporate tone.** Feels like having a monologue with an all-powerful, all-knowing, impersonal corporation.
- **Law enforcement tone.** Feels like being interrogated with a barrage of intrusive questions.
- **Lawyerly tone.** Feels like being asked to perform some legally significant act.
- **Sales rep tone.** Feels like being asked to buy or try something, with any concerns being glossed over.
- **Superior, condescending, or angry tone.** Feels like the software is belittling users, talking down to them, or upset with them. Typically, this tone is very technical, draws unnecessary attention to the user's mistakes, and feels rude.
- **Boastful tone.** Feels like the software is bragging about its accomplishments or otherwise drawing too much attention to itself.
- **Flippant tone.** Feels like users' goals and emotions aren't being taken seriously, or their use of the program is being taken for granted.

Use real-world language

- **Use everyday words when you can and avoid words you wouldn't say to someone else in person.** This is especially effective if you are explaining a complex technical concept or action. Imagine yourself looking over the user's shoulder and explaining how to accomplish the task.

  Acceptable:

  Use this procedure to change your password.

  Better:

  Follow these steps to change your password.

- **Use short, plain words whenever possible.** Shorter words are more conversational, save space on screen, and are easier to scan.

  Acceptable:

  In addition, this section shows you...

  Digital cameras employ tiny microchips...

  Digital cameras utilize tiny microchips...

  Better:

  This section also shows you...

  Digital cameras use tiny microchips...

- **Don't invent words or apply new meanings to standard words.** Assume that users are more familiar with a word's established meaning than with a special meaning given to it by the technology industry. When an industry term is required, provide an in-context definition. Avoid jargon, but remember that some expressions specific to computer usage—hacker, burn a CD, and so on—are already part of everyday speech.

  Incorrect:

  You can use folders to bucketize your favorites.

  Correct:

  You can use folders to categorize your favorites.

- **Don't use symbols as a substitute for simple words.**

  Incorrect:

  users & computers

  users + computers

  domain / workgroup

  # of users

  Correct:

  users and computers

domain or workgroup

number of users

Be precise

- Choose words with a clear meaning.

  Acceptable:

  Since you created the table, you can make changes to it.

  Keep your firewall turned on, as turning it off could create a security risk.

  Better:

  Because you created the table, you can make changes to it.

  Keep your firewall turned on, because turning it off could create a security risk.

- Omit needless words—don't use two or three words when one will do.

  Verbose:

  Follow these steps in order to change your password:

  Better:

  To change your password:

- Avoid unnecessary adverbs.

  Incorrect:

  It isn't terribly hard to change your password.

  Correct:

  It isn't hard to change your password.

- Choose single-word verbs over multi-word verbs.

  Acceptable:

  When you lock down your computer, ...

  Better:

  When you lock your computer, ...

- Don't convert verbs to nouns and nouns to verbs.

  Incorrect:

  To password-protect your computer...

  To establish connectivity...

  Correct:

  To protect your computer with a password...

  To connect...

Be consistent

- Consistent terminology promotes learning and a better understanding of technical concepts. Inconsistency forces users to figure out whether different words and actions mean the same thing.

  Examples:

  switch, toggle

  start, run, launch, boot, execute

  enable, activate, turn on

  burn, copy

- Consistent syntax helps set users' expectations. Once these expectations are set, users can more quickly parse text that uses consistent syntax. For example, if instructions are always written in the imperative form, users will learn to pay closer attention to imperative sentences.

Contractions

- Contractions lend a shorter, snappier, more conversational rhythm to writing. Use them as appropriate and in context. Don't use contractions with product names or other proper nouns.

Colloquialisms, idioms, and slang

- Consider using colloquialisms or slang only in special situations, such as product tours, setup screens, or content that won't be localized. Recent studies have shown

that users enjoy the unexpected word or familiar phrase. Bear in mind that using colloquialisms and slang can be difficult and costly to translate effectively, so such language is best used judiciously.

Examples:

On the other hand, there are certain values in the registry that shouldn't be changed.

With SDSL, the line is dedicated to your Internet use, so you won't experience rush-hour traffic jams on the Internet, as you might with cable service.

Person

- **Address the user as you, directly or indirectly.**
- **Use the second person (you, your) to tell users what to do.** Often the second person is implied.

  Examples:

  Choose the pictures you want to print.

  Choose an account. (implied)

- **Use the first person (I, me, my) to let users tell the program what to do.**

  Example:

  Print the photos on my camera.

- **Use we judiciously.** The first-person plural can suggest a daunting corporate presence. However, it can be preferable to using the name of your application. Use we recommend rather than it is recommended.
- **Avoid third-person references (the user)** because they create a more formal, less personal tone.

Voice

- **Use the active voice,** which emphasizes the person or thing doing the action. It is more direct and personal than the passive voice, which can be confusing or sound formal.

  Acceptable:

Icons can be arranged by name in alphabetical order.

When a Personal Digital Assistant (PDA) or laptop is plugged in...

Better:

You can arrange icons in alphabetical order by the icon name.

When you plug in any Personal Digital Assistant (PDA) or laptop...

- Use the passive voice only to avoid a wordy or awkward construction; when the action rather than the doer is the focus of the sentence; when the subject is unknown; or in error messages, when the user is the subject and might feel blamed for the error if the active voice were used.

  Correct:

  The new icon should appear in the upper-left corner.

  Updates must be downloaded and installed before they can work.

- **Phrase statements in the positive form**, and emphasize what users can accomplish, rather than what they can't.

Attitude toward the user

- **Be polite, supportive, and encouraging.** The user should never feel condescended to, blamed, or intimidated.

  Acceptable:

  Cannot delete New Text Document: Access is denied.

  Better:

  This file is protected and cannot be deleted without specific permission.

- **Strike the right balance: be warm toward the user without being too intimate or too business-like.** Imagine that you are helping a friend use the product for the first time. This person is not your best friend or significant other, but instead, a neighbor or family friend. Users should feel comfortable and at home when using your program, but the language should not feel presumptuous or too familiar.

- **Limit please to situations that inconvenience the user in some way,** such as:

- The user is asked to do something inconvenient, such as waiting, repeating a task or updating a program.

  Correct:

  Please wait while Windows copies the files to your computer.

- The user can't complete a task because of a missing feature, design flaw, or program bug.

  Correct:

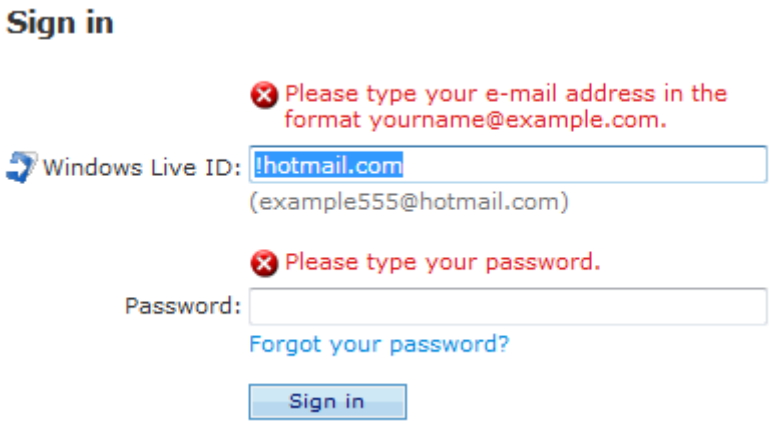
  Unable to save using the specified format. Please select another format.

- The user has gone out of his or her way to be helpful, as by participating in a customer feedback program or filing a bug report.

  Correct:

  To improve the Fabrikam Backup experience, please participate in the Fabrikam customer feedback program.

You should also use please whenever its absence would be considered curt.



In this example, the inline error messages would be curt without please.

- **Use sorry only in error messages that result in serious problems for the user** (for example, data loss, the user can't continue to use the computer, or the user must get help from a technical representative). Don't apologize if the issue occurred during the normal functioning of the program (for example, if the user needs to wait for a network connection to be found).

  Correct:

  We're sorry, but Fabrikam Backup detected an unrecoverable problem and was shut down.

Sentence structure and length

- Because users often scan text, **make every word count.** Simple, concise sentences (and paragraphs) not only save space on the screen but are the most effective means of conveying that an idea or action is important. Use your best judgment—make sentences tight, but not so tight that the tone seems abrupt and unfriendly.
- **Avoid repetition.** Review each window and eliminate duplicate words and statements. Don't avoid important text—be explicit whenever necessary—but don't be redundant and don't explain things that go without saying.
- **Use sentence fragments if appropriate.** Sentence fragments are short and punchy—and, as they typically take the interrogative form, they are a good way of directly engaging the user.

  Correct:

  Save changes to "My Photos"?

  Ever saved a file and then not remembered where you saved it?
- **Start sentences with conjunctions** (and, but, or) if you need to.
- **Substitute lists and tables for complex sentences.** Lists (whether numbered or bulleted) and tables are clearer and easier to scan.
- **Use parallel grammatical constructions.** Parallelism requires that words and phrases that have the same function have the same form. Use parallel language whenever you express ideas of equal weight, and for UI elements that are parallel in function (such as headings, labels, lists, or page titles).

  Correct:

  Listen

  Watch

  Share

  Collect

  These items are parallel because they are all single-word, imperative verbs.

  Incorrect:

  Music

Video

Share

Listen

These items are not parallel because Music and Video are nouns, but Share and Listen are verbs.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Messages (Design basics)

Article • 02/09/2021

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

Messages are any kind of message users need or want to see as they use your app. Learn how to present errors, warning, confirmations, and notifications in your app.

In this section

Expand table

Topic	Description
Error Messages	An error message alerts users of a problem that has already occurred. By contrast, a warning message alerts users of a condition that might cause a problem in the future. Error messages can be presented using modal dialog boxes, in-place messages, notifications, or balloons.
Warning Messages	A warning message is a modal dialog box, in-place message, notification, or balloon that alerts the user of a condition that might cause a problem in the future.
Confirmations	A confirmation is a modal dialog box that asks if the user wants to proceed with an action.
Notifications	A notification informs users of events that are unrelated to the current user activity, by briefly displaying a balloon from an icon in the notification area. The notification could result from a user action or significant system event, or could offer potentially useful information from Microsoft Windows or an application.

Feedback

Was this page helpful? Yes No

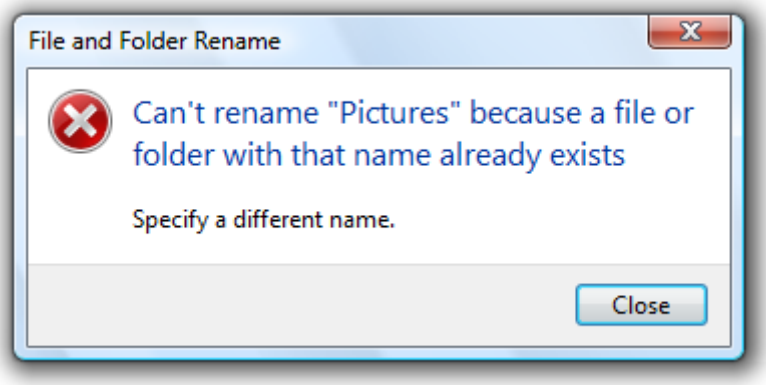
Provide product feedback | Get help at Microsoft Q&A

Error Messages in Windows 7

Article • 02/08/2022

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

Error messages in Windows 7 alert users of problems that have already occurred. In contrast, warning messages alert users of conditions that might cause problems in the future. Error messages can be presented using modal dialog boxes, in-place messages, notifications, or balloons.



A typical modal error message.

Effective error messages inform users that a problem occurred, explain why it happened, and provide a solution so users can fix the problem. Users should either perform an action or change their behavior as the result of an error message.

Well-written, helpful error messages are crucial to a quality user experience. Poorly written error messages result in low product satisfaction, and are a leading cause of avoidable technical support costs. Unnecessary error messages break users' flow.

Note: Guidelines related to dialog boxes, warning messages, confirmations, standard icons, notifications, and layout are presented in separate articles.

Is this the right user interface?

To decide, consider these questions:

- **Is the user interface (UI) presenting a problem that has already occurred?** If not, the message isn't an error. If the user being alerted of a condition that might cause a problem in the future, use a warning message.
- **Can the problem be prevented without causing confusion?** If so, prevent the problem instead. For example, use controls that are constrained to valid values instead of using unconstrained controls that may require error messages. Also, disable controls when clicking would result in error, as long as it's obvious why the control is disabled.
- **Can the problem be corrected automatically?** If so, handle the problem and suppress the error message.
- **Are users likely to perform an action or change their behavior as the result of the message?** If not, the condition doesn't justify interrupting the user so it's better to suppress the error.
- **Is the problem relevant when users are actively using other programs?** If so, consider showing the problem using a notification area icon.
- **Is the problem not related to the current user activity, does it not require immediate user action, and can users freely ignore it?** If so, use an action failure notification instead.
- **Does the problem relate to the status of a background task within a primary window?** If so, consider showing the problem using a status bars.
- **Are the primary target users IT professionals?** If so, consider using an alternative feedback mechanism, such as log file entries or e-mail alerts. IT professionals strongly prefer log files for non-critical information.

Design concepts

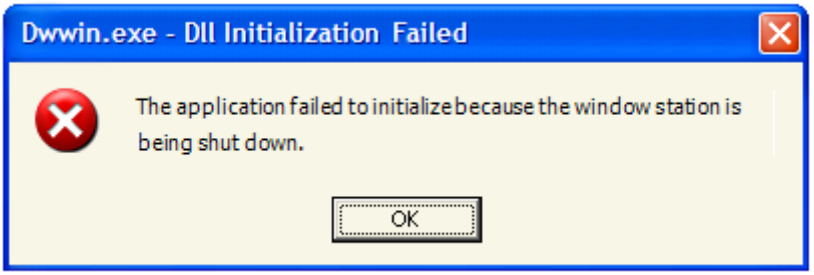
The characteristics of poor error messages

It should be no surprise that there are many annoying, unhelpful, and poorly written error messages. And because error messages are often presented using modal dialogs, they interrupt the user's current activity and demand to be acknowledged before allowing the user to continue.

Part of the problem is that there are so many ways to do it wrong. Consider these examples from the Error Message Hall of Shame:

Unnecessary error messages

Incorrect:



This example from Windows XP might be the worst error message ever. It indicates that a program couldn't launch because Windows itself is in the process of shutting down. There is nothing the user can do about this or even wants to do about this (the user chose to shut Windows down, after all). And by displaying this error message, Windows prevents itself from shutting down!

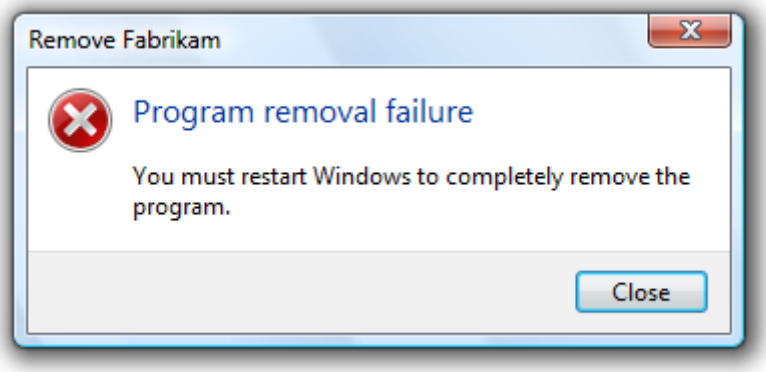
The problem: The error message itself is the problem. Aside from dismissing the error message, there is nothing for users to do.

Leading cause: Reporting all error cases, regardless of users' goals or point of view.

Recommended alternative: Don't report errors that users don't care about.

"Success" error messages

Incorrect:



This error message resulted from the user choosing not to restart Windows immediately after program removal. The program removal was successful from the user's point of view.

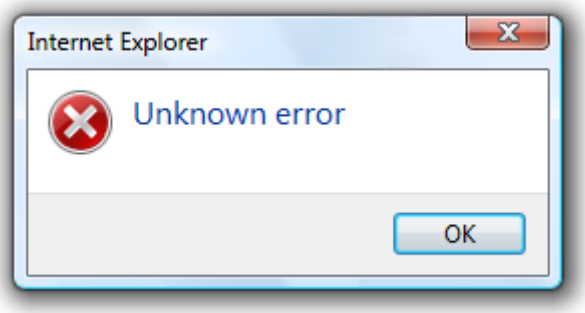
The problem: There's no error from the user's point of view. Aside from dismissing the error message, there is nothing for users to do.

Leading cause: The task completed successfully from the user's point of view, but failed from the uninstall program's point of view.

Recommended alternative: Don't report errors for conditions that users consider acceptable.

Completely useless error messages

Incorrect:



Users learn that there was an error, but have no idea what the error was or what to do about it. And no, it's not OK!

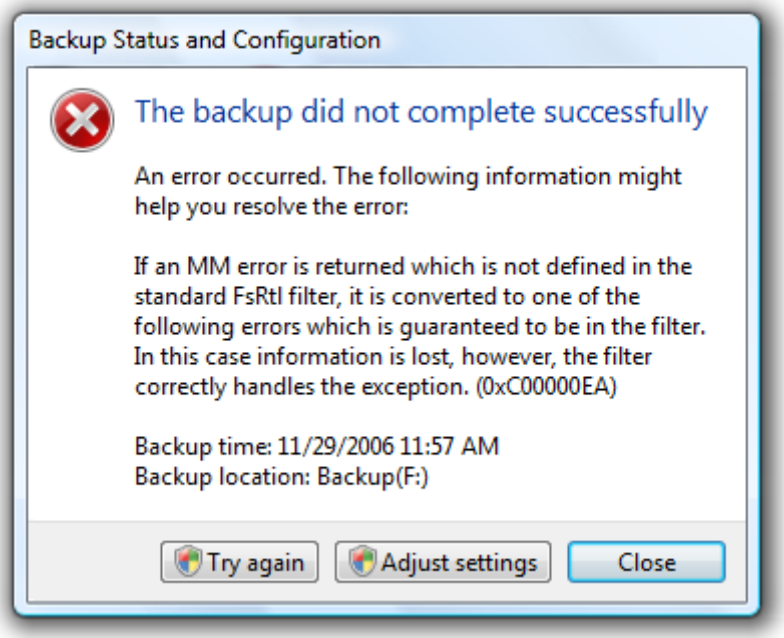
The problem: The error message doesn't give a specific problem and there is nothing users can do about it.

Leading cause: Most likely, the program has poor error handling.

Recommended alternative: Design good error handling into the program.

Incomprehensible error messages

Incorrect:



In this example, the problem statement is clear, but the supplemental explanation is utterly baffling.

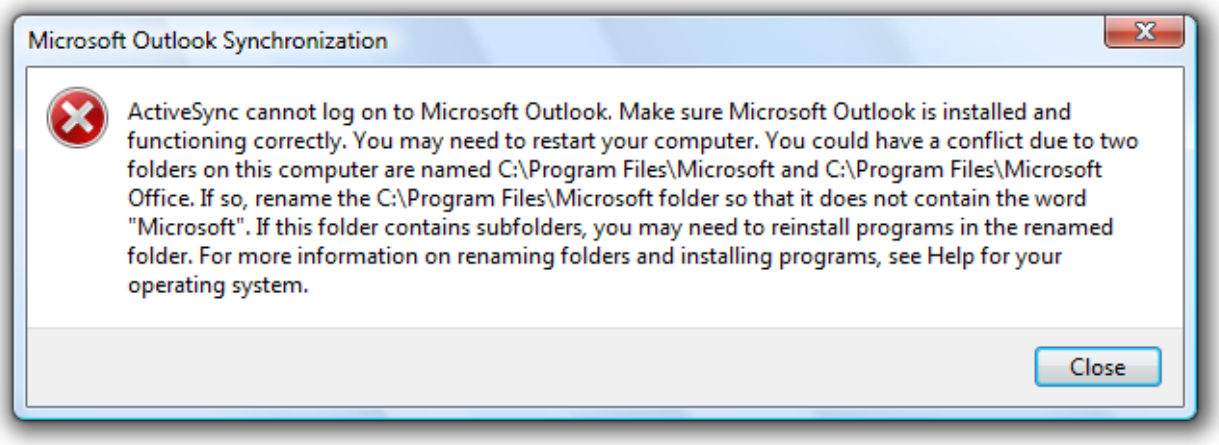
The problem: The problem statement or solution is incomprehensible.

Leading cause: Explaining the problem from the code's point of view instead of the user's.

Recommended alternative: Write error message text that your target users can easily understand. Provide solutions that users can actually perform. Design your program's error message experience don't have programmers compose error messages on the spot.

Error messages that overcommunicate

Incorrect:



In this example, the error message apparently attempts to explain every troubleshooting step.

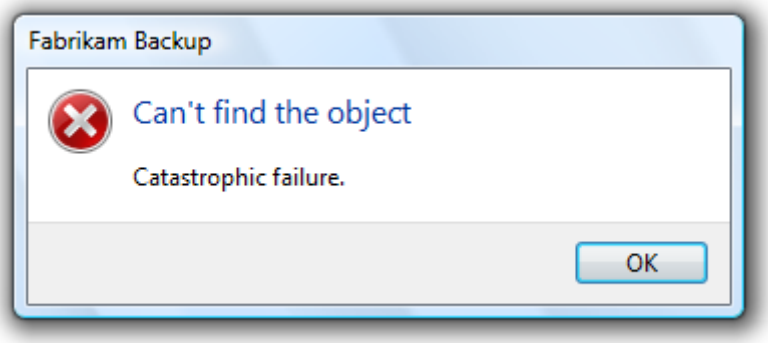
The problem: Too much information.

Leading cause: Giving too many details or trying to explain a complicated troubleshooting process within an error message.

Recommended alternative: Avoid unnecessary details. Also, avoid troubleshooters. If a troubleshooter is necessary, focus on the most likely solutions and explain the remainder by linking to the appropriate topic in Help.

Unnecessarily harsh error messages

Incorrect:



The program's inability to find an object hardly sounds catastrophic. And assuming it is catastrophic, why is OK the response?

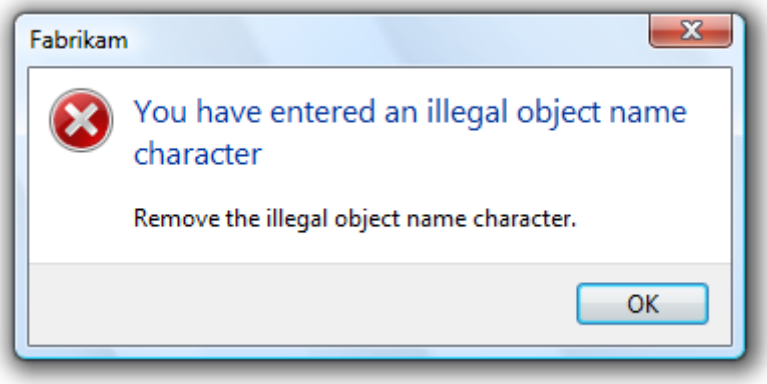
The problem: The program's tone is unnecessarily harsh or dramatic.

Leading cause: The problem is due to a bug that appears catastrophic from the program's point of view.

Recommended alternative: Choose language carefully based on the user's point of view.

Error messages that blame users

Incorrect:



Why make users feel like a criminal?

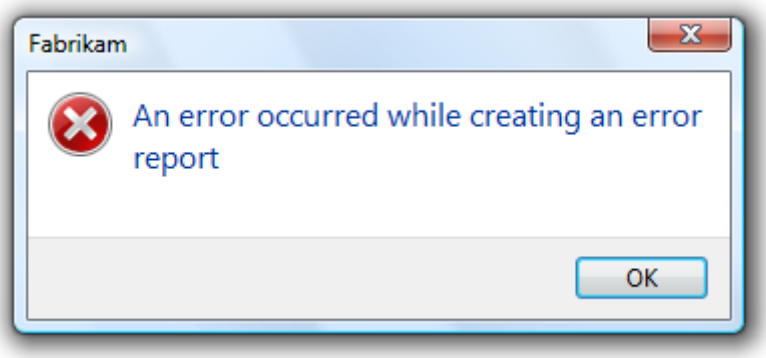
The problem: The error message is phrased in a way that accuses the user of making an error.

Leading cause: Insensitive phrasing that focuses on the user's behavior instead of the problem.

Recommended alternative: Focus on the problem, not the user action that led to the problem, using the passive voice as necessary.

Silly error messages

Incorrect:



In this example, the problem statement is quite ironic and no solutions are provided.

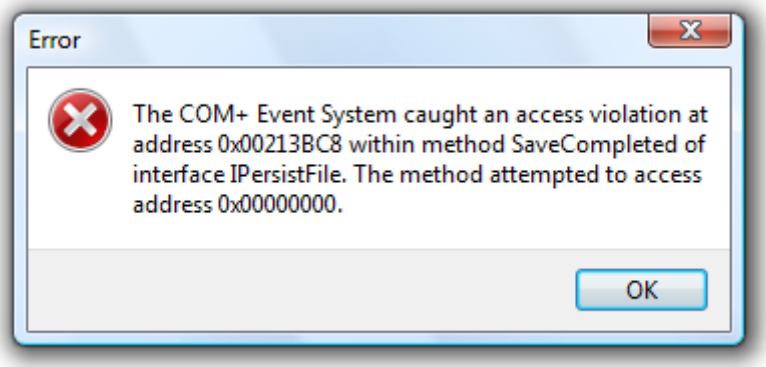
The problem: Error message statements that are silly or non-sequitors.

Leading cause: Creating error messages without paying attention to their context.

Recommended alternative: Have your error messages crafted and reviewed by a writer. Consider the context and the user's state of mind when reviewing the errors.

Programmer error messages

Incorrect:



In this example, the error message indicates that there is a bug in the program. This error message has meaning only to the programmer.

The problem: Messages intended to help the program's developers find bugs are left in the release version of the program. These error messages have no meaning or value to users.

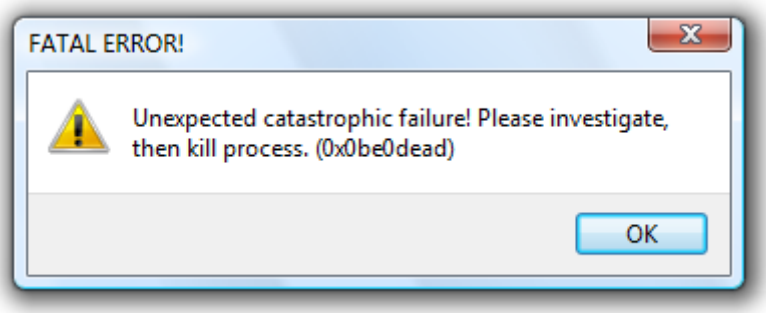
Leading cause: Programmers using normal UI to make messages to themselves.

Recommended alternative: Developers must conditionally compile all such messages so that they are automatically removed from the release version of a product. Don't waste

time trying to make errors like this comprehensible to users because their only audience is the programmers.

Poorly presented error messages

Incorrect:



This example has many common presentation mistakes.

The problem: Getting all the details wrong in the error message presentation.

Leading cause: Not knowing and applying the error message guidelines. Not using writers and editors to create and review the error messages.

The nature of error handling is such that many of these mistakes are very easy to make. It's disturbing to realize that most error messages could be nominees for the Hall of Shame.

The characteristics of good error messages

In contrast to the previous bad examples, good error messages have:

- **A problem.** States that a problem occurred.
- **A cause.** Explains why the problem occurred.
- **A solution.** Provides a solution so that users can fix the problem.

Additionally, good error messages are presented in a way that is:

- **Relevant.** The message presents a problem that users care about.
- **Actionable.** Users should either perform an action or change their behavior as the result of the message.
- **User-centered.** The message describes the problem in terms of target user actions or goals, not in terms of what the code is unhappy with.
- **Brief.** The message is as short as possible, but no shorter.
- **Clear.** The message uses plain language so that the target users can easily understand problem and solution.

- **Specific.** The message describes the problem using specific language, giving specific names, locations, and values of the objects involved.
- **Courteous.** Users shouldn't be blamed or made to feel stupid.
- **Rare.** Displayed infrequently. Frequently displayed error messages are a sign of bad design.

By designing your error handling experience to have these characteristics, you can keep your program's error messages out of the Error Message Hall of Shame.

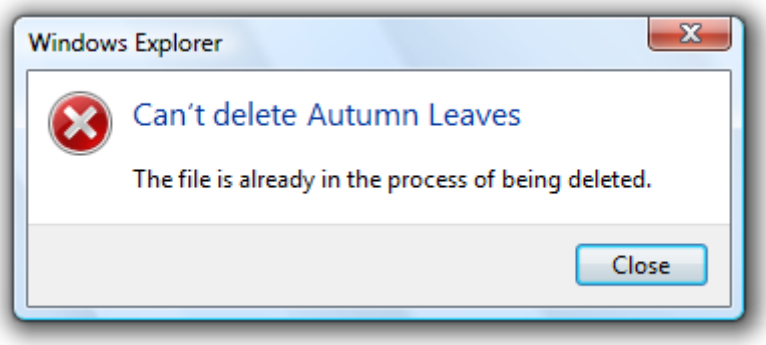
Avoiding unnecessary error messages

Often the best error message is no error message. Many errors can be avoided through better design, and there are often better alternatives to error messages. It's usually better to prevent an error than to report one.

The most obvious error messages to avoid are those that aren't actionable. If users are likely to dismiss the message without doing or changing anything, omit the error message.

Some error messages can be eliminated because they aren't problems from the user's point of view. For example, suppose the user tried to delete a file that is already in the process of being deleted. While this might be an unexpected case from the code's point of view, users don't consider this an error because their desired outcome is achieved.

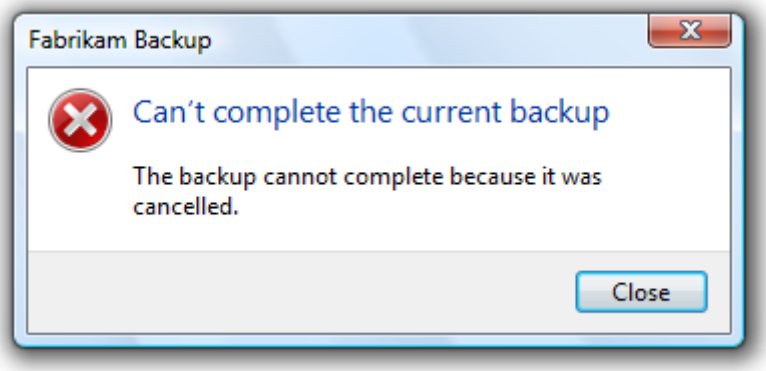
Incorrect:



This error message should be eliminated because the action was successful from the user's point of view.

For another example, suppose the user explicitly cancels a task. For the user's point of view, the following condition isn't an error.

Incorrect:



This error message should also be eliminated because the action was successful from the user's point of view.

Sometimes error messages can be eliminated by focusing on users' goals instead of the technology. In doing so, reconsider what an error really is. Is the problem with the user's goals, or with your program's ability to satisfy them? If the user's action makes sense in the real world, it should make sense in software too.

For example, suppose within an e-commerce program a user tries to find a product using search, but the literal search query has no matches and the desired product is out of stock. Technically, this is an error, but instead of giving an error message, the program could:

- Continue to search for products that most closely match the query.
- If the search has obvious mistakes, automatically recommend a corrected query.
- Automatically handle common problems such as misspellings, alternative spellings, and mismatching pluralization and verb cases.
- Indicate when the product will be in stock.

As long as the user's request is reasonable, a well designed e-commerce program should return reasonable results not errors.

Another great way to avoid error messages is by preventing problems in the first place. You can prevent errors by:

- **Using constrained controls.** Use controls that are constrained to valid values. Controls like lists, sliders, check boxes, radio buttons, and date and time pickers are constrained to valid values, whereas text boxes are often not and may require error messages. However, you can constrain text boxes to accept only certain characters and accept a maximum number of characters.
- **Using constrained interactions.** For drag operations, allow users to drop only on valid targets.
- **Using disabled controls and menu items.** Disable controls and menu items when users can easily deduce why the control or menu item is disabled.

- **Providing good default values.** Users are less likely to make input errors if they can accept the default values. Even if users decide to change the value, the default value lets users know the expected input format.
- **Making things just work.** Users are less likely to make mistakes if the tasks are unnecessary or performed automatically for them. Or if users make small mistakes but their intention is clear, the problem is fixed automatically. For example, you can automatically correct minor formatting problems.

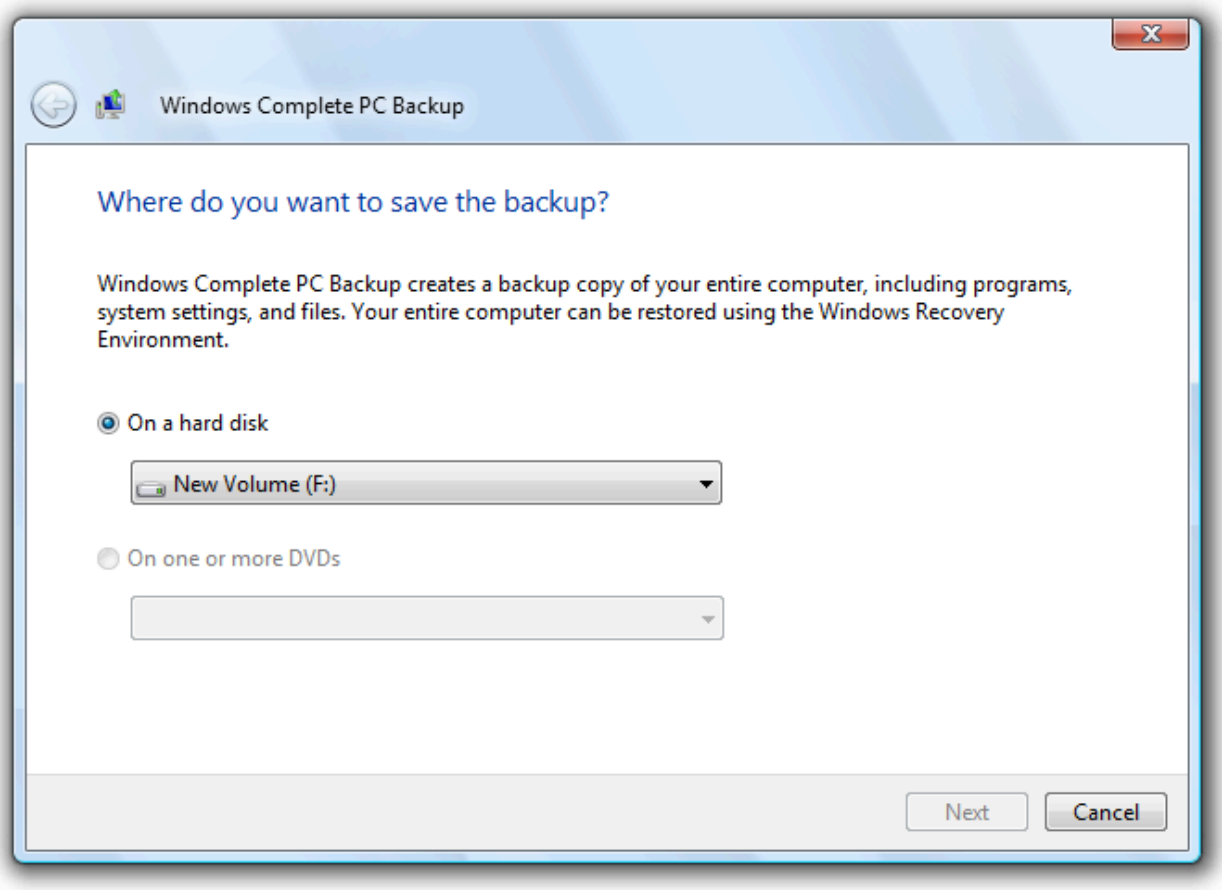
Providing necessary error messages

Sometimes you really do need to provide an error message. Users make mistakes, networks and devices stop working, objects can't be found or modified, tasks can't be completed, and programs have bugs. Ideally, these problems would happen less often for example, we can design our software to prevent many types of user mistakes but it isn't realistic to prevent all of these problems. And when one of these problems does happen, a helpful error message gets users back on their feet quickly.

A common belief is that error messages are the worst user experience and should be avoided at all costs, but it is more accurate to say that user confusion is the worst experience and should be avoided at all costs. Sometimes that cost is a helpful error message.

Consider disabled controls. Most of the time, it is obvious why a control is disabled, so disabling the control is a great way to avoid an error message. However, what if the reason a control is disabled isn't obvious? The user can't proceed and there is no feedback to determine the problem. Now the user is stuck and either has to deduce the problem or get technical support. In such cases, it's much better to leave the control enabled and give a helpful error message instead.

Incorrect:



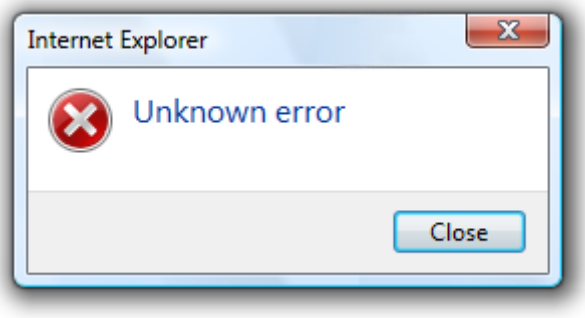
Why is the Next button disabled here? Better to leave it enabled and avoid user confusion by giving a helpful error message.

If you aren't sure whether you should give an error message, start by composing the error message that you might give. If users are likely either to perform an action or to change their behavior as a result, provide the error message. By contrast, if users are likely to dismiss the message without doing or changing anything, omit the error message.

Designing for good error handling

While crafting good error message text can be challenging, sometimes it is impossible without good error handling support from the program. Consider this error message:

Incorrect:



Chances are, the problem really is unknown because the program's error handling support is lacking.

While it's possible that this is a very poorly written error message, it more likely reflects the lack of good error handling by the underlying code there is no specific information known about the problem.

In order to create specific, actionable, user-centered error messages, your program's error handling code must provide specific, high-level error information:

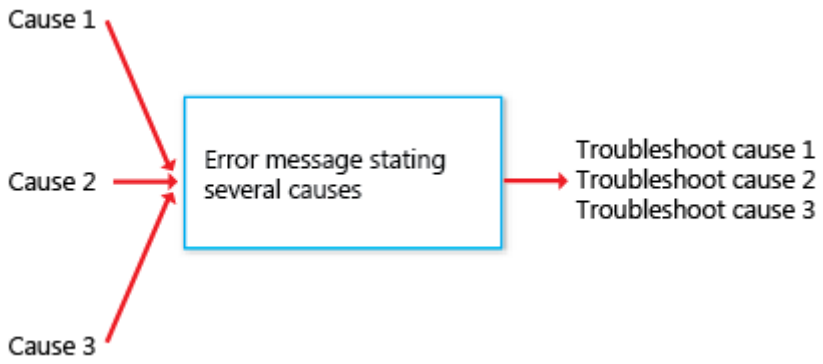
- Each problem should have a unique error code assigned.
- If a problem has several causes, the program should determine the specific cause whenever possible.
- If the problem has parameters, the parameters must be maintained.
- Low-level problems must be handled at a sufficiently high level so that the error message can be presented from the user's point of view.

Good error messages aren't just a UI problem, they are a software design problem. A good error message experience isn't something that can be tacked on later.

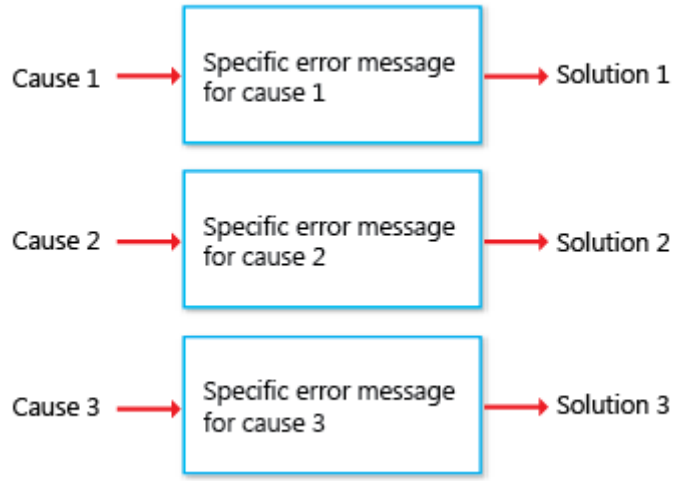
Troubleshooting (and how to avoid it)

Troubleshooting results when a problem with several different causes is reported with a single error message.

Incorrect:



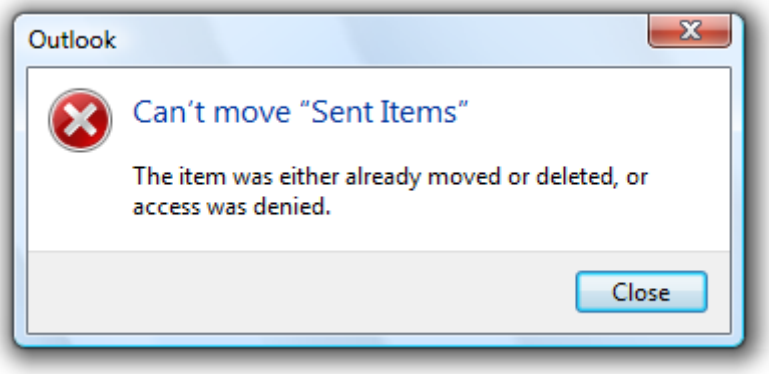
Correct:



Troubleshooting results when several problems are reported with a single error message.

In the following example, an item couldn't be moved because it was already moved or deleted, or access was denied. If the program can easily determine the cause, why put the burden on the user to determine the specific cause?

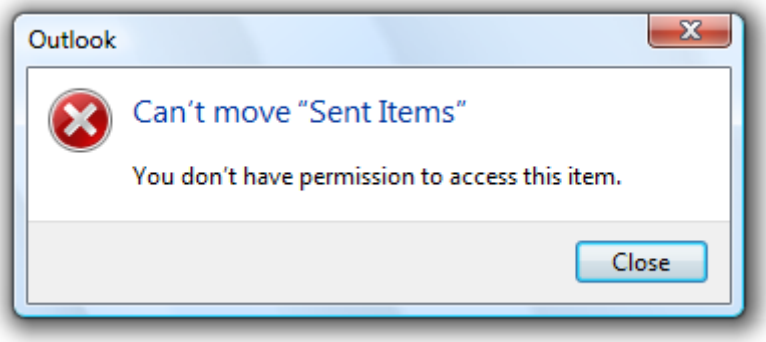
Incorrect:



Well, which is it? Now the user has to troubleshoot.

The program can determine if access was denied, so this problem should be reported with a specific error message.

Correct:



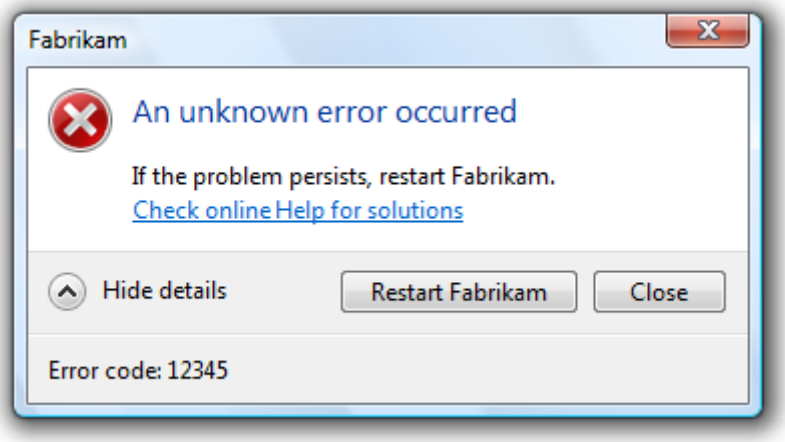
With a specific cause, no troubleshooting is required.

Use messages with multiple causes only when the specific cause cannot be determined. In this example, it would be difficult for the program to determine if the item was moved or deleted, so a single error message with multiple causes might be used here. However, it's unlikely that users are going to care if, for example, they couldn't move a deleted file. For these causes, the error message isn't even necessary.

Handling unknown errors

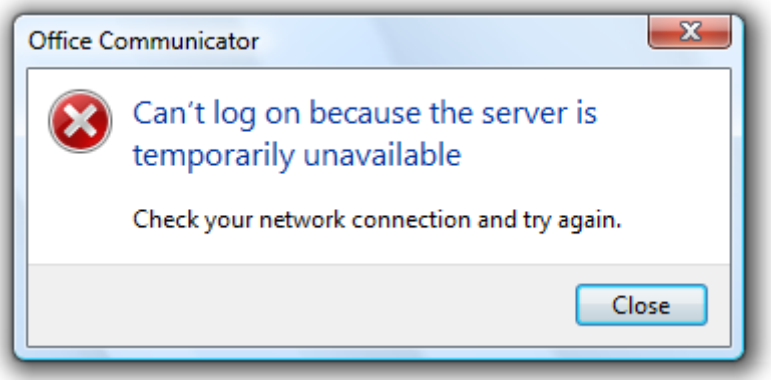
In some cases, you genuinely won't know the problem, cause, or the solution. If it would be unwise to suppress the error, it is better to be up front about the lack of information than to present problems, causes, or solutions that might not be right.

For example, if your program has an unhandled exception, the following error message is suitable:



If you can't suppress an unknown error, it is better to be up front about the lack of information.

On the other hand, do provide specific, actionable information if it is likely to be helpful most of the time.



This error message is suitable for an unknown error if network connectivity is usually the problem.

Determine the appropriate message type

Some issues can be presented as an error, warning, or information, depending on the emphasis and phrasing. For example, suppose a Web page cannot load an unsigned ActiveX control based on the current Windows Internet Explorer configuration:

- **Error.** "This page cannot load an unsigned ActiveX control." (Phrased as an existing problem.)
- **Warning.** "This page might not behave as expected because Windows Internet Explorer isn't configured to load unsigned ActiveX controls." or "Allow this page to install an unsigned ActiveX Control? Doing so from untrusted sources may harm your computer." (Both phrased as conditions that may cause future problems.)
- **Information.** "You have configured Windows Internet Explorer to block unsigned ActiveX controls." (Phrased as a statement of fact.)

To determine the appropriate message type, focus on the most important aspect of the issue that users need to know or act upon. Typically, if an issue blocks the user from proceeding, you should present it as an error; if the user can proceed, present it as a warning. Craft the main instruction or other corresponding text based on that focus, then choose an icon (standard or otherwise) that matches the text. The main instruction text and icons should always match.

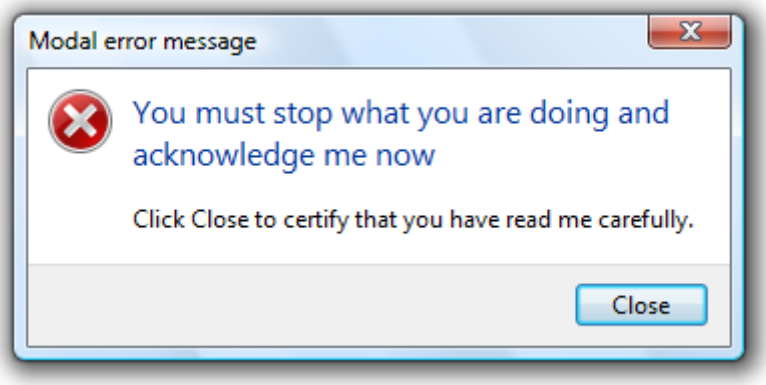
Error message presentation

Most error messages in Windows programs are presented using modal dialog boxes (as are most examples in this article), but there are other options:

- In-place
- Balloons
- Notifications
- Notification area icons
- Status bars

- Log files (for errors targeted at IT professionals)

Putting error messages in modal dialog boxes has the benefit of demanding the user's immediate attention and acknowledgement. However, this is also their primary drawback if that attention isn't necessary.



Do you really need to interrupt users so that they can click the Close button? If not, consider alternatives to using a modal dialog box.

Modal dialogs are a great choice when the user must acknowledge the problem immediately before continuing, but often a poor choice otherwise. Generally, you should prefer to use the lightest weight presentation that does the job well.

Avoid overcommunicating

Generally, users don't read, they scan. The more text there is, the harder the text is to scan, and the more likely users won't read the text at all. As a result, it is important to reduce the text down to its essentials, and use progressive disclosure and Help links when necessary to provide additional information.

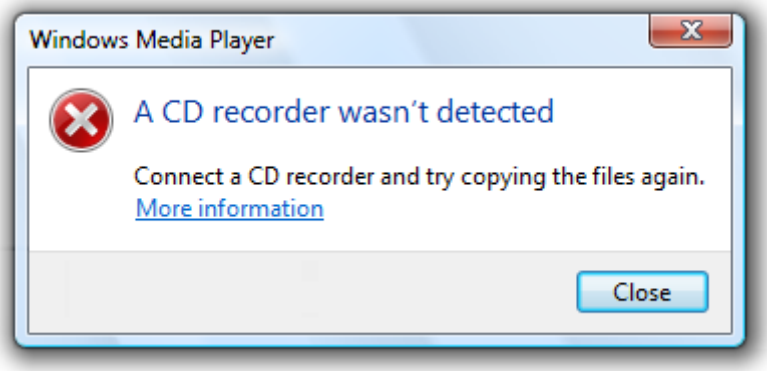
There are many extreme examples, but let's look at one more typical. The following example has most of the attributes of a good error message, but its text isn't concise and requires motivation to read.

Incorrect:

This example is a good error message, but it overcommunicates.

What is all this text really saying? Something like this:

Correct:



This error message has essentially the same information, but is far more concise.

By using Help to provide the details, this error message has an inverted pyramid style of presentation.

For more guidelines and examples on overcommunicating, see User Interface Text.

If you do only eight things

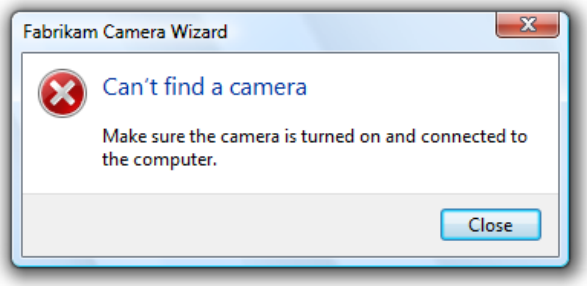
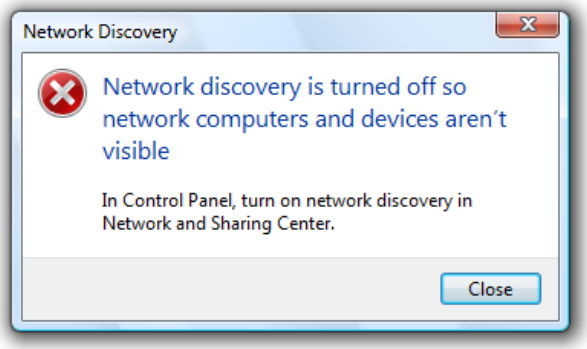
1. Design your program for error handling.
2. Don't give unnecessary error messages.
3. Avoid user confusion by giving necessary error messages.
4. Make sure the error message gives a problem, cause, and solution.
5. Make sure the error message is relevant, actionable, brief, clear, specific, courteous, and rare.
6. Design error messages from the user's point of view, not the program's point of view.

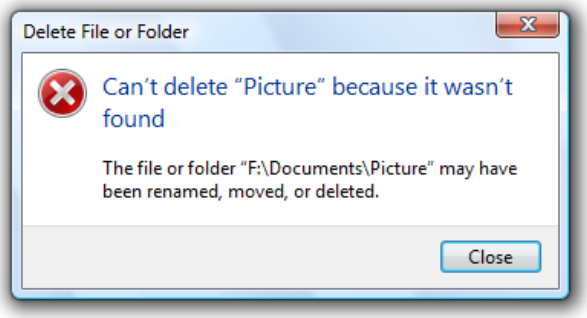
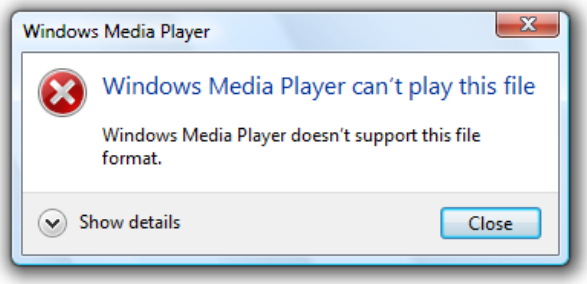
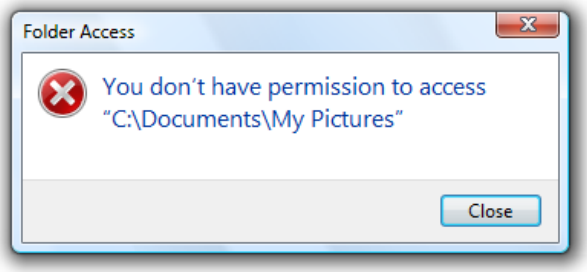
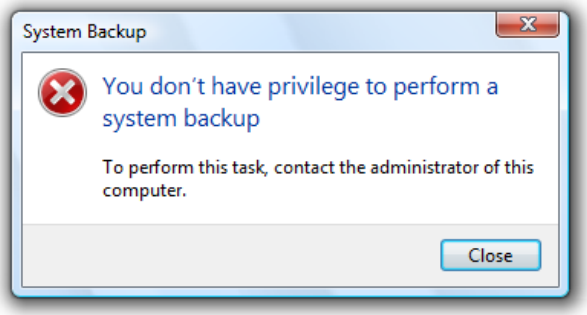
7. Avoid involving the user in troubleshooting use a different error message for each detectable cause.
8. Use the lightest weight presentation method that does the job well.

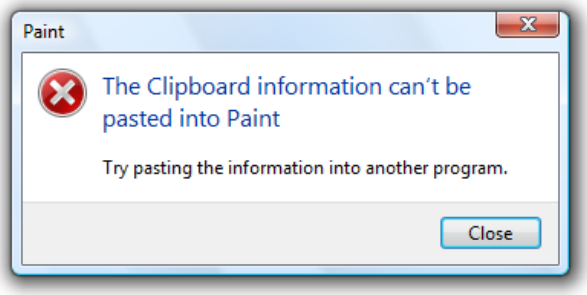
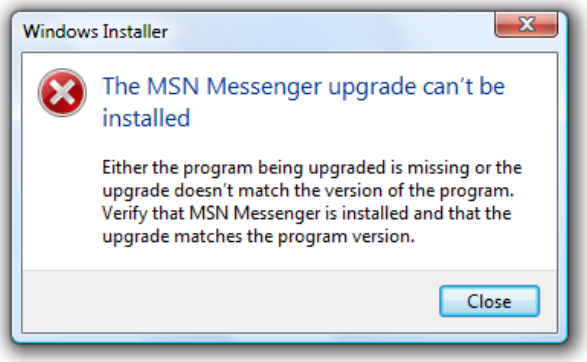
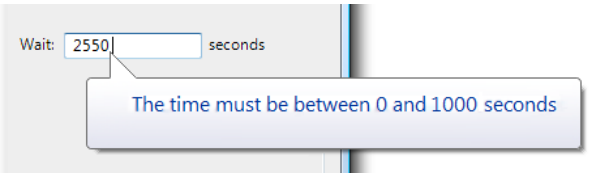
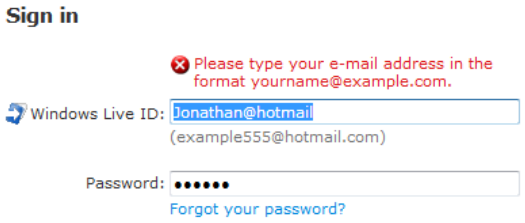
Usage patterns

Error messages have several usage patterns:

Expand table

Label	Value
System problems The operating system, hardware device, network, or program has failed or is not in the state required to perform a task.	Many system problems can be solved by the user: • Device problems can be solved by turning the device on, reconnecting the device, and inserting media. • Network problems can be solved by checking the physical network connect, and running **Network diagnose and repair**. • Program problems can be solved by changing program options or restarting the program.  In this example, the program can't find a camera to perform a user task.  In this example, a feature required to perform a task needs to be turned on.

Label	Value
File problems A file or folder required for a task initiated by the user was not found, is already in use, or doesn't have the expected format.	 In this example, the file or folder can't be deleted because it wasn't found.  In this example, the program doesn't support the given file format.
Security problems The user doesn't have permission to access a resource, or sufficient privilege to perform a task initiated by the user.	 In this example, the user doesn't have permission to access a resource.  In this example, the user doesn't have the privilege to perform a task.

Label	Value
Task problems There is a specific problem performing a task initiated by the user (other than a system, file not found, file format, or security problem).	 In this example, the Clipboard data can't be pasted into Paint.  In this example, the user can't install a software upgrade.
User input problems The user entered a value that is incorrect or inconsistent with other user input.	 In this example, the user entered an incorrect time value.  In this example, user input is not in the correct format.

Guidelines

Presentation

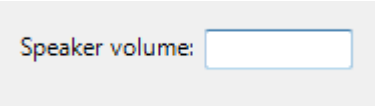
- **Use task dialogs whenever appropriate** to achieve a consistent look and layout. Task dialogs require Windows Vista or later, so they aren't suitable for earlier

versions of Windows. If you must use a message box, separate the main instruction from the supplemental instruction with two line breaks.

User input errors

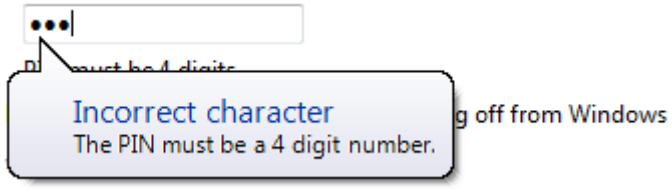
- **Whenever possible, prevent or reduce user input errors by:**
  - Using controls that are constrained to valid values.
  - Disabling controls and menu items when clicking would result in error, as long as it's obvious why the control or menu item is disabled.
  - Providing good default values.

Incorrect:



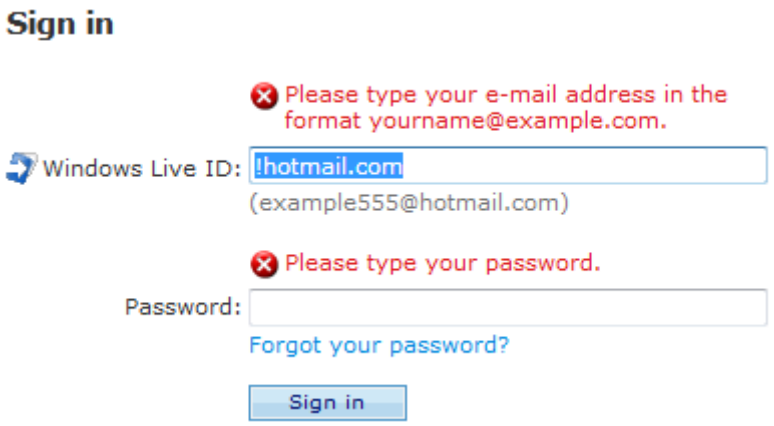
In this example, an unconstrained text box is used for constrained input. Use a slider instead.

- **Use modeless error handling (in-place errors or balloons) for contextual user input problems.**
- **Use balloons for non-critical, single-point user input problems detected while in a text box or immediately after a text box loses focus.** Balloons don't require available screen space or the dynamic layout that is required to display in-place messages. Display only a single balloon at a time. Because the problem isn't critical, no error icon is necessary. Balloons go away when clicked, when the problem is resolved, or after a timeout.



In this example, a balloon indicates an input problem while still in the control.

- **Use in-place errors for delayed error detection,** usually errors found by clicking a commit button. (Don't use in-place errors for settings that are immediately committed.) There can be multiple in-place errors at a time. Use normal text and a 16x16 pixel error icon, placing them directly next to the problem whenever possible. In-place errors don't go away unless the user commits and no other errors are found.



In this example, an in-place error is used for an error found by clicking the commit button.

- **Use modal error handling (task dialogs or message boxes) for all other problems,** including errors that involve multiple controls or are non-contextual or non-input errors found by clicking a commit button.
- **When a user input problem is reported, set input focus to the first control with the incorrect data.** Scroll the control into view if necessary. If the control is a text box, select the entire contents. It should always be obvious what the error message is referring to.
- **Don't clear incorrect input.** Instead, leave it so that the user can see and correct the problem without starting over.
  - **Exception:** Clear incorrect password and PIN text boxes because users can't correct masked input effectively.

Troubleshooting

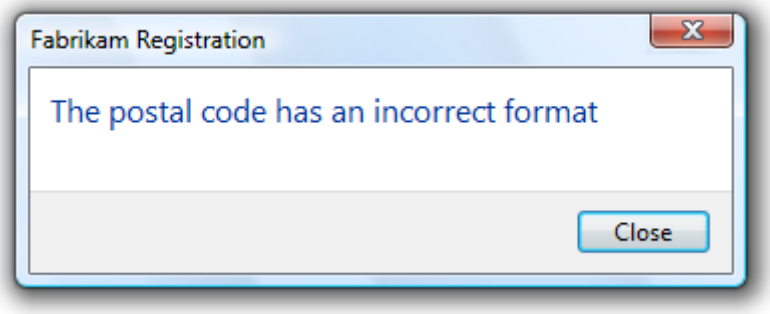
- **Avoid creating troubleshooting problems.** Don't rely on a single error message to report a problem with several different detectable causes.
- **Use a different error message (typically a different supplemental instruction) for each detectable cause.** For example, if a file cannot be opened for several reasons, provide a separate supplemental instruction for each reason.
- **Use a message with multiple causes only when the specific cause can't be determined.** In this case, present the solutions in order of likelihood of fixing the problem. Doing so helps users solve the problem more efficiently.

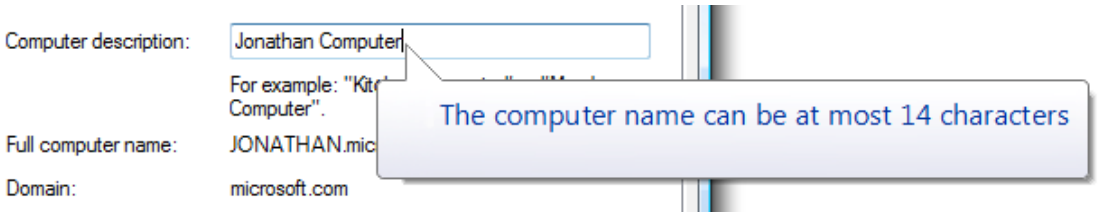
Icons

- **Modal error message dialogs don't have title bar icons.** Title bar icons are used as a visual distinction between primary windows and secondary windows.

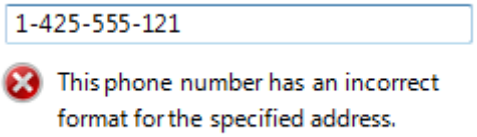
- **Use an error icon.** Exceptions:

- If the error is a user input problem displayed using a modal dialog box or balloon, don't use an icon. Doing so is counter to the encouraging tone of Windows. However, in-place error messages should use a small error icon (16x16 pixel) to clearly identify them as error messages.



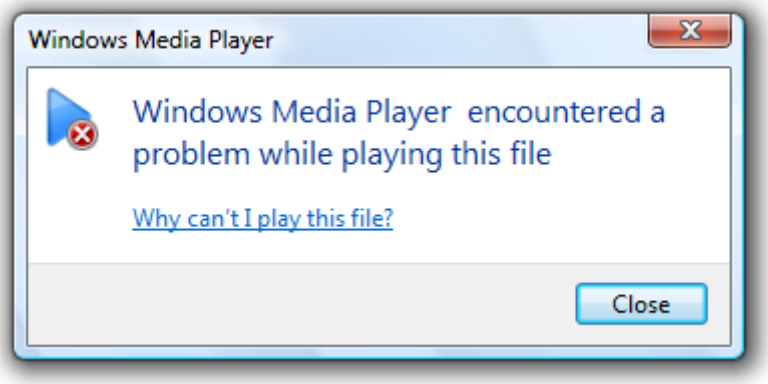


In these examples, user input problems don't need error icons.



In this example, an in-place error message needs a small error icon to clearly identify it as an error message.

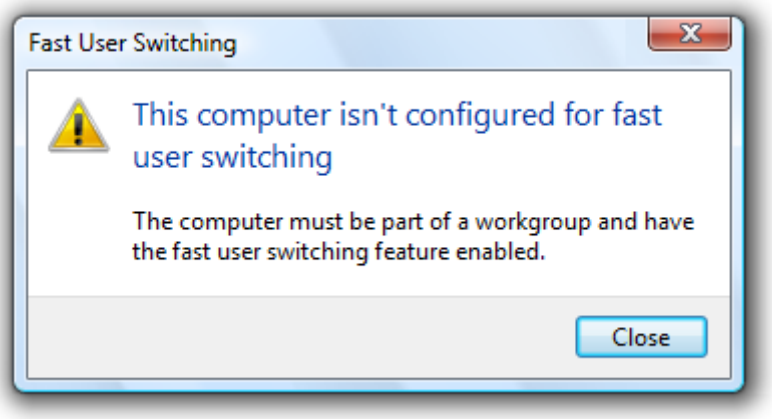
- If the problem is for a feature that has an icon (and not a user input problem), you can use the feature icon with an error overlay. If you do this, also use the feature name as the error's subject.



In this example, the feature icon has an error overlay, and the feature is the subject of the error.

- **Don't use warning icons for errors.** This is often done to make the presentation feel less severe. Errors aren't warnings.

  Incorrect:

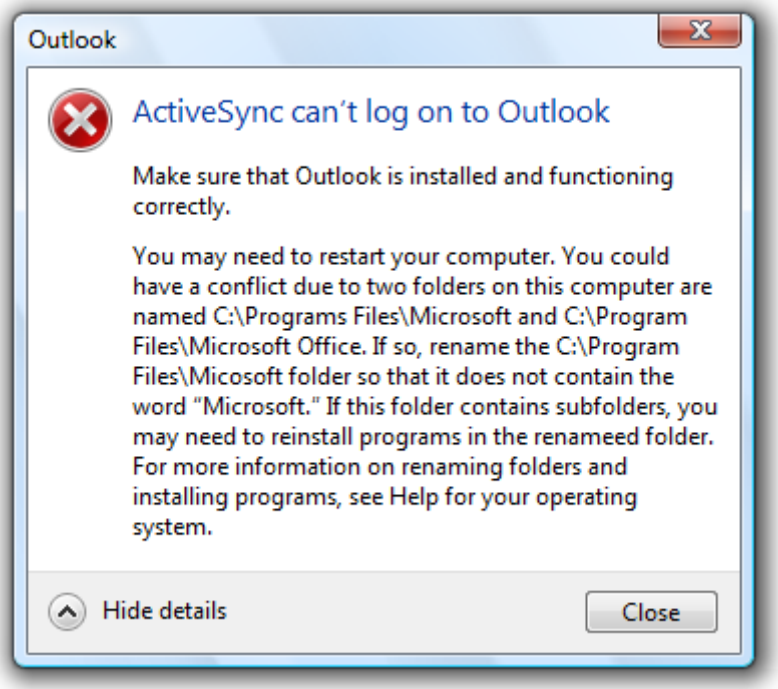
  

  In this example, a warning icon is incorrectly used to make the error feel less severe.

For more guidelines and examples, see Standard Icons.

Progressive disclosure

- **Use a Show/Hide details progressive disclosure button to hide advanced or detailed information in an error message.** Doing so simplifies the error message for typical usage. Don't hide needed information, because users might not find it.



In this example, the progressive disclosure button helps users drill down to more detail if they want it, or simplify the UI if they don't.

- **Don't use Show/Hide details unless there really is more detail.** Don't just restate the existing information in a more verbose format.
- **Don't use Show/Hide details to show Help information.** Use Help links instead.

For labeling guidelines, see Progressive Disclosure Controls.

Don't show this message again

- **If an error message needs this option, reconsider the error and its frequency.** If it has all the characteristics of a good error (relevant, actionable, and infrequent), it shouldn't make sense for users to suppress it.

For more guidelines, see Dialog Boxes.

Default values

- **Select the safest, least destructive, or most secure response to be the default.** If safety isn't a factor, select the most likely or convenient command.

Help

- **Design error messages to avoid the need for Help.** Ordinarily users shouldn't have to read external text to understand and solve the problem, unless the solution requires several steps.
- **Make sure the Help content is relevant and helpful.** It shouldn't be a verbose restatement of the error message rather, it should contain useful information that is beyond the scope of the error message, such as ways to avoid the problem in the future. Don't provide a Help link just because you can.
- **Use specific, concise, relevant Help links to access Help content.** Don't use command buttons or progressive disclosure for this purpose.
- **For error messages that you can't make specific and actionable, consider providing links to online Help content.** By doing so, you can provide users with additional information that you can update after the program is released.

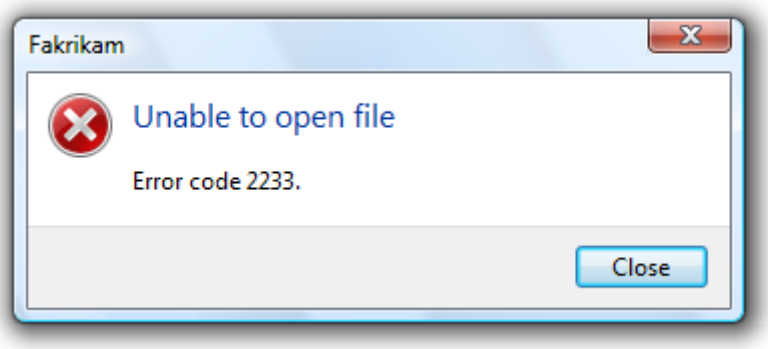
For more guidelines, see Help.

Error codes

- **For error messages that you can't make specific and actionable or they benefit from Help, consider also providing error codes.** Users often use these error codes to search the Internet for additional information.

- **Always provide a text description of the problem and solution.** Don't depend just on the error code for this purpose.

Incorrect:



In this example, an error code is used as a substitute for a solution text.

- **Assign a unique error code for each different cause.** Doing so avoids troubleshooting.
- **Choose error codes that are easily searchable on the Internet.** If you use 32-bit codes, use a hexadecimal representation with a leading "0x" and uppercase characters.

Correct:

1234

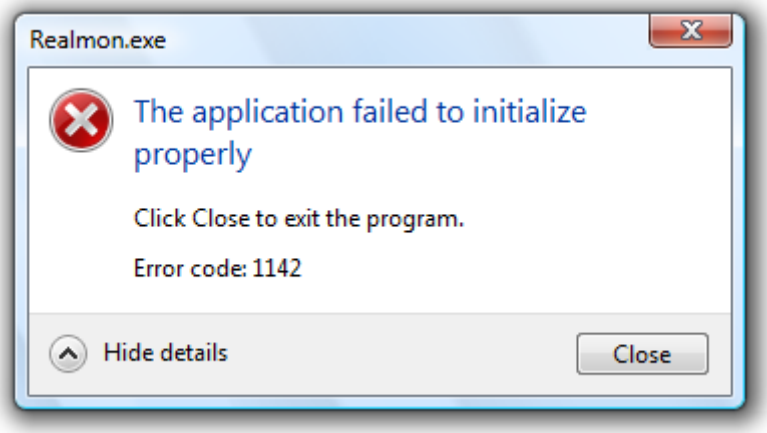
0xC0001234

Incorrect:

-1

-67113524

- **Use Show/Hide details to display error codes.** Phrase as Error code: `<error code>`.



In this example, an error code is used to supplement an error message that can benefit from further information.

Sound

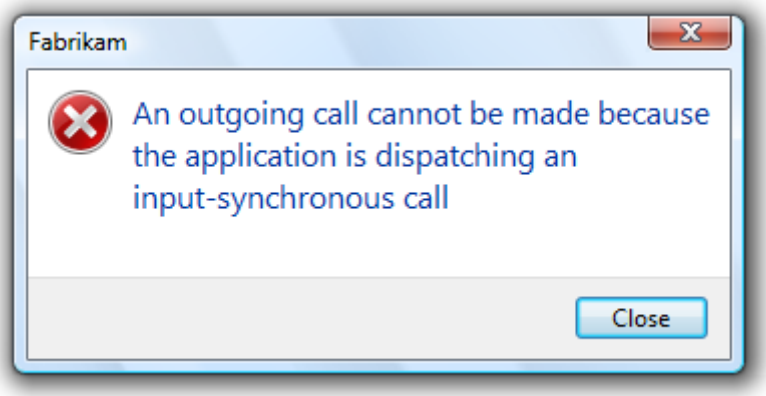
- **Don't accompany error messages with a sound effect or beep.** Doing so is jarring and unnecessary.
  - **Exception:** Play the Critical Stop sound effect if the problem is critical to the operation of the computer, and the user must take immediate action to prevent serious consequences.

Text

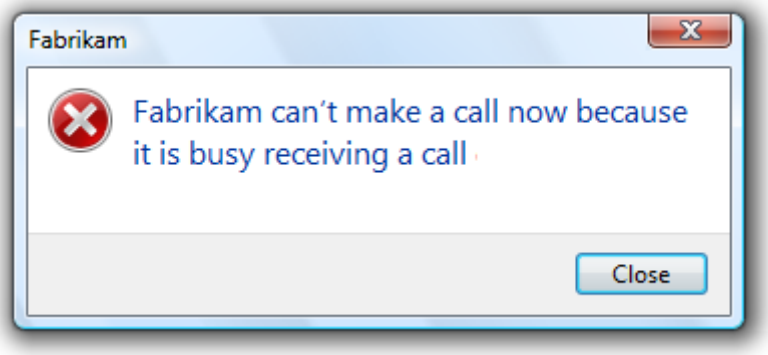
General

- **Remove redundant text.** Look for it in titles, main instructions, supplemental instructions, command links, and commit buttons. Generally, leave full text in instructions and interactive controls, and remove any redundancy from the other places.
- **Use user-centered explanations.** Describe the problem in terms of user actions or goals, not in terms of what the software is unhappy with. Use language that the target users understand and use. Avoid technical jargon.

Incorrect:



Correct:

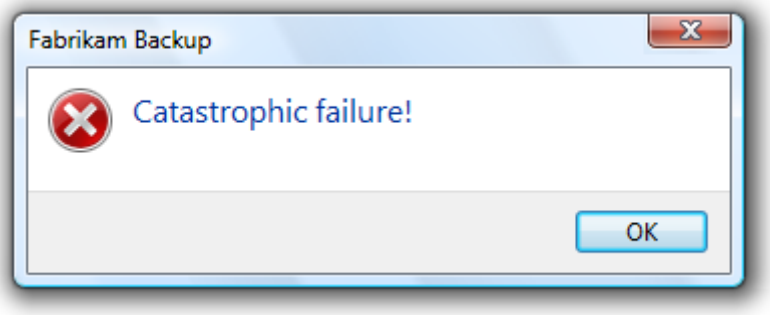


In these examples, the correct version speaks the user's language whereas the incorrect version is overly technical.

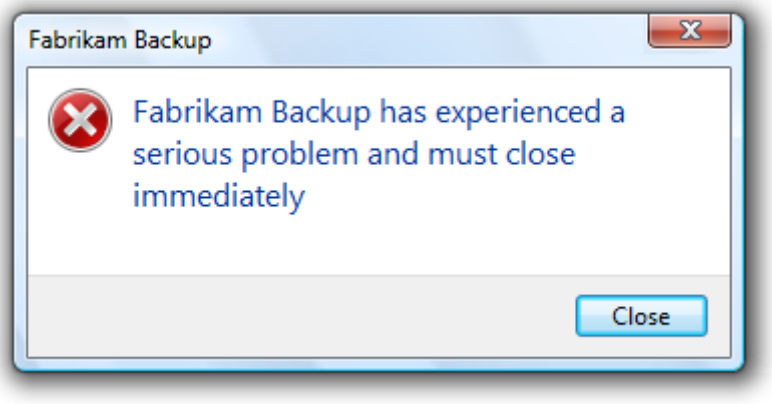
- **Don't use the following words:**
  - Error, failure (use problem instead)
  - Failed to (use unable to instead)
  - Illegal, invalid, bad (use incorrect instead)
  - Abort, kill, terminate (use stop instead)
  - Catastrophic, fatal (use serious instead)

These terms are unnecessary and contrary to the encouraging tone of Windows. When used correctly, the error icon sufficiently communicates that there is a problem.

Incorrect:



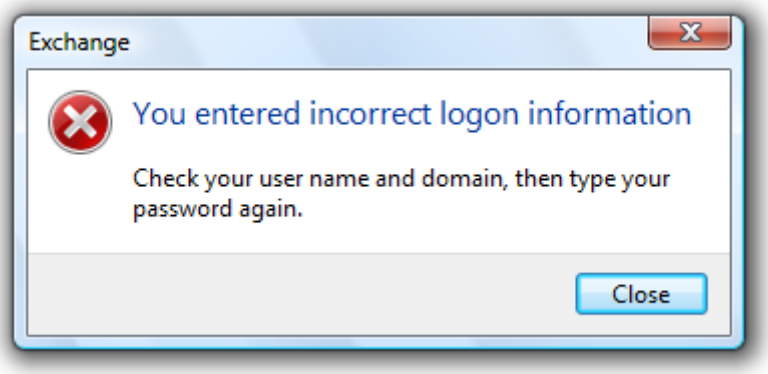
Correct:



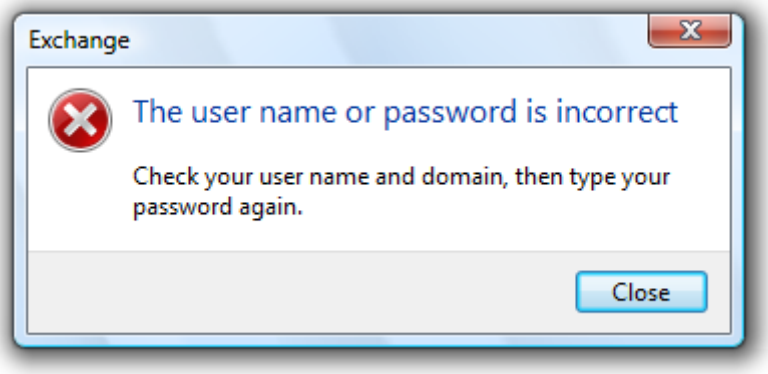
In the incorrect example, the terms "catastrophic" and "failure" are unnecessary.

- Don't use phrasing that blames the user or implies user error. Avoid using you and your in the phrasing. While the active voice is generally preferred, use the passive voice when the user is the subject and might feel blamed for the error if the active voice were used.

Incorrect:



Correct:



The incorrect example blames the user by using the active voice.

- **Be specific.** Avoid vague wording, such as syntax error and illegal operation. Provide specific names, locations, and values of the objects involved.

Incorrect:

File not found.

Disk is full.

Value out of range.

Character is invalid.

Device not available.

These problems would be much easier to solve with specific names, locations, and values.

- **Don't give possibly unlikely problems, causes, or solutions in an attempt to be specific.** Don't provide a problem, cause, or solution unless it is likely to be right. For example, it is better to say An unknown error occurred than something that is likely to be inaccurate.
- **Avoid the word "please,"** except in situations in which the user is asked to do something inconvenient (such as waiting) or the software is to blame for the situation.

Correct:

Please wait while Windows copies the files to your computer.

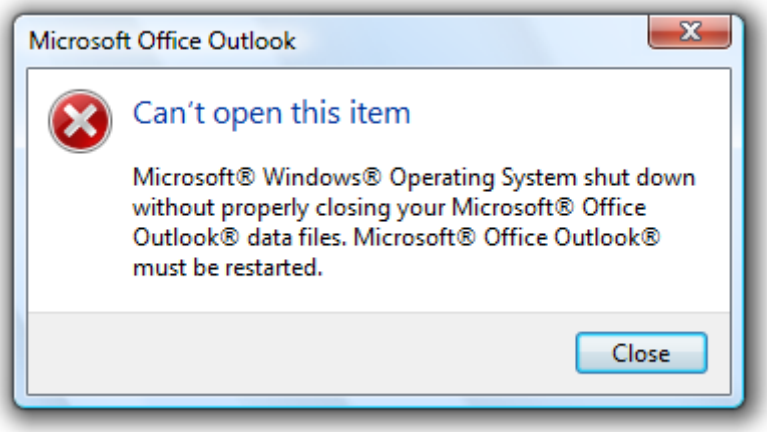
- **Use the word "sorry" only in error messages that result in serious problems for the user** (for example, data loss or inability to use the computer). Don't apologize if the issue occurred during the normal functioning of the program (for example, if the user needs to wait for a network connection to be found).

Correct:

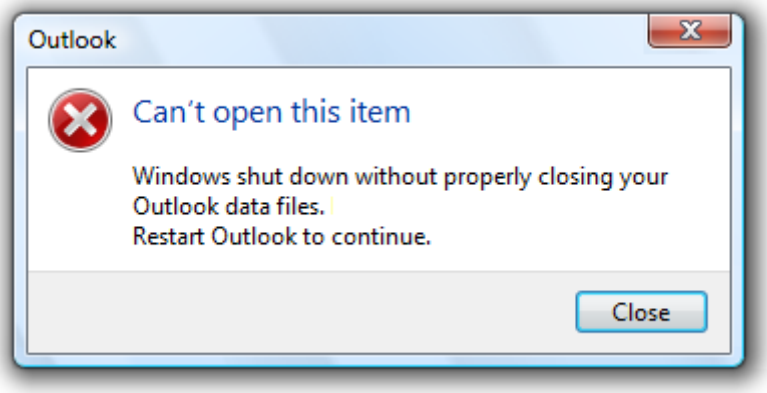
We're sorry, but Fabrikam Backup detected an unrecoverable problem and was shut down to protect files on your computer.

- **Refer to products using their short names.** Don't use full product names or trademark symbols. Don't include the company name unless users associate the company name with the product. Don't include program version numbers.

Incorrect:



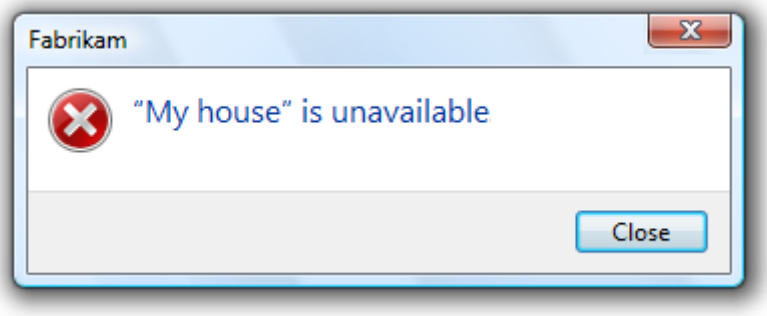
Correct:



In the incorrect example, full product names and trademark symbols are used.

- **Use double quotation marks around object names.** Doing so makes the text easier to parse and avoids potentially embarrassing statements.
  - **Exception:** Fully qualified file paths, URLs, and domain names don't need to be in double quotation marks.

Correct:



In this example, the error message would be confusing if the object name weren't in quotation marks.

- **Avoid starting sentences with object names.** Doing so is often difficult to parse.
- **Don't use exclamation marks or words with all capital letters.** Exclamation marks and capital letters make it feel like you are shouting at the user.

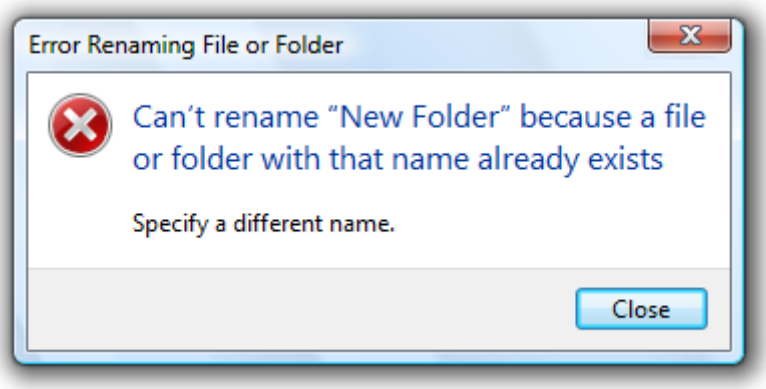
For more guidelines and examples, see Style and Tone.

Titles

- **Use the title to identify the command or feature from which the error originated.** Exceptions:
  - If an error is displayed by many different commands, consider using the program name instead.
  - If that title would be redundant or confusing with the main instruction, use the program name instead.

- **Don't use the title to explain or summarize the problem** that's the purpose of the main instruction.

Incorrect:



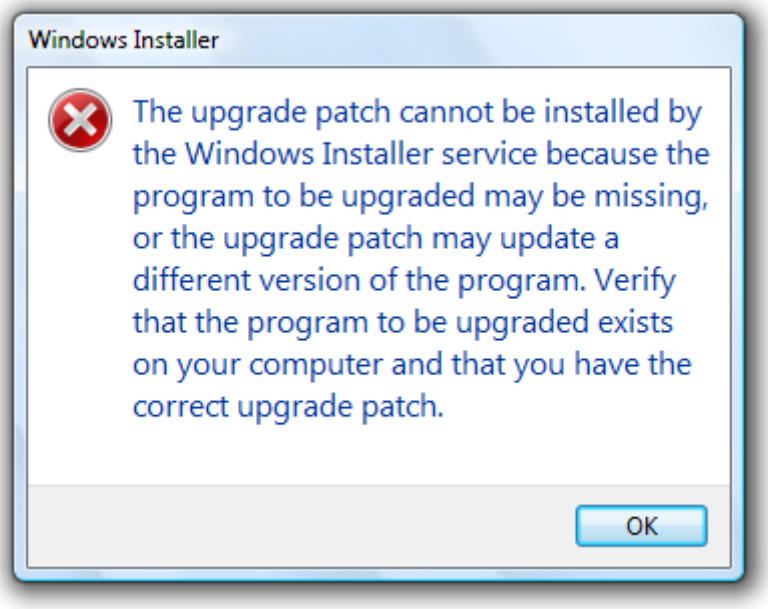
In this example, the title is being incorrectly used to explain the problem.

- Use title-style capitalization, without ending punctuation.

Main instructions

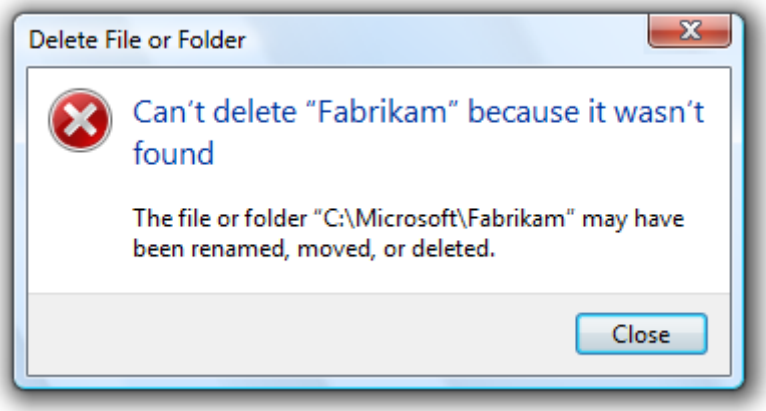
- **Use the main instruction to describe the problem in clear, plain, specific language.**
- **Be concise use only a single, complete sentence.** Pare the main instruction down to the essential information. You can leave the subject implicit if it is your program or the user. Include the reason for the problem if you can do so concisely. If you must explain anything more, use a supplemental instruction.

Incorrect:



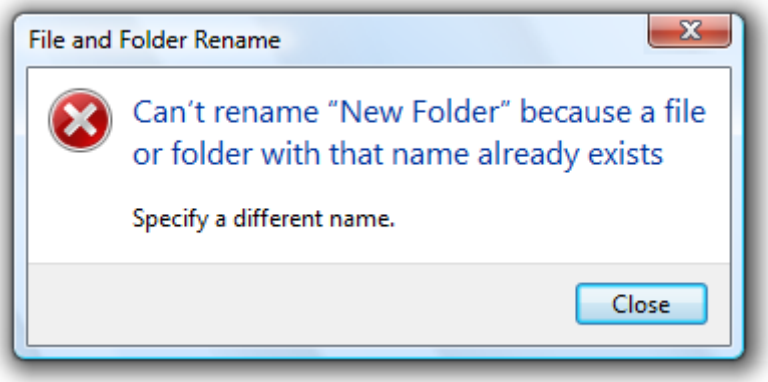
In this example, the entire error message is put in the main instruction, making it hard to read.

- **Be specific if there are objects involved, give their names.**
- **Avoid putting full file paths and URLs in the main instruction.** Rather, use a short name (such as the file name) and put the full name (such as the file path) in the supplemental instruction. However, you can put a single full file path or URL in the main instruction if the error message doesn't otherwise need a supplemental instruction.



In this example, only the file name is in the main instruction. The full path is in the supplemental instruction.

- **Don't give the full file path and URL at all if it's obvious from the context.**



In this example, the user is renaming a file from Windows Explorer. In this case, the full file path isn't needed because it's obvious from the context.

- Use present tense whenever possible.
- Use sentence-style capitalization.
- Don't include final periods if the instruction is a statement. If the instruction is a question, include a final question mark.

Main instruction templates

While there are no strict rules for phrasing, try using the following main instruction templates whenever possible:

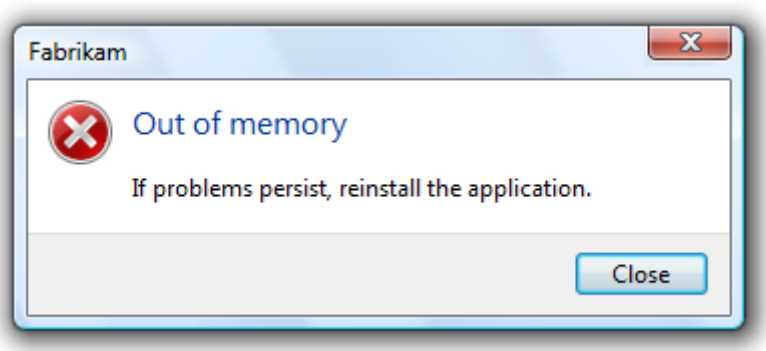
- [optional subject name] can't [perform action]
- [optional subject name] can't [perform action] because [reason]
- [optional subject name] can't [perform action] to "[object name]"
- [optional subject name] can't [perform action] to "[object name]" because [reason]
- There is not enough [resource] to [perform action]
- [Subject name] doesn't have a [object name] required for [purpose]
- [Device or setting] is turned off so that [undesired results]
- [Device or setting] isn't [available | found | turned on | enabled]
- "[object name]" is currently unavailable
- The user name or password is incorrect
- You don't have permission to access "[object name]"
- You don't have privilege to [perform action]
- [program name] has experienced a serious problem and must close immediately

Of course, make changes as needed for the main instruction to be grammatically correct and comply with the main instruction guidelines.

Supplemental instructions

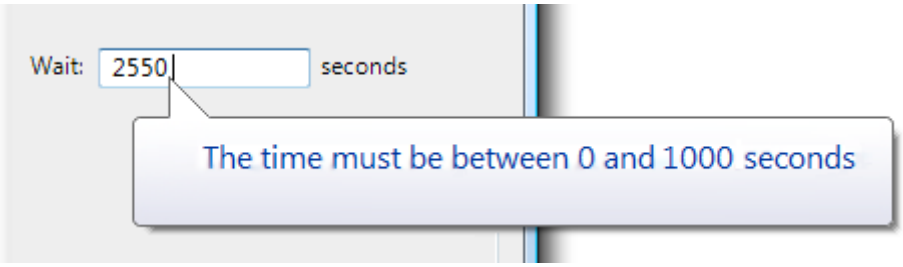
- Use the supplemental instruction to:
  - Give additional details about the problem.
  - Explain the cause of the problem.
  - List steps the user can take to fix the problem.
  - Provide measures to prevent the problem from reoccurring.
- **Whenever possible, propose a practical, helpful solution so users can fix the problem.** However, make sure the proposed solution is likely to solve the problem. Don't waste users' time by suggesting possible, but improbable, solutions.

Incorrect:



In this example, while the problem and its recommended solution are possible, they are very unlikely.

- **If the problem is an incorrect value that the user entered, use the supplemental instruction to explain the correct values.** Users shouldn't have to determine this information from another source.
- **Don't provide a solution if it can be trivially deduced from the problem statement.**



In this example, no supplemental instruction is necessary; the solution can be trivially deduced from the problem statement.

- **If the solution has multiple steps, present the steps in the order in which they should be completed.** However, avoid multi-step solutions because users have difficulty remembering more than two or three simple steps. If more steps are required, refer to the appropriate Help topic.
- **Keep supplemental instructions concise.** Provide only what users need to know. Omit unnecessary details. Aim for a maximum of three sentences of moderate length.
- **To avoid mistakes while users perform instructions, put the results before the action.**

Correct:

To restart Windows, click OK.

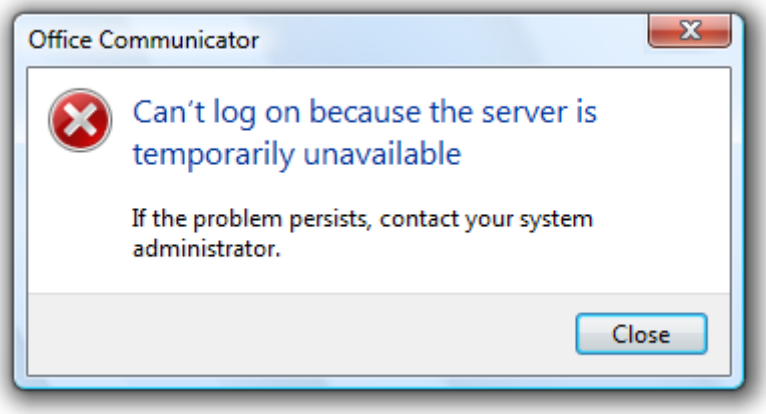
Incorrect:

Click OK to restart Windows.

In the incorrect example, users are more likely to click OK by accident.

- **Don't recommend contacting an administrator unless doing so is among the most likely solutions to the problem.** Reserve such solutions for problems that really can only be solved by an administrator.

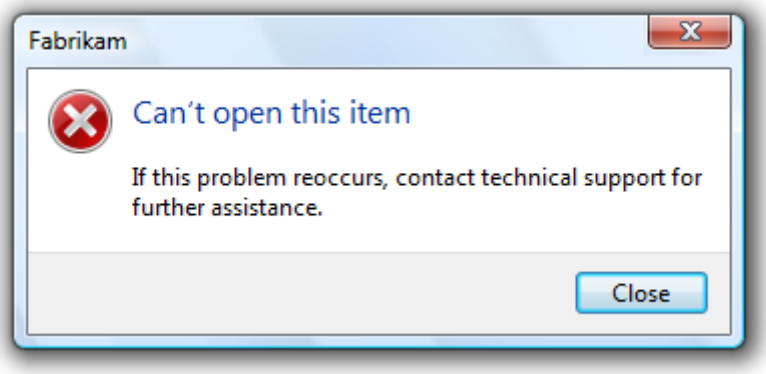
Incorrect:



In this example, most likely the problem is with the user's network connection, so it's not worth contacting an administrator.

- **Don't recommend contacting technical support.** The option to contact technical support to solve a problem is always available, and doesn't need to be promoted through error messages. Instead, focus on writing helpful error messages so that users can solve problems without contacting technical support.

Incorrect:



In this example, the error message incorrectly recommends contacting technical support.

- Use complete sentences, sentence-style capitalization, and ending punctuation.

Commit buttons

- If the error message provides command buttons or command links that solve the problem, follow their respective guidelines in Dialog Boxes.
- If the program must terminate as a result of the error, provide an Exit program button. To avoid confusion, don't use Close for this purpose.
- Otherwise, provide a Close button. Don't use OK for error messages, because this wording implies that problems are OK.
  - **Exception:** Use OK if your error reporting mechanism has fixed labels (as with the MessageBox API.)

Documentation

When referring to errors:

- Refer to errors by their main instruction. If the main instruction is long or detailed, summarize it.
- If necessary, you may refer to an error message dialog box as a message. Refer to as an error message only in programming and other technical documentation.
- When possible, format the text using bold. Otherwise, put the text in quotation marks only if required to prevent confusion.

Example: If you receive a **There is no CD disc in the drive** message, insert a new CD disc in the drive and try again.

Feedback

Was this page helpful? Yes No

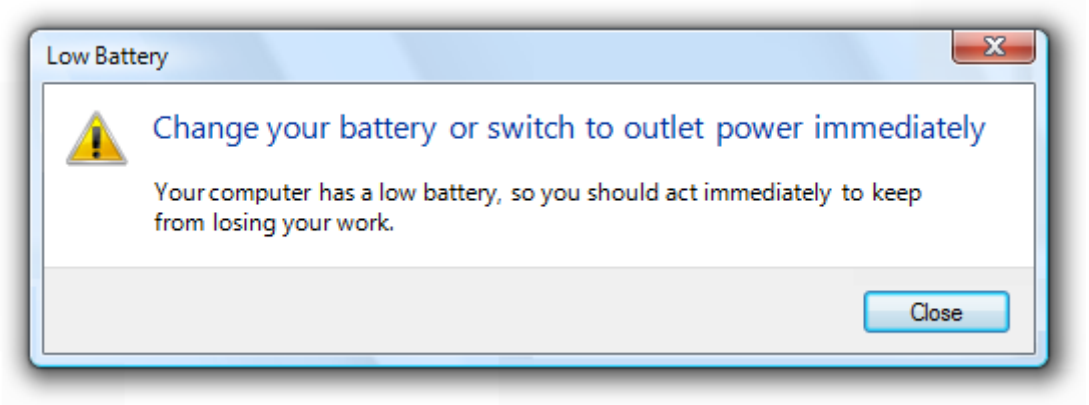
Provide product feedback | Get help at Microsoft Q&A

Warning Messages

Article • 08/26/2021

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

A warning message is a modal dialog box, in-place message, notification, or balloon that alerts the user of a condition that might cause a problem in the future.



A typical modal warning message.

The fundamental characteristic of warnings is that they involve the risk of losing one or more of the following:

- A valuable asset, such as important financial or other data.
- System access or integrity.
- Privacy or control over confidential information.
- User's time (a significant amount, such as 30 seconds or more).

By contrast, a confirmation is a modal dialog box that asks if the user wants to proceed with an action. Some types of warnings are presented as confirmations, and if so, the confirmation guidelines also apply.

Note: Guidelines related to dialog boxes, confirmations, error messagesstandard icons, notifications, and layout are presented in separate articles.

Is this the right user interface?

To decide, consider these questions:

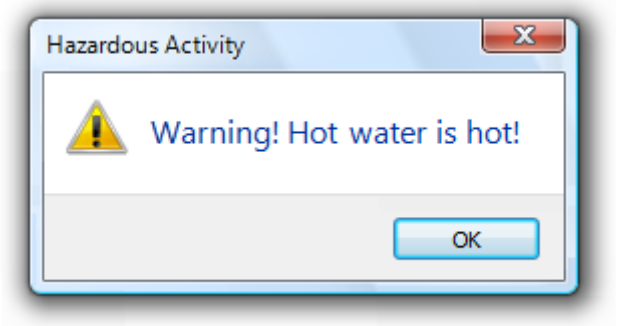
- **Is the user being alerted of a condition that might cause a problem in the future?** If not, the message isn't a warning.
- **Is the UI presenting an error or problem that has already occurred?** If so, use an error message instead.
- **Are users likely to perform an action or change their behavior as the result of the message?** If not, the condition doesn't justify interrupting the user so it's better to suppress the warning.
- **Is the condition the direct result of an action initiated by the user?** If not, consider using a non-critical event notifications.
- **Is the condition a special condition in a control?** If so, use a balloon instead.
- **For confirmations, is the user about to perform a risky action?** If so, a warning is appropriate if the action has significant consequences or cannot be easily undone.
- **For other types of warnings, does the user need to act now or in the immediate future?** Don't display warnings if users can continue to work productively without immediate problems. Postpone the warning until the condition is more immediate and relevant.

Design concepts

Avoid overwarning

We overwarn in Microsoft Windows programs. The typical Windows program has warnings seemingly everywhere, warning about things that have little significance. In some programs, nearly every question is presented as a warning. Overwarning makes using a program feel like a hazardous activity, and it detracts from truly significant issues.

Incorrect:



Overwarning makes your program feel hazardous and look like it was designed by lawyers.

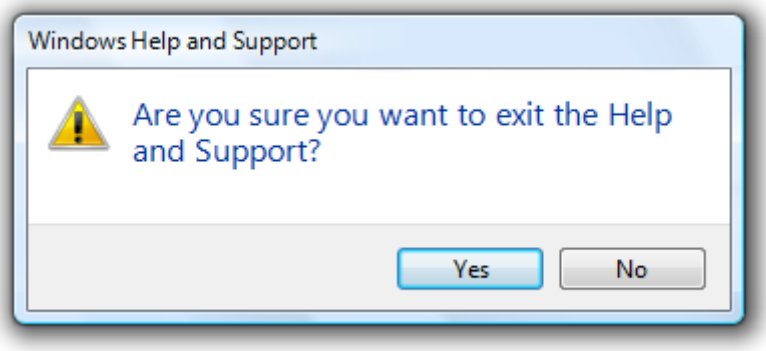
The mere potential for data loss or a future problem alone is insufficient to call for a warning. Additionally, any undesirable results should be unexpected or unintended, and not easily corrected. Otherwise, just about any user mistake could be construed to result in data loss or a potential problem of some kind and merit a warning.

Characteristics of good warnings

Good warnings:

- **Involve risk.** Good warnings alert users of something significant.

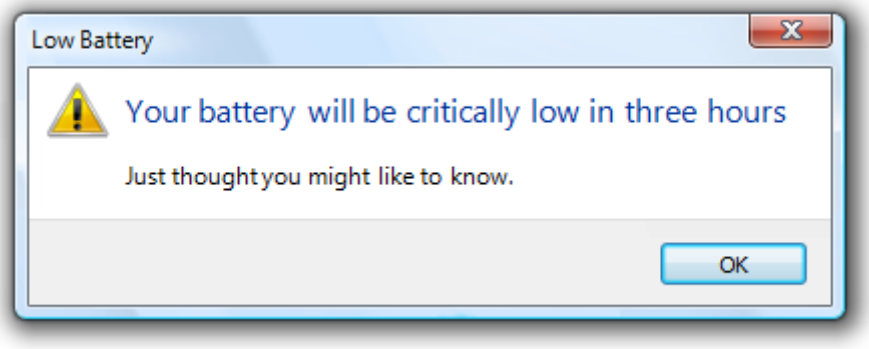
Incorrect:



So what? This confirmation assumes that users often exit programs by accident.

- **Have immediate relevance.** Not only do users have to care, they have to care now. Users typically aren't interested in problems they might have later as long as they can do their work now.

Incorrect:

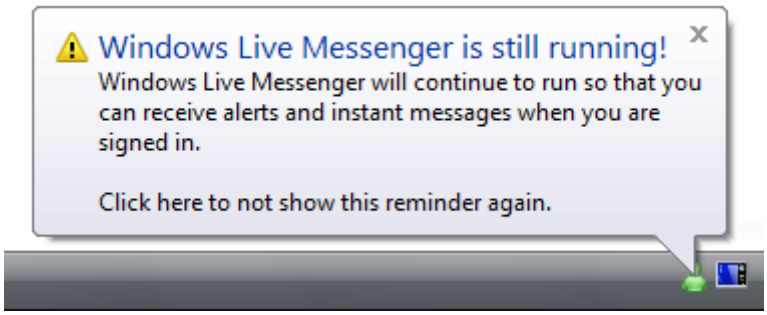


In this case, it's better just to warn the user in three hours.

- **Lead to action.** There is something users must do or be aware of as the result of the warning. Perhaps they must take an action now or sometime in the immediate future. Perhaps they will perform a task differently as a result. The consequence of

ignoring the warning should be clear. Warnings without actions just make users feel paranoid.

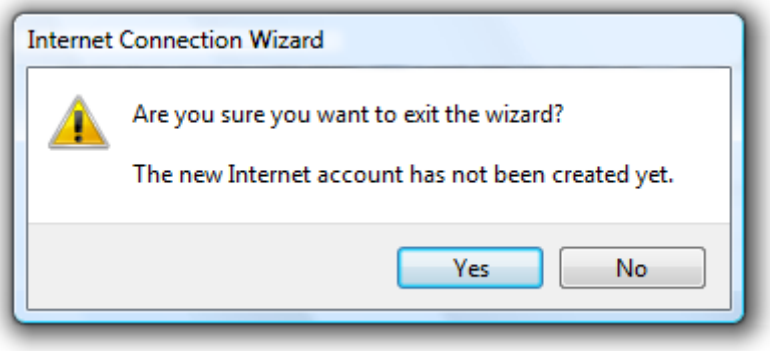
Incorrect:



Why is this notification a warning? What are users supposed to do (beside worry)?

- **Are not obvious.** Don't display a warning to state the obvious consequence of an action. For example, assume users understand the consequences of not completing a task.

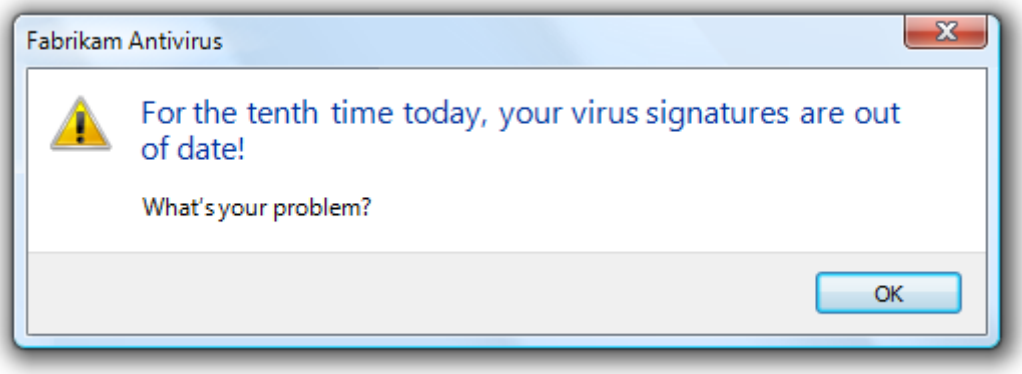
Incorrect:



Canceling an incomplete wizard means the task doesn't get done...who knew?

- **Occur infrequently.** Constant warnings quickly become ineffective and annoying. Users often become more focused on getting rid of the warning than addressing the problem.

Incorrect:



Users are more likely to focus on getting rid of the warning than fixing the underlying problem.

A message that doesn't have these characteristics might still be a good message, just not a good warning.

Determine the appropriate message type

Some issues can be presented as an error, warning, or information, depending on the emphasis and phrasing. For example, suppose a Web page cannot load an unsigned ActiveX control based on the current Windows Internet Explorer configuration:

- **Error.** "This page cannot load an unsigned ActiveX control." (Phrased as an existing problem.)
- **Warning.** "This page might not behave as expected because Windows Internet Explorer isn't configured to load unsigned ActiveX controls." or "Allow this page to install an unsigned ActiveX Control? Doing so from untrusted sources may harm your computer." (Both phrased as conditions that may cause future problems.)
- **Information.** "You have configured Windows Internet Explorer to block unsigned ActiveX controls." (Phrased as a statement of fact.)

To determine the appropriate message type, focus on the most important aspect of the issue that users need to know or act upon. Typically, if an issue blocks the user from proceeding, you should present it as an error; if the user can proceed, present it as a warning. Craft the main instruction or other corresponding text based on that focus, then choose an icon (standard or otherwise) that matches the text. The main instruction text and icons should always match.

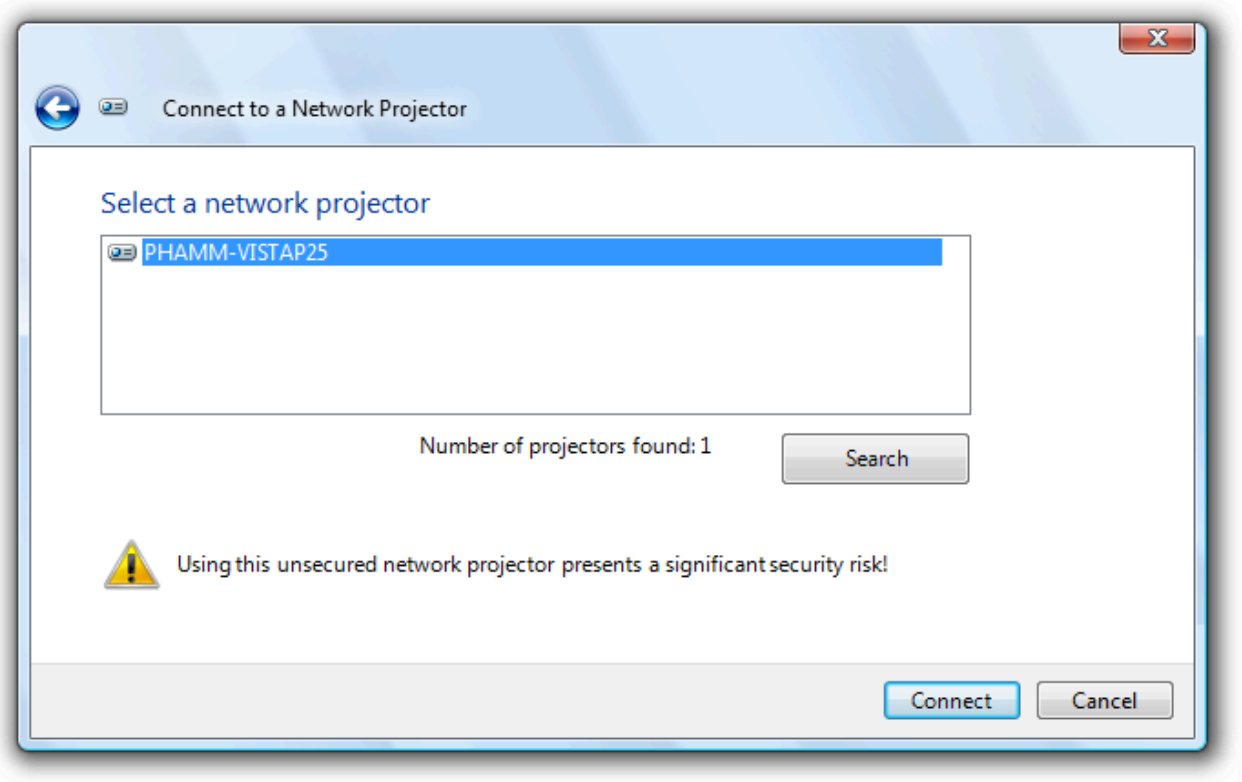
Be specific

Warnings are more compelling when the following information is specific and clear:

- The source of the warning.

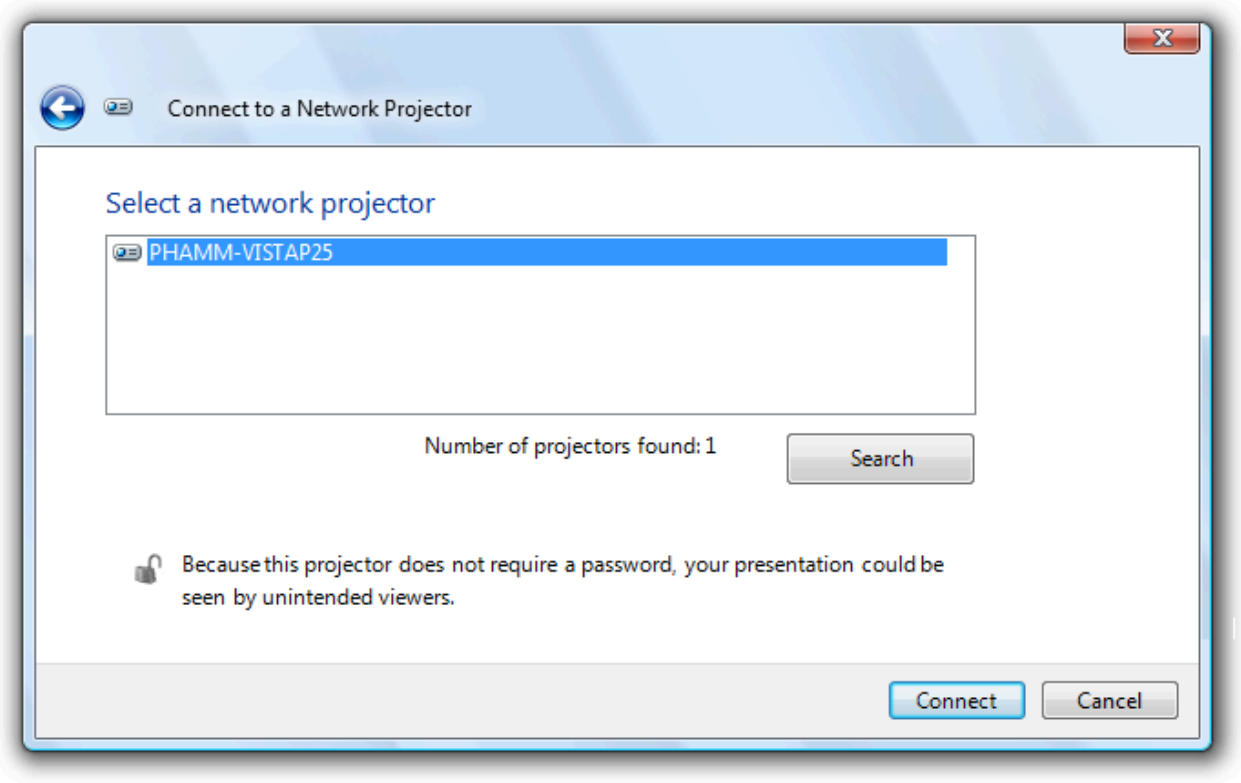
- The specific condition and potential problem.
- What the user should do about it.
- What happens if the user doesn't do anything.

Incorrect:



In this example, what is the potential problem? What is the user supposed to do, aside from not using the projector over the network? Without more specific information, all the user can do is feel bad about proceeding.

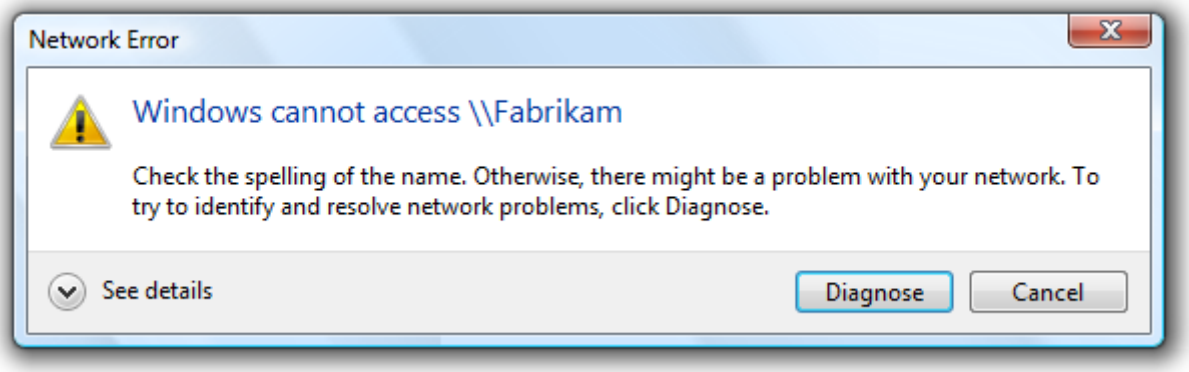
Correct:



In this example, the problem and consequences are clear.

Sometimes there is a legitimate potential problem worthy of informing users about, but the solution and consequences aren't known for sure. Rather than give a vague warning, be specific by giving the most likely information or the most common example.

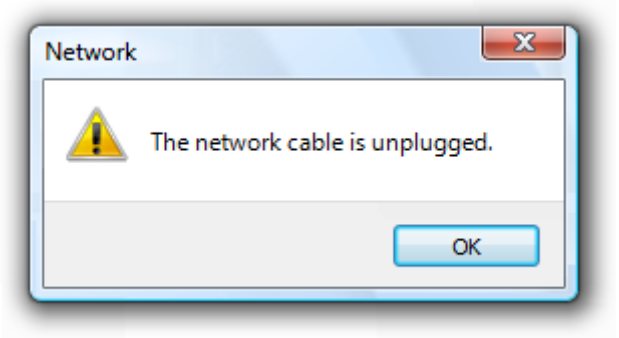
Correct:



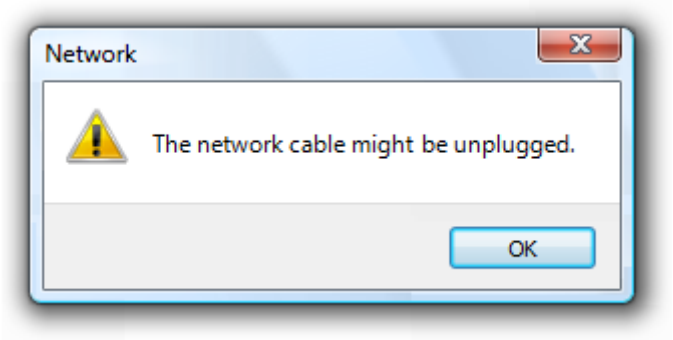
In this example, the warning is made specific by providing the most likely solution.

However, in such cases, use wording that indicates that there are other possibilities. Otherwise, users might be misled.

Incorrect:



Correct:



In the incorrect example, users will be confused if the cable is clearly plugged in.

If you do only two things...

1. Don't overwarn. Limit warnings to conditions that involve risk and are immediately relevant, actionable, not obvious, and infrequent. Otherwise, remove or rephrase the message.

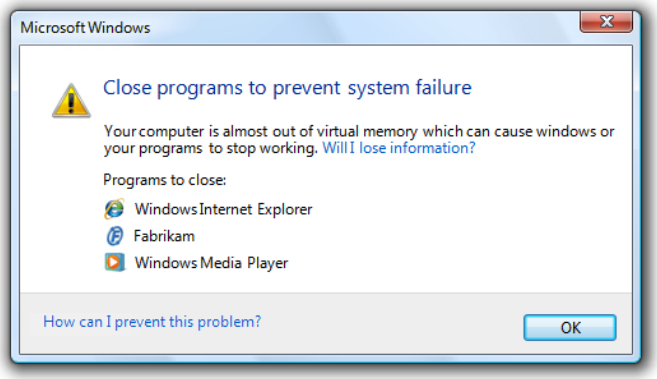
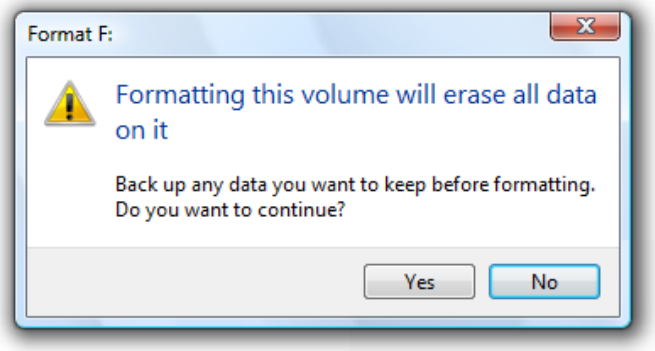
2. Provide specific, useful information.

Usage patterns

Warnings have several usage patterns:

Expand table

Label	Value
Awareness Make user aware of a condition or potential problem, but user may not have to do anything now.	 Examples of awareness warnings. Awareness warnings have the following presentation: • **Main instruction:** Describe the condition or potential problem. • **Supplemental instruction:** Explain the implication and why it is important. • **Commit buttons:** Close.
Error prevention Make user aware of information that might prevent a problem, especially when making choices.	Error prevention warnings are best presented using an in-place warning icon and explanatory text. Examples of error prevention warnings.

Label	Value
Imminent problem The user needs to do something now to prevent an imminent problem.	 An example of an imminent problem warning. Imminent problem warnings have the following presentation: • **Main instruction:** Describe what the user needs to do now. • **Supplemental instruction:** Explain the condition and why it is important. • **Commit buttons:** A command button or command link for each option, or OK if the action occurs outside the dialog box.
Risky action confirmation Confirm that the user wants to proceed with an action that has some risk and can't be easily undone.	 An example of risky action confirmation. Risky action confirmations have the following presentation: • **Main instruction:** Ask a question to determine if the user wants to proceed. • **Supplemental instruction:** Explain any non-obvious reasons why the user might not want to proceed. • **Commit buttons:** Yes, No. For guidelines on this pattern, see Confirmations.

Guidelines

Presentation

- **Choose the presentation UI based on the type of information:**

Expand table

User interface	Best used for
Modal dialog boxes	Critical warnings (including confirmations) that users must respond to now.
In-place	Information that might prevent a problem, especially when users are making choices.
Banners	Information that might prevent a problem, especially when related to completing a task.
Notifications	Significant events or status that can be safely ignored, at least temporarily.
Balloons	A control is in a state that affects input. This state is likely unintended and the user may not realize input is affected.

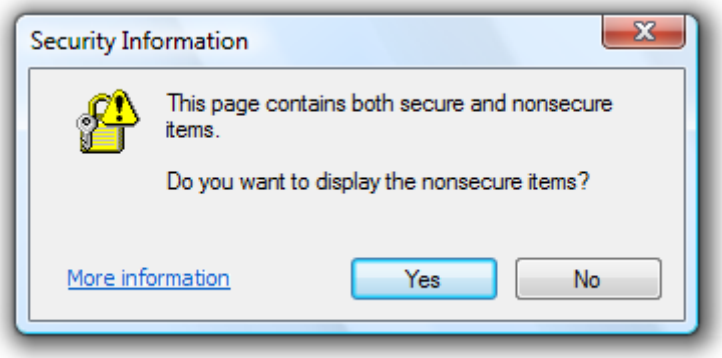
- **For modal dialog boxes:**
  - **Use task dialogs whenever appropriate to achieve a consistent look and layout.** Task dialogs require Windows Vista or later, so they aren't suitable for earlier versions of Windows.
  - **Display only one warning message per condition.** For example, display a single warning that completely explains a condition instead of describing it one detail at a time per message. Displaying a sequence of warning dialogs for a single condition is confusing and annoying.
  - **Don't display a warning more than once per condition.** Constant warnings quickly become ineffective and annoying. Users often become more focused on getting rid of the warning than addressing the problem. If you must warn repeatedly for a single condition, use progressive escalation.
- **Don't accompany warnings with a sound effect or beep.** Doing so is jarring and unnecessary.
  - **Exception:** If the user must respond immediately, you can use a sound effect.

Icons

- Don't place a warning icon in the title bar of a dialog box.
- **Use a warning icon.** Exceptions:

- If the warning is for a feature that has an icon, you can use the feature icon with a warning overlay.

  Correct:

  In this example, the feature icon has a warning overlay.

- For modal dialog boxes with a warning footnote, put the warning icon in the footnote instead of the content area.

  Correct:

  In this example, the footnote has the warning icon.

For more guidelines and examples, see Standard Icons.

Don't show this message again

- **If a warning dialog box needs this option, reconsider the warning and its frequency.** If it has all the characteristics of a good warning (involves risk, and is immediately relevant, actionable, not obvious, and infrequent), it shouldn't make sense for users to suppress it.

For more guidelines, see Dialog Boxes.

Progressive disclosure

- **If you must include advanced information in a warning message, reveal it by using progressive disclosure buttons** (for example, "Show details"). Doing so simplifies the warning for typical usage. Don't hide needed information because users might not find it.
- **Don't use "Show details" unless there really is more detail.** Don't just restate the existing information in a different format.

For labeling guidelines, see Progressive Disclosure.

Default values

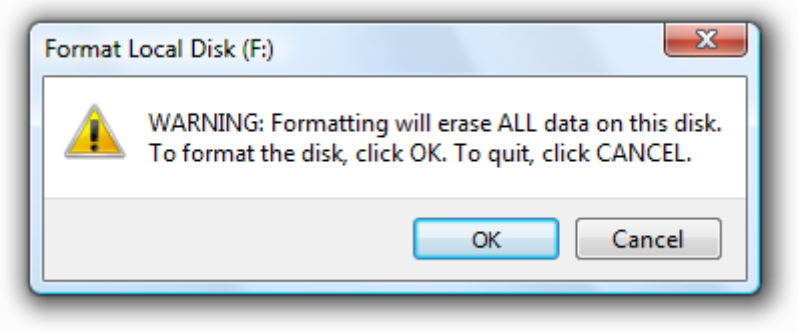
- **Select the safest, least destructive, or most secure response to be the default.**

Text

General

- **Remove redundant text.** Look for it in titles, main instructions, supplemental instructions, content areas, command links, and commit buttons. Generally, leave full text in instructions and interactive controls, and remove any redundancy from the other places.
- **Don't use the terms "warning" or "caution" in the text.** When used correctly, the warning icon sufficiently communicates that users must proceed with caution.

Incorrect:

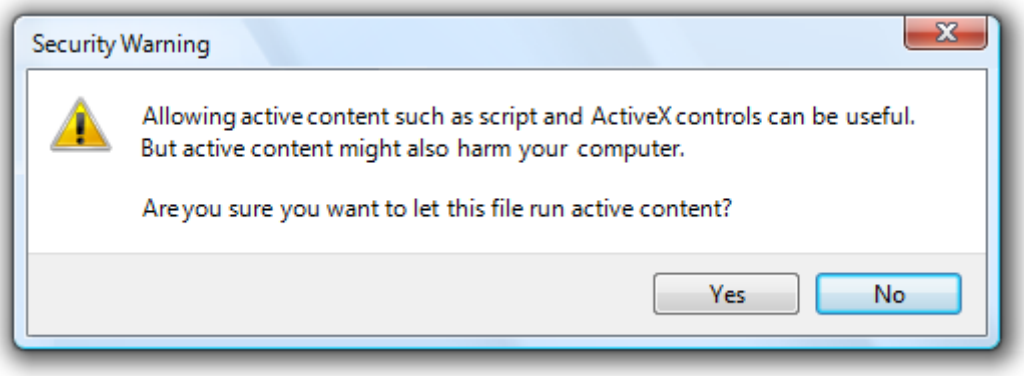


In this example, the term "warning" is unnecessary.

Titles

- **Use the title to identify the command or feature where the warning came from.** Exceptions:
  - If a warning is displayed by many different commands, consider using the program name instead.
  - If that title would be redundant or confusing with the main instruction, use the program name instead.

Incorrect:



In this example, "Security Warning" doesn't identify the command or feature where the warning came from.

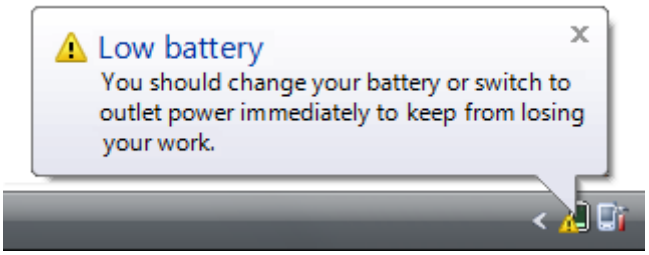
- **Don't use the title to explain what to do in the dialog** that's the purpose of the main instruction.
- Use title-style capitalization, without ending punctuation.

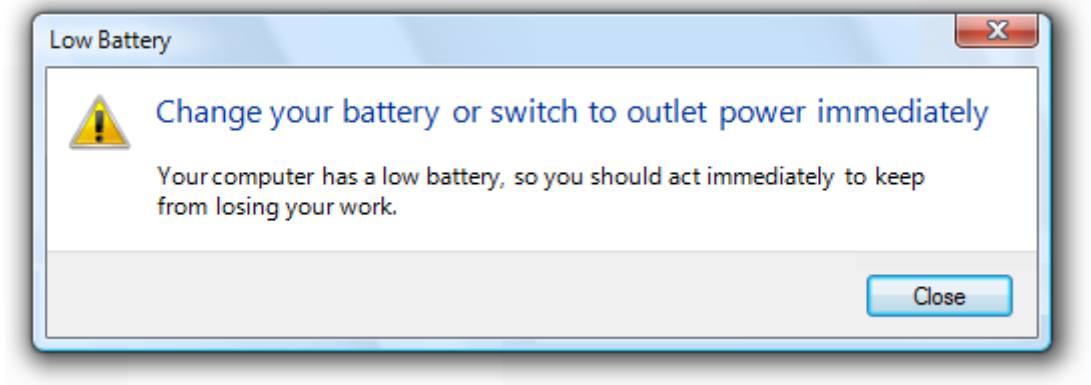
Main instructions

- The main instruction for a warning is based on its design pattern:

Expand table

Pattern	Main instruction
Awareness	Describe the condition or potential problem.
Imminent problem	Describe what the user needs to do now.
Risky action confirmation	Ask a question to determine if the user wants to proceed.

- 

- In this example, the low battery notification is an awareness warning, so the main instruction describes the condition.
- 

- In this example, the low battery dialog box is an imminent problem, so the main instruction describes what the user needs to do now.
- **Be concise use only a single, complete sentence.** Strip the main instruction down to the essential information. If you must explain anything more, use a supplemental instruction.
- **Use words like "now" and "immediately" if the user must act immediately.** Don't use these words if there is no urgency.
- **Be specific if there are objects involved, give their full names.**
- Use sentence-style capitalization.

Supplemental instructions

- The supplemental instruction for a warning is based on its design pattern:

Expand table

Pattern	Supplemental instruction
Awareness	Explain the implication and why it is important.
Imminent problem	Explain the condition and why it is important.
Risky action confirmation	Explain any non-obvious reasons why the user might not want to proceed.

- **Don't repeat the main instruction with slightly different wording.** Instead, omit the supplemental instruction if there is not more to add.

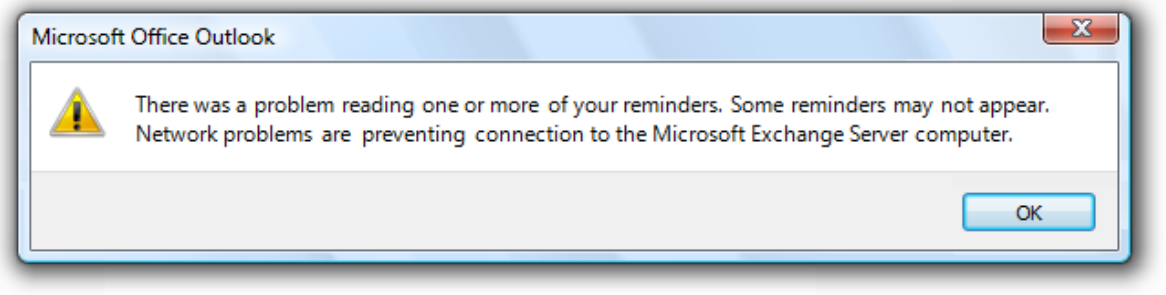
- Use complete sentences, sentence-style capitalization, and ending punctuation.

Commit buttons

- For warning dialog boxes, the commit buttons are based on its design pattern:

Expand table

Pattern	Commit buttons
Awareness	Close. Don't use OK because it suggests that potential problems are OK.
Imminent problem	A command button or command link for each option, or OK if the action occurs outside the dialog box.
Risky action confirmation	Yes, No.

- **Incorrect:**
- 

- Problems aren't OK, so use Close instead.

Documentation

When referring to warnings:

- If the warning asks a question, refer to a warning by its question; otherwise, use the main instruction. If the question or main instruction is long or detailed, summarize it.
- If necessary, you may refer to a warning dialog box as a message.
- When possible, format the text using bold. Otherwise, put the text in quotation marks only if required to prevent confusion.

Example: In the **Do you want to display the nonsecure items?** message, click Yes.

Feedback

Was this page helpful? Yes No

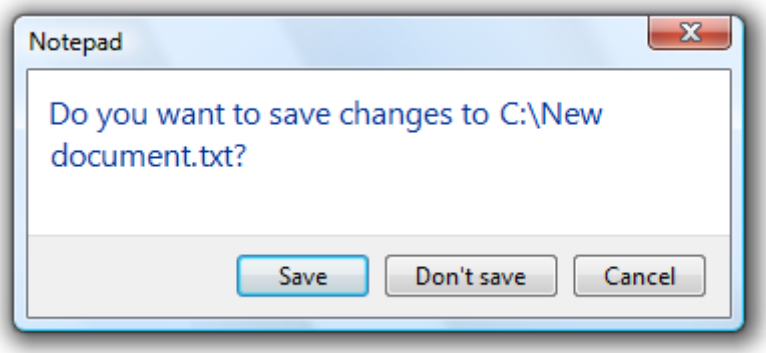
Provide product feedback | Get help at Microsoft Q&A

Confirmations

Article • 08/26/2021

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

A confirmation is a modal dialog box that asks if the user wants to proceed with an action.



A typical confirmation.

Confirmations have these essential characteristics:

- They are displayed as the direct result of an action initiated by the user.
- They verify that the user wants to proceed with the action.
- They consist of a simple question and two or more responses.

Confirmations are most useful when the action requires the user to make a relevant and distinct choice that can't be made later. That choice often involves some element of risk that isn't obvious to the user, but risk isn't essential to confirmations. These elements are necessary to justify the interruption of responding to a modal dialog.

By contrast, warning messages present a condition that might cause a problem in the future. Their fundamental characteristic is that they involve risk:

- They involve potential loss of one or more of the following:
  - A valuable asset, such as data loss or financial loss.
  - System access or integrity.
  - Privacy or control over confidential information.
  - User's time (a significant amount, such as 30 seconds or more).

- They have unexpected or unintended consequences.
- They require correct handling now because mistakes can't be easily fixed, and may even be irreversible.

If a confirmation involves risk, it can be considered a warning as well. Consequently, the warning message guidelines also apply.

Note: Guidelines related to dialog boxes, error messages, layout, and warning messages are presented in separate articles.

Is this the right user interface?

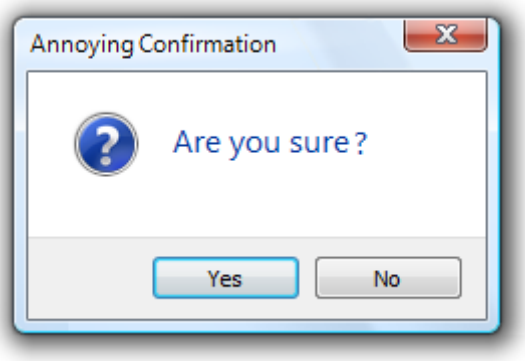
To decide, consider these questions:

- **Is the user being asked a question to proceed with an action that has two or more responses?** If not, the message isn't a confirmation.
- **Is the UI presenting an error or problem that has occurred?** If so, use an error message instead.
- **Does proceeding with the action require the user to make a choice that doesn't have a suitable default?** If so, a confirmation may be appropriate.
- **Is there an alternative design that eliminates the need for the confirmation?** The need for a confirmation sometimes indicates a design flaw. Often there is a better design alternative that doesn't need a confirmation.
- **Is the user about to perform a risky action?** If so, a confirmation is appropriate if the action has significant consequences or cannot be easily undone.
- **Is the user about to abandon a task?** If so, don't confirm. Assume users understand the consequences of not completing a task.
- **Does the action have consequences that users might not be aware of?** If so, a confirmation may be appropriate.
- **Given the current context, are users likely to be performing an action in error?** If so, a confirmation may be appropriate.
- **Do users perform the action frequently?** If so, consider an alternative design. Frequent confirmations are annoying and have little value because users learn to respond without thinking.
- **Does the action have security implications?** If so, a confirmation may be required even if the previous tests indicate otherwise.

Design concepts

Unnecessary confirmations are annoying

The first Windows confirmation ever created undoubtedly looked like this:



The original annoying confirmation.

This was a very bad start. If you want users to hate your program, just sprinkle confirmations like this throughout. To understand why, consider the user's point of view. The user just asked to perform an action by the definition of a confirmation so unless something was somehow clicked or pressed accidentally, of course the user wants to proceed.

Not only are unnecessary confirmations annoying, but they aren't effective in protecting the user from mistakes. Users quickly discover when a program has unnecessary confirmations and their natural response is to dismiss them as quickly as possible, often without reading. Consequently, such confirmations do little more than add an extra step to these tasks.

Don't use confirmations just because there is the possibility of users making a mistake. **Rather, confirmations are most effective when used to confirm actions that have significant or unintended consequences.** Good confirmations never state the obvious; they should communicate something users need to be aware of a good reason not to continue. And they are used only when they are really needed by an action, such as asking users to save changes only when there are changes worth saving. Doing so demands the user's attention only when it is truly warranted.

For other types of confirmations, there is often a better design alternative than forcing users to answer a question.

Consider the design alternatives

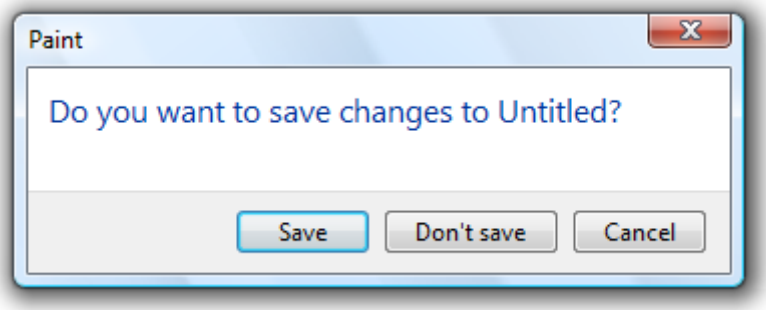
Here are some design alternatives that eliminate the need for routine confirmations:

- **Prevent errors.** Design tasks so that significant mistakes are difficult to do accidentally. For example, physically separate destructive commands from other commands, and require multiple actions to complete.

- **Provide undo.** Provide the ability to revert actions. For example, deleting a file in Microsoft Windows usually doesn't require a confirmation because deleted files can be recovered from the Recycle Bin. Note that if an action is very easy to perform, just having users redo the action may be sufficient.
- **Provide feedback.** Make undesirable outcomes obvious. Providing undo alone isn't sufficient if users don't realize when they make a mistake. For example, the effect of direct manipulation (such as a drag-and-drop operation) should always be obvious.
- **Assume the probable outcome, but make it easy to change.** If you aren't sure what users want but there is a likely, safe, and secure choice, assume that choice, make it clear what happened, and make it easy to change using a context menu. For example, Microsoft Word assumes that users want to spell words correctly. If it recognizes a misspelled word and it knows the likely correct spelling, Word automatically makes the correction but allows users to revert.
- **Eliminate the choice completely.** If the choice isn't important, users just won't care. Better to simplify your program and eliminate the choice.

Make confirmations require thought

For a confirmation to have value, users need to understand the reason not to proceed. Sometimes the reason is obvious, as when users are closing a document with changes that haven't been saved:

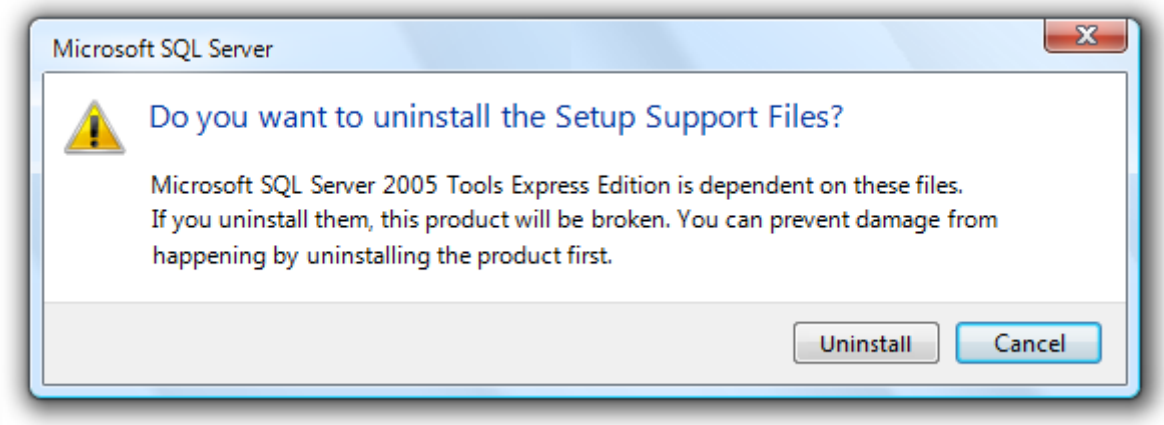


In this example, the reason for the confirmation is obvious.

In other situations, the reason might not be so obvious.

When choosing commit button labels for dialog boxes, the general guidelines is to choose labels that are specific responses to the main instruction. This leads to efficient decision making because users have to read a minimum amount of text to proceed. However, this efficiency goal can be counterproductive for confirmations. Consider this example:

Incorrect:

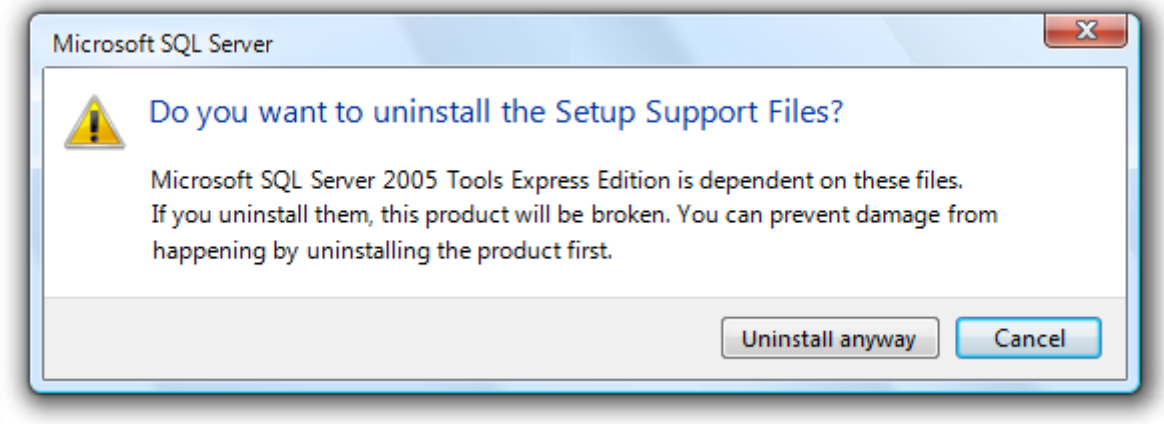


In this example, the correct response requires thought.

If you present this confirmation immediately after the user gave the Uninstall command, the user's response is likely to be "Of course I want to uninstall!" The user will click Uninstall without giving it a second thought.

For confirmations, we don't want users making hasty, emotional decisions. To encourage users to think about their response, we need to provide a small decision-making speed bump. When practical, it's usually better to do this by carefully phrasing commit buttons. For example, we can use additional language to indicate that there is a reason not to continue.

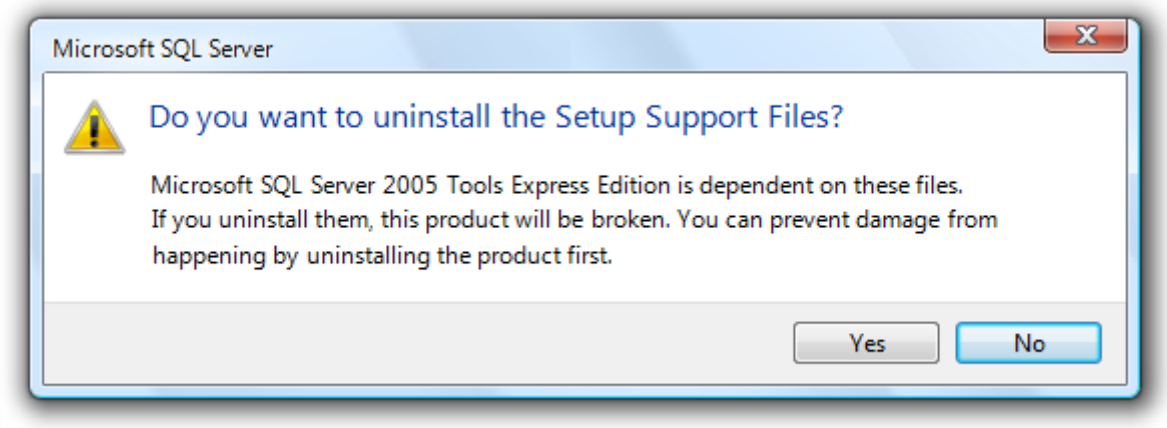
Better:



In this example, "anyway" is added to the commit button label to indicate that the confirmation gives a reason not to continue.

If that approach isn't practical, we can use Yes/No commit buttons.

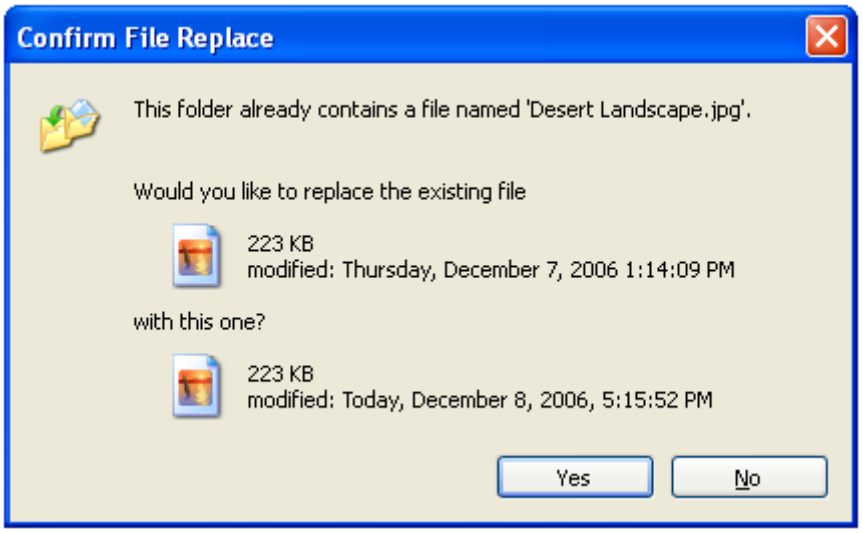
Also better:



In this example, using Yes/No commit buttons forces users to at least read the main instruction.

Provide all the information

If you are going to ask a question, you must provide sufficient information for users to answer that question intelligently. Consider the Confirm File Replace dialog from Windows XP:



The Windows XP Confirm File Replace dialog box.

Does this confirmation provide all the information users might need to answer the question? Before you answer, consider the most common user scenarios:

- Copy (or move) the other file, replacing the existing one.
- Keep the existing file, without copying or moving the other file.
- Keep or copy the newer file (top scenario).
- Either keep the existing file or copy the other file, depending on criteria such as file contents and size.
- Keep the existing file and copy the other file using a different name.

- Cancel the operation if something is wrong or unexpected.

Users can achieve scenario 1 by clicking Yes and scenario 2 by clicking No. They can achieve scenario 3 by comparing the file dates and clicking the appropriate button, but notice how much thought it takes to determine the newer file and then determine the appropriate button especially for what is likely to be the most common scenario.

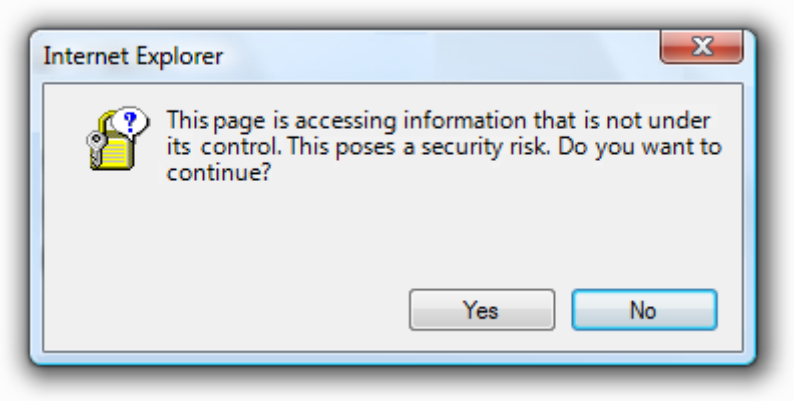
Scenarios 4, 5, and 6 are also surprisingly difficult. The file sizes are rounded off, so, for example, it is impossible to determine if these files have the same size or even if they are the same file. The icons are for the application used to open the file, so users would have to open the files to inspect and compare their content. Having thumbnails of the file content would be far more useful in answering the question.

The Copy File confirmation from Windows Vista does a much better job of handling these scenarios by providing more information and adding the option to keep both files:

The Windows Vista Copy File confirmation.

Provide specific, useful information

If you are going to ask a question, ensure that users understand the question and the implications of the alternative responses. Consider this Windows Internet Explorer security confirmation:



A vague security confirmation.

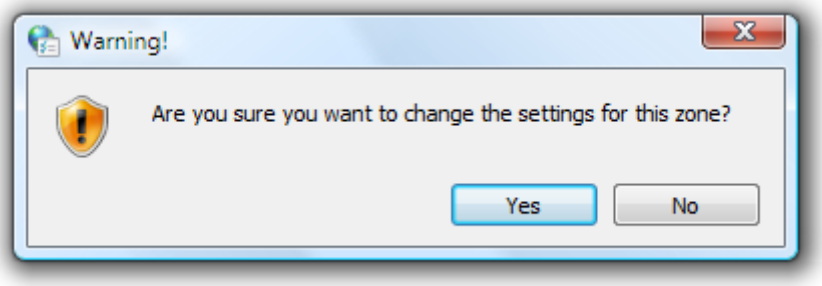
This confirmation asks a question that users can't possibly answer intelligently. The user has requested that Windows Internet Explorer display a page, and this message advises against it implicitly through the wording of the text and by highlighting No as the default choice.

The specific security issue that the page poses is not sufficiently explained, so the risk of continuing isn't clear. What information in the confirmation would cause the user ever to click No? Because of the vagueness of the message, the confirmation isn't likely to discourage users from continuing, but will make them feel bad about doing so.

For this confirmation to be useful, it must provide more information specific information that might cause the user to decide not to proceed. In general, for each response in a confirmation, consider the scenarios that require it and make sure that there is sufficient information provided for users to want to choose it. Provide choices, not dilemmas.

How to determine if a confirmation is necessary

Thinking through the scenarios and the likelihood of choosing each response suggests a systematic way to determine if a confirmation is necessary. If users are likely to select all of the responses, the confirmation is necessary and useful. However, if only one response is likely (say 98 percent of the time), the confirmation is clearly unnecessary and should be removed. Note that confirmations related to security, legal, and safety issues are possible exceptions.



Is this confirmation necessary? Will users ever select No? It's possible but very improbable. This confirmation should be removed.

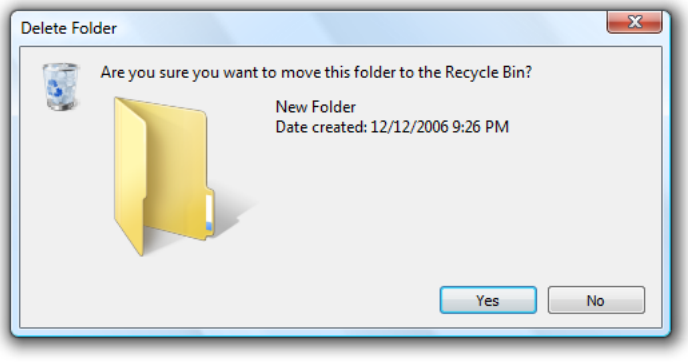
If you do only three things...

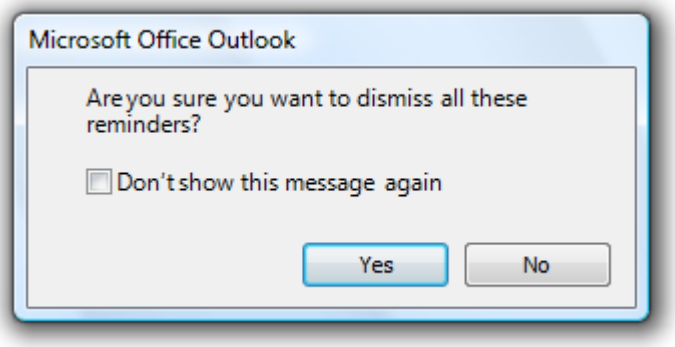
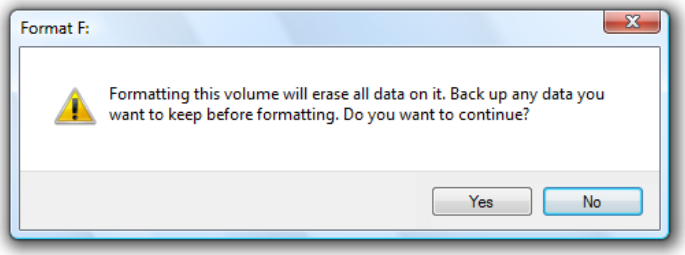
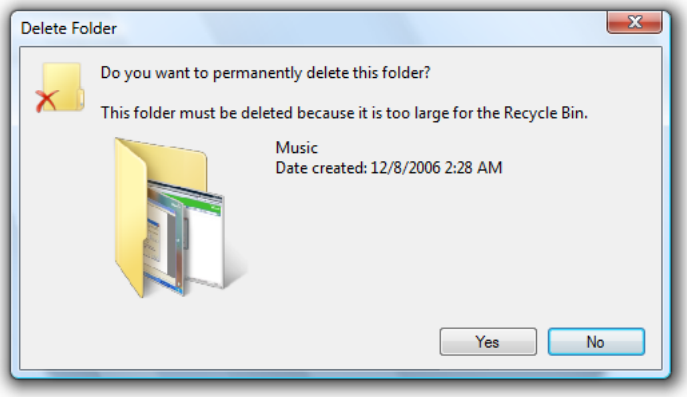
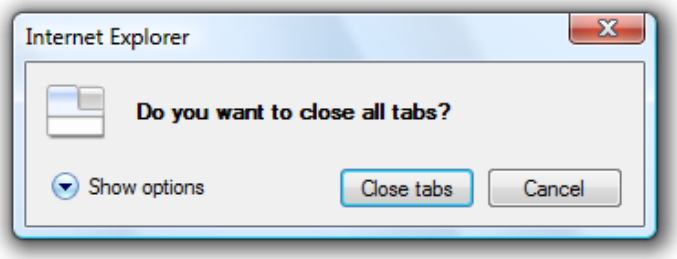
1. Make sure your confirmation is really necessary. There should be a legitimate and clear reason not to proceed, and a chance that sometimes users won't.
2. If the reason for the confirmation isn't immediately obvious, choose commit buttons that encourage users to think about their response. Typically, this is done by phrasing the confirmation as a yes or no question and providing completely self-explanatory or Yes/No answers.
3. Consider all the scenarios and provide the information required to answer the question intelligently.

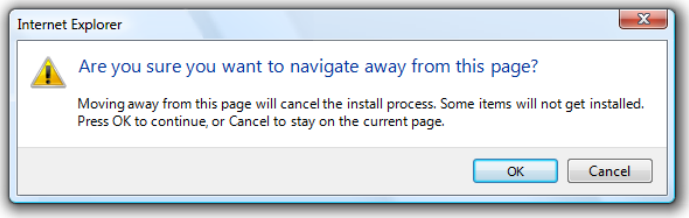
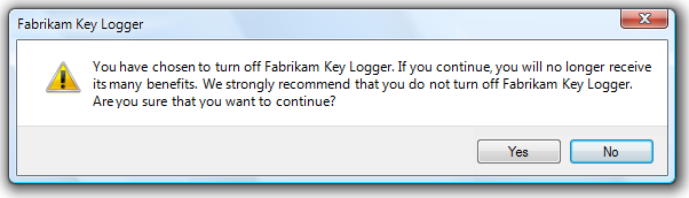
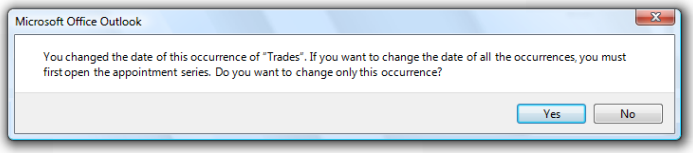
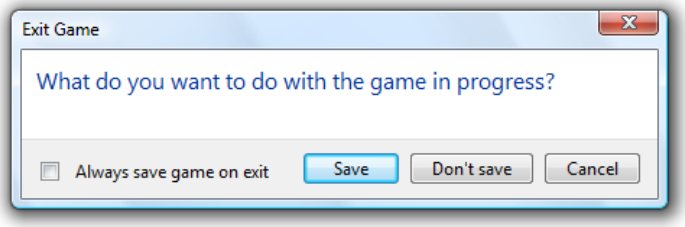
Usage patterns

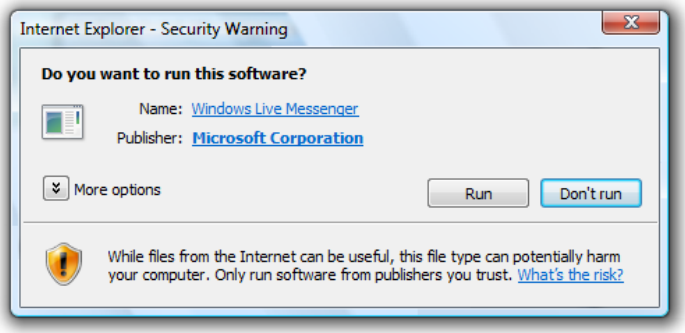
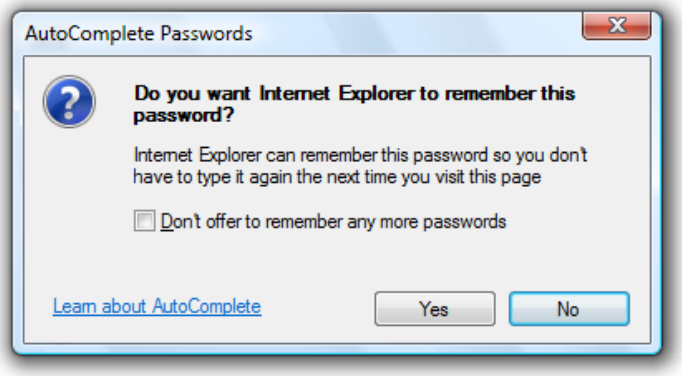
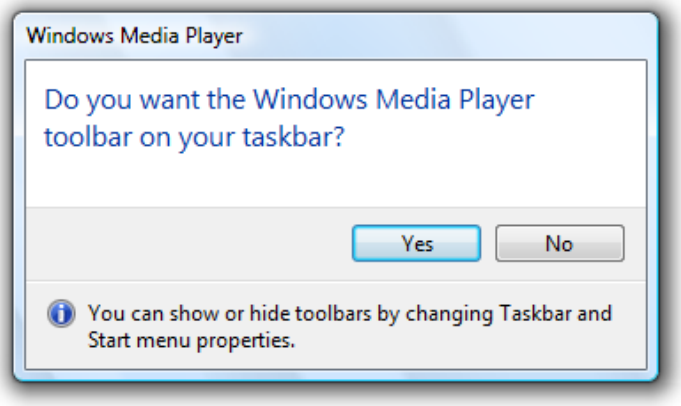
Confirmations have several usage patterns:

Expand table

Usage	Example
Routine confirmations confirm that the user wants to proceed with a routine, low risk action.	These confirmations are usually phrased "are you sure...?" and often have a don't show this message again check box to minimize their annoyance. 

Usage	Example
	 Examples of routine confirmations. **Note:** This pattern is usually unnecessary and should be avoided.
Risky action confirmations confirm that the user wants to proceed with an action that has some risk and can't be easily undone.	Because they have risk, these confirmations usually have a warning icon.   Examples of risky action confirmations.
Unintended consequence confirmations confirm that the user wants to proceed with an action that has unexpected or unintended consequences.	In addition to asking a question, these confirmations point out the unintended consequences. because they have unintended consequences, these confirmations usually have a warning icon. 

Usage	Example
	 examples of unintended consequence confirmations. however, this pattern requires that the consequences are truly unintended. **incorrect:**  The consequences are intended here, so this is a routine confirmation.
Clarifications clarify how the user wants to proceed with an action that has potentially ambiguous or unexpected consequences.	Drag-and-drop operations can result in clarifications if the effect of the operation can be misinterpreted.   Examples of clarifications. **Note:** This pattern should be avoided because it is better to design actions without ambiguous consequences and assume the most likely desired result.

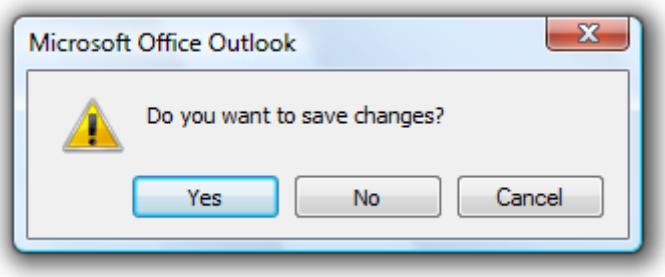
Usage	Example
Security confirmations confirm that the user wants to proceed with an action with security consequences.	  Examples of security confirmations.
Ulterior motive confirmations provide information about an action, but present it as a confirmation.	While these dialog boxes are presented as confirmations, their real goal is user education or advertisement of features.  An example of an ulterior motive confirmation. **Note:** This pattern is not recommended because there is usually a better, more direct alternative. For example, animations are a better way to show the relationship between cause and effect.

Guidelines

General

- **Use "Save changes" confirmations only when there are significant changes.** Don't confirm changes that weren't directly made by the user, such as automatic document reformatting.

Incorrect:



This example is incorrect when used for an empty e-mail or document that wasn't changed by the user.

Icons

- Confirmations don't use title bar icons.
- **The content area icon for a confirmation is based on its design pattern:**

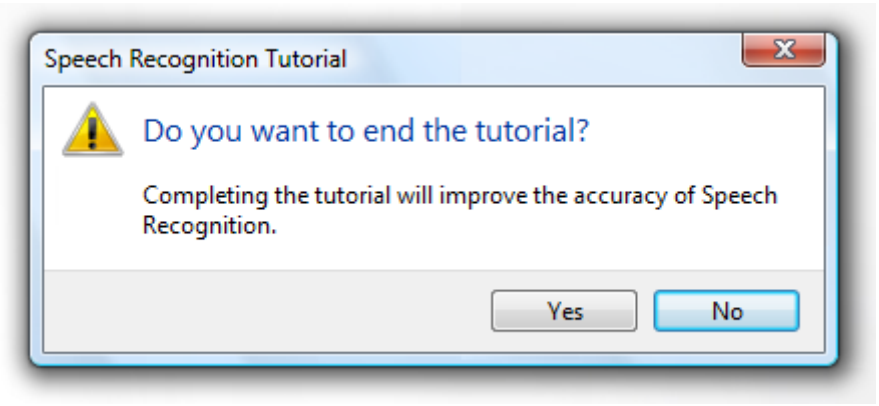
Expand table

Pattern	Icon
Routine confirmations	No icon.
Risky action confirmations	Warning icon.
Unintended consequence confirmations	Use a warning icon if there is risk, the feature icon if available; otherwise, no icon.
Clarifications	If the confirmation involves a document, use the document's thumbnail; otherwise, use the feature icon if available, or no icon.
Security confirmations	Warning icon.
Ulterior motive confirmations	No icon.

- **Don't use warning icons for routine questions.** Doing so is counter to the encouraging tone of Windows and makes using your program feel like a

hazardous activity. Assume users understand the consequences of canceling a task before it is finished.

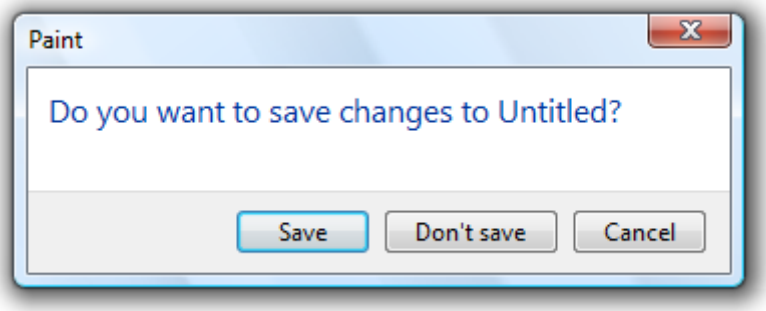
Incorrect:



In this example, a warning icon is used to ask a routine question.

Commit buttons

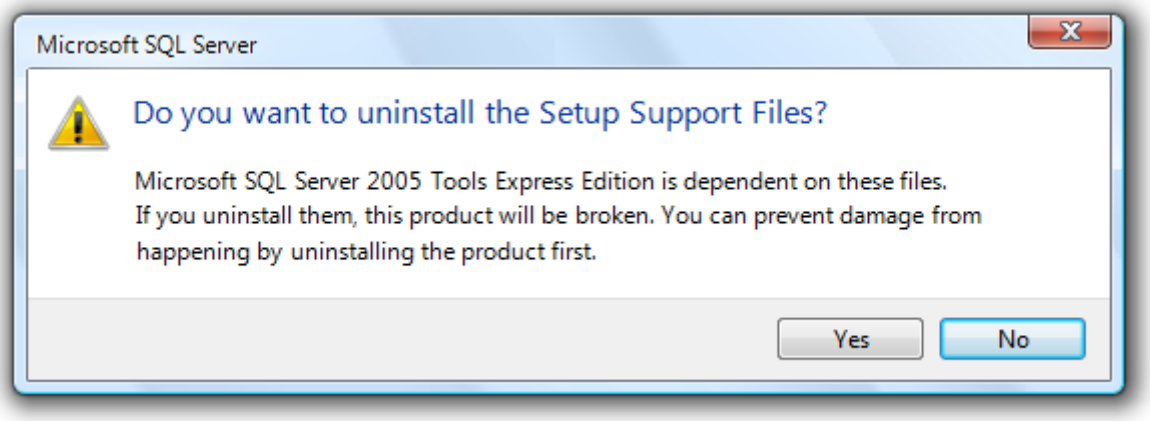
- **Use specific responses to the main instruction if the reason for the confirmation is obvious or can be made self explanatory.**



In this example, the reason for the confirmation is obvious, so Save and Don't save work well.

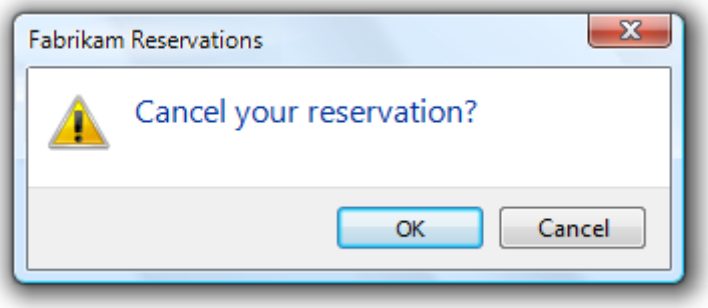
- **Otherwise, use Yes and No buttons for confirmation responses.** Doing so makes users give the confirmation some thought before responding. Never use OK and Cancel for confirmations.

Correct:



In this example, using Yes/No commit buttons forces users to at least read the main instruction.

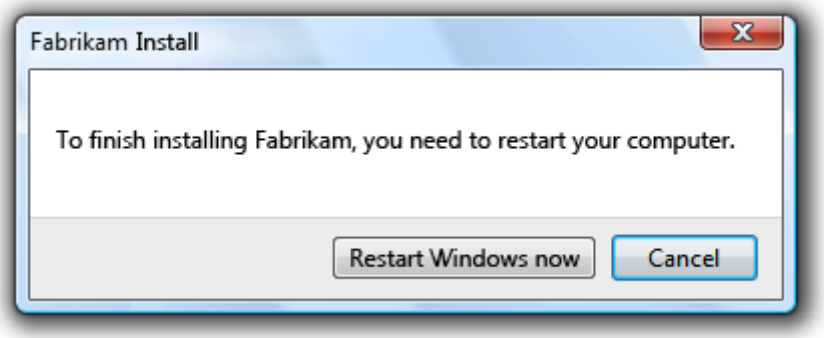
Incorrect:



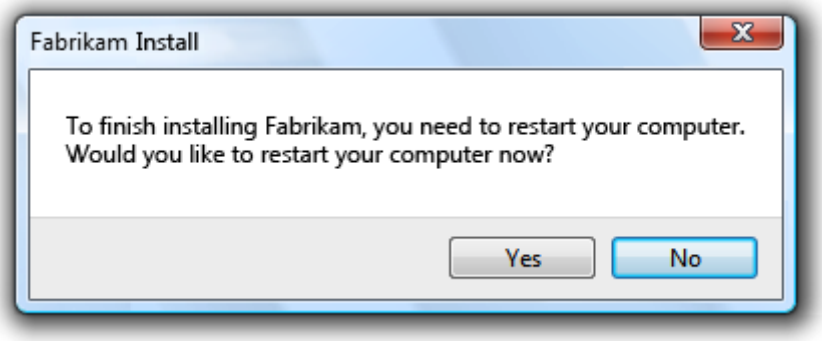
In this example, using OK/Cancel is confusing.

- **To close a program or restart Windows, use specific responses to the main instruction.** To prevent any misunderstanding, don't use Close or Yes/No for this purpose.

Correct:



Incorrect:

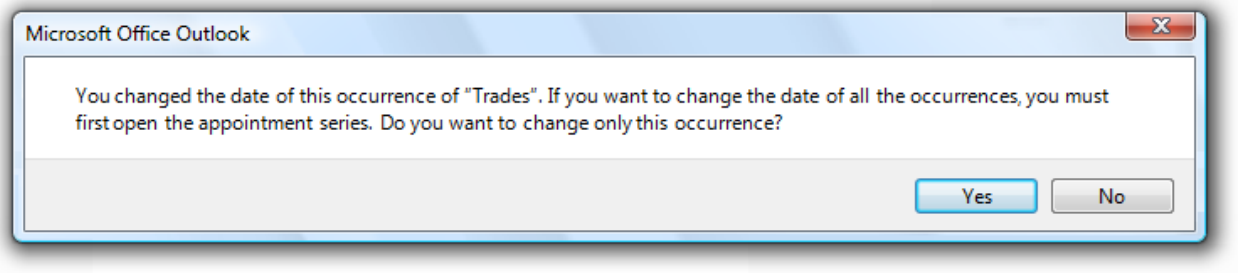


In the incorrect example, Yes is used to restart Windows.

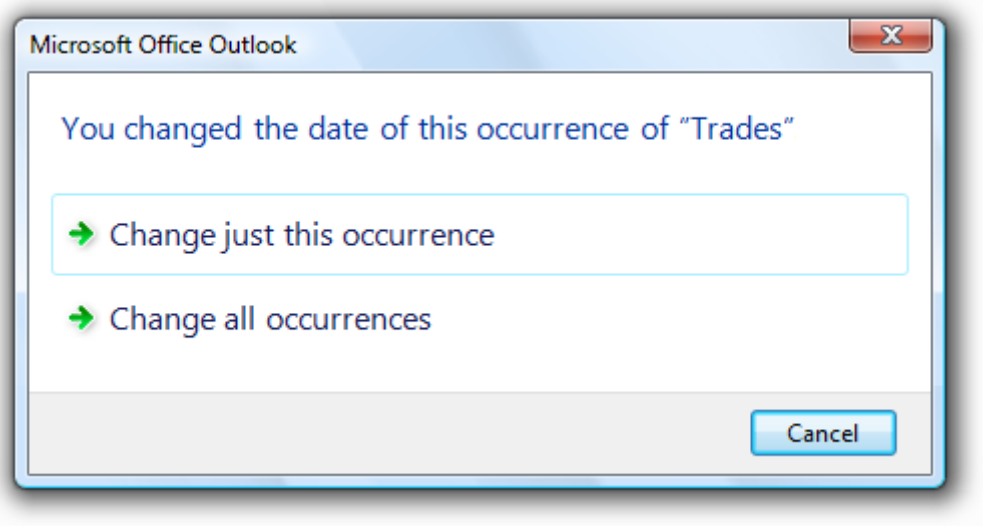
Command links

- **For the clarifications pattern, consider using command links to make the alternatives clear.**

Acceptable:



Better:



In the better example, command links make the alternatives clear.

- **Present the most commonly used command links first.** The resulting order should roughly follow the likelihood of use, but also have a logical flow.

- If a command link requires further explanation, **provide a supplemental explanation.** Supplemental explanations describe why users might want to choose the option or what happens if the option is chosen.

For more guidelines and examples, see Command Links.

Default values

- **The default response for a confirmation is based on its design pattern:**

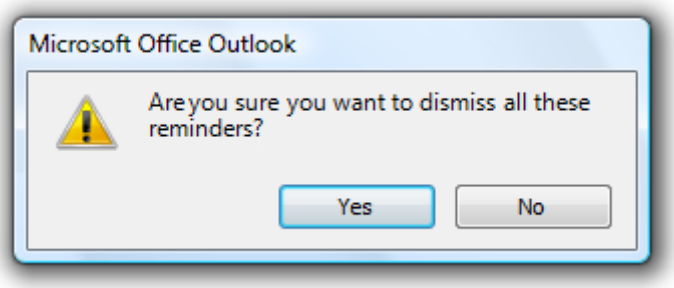
Expand table

Pattern	Default response
Routine confirmations	Proceed.
Risky action confirmations	Don't proceed (or the safe choice).
Unintended consequence confirmations	If consequences are significant, don't proceed; otherwise, proceed.
Clarifications	The most likely response.
Security confirmations	Don't proceed.
Ulterior motive confirmations	Proceed.

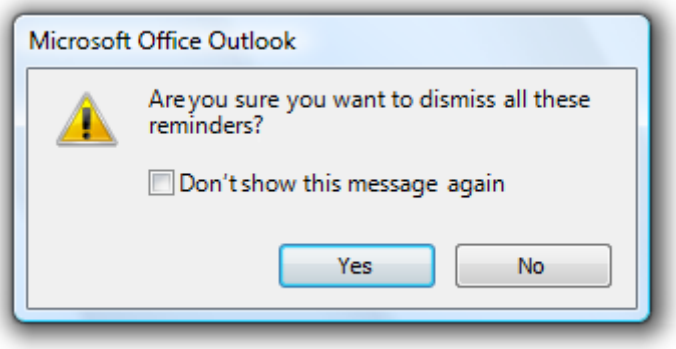
Don't show this message again

- **Use this option only for the routine and ulterior motive confirmation patterns.** For the other patterns, if the information is necessary, it should always be displayed.
- **Don't provide this option to justify displaying an unnecessary confirmation.** Just get rid of the confirmation instead.

Incorrect:



Still incorrect:

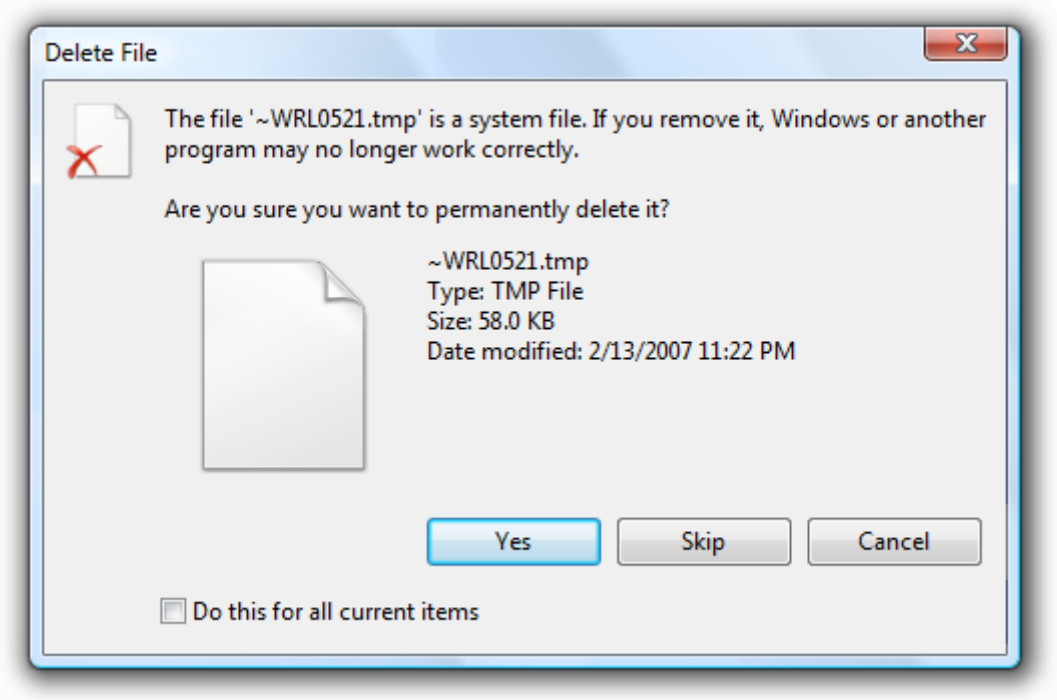


In these examples, adding a Don't show this message again option doesn't fix an unnecessary confirmation.

For more guidelines, see Dialog Boxes.

Bulk operations

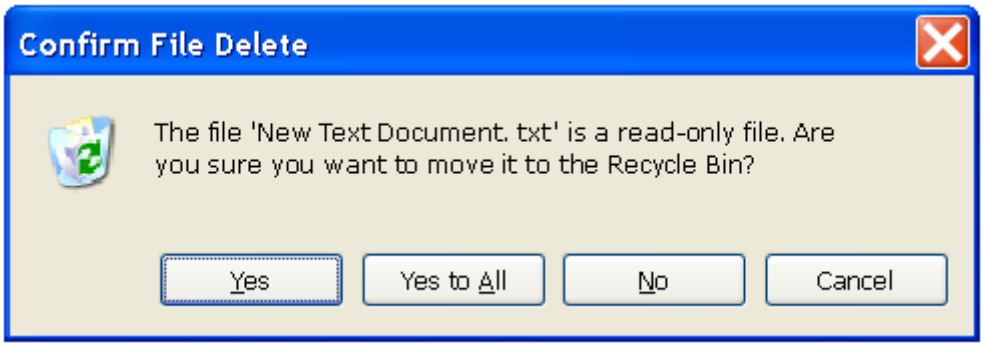
- For confirmations that apply to bulk operations, provide an option to apply the confirmation to the entire operation.



This example has an option for bulk operations.

- Eliminate or postpone confirmations in a bulk operation.

Incorrect:



In this example, Windows Explorer in Windows XP confirms each read-only file during a bulk file move. Better just to copy the read-only files without asking, or postpone handling these files and present the confirmation at the end of the task.

Progressive disclosure

- **If you must include advanced information in a confirmation message, reveal it by using progressive disclosure buttons (for example, "Show details").** Doing so simplifies the confirmation for typical usage. Don't hide needed information because users might not find it.
- **Don't use "Show details" unless there really is more detail.** Don't just restate the existing information in a different format.

For labeling guidelines, see Progressive Disclosure.

User Account Control

- **Don't use the User Account Control (UAC) elevation UI as a substitute for a confirmation.** If an action needs a confirmation, use a separate dialog box. During the elevation UI, users need to focus on whether they started task and if the program is trustworthy.
- **Display the confirmation before the elevation UI.** Doing so eliminates unnecessary elevations.

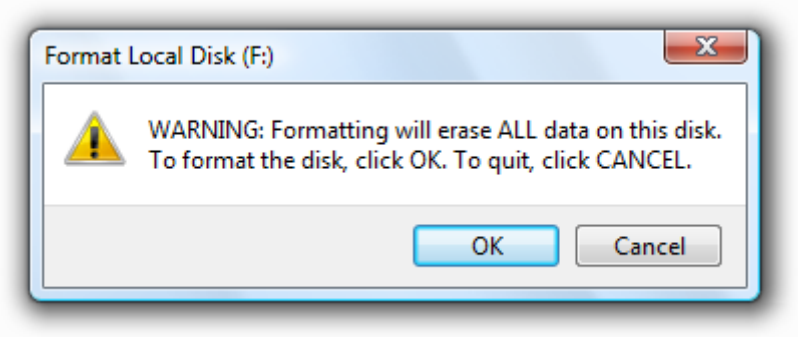
Text

General

- **Remove redundant text.** Look for redundant text in titles, main instructions, supplemental instructions, content areas, command links, and commit buttons. Generally, leave full text in instructions and interactive controls, and remove any redundancy from the other places.

- **Don't use "warning" or "caution" in the text.** If users need to proceed with caution, indicate this using a warning icon instead.

Incorrect:



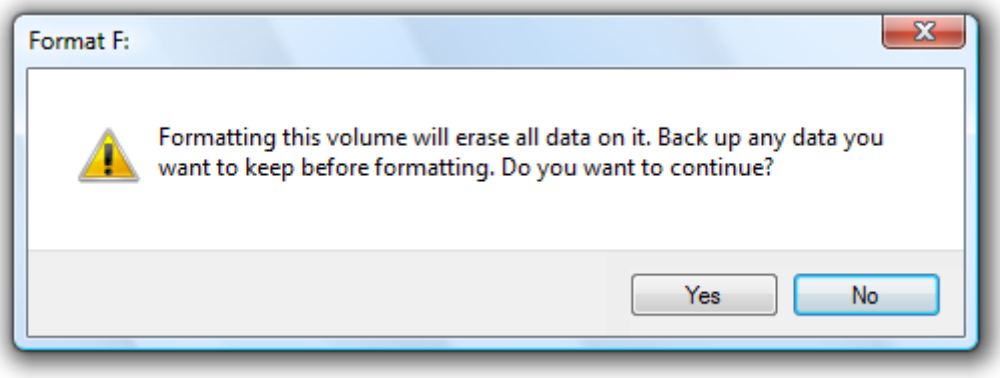
In this example, the term "warning" is unnecessary.

Titles

- **Use the title to identify the command or feature where the confirmation came from. Exceptions:**
  - If a confirmation is displayed by many different commands, consider using the program name instead.
  - If that title would be redundant or confusing with the main instruction, use the program name instead.

However, if the confirmation is from a long-running task and may display well after the task started, always use the command or feature to clearly identify the context.

- **Don't use the title to explain what to do in the dialog** that's the purpose of the main instruction.
- If it adds clarity, start the title with the word Confirm.
- **For risky action confirmations, you may add the name of the object involved for extra emphasis.**



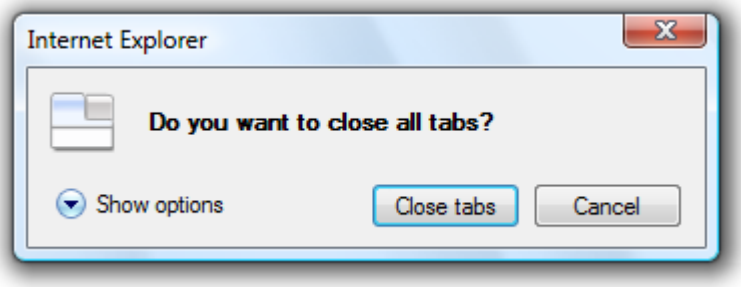
In this example, the drive to be formatted is included in the title.

- Use title-style capitalization, without ending punctuation.

Main instructions

- **The main instruction for a confirmation is based on its design pattern:**

Expand table

Pattern	Main instruction
Unintended consequence confirmations	state the unintended consequence. **exception:** if a question asking if the user wants to proceed clearly implies the unintended consequence, ask the question instead.  In this example, asking the user to proceed sufficiently conveys the consequences of the action.
All others	Ask a single question to determine if the user wants to proceed.

- **Be concise use only a single, complete sentence.** Strip the main instruction down to the essential information. If you must explain anything more, use a supplemental instruction.
- **Be specific if there are objects involved, give their full names.**
- **Use positive phrasing.** Positive phrasing is easier for users to understand.

Correct:

Do you want to enable file and printer sharing?

Incorrect:

Do you want to disable file and printer sharing?

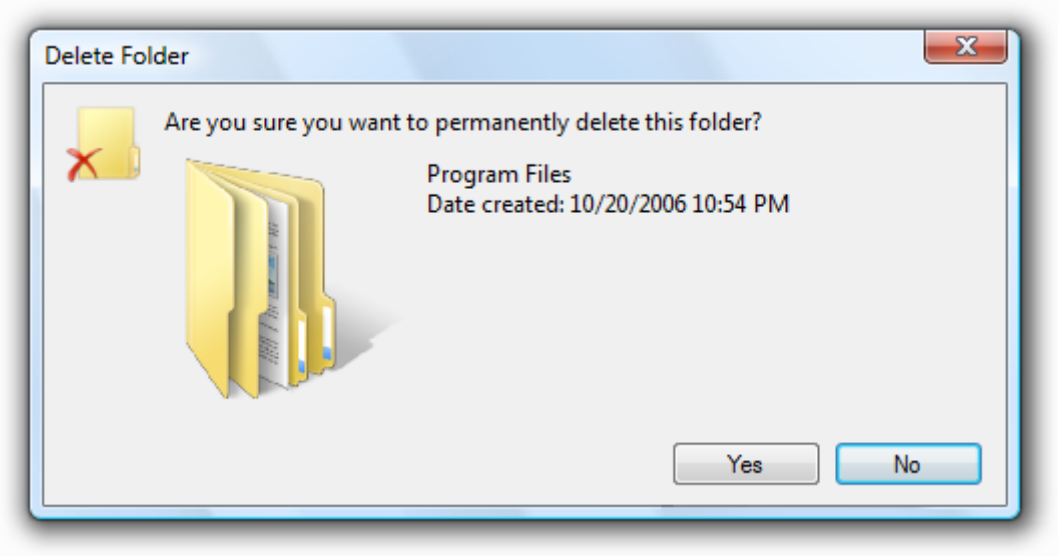
However, phrasing must match the associated command, even if the command is negatively phrased; so, for example, use disable to confirm a Disable command.

- While there are no strict rules for phrasing, **these common confirmation phrases have the indicated connotation:**

Expand table

Phrase	Connotation
Are you sure you want to [perform an action]?	Confirming the direct result of a user request.
Do you want to [perform an action]?	Confirming a side effect of a user request.
Would you like to [select a result]?	Need a clarification.
[Perform an action]?	No connotation.

- For risky action confirmations, use the term permanently to indicate that an action can't be undone.



In this example, "permanently" indicates that the action can't be undone.

- Use sentence-style capitalization.

Supplemental instructions

- **The supplemental instruction for a confirmation is based on its design pattern:**

Expand table

Label	Value
Pattern	Supplemental instruction

Label	Value
Unintended consequence confirmations	Ask a single question to determine if the user wants to proceed.

All others	Explain any non-obvious reasons why the user might not want to proceed. Such reasons include:
	• Potential loss of one or more of the following:
	  ◦ A valuable asset, such as data loss or financial loss.
	  ◦ System access or integrity.
	  ◦ Privacy or control over confidential information.
	• Actions that are irreversible.

- **Don't repeat the main instruction with slightly different wording.** Instead, omit the supplemental instruction if there is not more to add.
- **For unintended consequence confirmations, consider using the term anyway to concisely indicate that there is a reason not to continue** in case the user overlooked the main instruction. See Design Concepts for more information.
- Use complete sentences, sentence-style capitalization, and ending punctuation.

Documentation

When referring to confirmations:

- Refer to a confirmation by its title if the title is specific to the confirmation (that is, not the program name); otherwise, refer to it by its main instruction.
- If necessary, you may refer to a confirmation dialog box as a message.
- Use the exact text, including its capitalization.
- When possible, format the text using bold. Otherwise, put the text in quotation marks only if required to prevent confusion.

Example: In the **Copy File** message, click the newer file.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

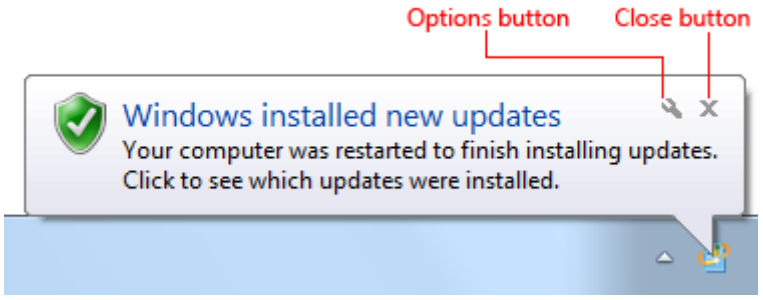
Notifications (Design basics)

Article • 03/18/2022

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

A notification informs users of events that are unrelated to the current user activity, by briefly displaying a balloon from an icon in the notification area. The notification could result from a user action or significant system event, or could offer potentially useful information from Microsoft Windows or an application.

The information in a notification is **useful and relevant, but never critical.** Consequently, notifications don't require immediate user action and users can freely ignore them.



A typical notification.

In Windows Vista and later, notifications are displayed for a fixed duration of 9 seconds. Notifications aren't displayed immediately when users are inactive or screen savers are running. Windows automatically queues notifications during these times, and displays the queued notifications when the user resumes regular activity. Consequently, you don't have to do anything to handle these special circumstances.

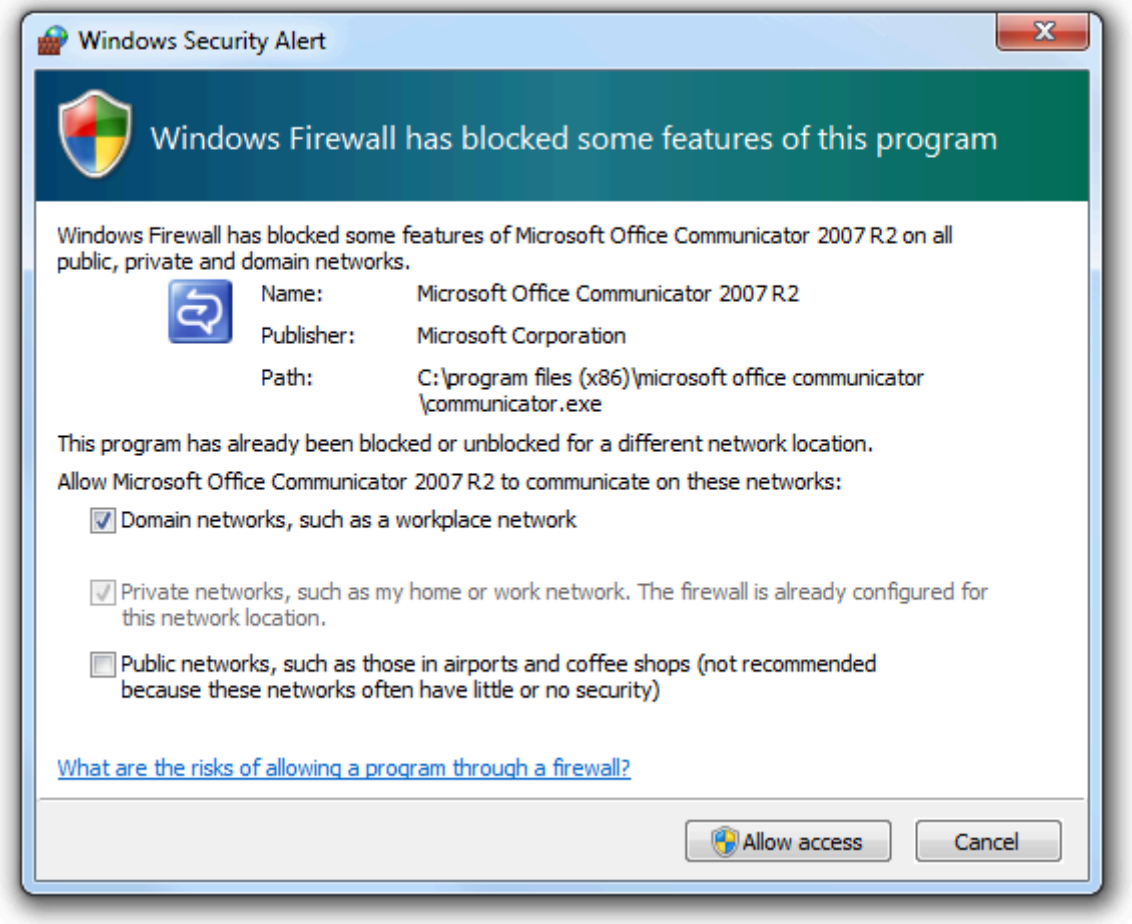
Developers: You can determine when the user is active using the SHQueryUserNotificationState API.

Note: Guidelines related to notification area, taskbar, and balloons are presented in separate articles.

Is this the right user interface?

To decide, consider these questions:

- **Is the information the immediate, direct result of users' interaction with your application?** If so, display this synchronous information directly within your application instead using a dialog box, message box, balloon, or in place UI. Notifications are for asynchronous information only.



In this example, the Windows Firewall exceptions dialog box is displayed as a direct result of user interaction. A notification wouldn't be appropriate here.

- **Is the information relevant only when users are actively using your application?** If so, display the information in your application's status bar or other status area.

In this example, Outlook displays its connection and synchronization state on its status bar.

- **Is the information rapidly changing, continuous, real-time information?** Examples include processing progress, stock quotes, and sports scores. If so, don't use notifications because they aren't suitable for rapidly changing information.
- **Is the information useful and relevant? Are users likely to change their behavior or avoid inconvenience as the result of receiving the information?** If not, either don't display the information or put it in a status window or log file.

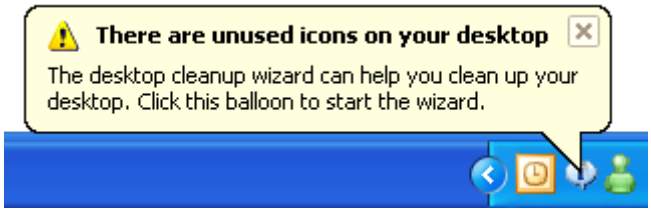
- **Is the information critical? Is immediate action required?** If so, display the information using an interface that demands attention and cannot be easily ignored, such as a modal dialog box or message box. If the program isn't active, you can draw attention to the critical information by flashing the program's taskbar button three times and leaving it highlighted until the program is active.
- **Are the primary target users IT professionals?** If so, use an alternative feedback mechanism such as log file entries or e-mail messages. IT professionals strongly prefer log files for non-critical information. Furthermore, servers are often managed remotely and typically run without any users logged on, making notifications ineffective.

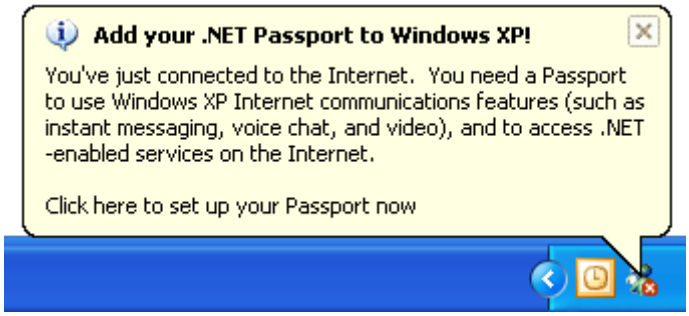
Design concepts

Effective notifications that promote a good user experience are:

- **Asynchronous.** The event is not an immediate, direct result of users' current interaction with Microsoft Windows or your application.
- **Useful.** There is a reasonable chance that users will perform a task or change their behavior as the result of the notification.
- **Relevant.** The notification displays helpful information that users care about and don't already know.
- **Not critical.** Notifications aren't modal and don't require user interaction, so users can freely ignore them.
- **Actionable.** For those notifications that suggest performing an action, that action is initiated by clicking on the notification. However, the action can always be postponed.
- **Appropriately presented.** The notification's presentation (duration, frequency, text, icon, and interactivity) matches its circumstances.
- **Not annoying!** There is a fine line between gently informing users of an event and pestering them.

Unfortunately, there are too many annoying, inappropriate, useless, irrelevant notifications out there. Consider these notifications from the Windows XP Hall of Shame:





In these examples, Windows XP is ostensibly attempting to assist users with their initial configuration. However, these notifications pop up far too often and well after they are useful, so they are little more than unsolicited feature advertisements.

User flow must be maintained

Ideally, users immersed in their work won't see your notifications at all. Rather, they'll see your notifications only when their flow is already broken.

In Flow: The Psychology of Optimal Experience, Mihaly Csikszentmihalyi says that users enter a flow state when they are fully absorbed in activity during which they lose their sense of time and have feelings of great satisfaction.

Effective notifications help users maintain their flow by presenting useful, relevant information that can easily be ignored. The notifications are presented in a low-key, peripheral way, and they don't require interaction.

Don't assume that if notifications are modeless a they can't be an annoying interruption. Notifications don't demand users' attention, but they certainly request it. You can break users flow by:

- Displaying notifications that users don't care about.
- Displaying a notification too often.
- Using several notifications when a single notification is sufficient.
- Using sound when displaying a notification.

In Windows 7, users have ultimate control over notifications. **If users find that a program's notifications are too annoying, they can choose to suppress all notifications from that program.** Make sure users don't do this to your program by presenting useful, relevant information and following these guidelines.

Notifications must be ignorable

Notifications don't require immediate user action and users can freely ignore them.

Developers and designers often want to present their notifications in a way that users can't ignore. This goal completely undermines the primary benefit of notifications because it would break users' flow. If users are distracted by your notifications or feel obligated to read them, your notification design has failed.

If you are concerned that users are ignoring your notifications, consider the following:

- If you are using notifications correctly and they don't require immediate user action, then having users choose to ignore them is by design. Don't change this.
- If the event requires immediate user action, use an alternative user interface (UI) that users cannot ignore. See Is this the right user interface? for the alternatives.

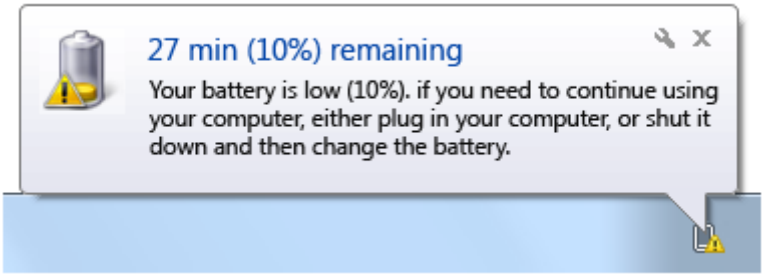
Use progressive escalation where applicable

If a notification is used for an event that users can safely ignore at first, but that must addressed eventually, an alternative UI should be used when the situation becomes critical. This technique is known as progressive escalation.

For example, the Windows power management system initially indicates a low battery by simply changing its notification area icon.

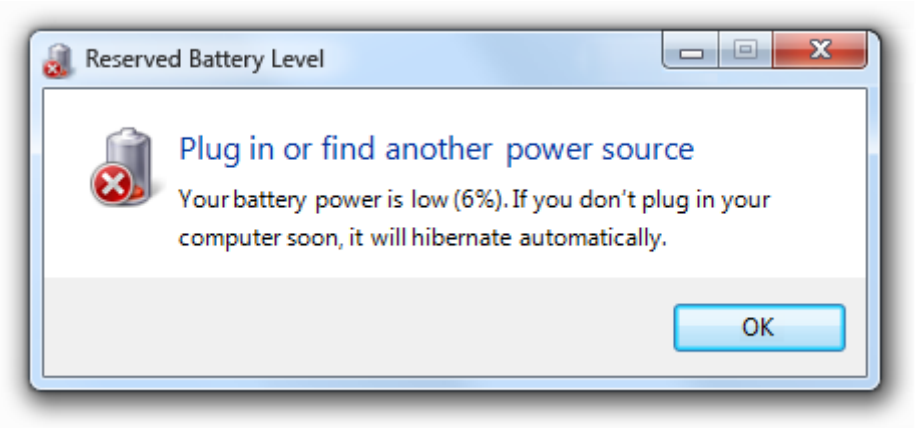
In these examples, Windows power management uses the notification area icon to notify users of progressively lower battery power.

As the battery power becomes lower, Windows warns users of weak battery power using a notification.



In this example, Windows power management uses a notification to tell users that their battery power is weak.

This notification appears while users still have several options. Users can plug in, change their power options, wrap up their work and shut down the computer, or ignore the notification and continue working. As the battery power continues to drain, the notification's text and icon reflect the additional urgency. However, once the battery power becomes so low that users must act immediately, Windows power management notifies users using a modal message box.



In this example, Windows power management uses a modal message box to notify users of critically low battery power.

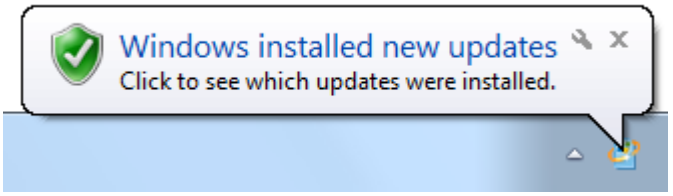
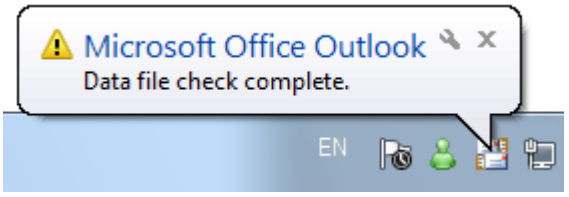
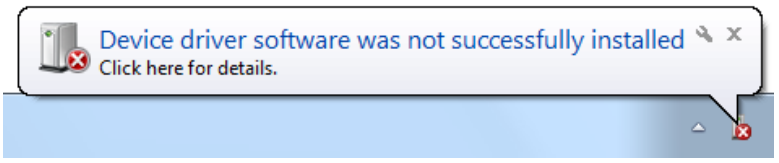
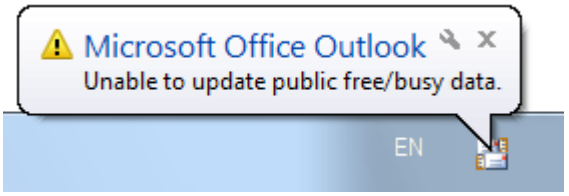
If you do only three things...

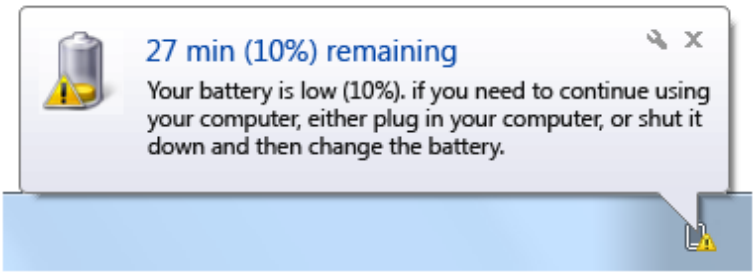
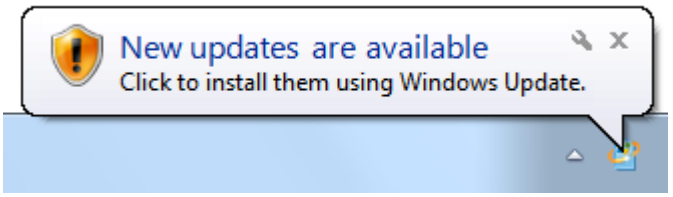
1. Use notifications only if you really need to. When you display a notification, you are potentially interrupting users or even annoying them. Make sure that interruption is justified.
2. Use notifications for non-critical events or situations that don't require immediate user action. For critical events or situations that require immediate user action, use an alternative UI (such as a modal dialog box).
3. If you use notifications, make it a good user experience. Don't try to force users to see your notifications. If users are so immersed in their work that they don't see your notifications, your design is good.

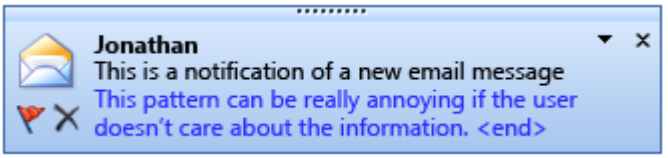
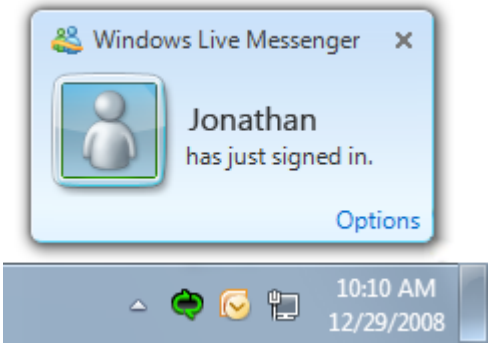
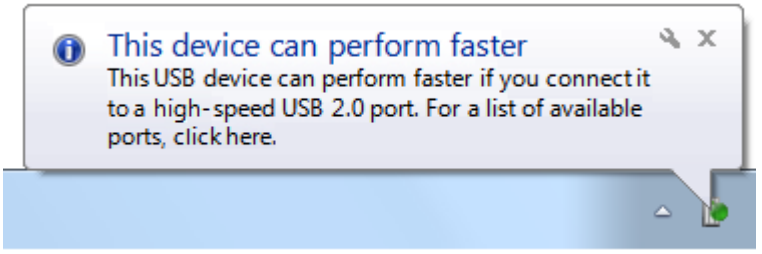
Usage patterns

Notifications have several usage patterns:

Expand table

Label	Value
Action success Notifies users when an asynchronous, user initiated action completes successfully.	**Correct:**  In this example, Windows Update notifies users when their computer has been updated successfully. **Incorrect:**  In this example, Microsoft Outlook notifies users when a data file check is complete. What are users supposed to do now? And why warn users about successful completion? **Show when:** Upon completion of an asynchronous task. Notify users of successful actions only if they are likely to be waiting for completion, or after recent failures. **Show how:** Use the real-time option so that these notifications aren't queued when users are running a full-screen application or aren't actively using their computer. **Show how often:** Once. **Annoyance factor:** Low if success isn't expected due to recent failures, success is after a critical or highly unusual failure so user needs additional feedback, or user is waiting for completion; high if not. **Alternatives:** Give feedback "on demand" by displaying an icon (or changing an existing icon) in the notification area while the operation is being performed; remove the icon (or restore the previous icon) when the operation is complete.
Action failure Notifies users when an asynchronous, user initiated action fails.	**Correct:**  In this example, Windows activation notifies users of failure. **Incorrect:**  In this example, Microsoft Outlook used to notify users of a failure that they are unlikely to care about.

Label	Value
	Show when: Upon failure of an asynchronous task. **Show how often:** Once. **Annoyance factor:** Low if useful and relevant; high if the problem will immediately resolve itself or users otherwise don't care. **Alternatives:** Use a modal dialog box if users must address the failure immediately.
Non-critical system event Notifies users of significant system events or status that can be safely ignored, at least temporarily.	 In this example, Windows warns users of low battery power, but there is still plenty of time before they have take action. **Show when:** When an event occurs and the user is active, or a condition continues to exist. If resulting from a problem, remove currently displayed notifications immediately once the problem is resolved. As with action notifications, notify users of successful system events only if users are likely to be waiting for the event, or after recent failures. **Show how often:** Once when the event first occurs. If this results from a problem that users need to solve, redisplay once a day. **Annoyance factor:** Low, as long as the notification isn't displayed too often. **Alternatives:** If users must eventually resolve a problem, use progressive escalation by ultimately displaying a modal dialog box when resolution becomes mandatory.
Optional user task Notifies users of asynchronous tasks they should perform. Whether optional or required, the task can be safely postponed.	 In this example, Windows Update is notifying users of a new security update. **Show when:** When the need to perform a task is determined and the user is active. **Show how often:** Once a day for a maximum of three times. **Annoyance factor:** Low, as long as users consider the task important and the notification isn't displayed too often. **Alternatives:** If users must eventually perform the task, use progressive escalation by ultimately displaying a modal dialog box when the task becomes mandatory.

Label	Value
FYI Notifies users of potentially useful, relevant information. You can notify users of information of marginal relevance if it is optional and users opt in.	**Correct:**  In this example, users are notified when a new e-mail message is received. **Correct:**  In this example, users are notified when contacts come online and they chose to receive this optional information. **Incorrect:**  In this example, the information is useful only if the user already has high-speed USB ports installed. Otherwise, the user isn't likely to do anything different as the result of it. **Show when:** When the triggering event occurs. **Show how:** Use the real-time option so that these notifications aren't queued when users are running a full-screen application or aren't actively using their computer. **Show how often:** Once. **Annoyance factor:** Medium to high, depending upon users' perception of usefulness and relevance. Not recommended if there is a low probability of user interest. **Alternatives:** Don't notify users.
Feature advertisement Notifies users of newly installed, unused system or application features.	**Don't use notifications for feature advertisements!** Instead, use another way to make the feature discoverable, such as: • Design the feature to be easier to discover in contexts where it is needed. • Don't do anything special and let users discover the feature on their own.

Label	Value
	Incorrect: Don't use notifications for feature advertisements.

Guidelines

General

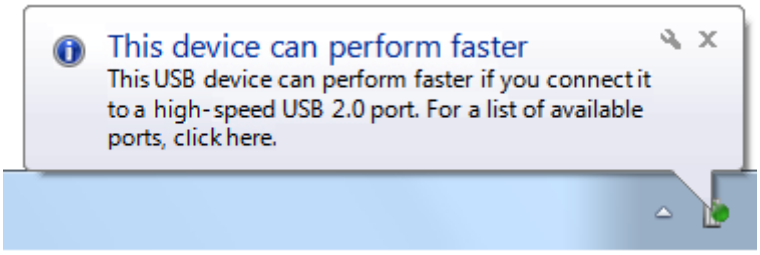
- **Select the notification pattern based on its usage.** For a description of each usage pattern, see the previous table.
- **Don't use any notifications during the initial Windows experience.** To improve its first experience, Windows 7 suppresses all notifications displayed during the first few hours of usage. Design your program assuming users won't see any such notifications.

What to notify

- **Don't notify of successful operations, except in the following circumstances:**
  - **Security.** Users consider security operations to be of the highest importance, so notify users of successful security operations.
  - **Recent failure.** Users don't take successful operations for granted if they were failing immediately before, so notify users of success when the operation was recently failing.
  - **Prevent inconvenience.** Report successful operations when doing so might avoid inconveniencing users. Consequently, notify users when a successful operation is performed in an unexpected way, such as when an operation is lengthy or completes earlier or later than expected.
- **In other circumstances, either give no feedback for success or give feedback "on demand."** Assume that users take successful operations for granted. You can give feedback on demand by displaying an icon (or changing an existing icon) in the notification area while the operation is being performed, and removing the icon (or restoring the previous icon) when the operation is complete.

- For the FYI pattern, **don't give a notification if users can continue to work normally or are unlikely to do anything different as the result of the notification.**

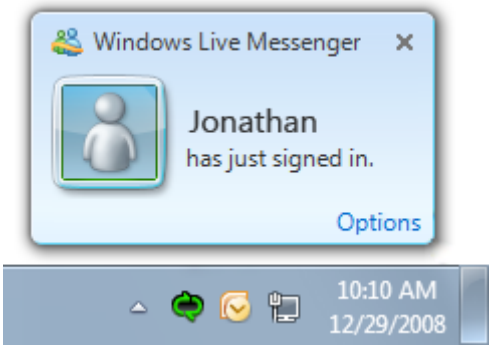
  Incorrect:

  In this example, the information is useful only if the user already has the ports installed. Otherwise, the user isn't likely to do anything different as the result of it.

  - Exception: **You can notify users of information of questionable relevance if it is optional and users opt in.**

    Correct:

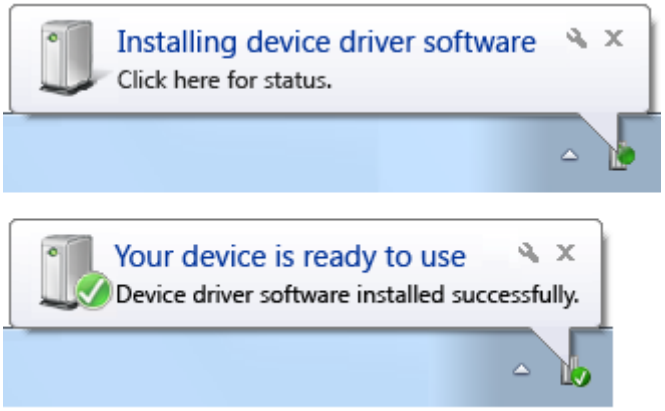
    In this example, users are notified when contacts come online and they chose to receive this optional information.

- For the non-critical system event and FYI patterns, **use complete notifications for a single event.** Don't present several partial ones.

  Incorrect:

These examples show just four of the eight notifications that were displayed by Windows XP when a user attaches a specific USB keyboard, each presenting incrementally more information.

Correct:



In this example, attaching a USB keyboard results in two complete notifications.

When to notify

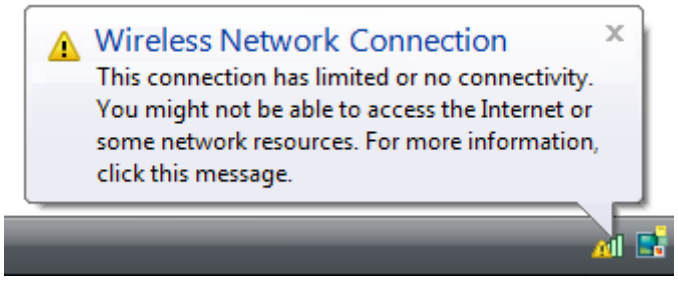
- **Display a notification based on its design pattern:**

Expand table

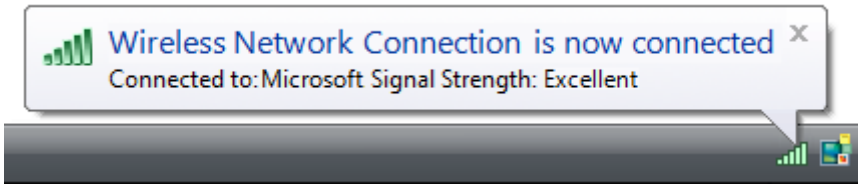
Pattern	When to notify
Action success	Upon completion of an asynchronous task. Notify users of successful actions only if they are likely to be waiting for completion, or after recent failures.
Action failure	Upon failure of an asynchronous task.
Non-critical system event	When an event occurs and the user is active, or the condition continues to exist. If this results from a problem, remove the currently displayed notification immediately once the problem is resolved.
Optional user task	When the need to perform a task is determined and the user is active.
FYI	When the triggering event occurs.

- For the action failure pattern, **if the problem might correct itself within seconds, delay the failure notification for an appropriate amount of time.** If the problem corrects itself, report nothing. Notify only after enough time has passed that the failure is noticeable. If you report too early, most likely users won't notice the problem reported, but they will notice the unnecessary notification.

Incorrect:



When immediately followed by:



In this example, in Windows Vista the notification of no wireless connectivity is premature because it is often immediately followed by a notification of good connectivity.

- For the action success and FYI patterns, **use the real-time option so that stale notifications aren't queued** when users are running a full-screen application or aren't actively using their computer.

- For the non-critical system event pattern, **don't create the potential for notification storms by staggering events tied to well-known events such as user logon.** Instead, tie the event to some time period after the event. For example, you could remind users to register your product five minutes after user logon.

How long to notify

In Windows Vista and later, notifications are displayed for a fixed duration of 9 seconds.

How often to notify

- **The number of times to display a notification is based on its design pattern:**

Expand table

Pattern	How often to notify
Action success	Once.
Action failure	Once.
Non-critical system event	Once when the event first occurs. If this results from a problem that users need to solve, redisplay once a day.
Optional user task	Once a day for a maximum of three times.
FYI	Once.

- **For optional user tasks, don't try to pester users into submission by constantly displaying notifications.** If the task is required, display a modal dialog box immediately instead of using notifications.

Notification escalation

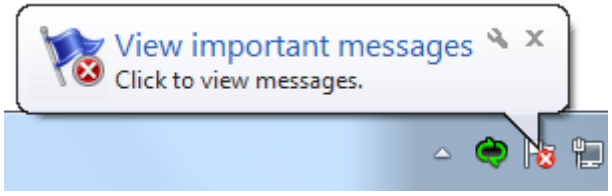
- **Don't assume that users will see your notifications.** Users won't see them when:
  - They are immersed in their work.
  - They aren't paying attention.
  - They are away from their computer.
  - They are running a full-screen application.
  - Their administrator has turned off all notifications for their computer.
- **If users must eventually take some kind of action, use progressive escalation** to display an alternative UI that users cannot ignore.

Interaction

- **Make notifications clickable when:**
  - **Users should perform an action.** Clicking the notification should display a window in which users can perform the action. This approach is preferred for the action failure and optional user task design patterns.
  - **Users may want to see more information.** Clicking the notification should display a window in which users can view additional information.
- **Always display a window when users click to perform an action.** Don't have clicking perform an action directly.
- **Clicking to show more information should always show more information.** Don't just rephrase the information already in the notification.

Icons

- **For the action failure pattern, use the standard error icon.**
- **For the non-critical system event patterns, use the standard warning icon.**
- **For other patterns, use icons showing objects that relate to or suggest the subject**, such as a shield for security or a battery for power.
- **Use icons based on your application or company branding if your target users will recognize them and there is no better alternative.**
- For progressive escalation, **consider using icons with a progressively more emphatic appearance as the situation becomes more urgent.**
- **Don't use the standard information icon.** That notifications are information goes without saying.
- **Consider using large icons (32x32 pixels) when:**
  - Users will quickly comprehend the icon rather than the text.
  - The large icons convey their meaning more clearly and effectively than the standard 16x16 pixel icons.
  - The icon uses the Aero-style.



In this example, users can quickly comprehend the nature of the notification with a glance at the large icon.

Notification queuing

Note: Notifications are queued whenever they cannot be displayed immediately, such as when another notification is being displayed, the user is running a full-screen application, or the user isn't actively using the computer. Real-time notifications remain in the queue for only 60 seconds.

- **For the action success and FYI patterns, use the real-time option** so that the notification isn't queued for long. These notifications have value only when they can be displayed immediately.
- **Remove queued notifications when they are no longer relevant.**
- **Developers:** You can do this by setting the NIF_INFO flag in uFlags and set szInfo to an empty string. There is no harm in doing this if the notification is no longer in the queue.

System integration

- If your application doesn't always have an icon in the notification area when it's running, **display an icon temporarily during the asynchronous task or event that caused the notification.**

Text

Title text

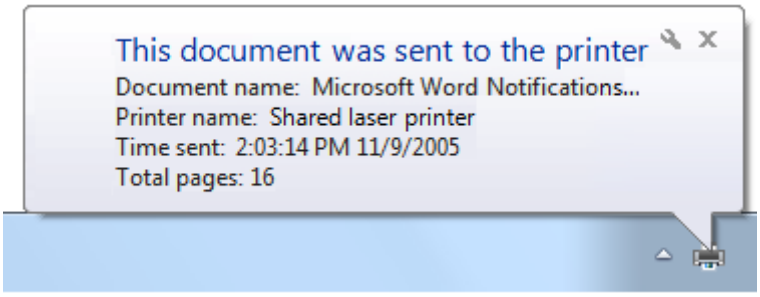
- **Use title text that briefly summarizes the most important information you need to communicate to users in clear, plain, concise, specific language.** Users should be able to understand the purpose of the notification information quickly and with minimal effort.
- **Use text fragments or complete sentences without ending punctuation.**
- **Use sentence-style capitalization.**
- **Use no more than 48 characters (in English) to accommodate localization.** The title has a maximum length of 63 characters, but you must allow for 30 percent expansion when the English-language text is translated.

Body text

- **Use body text that gives a description (without repeating the information in the title) and, optionally, that gives specific details about the notification, and also lets users know what action is available.**

- **Use complete sentences with ending punctuation.**

- **Use sentence-style capitalization.**
- **Use no more than 200 characters (in English) to accommodate localization.** The body text has a maximum length of 255 characters, but you must allow for 30 percent expansion when the English-language text is translated.
- **Include essential information in the body text, such as specific object names.** (Examples: user names, file names, or URLs.) Users shouldn't have to open another window to find such information.
- **Put double quotation marks around object names.**
  - **Exception:** Don't use quotation marks when:
    - The object name always uses title-style capitalization, such as with user names.
    - The object name is offset with a colon (example: Printer name: My printer).
    - The object name can be easily determined from the context.
- **If you must truncate object names to a fixed maximum size to accommodate localization, use an ellipsis to indicate truncation.**



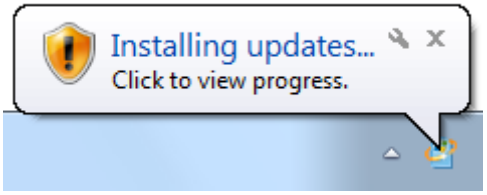
In this example, an object name is truncated using an ellipsis.

- **Use the following phrasing if the notification is actionable:**
  - If users can click the notification to perform an action:

    < brief description of essential information>

    <optional details>

    Click to <do something>.



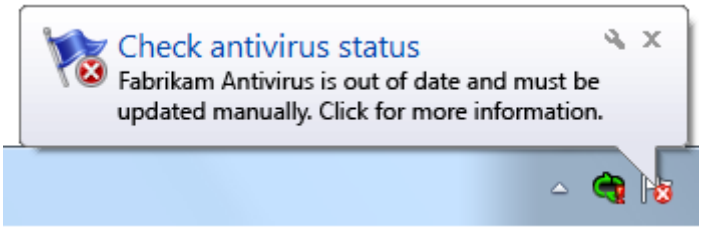
In this example, users can click to perform an action.

- If users can click the notification to see more information:

  < brief description of essential information>

  <optional details>

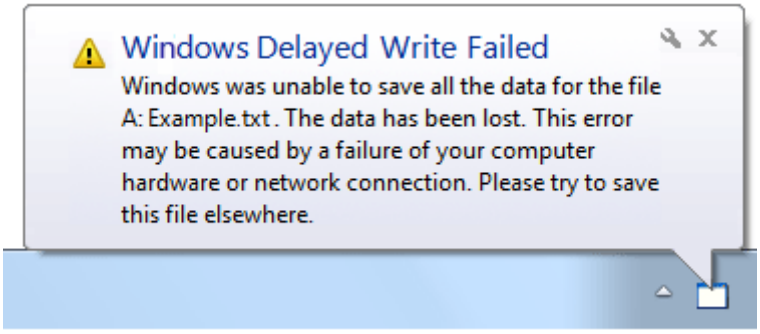
  Click for more information.

  In this example, users can click for more information.

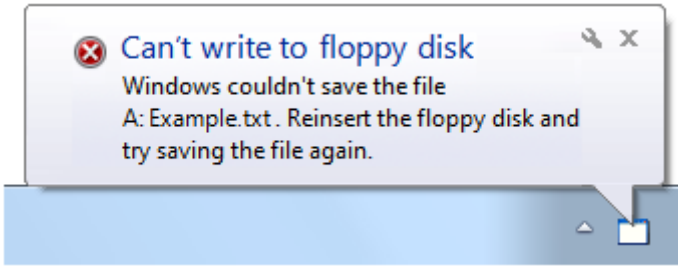
- **Don't say that the user "must" perform an action in a notification.** Notifications are for non-critical information that users can freely ignore. If users really must perform an action, don't use notifications.
- **If users should perform an action, make the importance clear.**
- For the action failure and non-critical system event patterns, **describe problems in plain language.**

  Incorrect:

  In this example, the problem is described using overly technical, yet unspecific language.

  Correct:

In this example, the problem is described in plain language.

- **Describe the event in a way that is relevant to the target users.** A notification is relevant if there's a reasonable chance that users will perform a task or change their behavior as the result of the notification. You can often accomplish this by describing notifications in terms of user goals instead of technological issues.

Documentation

When referring to notifications:

- Use the exact title text, including its capitalization.
- Refer to the component as a notification, not as a balloon or an alert.
- To describe user interaction, use click.
- When possible, format the title text using bold text. Otherwise, put the title in quotation marks only if required to prevent confusion.

Example: When the **Critical updates are ready to install** notification appears, click the notification to start the process.

When referring to the notification area:

- Refer to the notification area as the notification area, not the system tray.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Interaction

Article • 06/04/2021

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

Interaction is the variety of ways users interact with your app, including touch, keyboard, mouse, and so on. Use these guidelines to create intuitive and distinctive experiences that are optimized for touch but work consistently across input devices.

Design for a touch-first experience

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 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 unique to the technology and application. In the move to a touch-first UI, it's important to understand the core differences between touch (including touchpad), pen/stylus, mouse, and keyboard input. Don't take familiar input device properties and behaviors for granted, as touch in Windows 8 does more than simply emulate that functionality.

As you'll soon discover, touch input requires a different approach to UI design.

Compare touch interaction requirements

This table shows some of the differences between input devices that you should consider when you design touch-optimized Windows Store apps.

Expand table

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.

Factor	Touch interactions	Mouse, keyboard, pen/stylus interactions	Touchpad
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 tapping, dragging, sliding, pinching, and rotating.	No support for direct manipulation as mouse, pen/stylus, and keyboard are indirect input devices.	Same as mouse.

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.

We provide some general user interaction guidelines here and cover device-specific guidelines in these topics.

- Touch
- Mouse
- Pen
- Keyboard
- Accessibility

Visuals for 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 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.

Optimize for accuracy

Touch input involves the entire contact area of the finger. This contact geometry is used to determine the most likely target object. Size your controls to ensure a comfortable UI with objects and controls that are easy and safe to target.

Size, space, and position your controls to help eliminate finger and hand occlusion, where the UI is obscured by the user interaction itself.

Position menus, pop-ups, and tooltips above the contact area whenever possible.

Constrain for confidence

Avoid, or minimize, sloppy interactions by using UI constraints.

- Snap points can make it easier to stop at desired locations. 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, swiping, or rotating. With them, you can handle imprecise user input and ensure a specific subset of content or key information is displayed.
- Directional "rails" that emphasize the axis of motion (vertical or horizontal).

Avoid timed interactions

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.

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 and users must reach arbitrary and invisible thresholds to achieve a result. Instant visual feedback during interactions make users feel more engaged, confident, and in control.

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.

Note: An exception to this is where you use specific timed interactions to assist in learning and exploration (for example, press and hold). Using appropriate descriptions and visual cues has a great effect on the use of advanced interactions.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Touch

Article • 02/09/2021

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

All Microsoft Windows applications should have a great touch experience. And creating such an experience is easier than you think.

Touch refers to using one or more fingers to provide input through a device display and interact with Windows and apps. A touch-optimized app has a UI and interaction model designed to accommodate the larger, less precise contact areas of touch, the various form factors of touch devices, and the many postures and grips users might adopt when using a touch device.

Each input device has its strengths. The keyboard is best for text input and giving commands with minimal hand movement. The mouse is best for efficient, precise pointing. Touch is best for object manipulation and giving simple commands. A pen is best for freeform expression, as with handwriting and drawing.

Windows 8.1 is optimized for responsiveness, accuracy, and ease of use with touch while fully supporting traditional input methods (such as mouse, pen, and keyboard). The speed, accuracy, and tactile feedback that traditional input modes provide are familiar and appealing to many users and potentially better suited to specific interaction scenarios.

You can find guidelines related to mouse, pen, and accessibility in separate topics.

When you think about the interaction experience for your app:

Don't assume that if a UI works well for a mouse, it also works well for touch. While good mouse support is a start, a good touch experience has a few additional requirements.

Do assume that if a UI works well for a finger, it also works well for a pen. Making your app touchable goes a long way to also provide good pen support. The primary difference is that fingers have a blunter tip, so they need larger targets.

With touch, you can manipulate objects and UI directly, which makes for a quicker, more natural and engaging experience.

Provide a great touch experience

You should ensure users can perform critical and important tasks efficiently using touch input. However, specific app functionality, like text or pixel manipulation, might not be suitable for touch and can be reserved for the most suitable input device.

If you don't have much experience developing touch apps, it's best to learn by doing. Get a touch-enabled computer, put the mouse and keyboard aside, and use only your fingers to interact with your app. If you have a tablet, experiment with holding it in different positions, such as on your lap, lying flat on a table, or in your arms while you're standing. Try using it in portrait and landscape orientation.

Touch-optimized apps that work best with touch interaction are typically:

- **Natural and intuitive.** Interactions are designed to correspond to how users interact with objects in the real world.
- **Less intrusive.** Using touch is silent, and consequently much less distracting than typing or clicking.
- **Portable.** Touch devices are more compact because many tasks can be completed without a keyboard, mouse, pen, or touchpad. They're also more flexible because a work surface isn't required.
- **Direct and engaging.** Touch makes you feel like you are directly manipulating objects on the screen.
- **Less accurate.** Users can't target objects as accurately with touch, compared to a mouse or pen.

Touch provides a natural, real-world feel to interaction. Direct manipulation and animation complete this impression, by giving objects a realistic, dynamic motion and feedback. For example, consider a card game. Not only is it convenient and easy to drag cards using a finger, the experience takes on an engaging real-world feel when you can toss, glide, and spin the cards just like you would a physical deck. And when you try to

move a card that can't be moved, it's a better experience to have the card resist but not prevent movement, and settle back in place when released, to clearly indicate that the action was recognized but can't be done.

Fortunately, if your app is already well designed, providing a great touch experience is easy to do. For this purpose, a well-designed program:

- **Ensures most important tasks can be performed efficiently using a finger** (at least the tasks that don't involve a lot of typing or detailed pixel manipulation).
- **Uses large controls for touch.** Common controls have a minimum size of 23x23 pixels (13x13 DLUs), and the most commonly used controls are at least 40x40 pixels (23x22 DLUs). To avoid unresponsive behavior, UI elements should have at least 5 pixels (3 DLUs) of space between them. For other controls, make sure they have at least a 23x23 pixel (13x13 DLU) click target, even if their static appearance is much smaller. See standard control sizing.
- **Supports Mouse input.** The interactive controls have clear, visible affordances. Objects have standard behaviors for the standard mouse interactions (single and double left-click, right-click, drag, and hover).
- **Supports keyboard input.** The app provides standard shortcut key assignments, especially for navigation and editing commands that can also be generated through touch gestures.
- **Ensures accessibility.** Uses UI Automation or Microsoft Active Accessibility (MSAA) to provide programmatic access to the UI for assistive technologies. The app responds appropriately to orientation, theme, locale, and system metric changes.
- **Eliminates unnecessary interactions.** To prevent loss of data or system access, use the safest and most secure default values. If safety and security aren't factors, the app selects the most likely or convenient option.
- **Provides touch equivalent for hover.** Don't rely on hover as the only way to perform an action.
- **Ensures gestures take effect immediately.** Keep contact points under the user's fingers smoothly throughout the gesture, which provides the effect of the gesture mapping directly to the user's motion.
- **Uses standard gestures whenever possible.** Custom gestures only for interactions unique to your app.
- **Ensures undesired or destructive commands can be reversed or corrected.** Accidental actions are more likely when using touch.

Guidelines for touch input

With touch, your Windows app can use physical gestures to emulate the direct manipulation of UI elements.

Consider these best practices when designing your touch-enabled app:

Responsiveness is essential for creating touch experiences that feel direct and engaging. To feel direct, gestures must take effect immediately, and an object's contact points must stay under the user's fingers smoothly throughout the gesture. The effect of touch input should map directly to the user's motion, so, for example, if the user rotates his fingers 90 degrees, the object should rotate 90 degrees as well. Any lag, choppy response, loss of contact, or inaccurate results destroys the perception of direct manipulation and also of quality.

Consistency is essential for creating touch experiences that feel natural and intuitive. Once users learn a standard gesture, they expect that gesture to have the same effect across all apps. To avoid confusion and frustration, never assign non-standard meanings to standard gestures. Instead, use custom gestures for interactions unique to your program.

Next we'll describe the Windows touch language, but before we go on, here's a short list of basic touch input terms.

- **Gesture**

  A gesture is the physical act or motion performed on, or by, the input device (finger, fingers, pen/stylus, mouse, and so on). For example, to launch, activate, or invoke a command, you use a single finger tap for a touch or touchpad device (equivalent to a left-click with a mouse, a tap with a pen, or Enter on a keyboard).

- **Manipulation**

  A manipulation is the immediate, real-time reaction or response an object or UI has to a gesture. For example, both the slide and swipe gestures typically cause an element or UI to move in some way.

  The final outcome of a manipulation, how it's manifested by the object on the screen and in the UI, is the interaction.

- **Interaction**

  Interactions depend on how a manipulation is interpreted and the command or action that results from the manipulation. For example, objects can be moved using both the slide and swipe gestures, but the results differ depending on whether a distance threshold is crossed. Slide can be used to drag an object or pan a view while swipe can be used to select an item or display an app bar.

The Windows touch language

Windows provides a concise set of touch interactions that are used throughout the system. Applying this touch language consistently makes your app feel familiar to what users already know. This increases user confidence by making your app easier to learn and use. To learn more about touch language implementation, see Gestures, manipulations, and interactions.

Press and hold to learn

The press and hold gesture displays detailed info or teaching visuals (for example, a tooltip or context menu) without committing to an action or command. Panning is still possible if a sliding gesture is started while the visual is displayed.

> **ⓘ Important**
>
> You can use press and hold for selection in cases where both horizontal and vertical panning is enabled.

Entry state: One or two fingers in contact with the screen.

Motion: No motion.

Exit state: Last finger up ends the gesture.

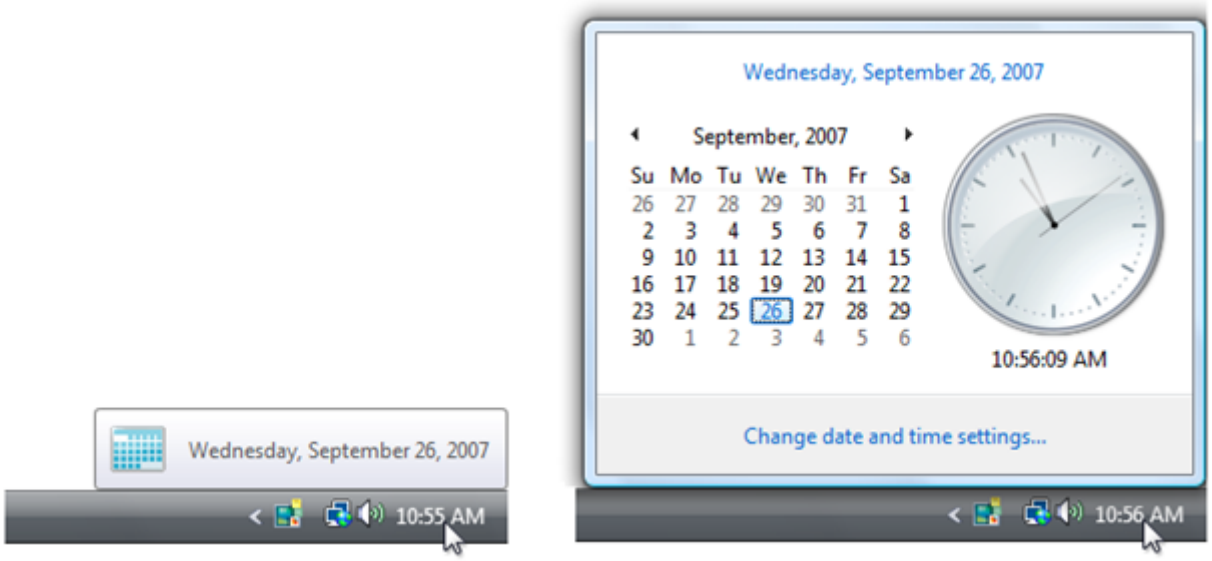
Effect: Display more information.

The press and hold gesture.

Hover

Hover is a useful interaction because it allows users to get additional information through tips before initiating an action. Seeing these tips makes users feel more confident and reduces errors.

Unfortunately, hover isn't supported by touch technologies, so users can't hover when using a finger. The simple solution to this problem is to take full advantage of hover, but only in ways that are not required to perform an action. In practice, this usually means that the action can also be performed by clicking, but not necessarily in exactly the same way.



In this example, users can see today's date by either hovering or clicking.

Tap for primary action

Tapping on an element invokes its primary action, for instance launching an app or executing a command.

Entry state: One finger in contact with the screen or touchpad and lifted before the time threshold for a press and hold interaction occurs.

Motion: No motion.

Exit state: Finger up ends the gesture.

Effect: Launch an app or execute a command.

The tap gesture.

Slide to pan

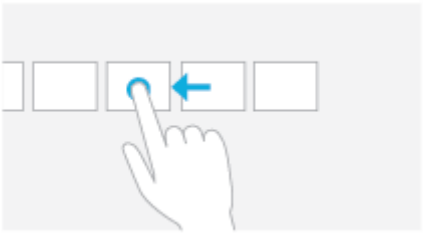
Slide is used primarily for panning interactions but can also be used for moving (where panning is constrained to one direction), drawing, or writing. Slide can also be used to target small, densely packed elements by scrubbing (sliding the finger over related objects such as radio buttons).

Entry state: One or two fingers in contact with the screen.

Motion: Drag, with any additional fingers remaining in same position relative to each other.

Exit state: Last finger up ends the gesture.

Effect: Move the underlying object directly and immediately as the fingers move. Be sure to keep the contact point under the finger throughout the gesture.



The pan gesture.

Swipe to select, command, and move

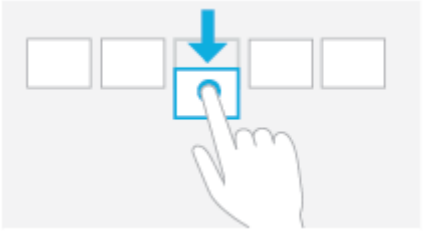
Sliding the finger a short distance, perpendicular to the panning direction (where panning is constrained to one direction), selects objects in a list or grid. Display the app bar with relevant commands when objects are selected.

Entry state: One or more fingers touch the screen.

Motion: Drag a short distance and lift before the distance threshold for a move interaction occurs.

Exit state: Last finger up ends the gesture.

Effect: Underlying object is selected, or moved, or app bar is displayed. Be sure to keep the contact point under the finger throughout the gesture.



The swipe gesture.

Pinch and stretch to zoom

The pinch and stretch gestures are used for three types of interactions: optical zoom, resizing, and semantic zoom.

Optical zoom adjusts the magnification level of the entire content area to get a more detailed view of the content. In contrast, resizing is a technique for adjusting the relative size of one or more objects within a content area without changing the view into the content area.

Semantic zoom is a touch-optimized technique for presenting and navigating structured data or content within a single view (such as the folder structure of a computer, a library of documents, or a photo album) without the need for panning, scrolling, or tree view controls. Semantic zoom provides two different views of the same content by letting you see more detail as you zoom in and less detail as you zoom out.

Entry state: Two fingers in contact with the screen at the same time.

Motion: Fingers move apart (stretch) or together (pinch) along an axis.

Exit state: Any finger up ends the gesture.

Effect: Zoom the underlying object in or out directly and immediately as the fingers separate or approach on the axis. Be sure to keep the contact points under the finger throughout the gesture.

The zoom gesture.

Turn to rotate

Rotating with two or more fingers causes an object to rotate. Rotate the device itself to rotate the entire screen.

Entry state: Two fingers in contact with the screen at the same time.

Motion: One or both fingers rotate around the other, moving perpendicular to the line between them.

Exit state: Any finger up ends the gesture.

Effect: Rotate the underlying object the same amount as the fingers have rotated. Be sure to keep the contact points under the finger throughout the gesture.

The rotation gesture.

Rotation makes sense only for certain types of objects, so it's not mapped to a system Windows interaction.

Rotation is often done differently by different people. Some people prefer to rotate one finger around a pivot finger, while others prefer to rotate both fingers in a circular motion. Most people use a combination of the two, with one finger moving more than the other. While smooth rotation to any angle is the best interaction, in many contexts, such as photo viewing, it's best to settle to the nearest 90 degree rotation once the user lets go. In photo editing, you can use a small rotation to straighten the photo.

Swipe from edge for app commands

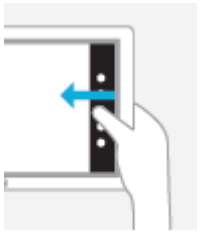
Swiping the finger a short distance from the bottom or top edge of the screen reveals app commands in an app bar.

Entry state: One or more fingers touch the bezel.

Motion: Drag a short distance onto the screen and lift.

Exit state: Last finger up ends the gesture.

Effect: App bar is displayed.



The swipe from edge gesture.

Developers: For more info, see **DIRECTMANIPULATION_CONFIGURATION** enumeration.

Control usage

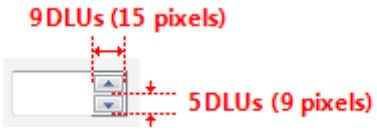
Here, we provide some guidelines for optimizing controls for touch usage.

- **Use common controls.** Most common controls are designed to support a good touch experience.
- **Choose custom controls that are designed to support touch.** You might need custom controls to support your program's special experiences. Choose custom controls that:
  - Can be sized large enough to for easy targeting and manipulation.
  - When manipulated, move and react the way real-world objects move and react, such as by having momentum and friction.
  - Are forgiving by allowing users to easily correct mistakes.
  - Are forgiving of inaccuracy with clicking and dragging. Objects that are dropped near their destination should fall into the correct place.
  - Have clear visual feedback when the finger is over the control.
- **Use constrained controls.** Constrained controls like lists and sliders, when designed for easy touch targeting, can be better than unconstrained controls like text boxes because they reduce the need for text input.
- **Provide appropriate default values.** Select the safest (to prevent loss of data or system access) and most secure option by default. If safety and security aren't factors, select the most likely or convenient option, thereby eliminating unnecessary interaction.
- **Provide text auto completion.** Provides a list of most likely values, or most recently input values, to make text input much easier.
- **For important tasks that use multiple selection**, if a standard multiple-selection list is normally used, provide an option to use a check box list instead.

Control sizes and touch targeting

Due to the large surface area of the fingertip, small controls that are too close together can be difficult to target precisely.

As a general rule, a control size of 23x23 pixels (13x13 DLUs) is a good minimum interactive control size for any input device. By contrast, the spin controls at 15x11 pixels are much too small to be used effectively with touch.



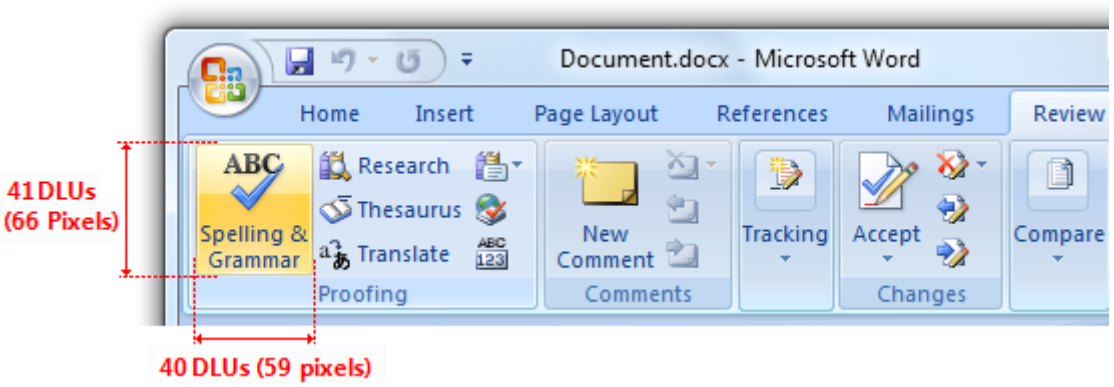
Keep in mind that the minimum size is really based on physical area, not layout metrics such as pixels or DLUs. Research indicates that the minimum target area for efficient, accurate interaction using a finger is 6x6 millimeters (mm). This area translates to layout metrics like this:

Expand table

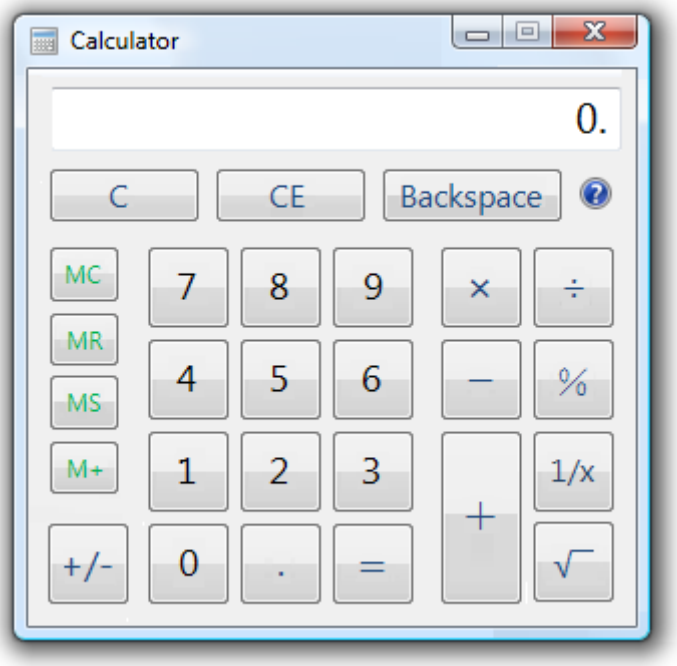
Font	Millimeters	Relative pixels	DLUs
9 point Segoe UI	6x6	23x23	13x13
8 point Tahoma	6x6	23x23	15x14

Furthermore, research shows that a minimum size of 10x10 mm (about 40x40 pixels) enables better speed and accuracy, and also feels more comfortable to users. When practical, use this larger size for command buttons used for the most important or frequently used commands.

The goal isn't to have giant controls, just ones that are easily used with touch.



In this example, Microsoft Word uses buttons larger than 10x10 mm for the most important commands.

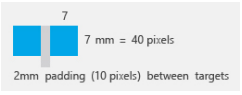
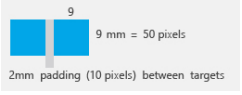
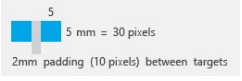


This version of Calculator uses buttons larger than 10x10 mm for its most frequently used commands.

There's no perfect size for touch targets. Different sizes work for different situations. Actions with severe consequences (such as delete and close) or frequently used actions should use large touch targets. Infrequently used actions with minor consequences can use small targets.

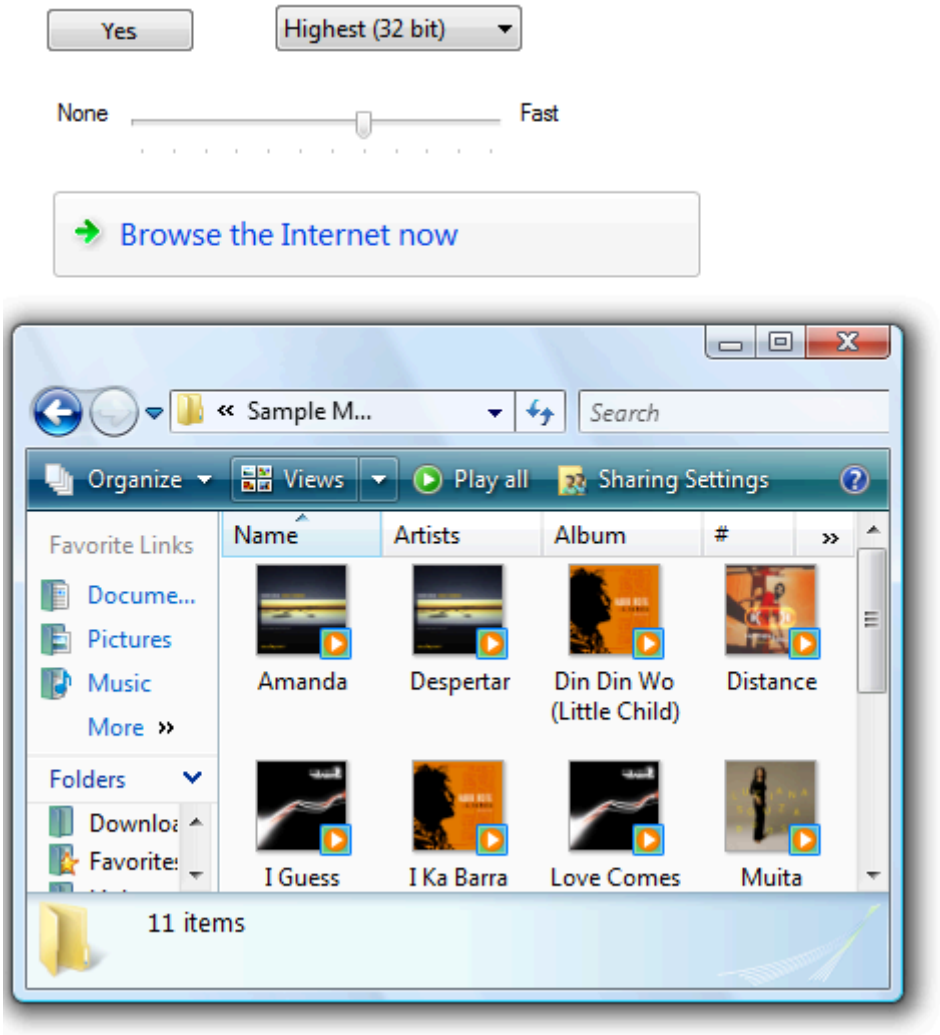
Target size guidelines for custom controls

Expand table

Size guideline	Description
	**7x7 mm: Recommended minimum size** 7x7 mm is a good minimum size if touching the wrong target can be corrected in one or two gestures or within five seconds. Padding between targets is just as important as target size.
	**When accuracy matters** Close, delete, and other actions with severe consequences can't afford accidental taps. Use 9x9 mm targets if touching the wrong target requires more than two gestures, five seconds, or a major context change to correct.
	**When it just won't fit** If you find yourself cramming things to fit, it's okay to use 5x5 mm targets as long as touching the wrong target can be corrected with one gesture. Using 2 mm of padding between targets is extremely important in this case.

Target size guidelines for common controls

- **For common controls, use the recommended control sizes.** The recommended control sizing satisfies the 23x23 pixel (13x13 DLU) minimum size, except for check boxes and radio buttons (their text width compensates somewhat), spin controls (which aren't usable with touch, but are redundant), and splitters.



The recommended control sizes are easily touchable.

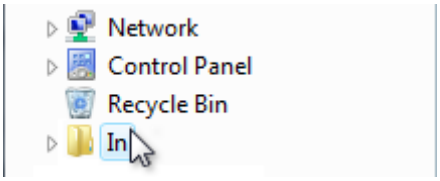
- **For command buttons used for the most important or frequently used commands, use a minimum size of 40x40 pixels (23x22 DLUs) whenever practical.** Doing so yields better speed and accuracy, and also feels more comfortable to users.

Whenever practical, use larger command buttons for important or frequently used commands.

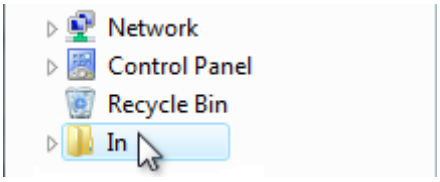
- **For other controls:**
  - **Use larger click targets.** For small controls, make the target size larger than the statically visible UI element. For example, 16x16 pixel icon buttons can have a 23x23 pixel click target buttons, and text elements can have selection rectangles 8 pixels wider than the text and 23 pixels high.

    Correct:

    Incorrect:

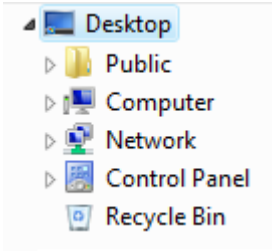
    Correct:

    In the correct examples, the click targets are larger than the statically visible UI elements.

  - **Use redundant click targets.** It's acceptable for click targets to be smaller than the minimum size if that control has redundant functionality.

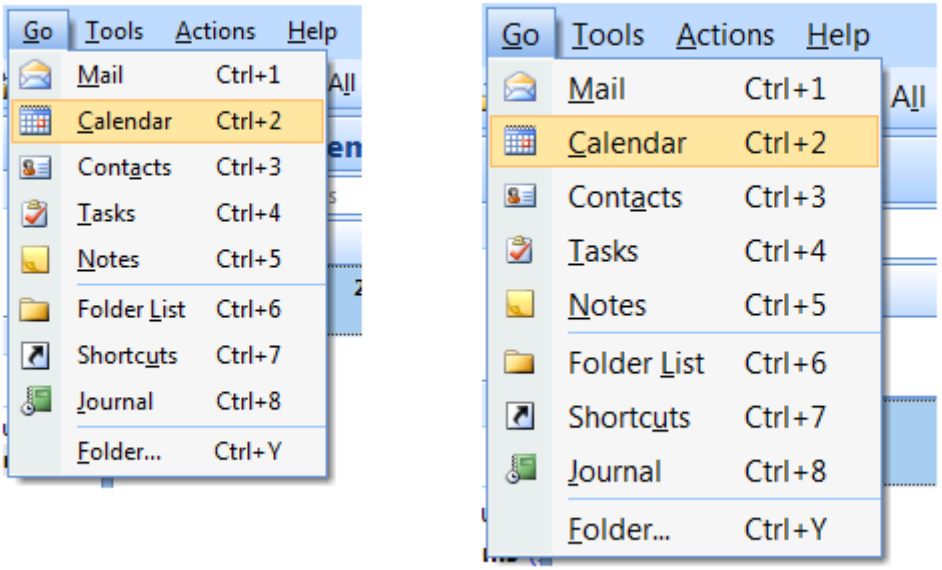
    For example, the progressive disclosure triangles used by the tree view control are only 6x9 pixels, but their functionality is redundant with their associated item labels.

    The tree view triangles are too small to be easily touchable, but they are redundant in functionality with their larger associated labels.

    Respect system metrics. Don't hardcode sizes. If necessary, users can change the system metrics or dpi to accommodate their needs. However, treat this as a last

resort because users shouldn't normally have to adjust system settings to make UI usable.

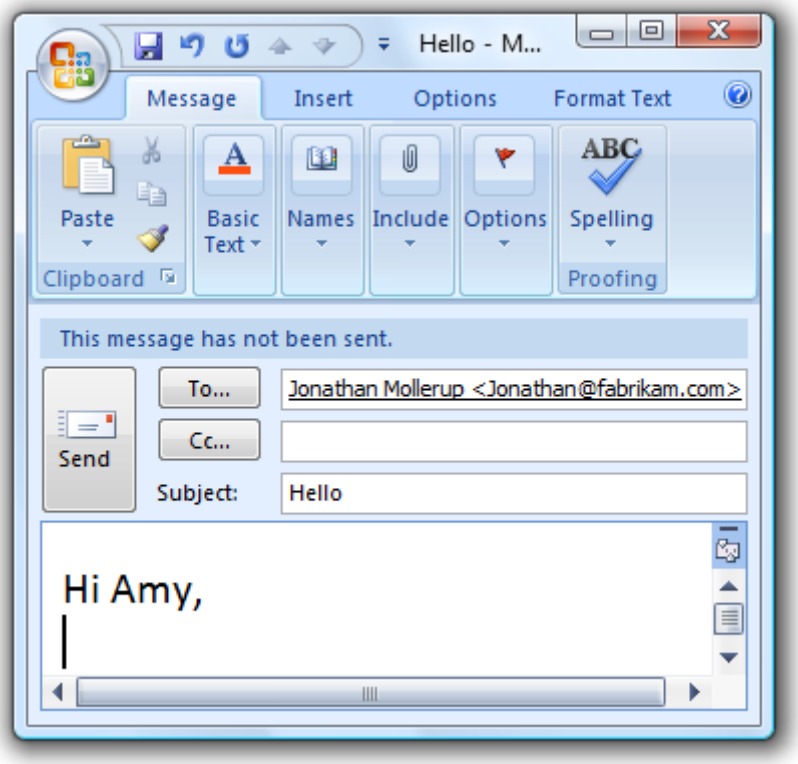


In this example, the system metric for menu height was changed.

Editing text

Editing text is one of the most challenging interactions when using a finger. Using constrained controls, appropriate default values, and auto-completion eliminates or reduces the need to input text. But if your app involves editing text, you can make users more productive by automatically zooming input UI up to 150 percent by default when touch is used.

For example, an e-mail program could display UI at normal touchable size, but zoom the input UI to 150 percent to compose messages.



In this example, the input UI is zoomed to 150 percent.

Control layout and spacing

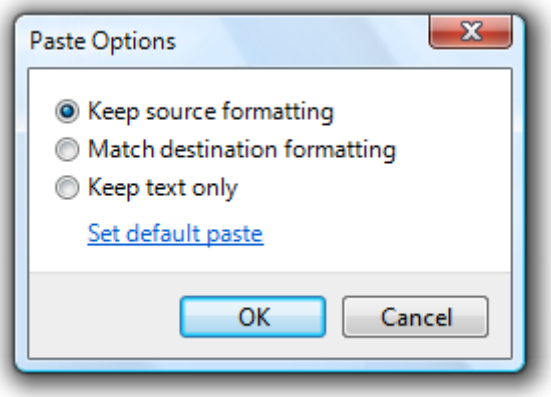
The spacing between controls is a significant factor in making controls easily touchable. Targeting is quicker but less precise when using a finger as the pointing device, resulting in users more often tapping outside their intended target. When interactive controls are placed very close together but are not actually touching, users may click on inactive space between the controls. Because clicking inactive space has no result or visual feedback, users are often uncertain what went wrong.

Dynamically adjust spacing based on the input device used. This is particularly useful with transient UI such as menus and flyouts.

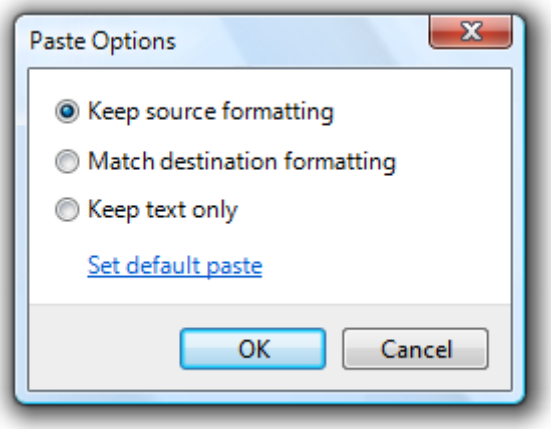
Provide a minimum of 5 pixels (3 DLUs) of space between the target regions of interactive controls. If small controls are too closely spaced, the user needs to tap with precision to avoid tapping the wrong object.

Make controls within groups easier to differentiate by using more than the recommended vertical spacing between controls. For example, radio buttons at 19 pixels high are shorter than the minimum recommended size of 23 pixels. When you have vertical space available, you can achieve roughly the same effect as the recommended sizing by adding an additional 4 pixels of spacing to the standard 7 pixels.

Correct:



Better:

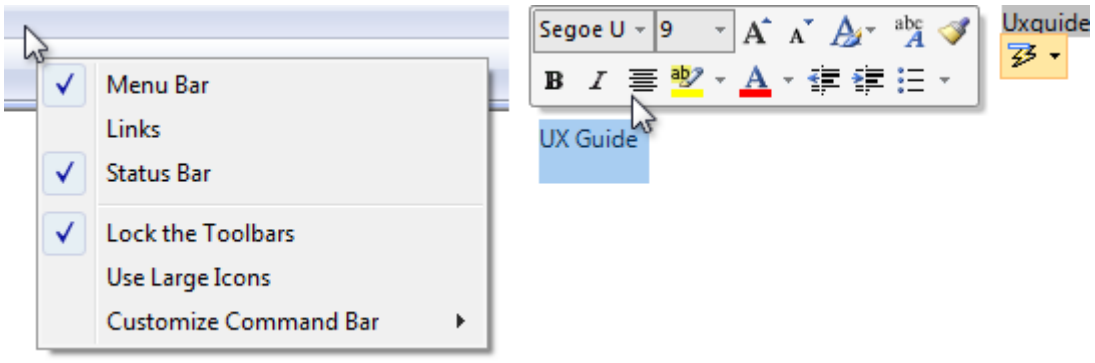


In the better example, the extra spacing between the radio buttons makes them easier to differentiate.

There may be situations in which extra spacing would be desirable when using touch, but not when using the mouse or keyboard. In such cases, only use a more spacious design when an action is initiated using touch.

Choose a layout that places controls close to where they are most likely going to be used. Keep task interactions within a small area whenever possible and locate controls close to where they are most likely going to be used. Avoid long distance hand movements, especially for common tasks and for drags.

Consider that the current pointer location is the closest a target can be, making it trivial to acquire. Thus, context menus take full advantage of Fitts' law, as do the mini-toolbars used by Microsoft Office.



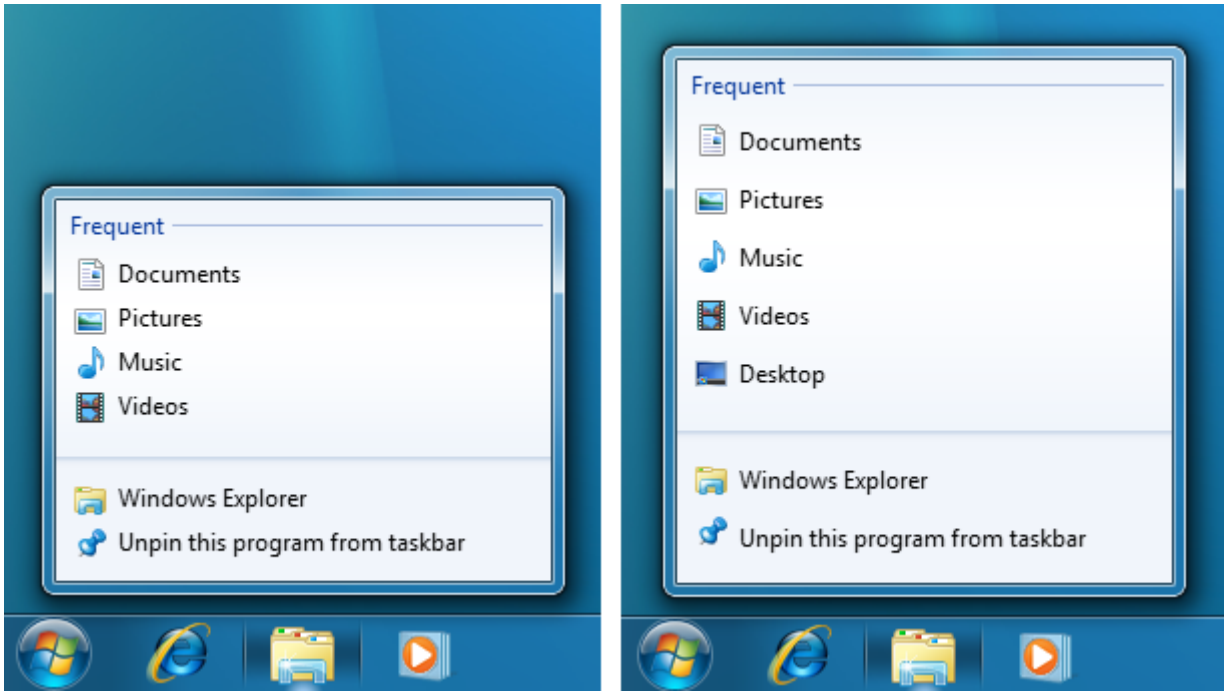
Avoid placing small controls near the edge of the app or the display. Small targets near edges can be difficult to touch (display bezels can interfere with edge gestures). To ensure that controls are easy to target when a window is maximized, either make them at least 23x23 pixels (13x13 DLUs) or place them away from the window edge.

Use the recommended spacing. The recommended spacing is touch-friendly. However, if your app can benefit from larger sizing and spacing, consider the recommended sizing and spacing to be minimums when appropriate.

Provide at least 5 pixels (3 DLUs) of space between interactive controls. Doing so prevents confusion when users tap outside their intended target.

Consider adding more than the recommended vertical spacing within groups of controls, such as command links, check boxes, and radio buttons, as well as between the groups. Doing so makes them easier to differentiate.

Consider adding more than the recommended vertical spacing dynamically when an action is initiated using touch. Doing so makes objects easier to differentiate, but without taking more space when using a keyboard or mouse. Increase the spacing by a third of its normal size or at least 8 pixels.



In this example, Windows 7 taskbar Jump Lists are more spacious when displayed using touch.

Interaction

Using the correct controls gets you only part of the way to a touch-optimized app, you also need to consider the overall interaction model those controls support. Here are a few guidelines to help you with this.

- **Make hover redundant.** Hover isn't supported by most touch technologies, so users with such touchscreens can't perform any tasks that require hovering.
- **For apps that need text input, fully integrate the touch keyboard feature** by:
  - Providing appropriate default values for user input.
  - Providing auto-complete suggestions when appropriate.

  > **Note**
  >
  > Developers: For more info about integrating the touch keyboard, see ITextInputPanel.

- **Allow users to zoom the content UI if your program has tasks that require editing text.** Consider automatically zooming to 150 percent when touch is used.
- **Provide smooth, responsive panning and zooming wherever appropriate.** Redraw quickly after a pan or zoom to remain responsive. Doing so is necessary to make direct manipulation feel truly direct.
- **During a pan or zoom, make sure that the contact points stay under the finger throughout the gesture.** Otherwise, the pan or zoom is difficult to control.
- **Because gestures are memorized, assign them meanings that are consistent across apps.** Don't give different meanings to gestures with fixed semantics. Use an appropriate app-specific gesture instead.

Forgiveness

Direct manipulation makes touch natural, expressive, efficient, and engaging. However, where there is direct manipulation, there can be accidental manipulation and therefore the need for forgiveness.

Forgiveness is the ability to reverse or correct an undesired action easily. You make a touch experience forgiving by providing undo, giving good visual feedback, having a clear physical separation between frequently used commands and destructive commands, and allowing users to correct mistakes easily. Associated with forgiveness is preventing undesired actions from happening in the first place, which you can do by using constrained controls and confirmations for risky actions or commands that have unintended consequences.

- **Provide an Undo command.** It's best to provide a simple way to undo all commands, but your app may have some commands whose effect cannot be undone.
- **Whenever practical, provide good feedback on finger down, but don't take actions until finger up.** Doing so allows users to correct mistakes before they make them.
- **Whenever practical, allow users to correct mistakes easily.** If an action takes effect on finger up, allow users to correct mistakes by sliding while the finger is still down.
- **Whenever practical, indicate that a direct manipulation can't be performed by resisting the movement.** Allow the movement to happen, but have the object settle back in place when released to clearly indicate the action was recognized but can't be done.
- **Have clear physical separation between frequently used commands and destructive commands.** Otherwise, users might touch destructive commands accidentally. A command is considered destructive if its effect is widespread and either it cannot be easily undone or the effect isn't immediately noticeable.
- **Confirm commands for risky actions or commands that have unintended consequences.** Use a confirmation dialog box for this purpose.
- **Consider confirming any other actions that users tend to do accidentally when using touch, and which either go unnoticed or are difficult to undo.** Normally, these are called routine confirmations and are discouraged based on the assumption that users don't often issue such commands by accident with a mouse or keyboard. To prevent unnecessary confirmations, present these confirmations only if the command was initiated using touch.

  Routine confirmations are acceptable for interactions that users often do accidentally using touch.

Developers: You can distinguish between mouse events and touch events using the **INPUT_MESSAGE_SOURCE** API.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Keyboard

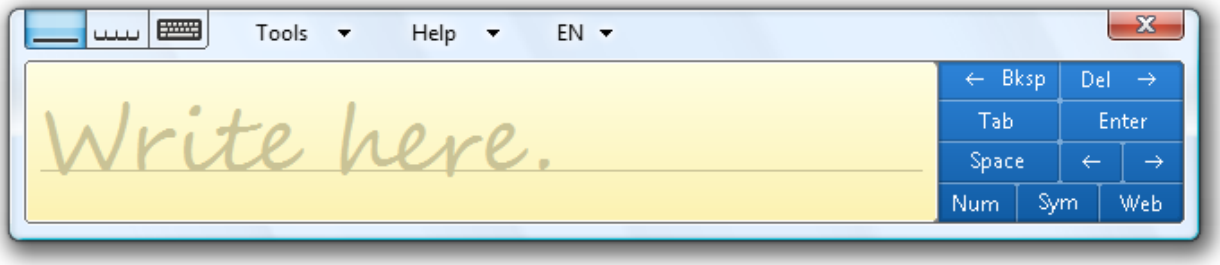
Article • 06/04/2021

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

The keyboard is the primary input device used for text input in Microsoft Windows. For accessibility and efficiency, most actions can be performed using the keyboard as well.

Keyboards can also refer to virtual, on-screen keyboards and writing pads used by computers without a physical keyboard, such as tablet-based computers.

The Windows Tablet and Touch Technology on-screen keyboard.



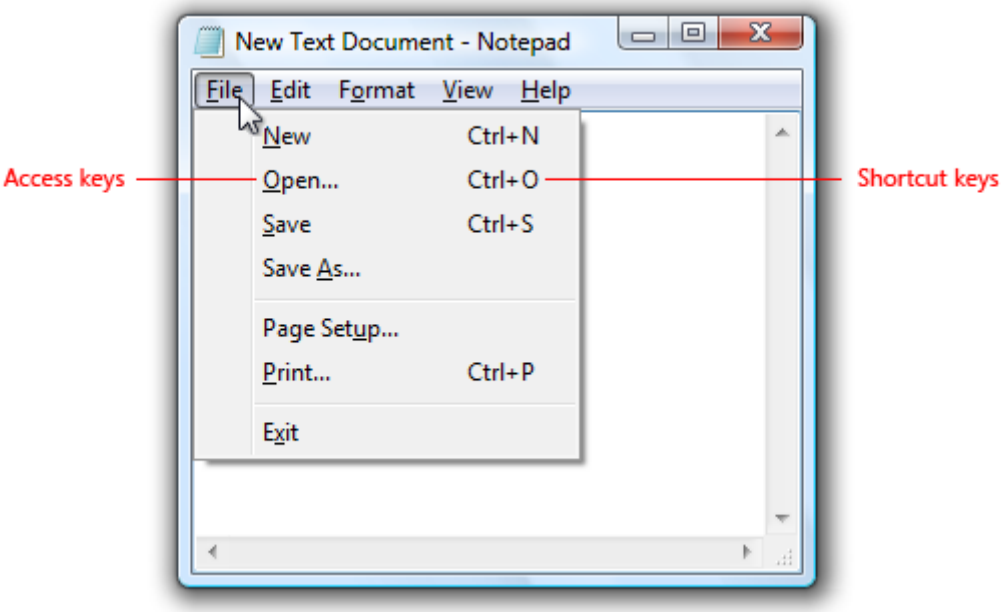
The Windows Tablet and Touch Technology writing pad.

There are six basic types of keys:

- A character key sends a literal character to the window with input focus.
- A modifier key combined with another key alters the meaning of its associated key, such as Ctrl, Alt, Shift, and the Windows logo key.
- The navigation keys are the directional arrows, plus Home, End, Page Up, and Page Down.
- The editing keys are Insert, Backspace, and Delete.

- The function keys are F1 through F12.
- System keys put the system into a mode or perform a system task, such as Print Screen, Caps Lock, and Num Lock.

Access keys are keys or key combinations used for accessibility to interact with all controls or menu items using the keyboard. Shortcut keys are keys or key combinations used by advanced users to perform frequently used commands for efficiency. Windows indicates access keys by underlining the access key assignment.



This example shows both access keys and shortcut keys.

To eliminate visual clutter, Windows hides access key underlines by default and displays them only when the Alt key is pressed. To maintain consistency with Windows, the images in UX Guide are also shown with the access key underlines hidden unless the guideline involves access keys.

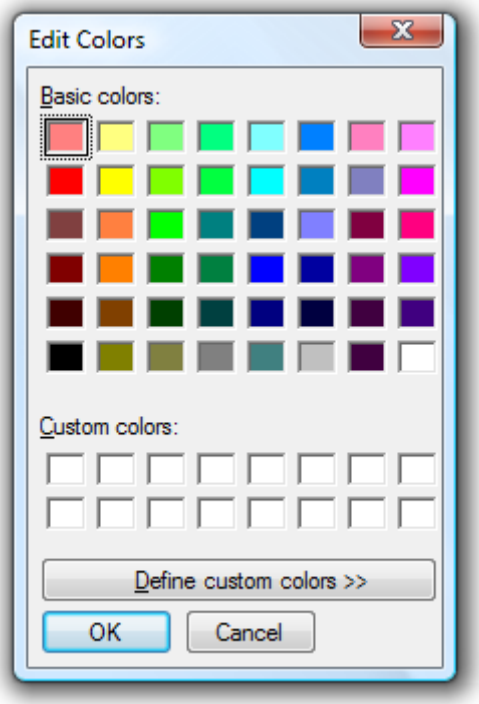
To improve awareness of the access key assignments in your program throughout the development process, you can display them at all times. In Control Panel, go to the Ease of Access Center, and click **Make the keyboard easier to use**; then select the **Underline keyboard shortcuts and access keys** check box.

Note: Guidelines related to accessibility are presented in a separate article.

Design concepts

Elements of keyboard navigation

Users interact with a window using the keyboard by navigating to controls, making selections, and performing commands. The following elements work together to make this happen.



To illustrate the elements of keyboard navigation in the following list, we'll refer to this dialog box.

- **Input focus.** The control with input focus receives most keyboard input. Input focus is indicated with a dotted rectangle called the focus rectangle. Some keyboard input is sent to controls that don't have input focus, as explained later.

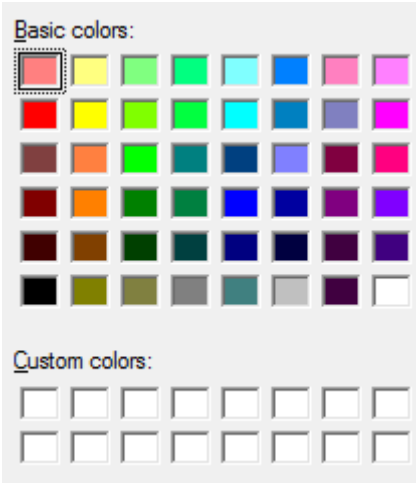
  The first Basic colors control has input focus, as indicated with a dotted rectangle.

- **Tab key and tab stops.** The Tab key is the primary mechanism for navigating within a window. The Tab key visits only those controls with a tab stop. All interactive controls should have tab stops (unless they are in a group), whereas non-interactive controls, such as labels, should not.

- **Tab order.** All controls with tab stops are visited in tab order. Pressing Tab moves input focus to the next control in tab order, whereas pressing Shift+Tab moves input focus to the previous control.

- **Control groups.** A set of related controls can be made into a group and be assigned a single tab stop. Control groups are used for sets of controls that behave

like a single control, such as radio buttons. They can also be used when there too many controls to navigate efficiently with the Tab key alone.



Basic colors and Custom colors are control groups, giving this dialog box five tab stops. There are so many controls that navigation would be inefficient without using control groups.

- **Arrow keys.** The arrow keys move input focus among the controls within a group. Pressing the right arrow key moves input focus to the next control in tab order, whereas pressing the left arrow moves input focus to the previous control. Home, End, Up, and Down also have their expected behavior within a group. Users can't navigate out of a control group using arrow keys.

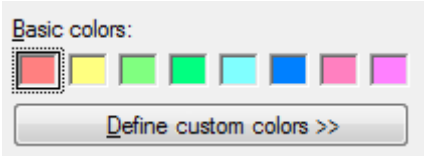
- **Default buttons.** Windows with command buttons and command links have a single default button indicated by a highlighted border, which is the button that is clicked when the Enter key is pressed. There is a single default command button or command link assigned by default. However, the default button moves when the user tabs to another command button or command link. Consequently, any command button or command link with input focus is also always the default button.

The OK button is normally the default button, as indicated by its highlighted border. However, if the user were to tab to the Cancel button, it would become the default button and would be activated with the Enter key.

- **Spacebar, Enter, and Esc keys.** The spacebar activates the control with input focus, whereas the Enter key activates the default button. Pressing the Esc key cancels or closes the window.

- **Access keys.** Access keys are used to interact with controls directly instead of navigating with Tab. They are combined with the Alt key and indicated with an underlined letter in their label.

- **Access key labels.** While some controls contain their own labels, such as command buttons, check boxes, and radio buttons, other controls have external labels, such as list boxes and tree views. For external labels, the access key is assigned to the label, and if invoked, navigates to the next control in tab order. Buttons labeled OK, Cancel, and Close aren't assigned access keys because they are invoked with Enter and Esc.

  Pressing Alt+B navigates to the selected basic color, pressing Alt+D clicks the Define Custom Colors button, Enter invokes the OK button, and Esc invokes Cancel.

- **Access key behavior.** When an access key is invoked and it is assigned uniquely, the associated control is clicked. If the assignment isn't unique, the associated control is given input focus. If the user types the same access key again, the next control in tab order with the same assignment is given input focus.

While this mechanism is fairly complicated, it is also fairly intuitive. Users pick up most these details right away, even though few can explain exactly how they work.

Keyboard support for accessibility and advanced users

In Windows, designing for the keyboard boils down to providing well-designed keyboard navigation, access keys for accessibility, and shortcut keys for advanced users.

To ensure that your program's functionality is easily available to the widest range of users, including those who have disabilities and impairments, all interactive user interface (UI) elements must be keyboard accessible. Generally, this means that the most commonly used UI elements are accessible using a single access key or key combination, whereas less frequently used elements may require additional tab or arrow key navigation. For these users, comprehensiveness is more important than consistency.

To ensure that your program's functionality is efficient for experienced users, commonly used UI elements should also have shortcut keys for direct keyboard access. Experienced users often have a strong preference for using the keyboard, because keyboard-based

commands can be entered more quickly and don't require removing their hands from the keyboard. For these users, efficiency and consistency are crucial; comprehensiveness is important only for the most frequently used commands.

There are subtle distinctions when designing keyboard access for these two groups, which is why Windows provides two independent direct keyboard access mechanisms. By using both access and shortcut keys effectively, you can give your programs efficient, consistent, comprehensive keyboard access that benefits everyone.

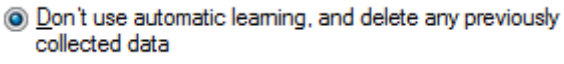
Access keys

Access keys have the following characteristics:

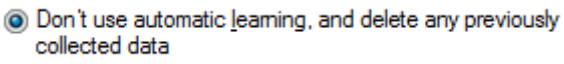
- They use the Alt key plus an alphanumeric key.
- They are primarily for accessibility.
- They are assigned to all menus and most dialog box controls.
- They aren't intended to be memorized, so they are documented directly in the UI by underlining the corresponding control label character.
- They have effect only in the current window, and navigate to the corresponding menu item or control.
- They aren't assigned consistently because they can't always be. However, access keys should be assigned consistently for commonly used commands, especially commit buttons.
- They are localized.

Because access keys aren't intended to be memorized, they are assigned to a character that is early in the label to make them easy to find, even if there is a keyword that appears later in the label.

Correct:



Incorrect:



In the correct example, the access key is assigned to a character that is early in the label.

Shortcut keys

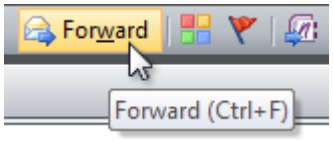
By contrast, shortcut keys have the following characteristics:

- They primarily use Ctrl and Function key sequences (Windows system shortcut keys also use Alt+non-alphanumeric keys and the Windows logo key).
- They are primarily for efficiency for advanced users.
- They are assigned only to the most commonly used commands.
- They are intended to be memorized, and are documented only in menus, tooltips, and Help.
- They have effect throughout the entire program, but have no effect if they don't apply.
- They must be assigned consistently because they are memorized and not directly documented.
- They aren't localized.

Because shortcut keys are intended to be memorized, the most frequently used shortcut keys ideally use letters from the first or most memorable characters within the command's keywords, such as Ctrl+C for Copy and Ctrl+Q for Request.

Inconsistent meanings for well-known shortcut keys are frustrating and cause errors.

Incorrect:



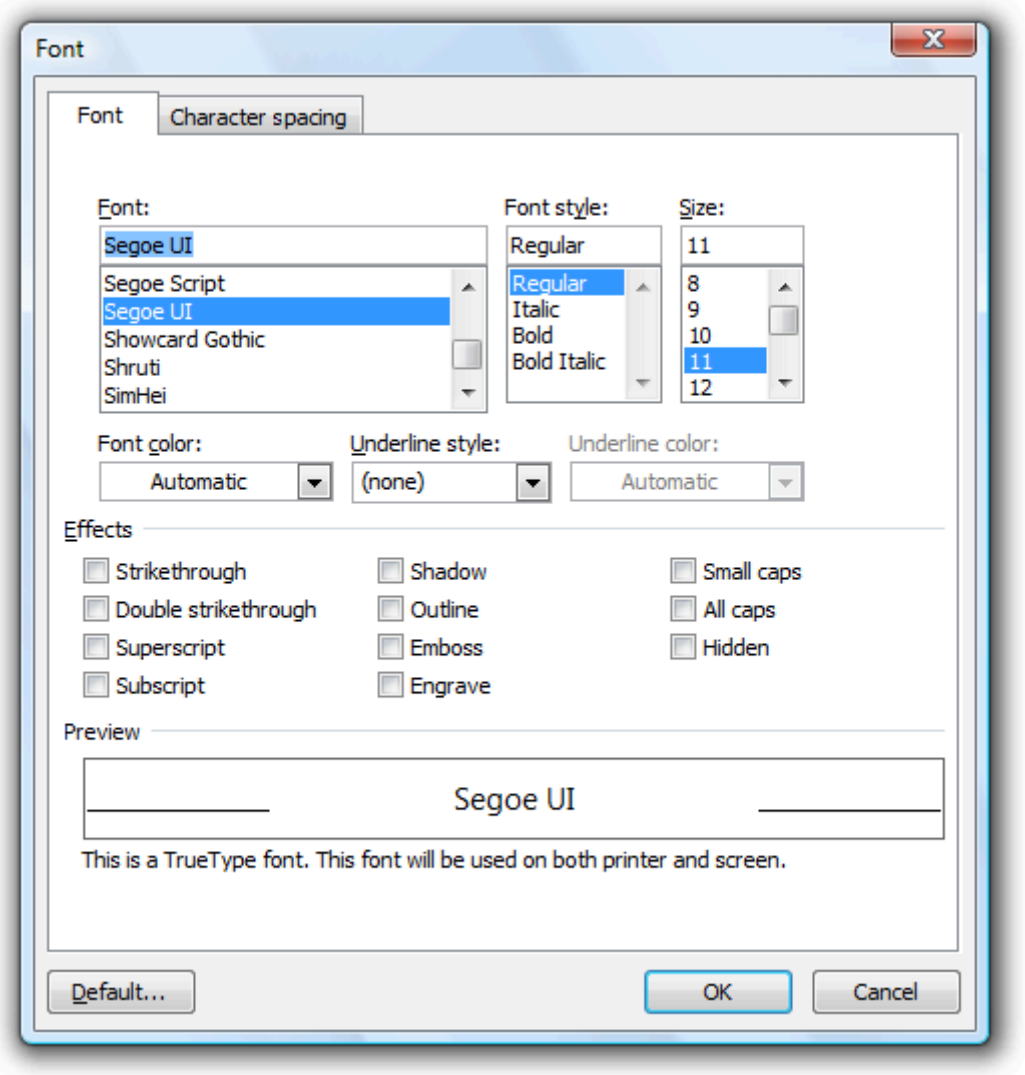
In this example, Ctrl+F is the standard shortcut for Find, so assigning it to Forward is frustrating and error prone. Ctrl+W would be a better, memorable choice.

Finally, because they are intended to be memorized, **application-specific shortcut keys make sense only for programs and features that are run frequently enough for motivated users to memorize.** Infrequently used programs and features don't need shortcut keys. For example, setup programs and most wizards don't need any special shortcut key assignments, nor do infrequently used commands in a productivity application.

Assigning access keys in dialog boxes

Whenever possible, assign unique access keys to all interactive controls except those that normally aren't assigned access keys. However, in English there are only 26 characters. Some characters may not appear in any of the labels, and there may not be distinctive characters in all the labels, reducing this number further. Also, you should plan to have a few unassigned characters to facilitate localization. Consequently, you can assign only about 20 unique access keys in a single dialog box.

If you have a dialog box with more than 20 interactive controls, either don't assign access keys to some controls, or, in rare situations, assign duplicate access keys.



When there are this many interactive controls, not all of them need an access key assigned.

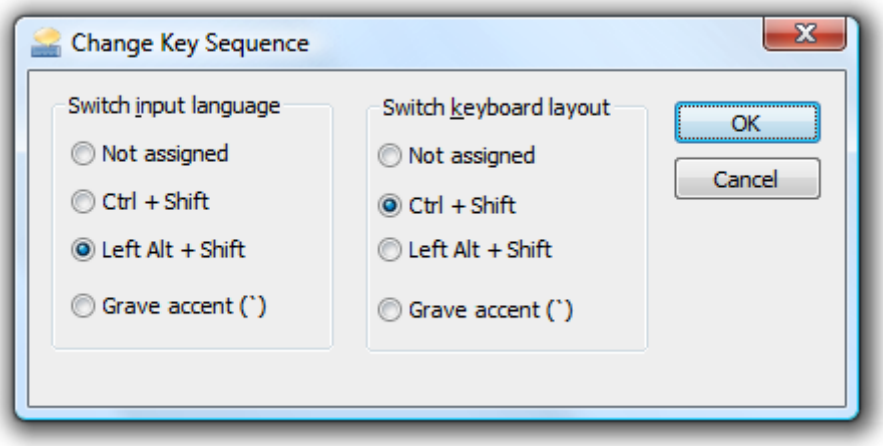
Use the following general procedure to assign access keys:

- First, assign access keys to the commit buttons and command links. Use the standard access key assignments table when it applies, otherwise use the first letter of the first word.
- Skip the controls that aren't assigned access keys.
- Assign unique access keys to the remaining controls (starting with the most frequently used):
  - If possible, assign the access key according to the standard access key assignments table.
  - Otherwise:
    - Prefer characters that appear early in the label, ideally the first character of the first or second word.
    - Prefer a distinctive consonant or a vowel, such as "x" in "Exit."

    - Prefer characters with wide widths, as w, m, and capital letters.
    - Avoid using characters that make the underline difficult to see, such as letters that are one pixel wide, letters with descenders, and letters next to a letter with a descender.
- If not all controls can have unique access keys (start with the least frequently used):
  - If there are groups of related controls, such as:
    - A single set of radio buttons
    - A set of related check boxes
    - A set of related controls within a group box

Assign access keys to group labels instead of the individual controls. Normally, you would do the opposite. (In doing so, make sure there is a control group defined for these controls.)

- If still not all controls can have unique access keys:
  - You may assign non-unique access keys if:
    - The controls would otherwise be too difficult to navigate to.
    - The non-unique access keys don't conflict with the access keys of commonly used controls.
  - Otherwise, the remaining controls can be accessed using Tab and arrow key navigation.



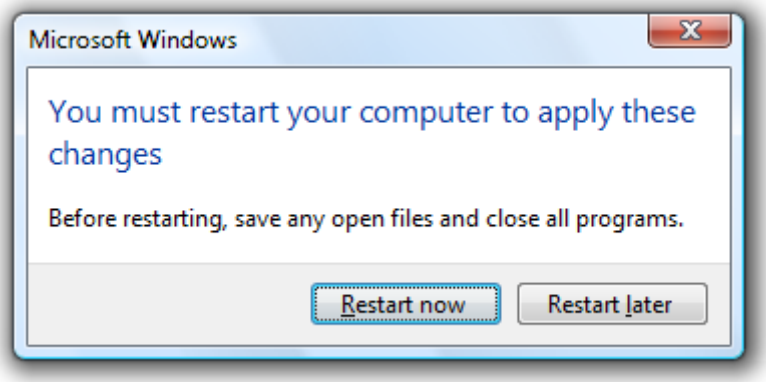
In this example, there are repetitive controls so access keys are assigned to the radio button groups.

Preventing accidental commands

If a window displayed out of context (not user initiated) steals input focus, there is a good chance that this window will receive input intended for another window. Furthermore, access keys take effect when pressed without depressing the Alt key if the dialog box doesn't have any controls that take text input (such as text boxes and lists). So, in the following example, pressing "r" activates the Restart now button.

Clearly, such input may have significant unintended consequences.

Incorrect:

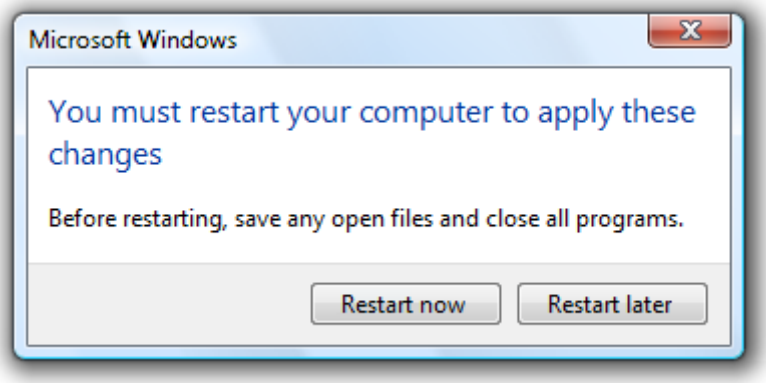


In this example, typing text with space, "r", or Enter accidentally restarts Windows.

Of course, the best solution to this problem is not to steal input focus. Instead, either flash the program's taskbar button or display a notification to get the user's attention.

However, if you must display such a window, the best approach is to not assign a default button or access keys, and give initial input focus to a control other than a commit button.

Correct:



In this example, accidentally restarting Windows is much harder to do.

If you do only six things...

1. Design good keyboard navigation, with a sensible tab order and appropriate control groups, initial input focus, and default buttons.
2. Assign access keys to all menus and most controls.
3. Assign the access keys to a character that appears early in the label, to make them easy to find.
4. Assign shortcut keys to the most commonly used commands.

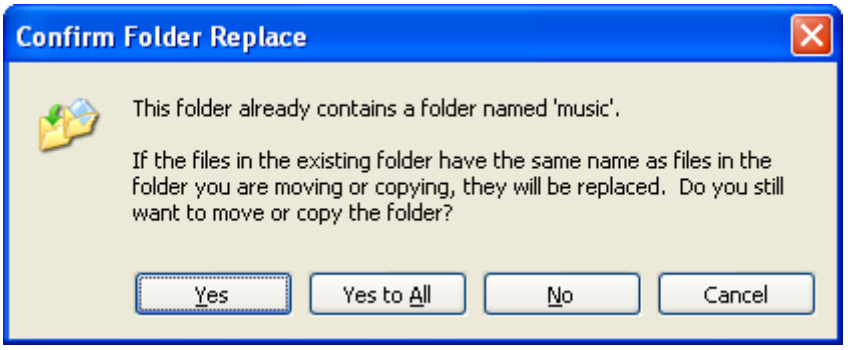
5. Try to assign the shortcut keys to the first or most memorable characters within keywords.
6. Give well-known shortcut keys a consistent meaning.

Guidelines

Interaction

- **Don't use the Shift key to modify commands in menus or dialog boxes.** Doing so is undiscoverable and unexpected.

  Incorrect:

  In this example from Windows XP, holding the Shift key replaces Yes to All with No to All.

- **Don't disable a control with input focus. Doing so may prevent the window from receiving keyboard input.** Instead, before disabling a control with input focus, move input focus to another control.

- **If a window is displayed out of context, potentially surprising users, you may need to prevent significant unintended consequences:**
  - Don't assign a default button.
  - Don't assign access keys.
  - Give initial input focus to a control other than a commit button.

Keyboard navigation

- **Always show the input focus indicator. Exception:** You can temporarily suppress the input focus indicator if:
  - The input focus indicator is visually distracting (as with a large list view not in Details view).
  - The Enter key's usage is likely preceded by other keyboard input, such as Alt or arrow keys.

    - The input focus indicator is displayed upon any keyboard input.
- **Assign initial input focus to the control that users are most likely to interact with first,** which is often the first interactive control. If the first interactive control isn't a good choice, consider changing the window's layout.
- **Assign tabs stops to all interactive controls, including read-only edit boxes. Exceptions:**
    - Group sets of related controls that behave as a single control, such as radio buttons. Such groups have a single tab stop.
    - Properly contain groups so that the arrow keys cycle both forward and backward within the group and stay within the group.
- **Tab order should follow reading order, which generally flows from left to right, top to bottom.** Consider making exceptions for commonly used controls by putting them earlier in the tab order. Tab should cycle through all the tab stops in both directions without stopping.
- **Within a tab stop, the arrow key order should flow from left to right, top to bottom,** without exceptions. The arrow keys should cycle through all items in both directions without stopping.
- **Present the commit buttons in the following order:**
    - OK/[Do it]/Yes
    - [Don't do it]/No
    - Cancel
    - Apply (if present)

where [Do it] and [Don't do it] are specific responses to the main instruction.

- **Select the safest (to prevent loss of data or system access) and most secure command button or command link to be the default.** If safety and security aren't factors, select the most likely or convenient response.
- **Keyboard navigation shouldn't change control values or result in an error message.** Never require users to change a control's initial value during navigation. Instead, initialize controls that validate on exit with valid values, and validate a control's value only when it has changed.

Access keys

- **Whenever possible, assign access keys for commonly used commands according to the following table.** While consistent access key assignments aren't always possible, they are certainly preferred especially for frequently used commands.

Expand table

Access key	Command
A	About
A	Always on top
A	Apply
B	Back
B	Bold
B or r	Browse
C	Close
C	Copy
C	Copy here
s	Create shortcut
s	Create shortcut here
t	Cut
D	Delete
D	Don't show this [item] again
E	Edit
x	Exit
E	Explore
F	Fewer
F	File
F	Find
n	Find next
F	Font
F	Forward
H	Help
t	Help topics
H	Hide

Access key	Command
I	Insert
o	Insert object
I	Italic
L	Link here
x	Maximize
n	Minimize
M	More
M	Move
M	Move here
N	New
N	Next
N	No
O	Open
w	Open with
O	Options
u	Page setup
P	Paste
l	Paste link
s	Paste shortcut
s	Paste special
P	Pause
P	Play
P	Print
P	Print here
r	Properties
R	Redo

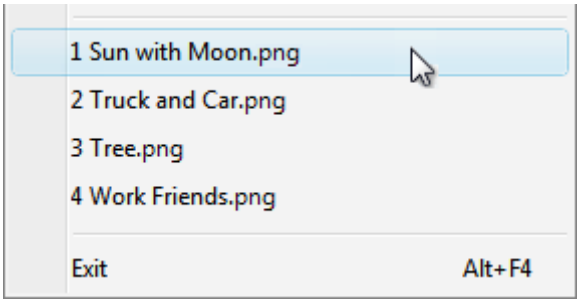
Access key	Command
R	Repeat
R	Restore
R	Resume
R	Retry
R	Run
S	Save
a	Save as
a	Select all
n	Send to
S	Show
S	Size
p	Split
S	Stop
T	Tools
U	Underline
U	Undo
V	View
W	Window
Y	Yes

- **Prefer characters with wide widths,** such as w, m, and capital letters.
- **Prefer a distinctive consonant or a vowel,** such as "x" in "Exit."
- **Avoid using characters that make the underline difficult to see,** such as (from most problematic to least problematic):
  - Characters that are only one pixel wide, such as i and l.
  - Characters with descenders, such as g, j, p, q, and y.
  - Characters next to a letter with a descender.

- When assigning access keys in wizard pages, remember to reserve "B" for Back and "N" for Next.
- When assigning access keys in property pages, remember to reserve "A" for Apply, if used.

Menu access keys

- **Assign access keys to all menu items.** No exceptions.
- **For dynamic menu items (such as recently used files), assign access keys numerically.**

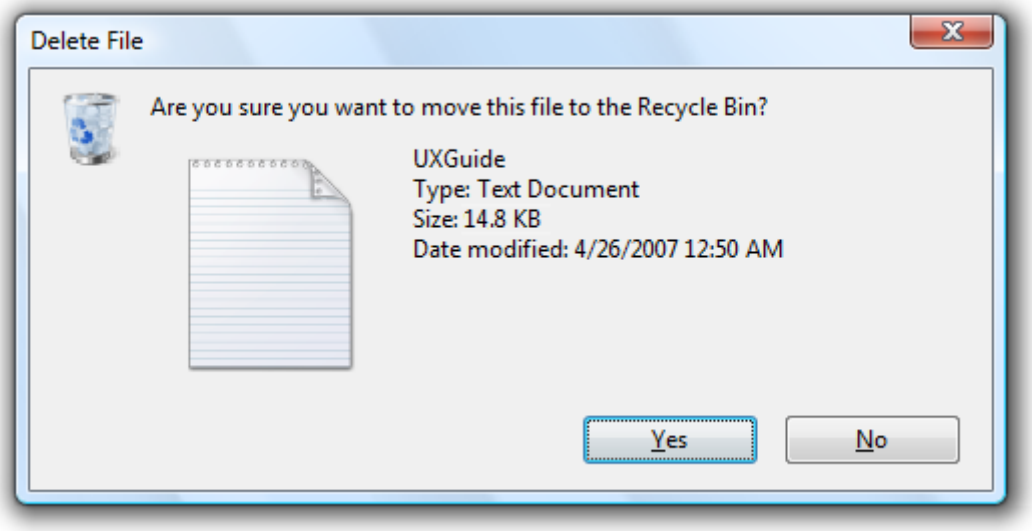
  

  In this example, the Paint program in Windows assigns numeric access keys to recently used files.

- **Assign unique access keys within a menu level.** You can reuse access keys across different menu levels.
- **Make access keys easy to find:**
  - For the most frequently used menu items, choose characters at the beginning of the first or second word of the label, preferably the first character.
  - For less frequently used menu items, choose letters that are a distinctive consonant or a vowel in the label.

Dialog box access keys

- **Whenever possible, assign unique access keys to all interactive controls or their labels.**Read-only text boxes are interactive controls (because users can scroll them and copy text), so they benefit from access keys. **Don't assign access keys to:**
  - **OK, Cancel, and Close buttons.** Enter and Esc are used for their access keys. However, always assign an access key to a control that means OK or Cancel, but has a different label.



In this example, the positive commit button has an access key assigned.

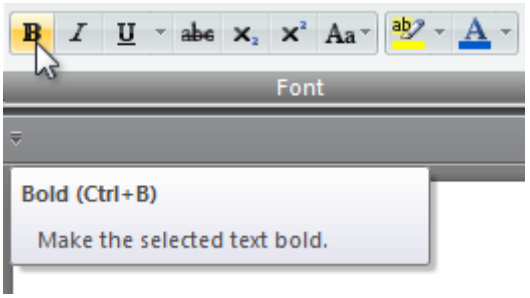
- **Group labels.** Normally, the individual controls within a group are assigned access keys, so the group label doesn't need one. However, assign an access key to the group label and not the individual controls if there is a shortage of access keys.
- **Generic Help buttons,** which are accessed with F1.
- **Link labels.** There are often too many links to assign unique access keys, and link underscores hide the access key underscores. Have users access links with the Tab key instead.
- **Tab names.** Tabs are cycled using Ctrl+Tab and Ctrl+Shift+Tab.
- **Browse buttons labeled "...".** These can't be assigned access keys uniquely.
- **Unlabeled controls,** such as spin controls, graphic command buttons, and unlabeled progressive disclosure controls.
- **Non-label static text or labels for controls that aren't interactive,** such as progress bars.

- **Assign commit button access keys first to ensure that they have the standard key assignments.** If there isn't a standard key assignment, use the first letter of the first word. For example, the access key for Yes and No commit buttons should always be "Y" and "N", regardless of the other controls in the dialog box.
- **For negative commit buttons (other than Cancel) phrased as a "Don't", assign the access key to the "n" in "Don't".** If not phrased as a "Don't", use the standard access key assignment or assign the first letter of the first word. By doing so, all Don'ts and No's have a consistent access key.

- **To make access keys easy to find, assign the access keys to a character that appears early in the label,** ideally the first character, even if there is a keyword that appears later in the label.
- **Assign at most 20 access keys,** so you have a few unassigned characters to facilitate localization.
- **If there are too many interactive controls to assign unique access keys, you may assign non-unique access keys** if:
  - The controls would otherwise be too difficult to navigate to.
  - The non-unique access keys don't conflict with the access keys of commonly used controls.
- **Don't use menu bars in dialog boxes.** It's difficult to assign unique access keys in this case, because the dialog box controls and menu items share the same characters.

Shortcut keys

- **Assign shortcut keys to the most commonly used commands.** Infrequently used programs and features don't need shortcut keys because users can use access keys instead.
- **Don't make a shortcut key the only way to perform a task.** Users should also be able to use the mouse or the keyboard with Tab, arrow, and access keys.
- **Don't assign different meanings to well-known shortcut keys.** Because they are memorized, inconsistent meanings for well-known shortcuts are frustrating and error prone.
- **Don't try to assign system-wide program shortcut keys.** Your program's shortcut keys will have effect only when your program has input focus.
- **Document all shortcut keys.** Document shortcuts in menu bar items, toolbar tooltips, and a single Help article that documents all shortcut keys used. Doing so helps users learn the shortcut key assignments they shouldn't be a secret.
  - **Exception:** Don't display shortcut key assignments within context menus. Context menus don't display the shortcut key assignments because these menus are optimized for efficiency.



The shortcut key is documented in the tooltip.

- **If your program assigns many shortcut keys, provide the ability to customize the assignments.** Doing so allows users to reassign conflicting shortcut keys and migrate from other products. Most programs don't assign enough shortcut keys to need this feature.

Choosing shortcut keys

- **For well-known shortcut keys, use the standard assignments.**
- **For non-standard key assignments, use the following recommended shortcut keys for more frequently used commands.** These shortcut keys are recommended because they don't conflict with the well-known shortcuts and are easy to press.
  - Ctrl+G, J, K, L M, Q, R, or T
  - Ctrl+any number
  - F7, F8, F9, or F12
  - Shift+F2, F3, F4, F5, F7, F8, F9, F11, or F12
  - Alt+any function key except F4
- **Use the following recommended shortcut keys for less frequently used commands.** These shortcut keys don't have conflicts, but are harder to press often requiring two hands.
  - Ctrl+any function key except F4 and F6
  - Ctrl+Shift+any letter or number
- **Make frequently used shortcut keys easy to remember:**
  - Use letters instead of numbers or function keys.
  - Try to use a letter that is in the first word or most memorable character within the command's keywords.
- **Use function keys for commands that have a small-scale effect,** such as commands that apply to the selected object. For example, F2 renames the selected item.
- **Use Ctrl key combinations for commands that have a large-scale effect,** such as commands that apply to an entire document. For example, Ctrl+S saves the current document.

- **Use Shift key combinations for commands that extend or complement the actions of the standard shortcut key.** For example, the Alt+Tab shortcut key cycles through open primary windows, whereas Alt+Shift+Tab cycles in the reverse order. Similarly, F1 displays Help, whereas Shift+F1 displays context-sensitive Help.
- **When using arrow keys to move or resize an item, use Ctrl+arrow keys for more granular control.**

Choosing shortcut keys (what not to do)

- **Don't distinguish between key locations.** For example, Windows can distinguish between left and right Shift, Alt, Ctrl, Windows logo, and Application keys, as well as keys on the numeric keypad. Assigning behavior to only one key location is confusing and unexpected.
- **Don't use the Windows logo modifier key for program shortcut keys.** Windows logo key is reserved for Windows use. Even if a Windows logo key combination isn't being used by Windows now, it may be in the future.
- **Don't use the Application key as a shortcut key modifier.** Use Ctrl, Alt, and Shift instead.
- **Don't use shortcut keys used by Windows for program shortcut keys.** Doing so will conflict with the Windows system shortcut keys when your program has input focus.
- **Don't use Alt+alphanumeric key combinations for shortcut keys.** Such shortcut keys may conflict with access keys.
- **Don't use the following characters for shortcut keys:** @ $ {} [] \ ~ | ^ ' < >. These characters require different key combinations across languages or are locale specific.
- **Avoid complex key combinations,** such as three or more keys together (example: Ctrl+Alt+spacebar) or keys that are far apart on the keyboard (example: Ctrl+F5). Use simple shortcut keys for frequently used commands.
- **Don't use Ctrl+Alt combinations,** because Windows interprets this combination in some language versions as an AltGR key, which generates alphanumeric characters.

Keyboard and mouse combinations

- For links, use Shift+click to navigate using a new window and Ctrl+click to navigate using a new tab. This approach is consistent with Windows Internet Explorer .

Documentation

When referring to the keyboard:

- Use on-screen keyboard to refer to a keyboard representation on the screen that the user touches to input characters.
- Give keyboard combinations starting with the modifier key. Present modifier keys in the following order: Windows logo, Application, Ctrl, Alt, Shift. If the Numpad modifier is used, put that just before the key it modifies.
- Don't use all capital letters for keyboard keys. Instead, follow the capitalization used by standard keyboards, or lowercase if the key is not labeled on the keyboard.
  - For alphabetical key combinations, use an uppercase letter.
  - Spell out Page Up, Page Down, Print Screen, and Scroll Lock.
  - Spell out plus sign, minus sign, hyphen, period, and comma.
  - For arrow keys, use left arrow, right arrow, up arrow, and down arrow. Don't use graphic labels for the arrow keys.
  - Use Windows logo key and Application key to refer to the keys labeled with icons. Don't use graphic labels for these keys.

Correct:

spacebar, Tab, Enter, Page Up, Ctrl+Alt+Del, Alt+W, Ctrl+plus sign

Incorrect:

SPACEBAR, TAB, ENTER, PG UP, Ctrl+Alt+DEL, Alt+w, Ctrl++

- Indicate key combinations with a plus sign, without spaces.

Correct:

Ctrl+A, Shift+F5

Incorrect:

Ctrl-A, Shift + F5

- To show a key combination that includes punctuation that requires use of the Shift key, such as the question mark, add Shift to the combination and give the name or symbol of the shifted key. Using the name of the unshifted key, such as 4 rather than $, could be confusing to users or even wrong; for instance, the ? and / characters are not always shifted keys on every keyboard.

Correct:

Ctrl+Shift+?, Ctrl+Shift+*, Ctrl+Shift+comma

Incorrect:

Ctrl+Shift+/, Ctrl+?, Ctrl+Shift+8, Ctrl+*

- At first mention, use the and key with the key name if necessary for clarity for example, the F1 key. At all subsequent references, refer to the key only by its name for example, press F1.
- Refer specifically to access keys and shortcut keys in programming and other technical documentation. Don't use accelerator, mnemonic, or hot keys. Everywhere else use keyboard shortcut, especially in user documentation.

When referring to interaction:

- Use press, not depress, strike, hit, or type, when pressing and immediately releasing a key initiates an action within the program or navigates within a document or UI.
- Use type, not enter, to direct users to type text.
- Use use in situations when press might be confusing, such as when referring to a type of key such as the arrow keys or function keys. In such cases, press might make users think they need to press all the keys simultaneously.
- Use hold when pressing and holding a key, such as a modifier key.
- Don't use press as a synonym for click.

Examples:

- Type your name, and then press Enter.
- Press Ctrl+F, and then type the text you want to search for.
- To save your file, press Y.
- To move the insertion point, use the arrow keys.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Windows 7 Mouse and Pointers

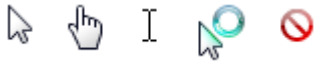
Article • 02/08/2022

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

The mouse is the primary input device used to interact with objects in Windows. Mouse functionality can also encompass other pointing devices, such as trackballs, touchpads and pointing sticks built into notebook computers, pens used with Windows Tablet and Touch Technology, and, on computers with touchscreens, even a user's finger.

> **Note**
>
> Guidelines related to accessibility, pen, and touch are presented in separate articles.

Physically moving the mouse moves the graphic pointer (also referred to as the cursor) on the screen. The pointer has a variety of shapes to indicate its current behavior.

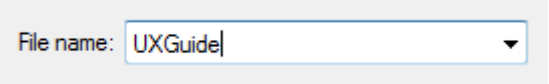


Typical mouse pointers

Mouse devices often have a primary button (usually the left button), a secondary button (usually the right), and a mouse wheel between the two. By positioning the pointer and clicking the primary and secondary buttons on the mouse, users can select objects and perform actions on them. For most interactions, pressing a mouse button while the cursor is over a target indicates the selected target, and releasing the button performs any action associated with the target.

All pointers, except the busy pointer, have a single pixel hot spot that defines the exact screen location of the mouse. The hot spot determines which object is affected by mouse actions. Objects define a hot zone, which is the area where the hot spot is considered to be over the object. Typically, the hot zone coincides with the borders of an object, but it may be larger to make user's intent easier to perform.

The caret is the flashing vertical bar that is displayed when the user is typing into a text box or other text editor. The caret is independent of the pointer (by default, Windows hides the pointer while the user is typing).



The caret

Design concepts

The mouse is intuitive

The mouse has been a successful input device because it is easy to use for the typical human hand. Pointer-based interaction has been successful because it is intuitive and allows for a rich variety of experiences.

Well-designed user interface (UI) objects are said to have affordance, which are visual and behavioral properties of an object that suggest how it is used. The pointer acts as a proxy for the hand, allowing users to interact with screen objects much like they would with physical objects. We humans have an innate understanding of how the human hand works, so if something looks like it can be pushed, we try to push it; if it looks like it can be grabbed, we try to grab it. Consequently, users can figure out how to use objects with strong affordance just by looking at them and trying them.

Buttons and sliders have strong affordance

By contrast, objects with poor affordance are harder to figure out. Such objects often require a label or instruction to explain them.

link text and icons have poor affordance

Some aspects of mouse use are not intuitive

Right-clicking, double-clicking, and clicking with Shift or Ctrl key modifiers are three mouse interactions that aren't intuitive, because they have no real world counterparts. Unlike keyboard shortcuts and access keys, these mouse interactions usually aren't documented anywhere in the UI. This suggests that right-click, double-click, and keyboard modifiers shouldn't be required to perform basic tasks, especially by novice users. It also suggests that these advanced interactions must have consistent, predictable behavior to be used effectively.

Single-click or double-click?

Double-clicking is used so extensively on the Windows desktop that it may not seem like an advanced interaction. For example, opening folders, programs, or documents in the file pane of Windows Explorer is performed by double-clicking. Opening a shortcut on the Windows desktop also uses double-clicking. By contrast, opening folders or programs in the Start menu requires a single click.

Selectable objects use single-click to perform selection, so they require a double-click to open, whereas non-selectable objects require only a single click to open. This distinction isn't understood by many users (clicking a program icon is clicking a program icon, right?) and as a result, some users just keep clicking on icons until they get what they want.

Direct manipulation

Interacting with objects directly is referred to as direct manipulation. Pointing, clicking, selecting, moving, resizing, splitting, scrolling, panning, and zooming are common direct manipulations. By contrast, interacting with an object through its properties window or other dialog box could be described as indirect manipulation.

However, where there is direct manipulation, there can be accidental manipulation and therefore the need for forgiveness. Forgiveness is the ability to reverse or correct an undesired action easily. You make direct manipulations forgiving by providing undo, giving good visual feedback, and allowing users to correct mistakes easily. Associated with forgiveness is preventing undesired actions from happening in the first place, which you can do by using constrained controls and confirmations for risky actions or commands that have unintended consequences.

Standard mouse button interactions

The standard mouse interactions depend on a variety of factors, including the mouse key clicked, the number of times it is clicked, its position during the clicks, and whether

any keyboard modifiers were pressed. Here is a summary of how these factors usually affect interaction:

- For most objects, left double-clicking performs a single left click and performs the default command. The default command is identified in the context menu.
- For some types of selectable objects, each click expands the effect of the click. For example, single-clicking in a text box sets the input location, double-clicking selects a word, and triple-clicking selects a sentence or paragraph.
- Right-clicking displays an object's context menu.
- Keeping the mouse still while pointing results in hovering.
- Keeping the mouse still while pressing the mouse buttons indicates clicking and single object selection. Moving the mouse indicates moving, resizing, splitting, dragging, and multiple object selection.
- The Shift key extends selection contiguously.
- The Ctrl key extends selection by toggling the selection state of the clicked item without affecting the selection of other objects.

Simple mouse interactions

The following table describes common mouse interactions and effects.

Expand table

Simple action	Interaction	Typical effect
Pointing	Position the pointer to a specific object without clicking any mouse buttons.	Target displays its hover state and any dynamic affordances.
Hovering	Position the pointer to a specific object without clicking any mouse buttons and without moving for at least a second.	Target displays its tooltip, infotip, or equivalent.
Clicking	Position the pointer to a specific, non-selectable object and press and release a mouse button without moving. Clicking takes effect on the mouse button release to allow users the opportunity to cancel the click by moving the mouse off the target. Therefore, mouse press only indicates the selected target.	For single clicks with the primary button, activate the object. For double-clicks with the primary button, activate the object and perform the default command. For the secondary button, display the object's context menu.
Selecting	Position the pointer to a specific, selectable object and press and release a mouse button.	For single clicks with the primary button, select the object. If the users drags the mouse, select a contiguous range of objects. For

Simple action	Interaction	Typical effect
		double-clicks with the primary button, select the object and perform the default command. For text, the right primary button click sets the insertion point, the second selects word at the insertion point, and the third click selects the sentence or paragraph.
Pressing	Position the pointer to a specific object and press a mouse button without releasing.	For auto-repeat functions (such as pressing a scroll arrow to continuously scroll), activate repeatedly. Otherwise indicates the start of a move, resize, split, or drag, unless followed by a release without moving.
Wheeling	Move mouse wheel.	Window scrolls vertically in direction of mouse wheel movement.

Pointer shapes

The following table describes common pointer shapes and usages.

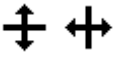
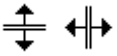
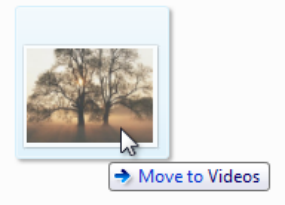
Expand table

Shape	Name	When used
	Normal select	Used for most objects.
	Link select	Used for text and graphics links because of their weak affordance.
	Text select	Used for text to indicate a location between characters.
	Precision select	Used for graphic and other two-dimensional interaction.

Compound mouse interactions

The following table describes common mouse interactions.

Expand table

Compound action	Interaction	Typical effect	Pointers
Moving	If moving is a mode (entered by giving a command), enter the mode, position the pointer over a movable object, press button and move mouse, release mouse button. in this case, the pointer changes shape to indicate the mode. otherwise, position the pointer over a movable object's grabber, press button and move mouse, release mouse button. in this case, the pointer doesn't need to change shape.	object moves in direction of pointer movement.	move used to move a window in any direction. pan Used to move an object within a window in any direction.
Resizing	Position the pointer over a resizable border or resize handle, press a mouse button and move mouse, and then release the mouse button.	object resizes in direction of pointer movement.	vertical and horizontal resize used to resize a single dimension. diagonal resize used to resize two dimensions simultaneously. row and column resize  Used to resize a row or column in a grid.
Splitting	Position the pointer over a splitter, press a mouse button and move mouse, and then release the mouse button.	split pane border moves in direction of pointer movement.	window splitters  Used to resize a split pane vertically or horizontally.
Dragging and dropping	Position the pointer over a valid object for dragging, press a mouse button and move mouse to a drop target, and then release the mouse button.	object is moved or copied to the drop target.	normal select  used over valid drag

Compound action	Interaction	Typical effect	Pointers
			targets. may also have an infotip to indicate specific effect. unavailable Used to indicate a surface isn't a valid drop target.

Activity indicators

The following table shows pointers that users see when performing an action that takes longer than a couple of seconds to complete.

Expand table

Shape	Name	When used
	Busy pointer	Used to wait for a window to become responsive.
	Working in background pointer	Used to point, click, press, or select while a task completes in the background.

Hand pointers

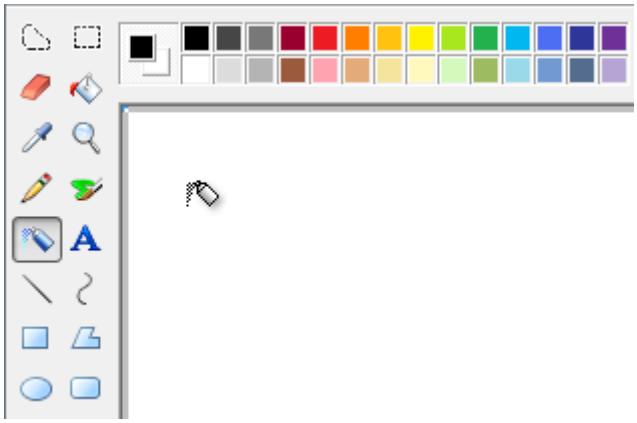
Text and graphics links use a hand or "link select" pointer (a hand with the index finger pointing) because of their weak affordance. While links may have other visual clues to indicate that they are links (such as underlines and special placement), displaying the hand pointer on hover is the definitive indication of a link.

To avoid confusion, it is imperative not to use the hand pointer for other purposes. For example, command buttons already have a strong affordance, so they don't need a hand pointer. The hand pointer must mean "this target is a link" and nothing else.

Custom pointers

Windows supports the creation of custom pointers. For more details see, Setting the Cursor Image and User Input: Extended Example.

Many applications provide a palette of controls with custom pointers to support application functionality.



Microsoft Paint includes a palette of different functions, each with a unique pointer

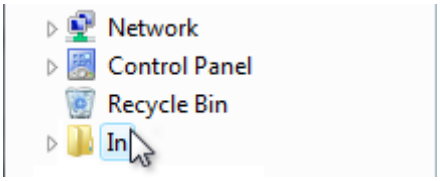
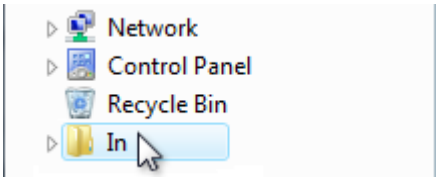
Fitts' Law

Fitts' Law is a well known principle in graphical user interface design ergonomics that essentially states:

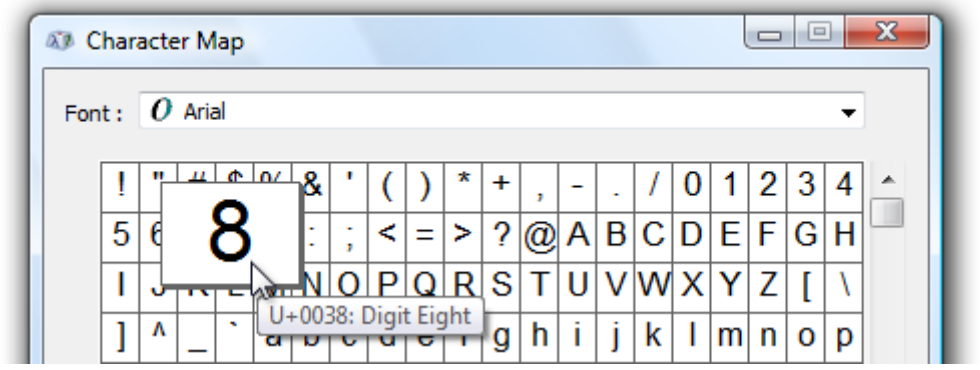
- The farther away a target is, the longer it takes to acquire it with the mouse.
- The smaller a target is, the longer it takes to acquire it with the mouse.

Thus, large targets are good. Be sure to make the entire target area clickable.

Expand table

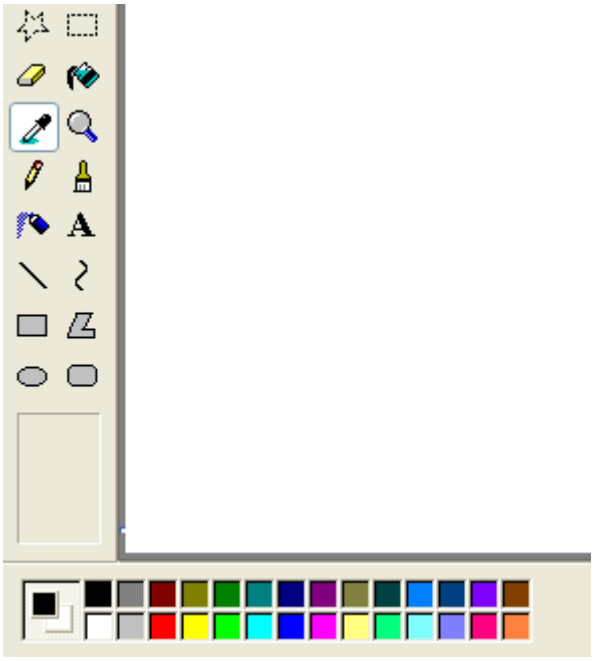
Incorrect	Correct (the entire target is clickable)
	

You can dynamically change the size of a target when pointing to make it easier to acquire.



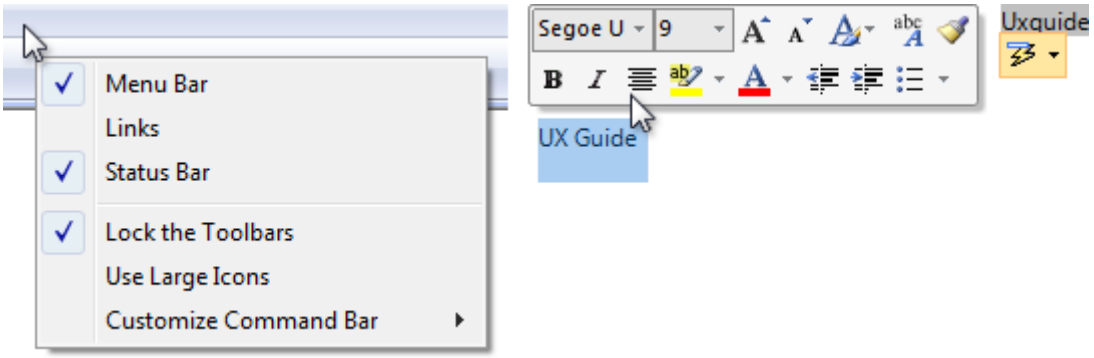
A target becomes larger when the user is pointing to make it easier to acquire

And close targets are also good. Locate clickable items close to where they are most likely going to be used. In the following image, the color palette is too far away from the tool selector.



The color palette is too far from where it is likely to be used

Consider the fact that the user's current pointer location is as close as a target can be, making it trivial to acquire. Thus, context menus take full advantage of Fitts' law, as do the mini toolbars used by Microsoft Office.



The current pointer location is always the easiest to acquire

Also, consider alternative input devices when determining object sizes. For example, the minimum target size recommended for touch is 23x23 pixels (13x13 DLUs).

Environments without a mouse

Not all Windows environments have a mouse. For example, kiosks rarely have a mouse and usually have a touchscreen instead. This means that users can perform simple interactions such as left-clicking and perhaps dragging-and-dropping. However, they can't hover, right-click, or double-click. This situation is easy to design for because these limitations are usually known in advance.

Using a mouse requires fine motor skills, and as a result, not all users can use a mouse. To make your software accessible to the broadest audience, make sure all interactions for which fine motor skills aren't essential can be performed using the keyboard instead.

For more information and guidelines, see Accessibility.

If you do only four things...

1. Give mouse interactions behaviors consistent with their standard effects, using the standard pointers whenever appropriate.
2. Limit advanced mouse interactions (those requiring right clicks, multiple clicks, or modifier keys) to advanced tasks targeted at advanced users.
3. Assign advanced mouse interactions consistent, predictable behaviors so that they can be used effectively.
4. Make sure your program provides the ability to reverse or correct any undesired actions especially for destructive commands. Accidental actions are more likely when using direct manipulation.

Guidelines

Click affordance

- **Never require users to click an object to determine if it is clickable.** Users must be able to determine clickability by visual inspection alone.
  - Primary UI (such as commit buttons) must have a static click affordance. Users shouldn't have to hover to discover primary UI.
  - Secondary UI (such as secondary commands or progressive disclosure controls) can display their click affordance on hover.

    - Text links should statically suggest link text, then display their click affordance (underline or other presentation change, with hand pointer) on hover.
    - Graphics links only display a hand pointer on hover.
- **Use the hand (or "link select") pointer only for text and graphic links.** Otherwise, users would have to click on objects to determine if they are links.

Standard mouse button interactions

The following table summarizes the mouse button interactions that apply in most cases:

Expand table

Interaction	Effect
Hover	Target displays its tooltip, infotip, or equivalent.
Single left-click	Activates or selects the object. For text, sets the insertion point.
Single right-click	Selects the object and displays its context menu.
Double left-click	Activates or selects the object, and performs the default command. For text, selects word at the insertion point (a third click selects the sentence or paragraph).
Double right-click	Same as single right-click.
Shift single left-click	For selectable objects, contiguously extends the selection. Otherwise, same as single left-click with possible modifications. For example, in Paint, drawing an oval with the Shift key modifier results in drawing a circle.
Shift single right-click	Same as Shift single left-click.
Shift double left-click	Same as Shift single left-click, and performs the default command on the entire selection.
Shift double right-click	Same as Shift single left-click.
Ctrl single left-click	For selectable objects, extends the selection by toggling the selection state of the clicked item without affecting the selection of other objects (therefore allowing selection that isn't contiguous). Otherwise, same as single left-click.
Ctrl single right-click	Same as Ctrl single left-click.

Interaction	Effect
Ctrl double left-click	Same as Ctrl single left-click, and performs the default command on the entire selection.
Ctrl double right-click	Same as Ctrl single left-click.

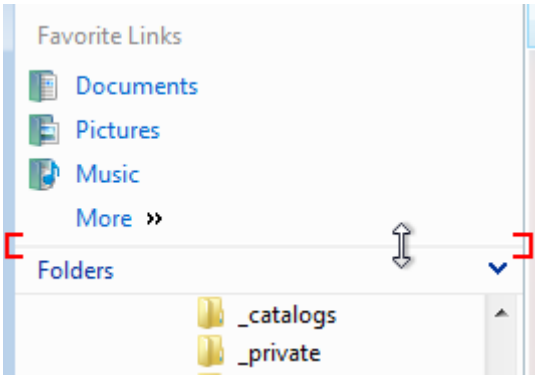
Mouse interaction

- **Make click targets at least 16x16 pixels so that they can be easily clicked by any input device.** For touch, the recommended minimum control size is 23x23 pixels (13x13 DLUs). Consider dynamically changing the size of small targets when the user is pointing to make them easier to acquire.

  In this example, the spin control buttons are too small to be used effectively with touch or a pen.

- **Make splitters at least five pixels wide so that they can be easily clicked by any input device.** Consider dynamically changing the size of small targets when the user is pointing to make them easier to acquire.

  In this example, the splitter in the Windows Explorer navigation pane is too narrow to be used effectively with a mouse or pen.

- **Provide users a margin of error spatially.** Allow for some mouse movement (for example, three pixels) when users release a mouse button. Users sometimes move the mouse slightly as they release the mouse button, so the mouse position just before button release better reflects the user's intention than the position just after.

- **Provide users a margin of error temporally.** Use the system double-click speed to distinguish between single and double clicks.

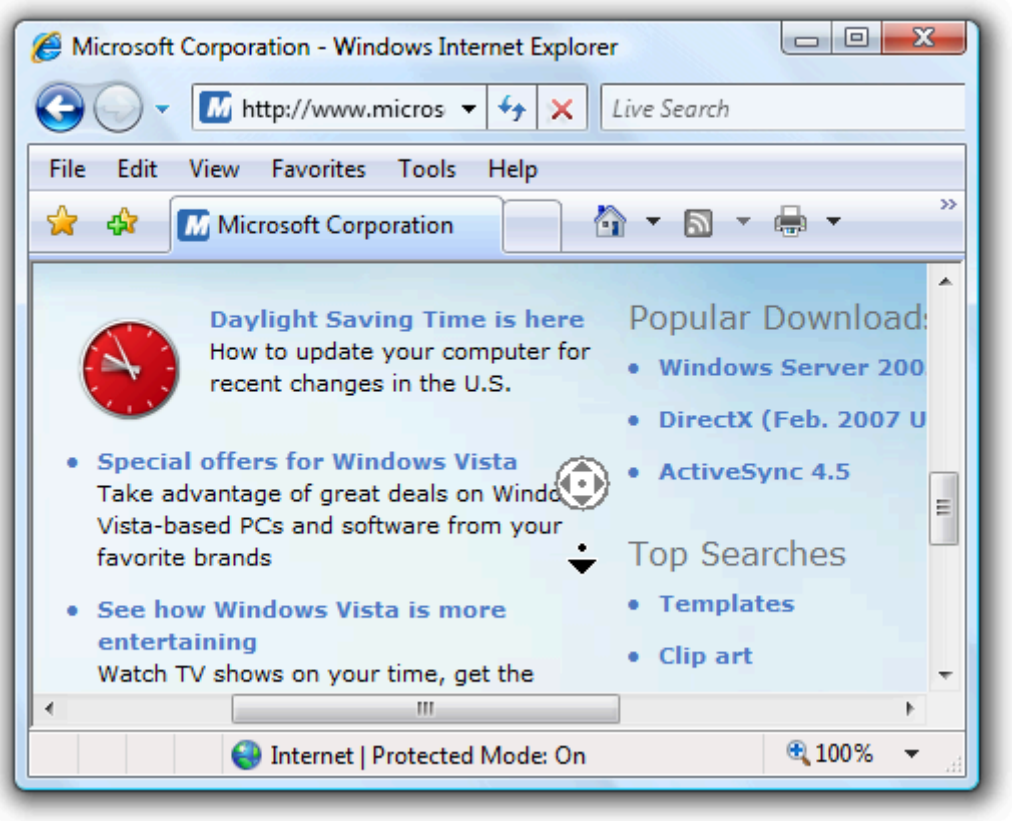
- **Have clicks take effect on mouse button up.** Allow users to abandon mouse actions by removing the mouse from valid targets before releasing the mouse button. For most mouse interactions, pressing a mouse button only indicates the selected target and releasing the button activates the action. Auto-repeat functions (such as pressing a scroll arrow to continuously scroll) are an exception.
- Capture the mouse for selecting, moving, resizing, splitting, and dragging.
- Use the Esc key to let users abandon compound mouse interactions such as moving, resizing, splitting, and dragging.
- **If an object doesn't support double clicks but users are likely to assume it does, interpret a "double click" as one single click.** Assume the user intended a single action instead of two.

  Because users are likely to assume that taskbar buttons support double clicks, a "double click" should be handled as a single click.

- **Ignore redundant mouse clicks while your program is inactive.** For example, if the user clicks a button 10 times while a program is inactive, interpret that as a single click.
- **Don't use double drags or chords.** A double drag is a drag action commenced with a double-click, and a chord is when multiple mouse buttons are pressed simultaneously. These interactions aren't standard, aren't discoverable, are difficult to perform, and are most likely performed accidentally.
- **Don't use Alt as a modifier for mouse interactions.** The Alt key is reserved for toolbar access and access keys.
- **Don't use Shift+Ctrl as a modifier for mouse interactions.** Doing so would be too difficult to use.
- **Make hover redundant.** To make your program touchable, take full advantage of hover but only in ways that are not required to perform an action. This usually means that an action can also be performed by clicking, but not necessarily in exactly the same way. Hover isn't supported by most touch technologies, so users with such touchscreens can't perform any tasks that require hovering.

Mouse wheel

- **Make the mouse wheel affect the control, pane, or window that the pointer is currently over.** Doing so avoids unintended results.
- **Make the mouse wheel take effect without clicking or having input focus.** Hovering is sufficient.
- **Make the mouse wheel affect the object with the most specific scope.** For example, if the pointer is over a scrollable list box control in a scrollable pane within a scrollable window, the mouse wheel affects the list box control.
- **Don't change the input focus when using the mouse wheel.**
- Give the mouse wheel the following effects:
  - For scrollable windows, panes, and controls:
    - **Rotating the mouse wheel scrolls the object vertically, where rotating up scrolls up.** For the wheel to have natural mapping, rotating the mouse wheel should never scroll horizontally because doing so is disorienting and unexpected.
      - **If the Ctrl key is pressed, rotating the mouse wheel zooms the object,** where rotating up zooms in and rotating down zooms out.
      - **Tilting the mouse wheel scrolls the object horizontally.**
  - For zoomable windows and panes (without scrollbars):
    - **Rotating the mouse wheel zooms the object,** where rotating up zooms in and rotating down zooms out.
    - Tilting the mouse wheel has no effect.
  - For tabs:
    - **Rotating the mouse wheel can change the current tab,** regardless of the orientation of the tabs.
    - Tilting the mouse wheel has no effect.
  - If the Shift and Alt keys are depressed, the mouse wheel has no effect.
- **Use the Windows system settings for the vertical scroll size (for rotating) and horizontal scroll size (for tilting).** These settings are configurable through the Mouse control panel item.
- **Make rotating the mouse wheel more rapidly result in scrolling more rapidly.** Doing so allows users to scroll large documents more efficiently.
- **For scrollable windows, consider having clicking the mouse wheel button put the window in "reader mode."** Reader mode plants a special scroll origin icon and scrolls the window in a direction and speed relative to the scroll origin.



Internet Explorer supports reader mode, which features the scroll-origin icon

Hiding the pointer

- **Don't hide the pointer.** Exceptions:
  - Presentation applications running in full screen presentation mode may hide the pointer. However, the pointer must be restored immediately when users move the mouse, and can be rehidden after two seconds of inactivity.
  - Environments without a mouse (such as kiosks) can permanently hide the pointer.
- By default, Windows hides the pointer while the user is typing in a text box. This Windows system setting is configurable through the Mouse control panel item.

Activity pointers

The activity pointers in Windows are the busy pointer ( ) and the working in background pointer ( ).

- Display the busy pointer when users have to wait more than one second for an action to complete. Note that the busy pointer has no hot spot, so users can't click anything while it is displayed.
- Display the working in background pointer when users have to wait more than one second for an action to complete, but the program is responsive and there is no

other visual feedback that the action isn't complete.
- Don't combine activity pointers with progress bars or progress animations.

Caret

- **Don't display the caret until the text input window or control has input focus.** The caret suggests input focus to users, but a window or control can display the caret without input focus. Of course, don't steal input focus so that an out-of-context dialog box can display the caret.

  The Windows Credential Manager is displayed out of context with the caret but without input focus. As a result, users end up typing their password in unexpected places.

- **Place the caret where users are most likely to type first.** Usually this is either the last place the user was typing or at the end of the text.

Accessibility

- For users who can't use the mouse at all, make the mouse redundant with the keyboard.
  - Users should be able to do everything with the keyboard that they can with the mouse, except actions for which fine motor skills are essential, such as drawing and game playing.
  - Users should be able to do everything with the mouse that they can with the keyboard, except efficient text entry.
- For users with limited ability to use the mouse:

    - Don't make double-clicking and dragging the only way to perform an action.

For more information and guidelines, see Accessibility.

Documentation

When referring to the mouse:

- Avoid using the plural mice; if you need to refer to more than one mouse, use mouse devices.
- Use mouse button to indicate the left mouse button. Don't use primary mouse button. Similarly, use right mouse button instead of secondary mouse button. Regardless of accuracy, users understand these terms and users who reprogram their buttons make the mental shift.
- Use wheel for the rotating part of the mouse wheel, and wheel button to refer to the clickable part.
- Use verbs such as click, point, and drag to refer to mouse actions. Users rotate the wheel vertically, tilt it horizontally, and click the wheel button.
- Use drag, not drag and drop, for the action of moving a document or folder. It is acceptable to use drag-and-drop as an adjective, as in "moving the folder is a drag-and-drop operation."
- Always hyphenate double-click and right-click as verbs.
- Use click, not click on. Click in (as in "click in the window") is acceptable.

When referring to mouse pointers:

- Refer to the mouse pointer as the pointer. Use cursor only in technical documentation.
- For pointers with activity indicators, use busy pointer for the pointer consisting of only an activity indicator, and working in background pointer for the combination pointer and activity indicator.
- For the other types of pointers, don't use descriptive labels to refer to the pointer. If necessary, use a graphic to describe how the mouse pointer can appear on the screen.

Examples:

- Point to the window border.
- Using the mouse, click the **Minimize** button.
- Hold down Shift and click the right mouse button.
- When the pointer becomes a , drag the pointer to move the split line.

See also

- Accessibility interaction guidelines
- Touch interaction guidelines
- Pen interaction guidelines
- Text links
- Graphics links
- Capture the mouse
- Setting the Cursor Image
- User Input: Extended Example

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Pen

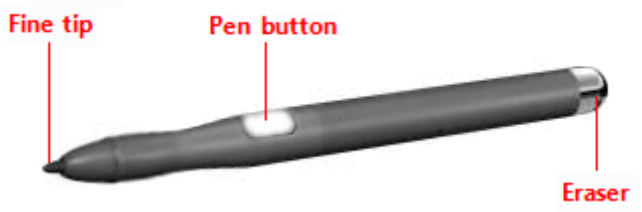
Article • 02/09/2021

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

All Microsoft Windows applications should be pen enabled. And doing so is easier than you think.

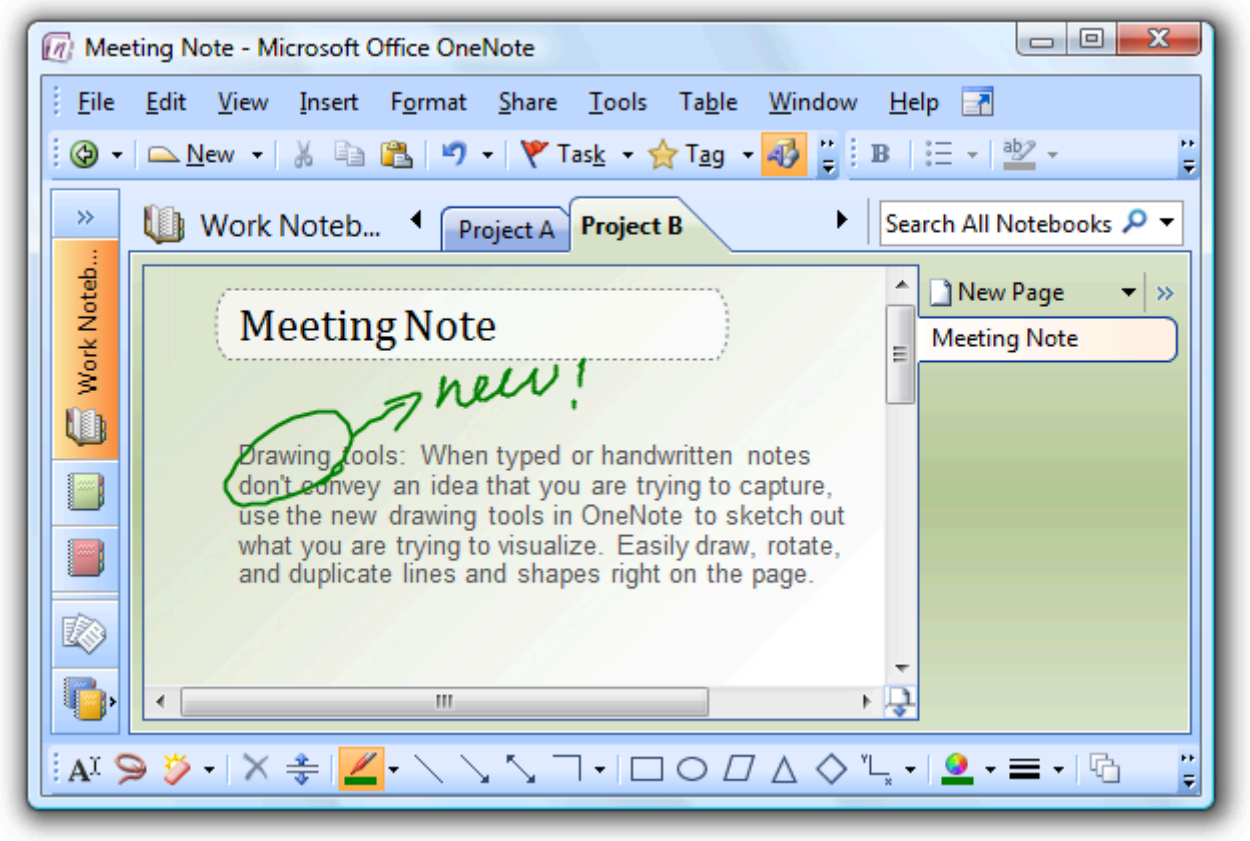
Pen input refers to the way Windows lets you interact directly with a computer using a pen. A pen can be used for pointing and also for gestures, simple text entry, and capturing free-form thoughts in digital ink.

The pen used for input has a fine, smooth tip that supports precise pointing, writing, or drawing in ink. The pen may also have an optional pen button (used to perform right-clicks) and eraser (used to erase ink). Most pens support hover.



A typical pen.

When the pen is used for handwriting, the user's strokes can be converted to text using handwriting recognition. The strokes can be kept just as they were written, with recognition performed in the background to support searching and copying as text. Such unconverted strokes are called digital ink.



An example of ink input.

Most Windows programs are already pen-friendly in that a pen can be used instead of a mouse, the pen works smoothly for most important tasks and interactions, and the program responds to gestures. A program becomes handwriting-friendly when it assists with handwritten text input. A program becomes ink-enabled when it can handle ink directly, instead of requiring that pen strokes be translated to text or equivalent mouse movements. This allows users to write, draw, and add comments in free-flowing, high-quality digital ink. Collecting ink is different than collecting mouse events, because ink requires higher resolution and a higher sample rate, and it can also add nuance with pressure and tilt. For information about creating handwriting friendly and ink-enabled programs, see Integrating Ink and Text Input Using the Pen.

When positioning a pen, there is less need for a cursor because the tip represents itself. However, for targeting assistance, Windows provides a tiny pen cursor that indicates the current pen location. Unlike the mouse pointer it replaces, the pen cursor is not needed unless the pen is near the display, so it disappears after a few seconds of inactivity to allow an unobstructed view of information.

Most pen-friendly programs support gestures. A gesture is a quick movement of the pen on a screen that the computer interprets as a command, rather than as a mouse movement, writing, or drawing. One of the quickest and easiest gestures to perform is a flick. A flick is a simple gesture that results in navigation or an editing command.

Navigational flicks include drag up, drag down, move back, and move forward, whereas editing flicks include copy, paste, undo, and delete.

All pointers except the busy pointer have a single pixel hot spot that defines the exact screen location of the pointer. The hot spot determines which object is affected by the interaction. Objects define a hot zone, which is the area where the hot spot is considered to be over the object. Typically, the hot zone coincides with the borders of an object, but it may be larger to make interaction easier.

Because a pen can point more precisely than a finger, if your user interface works well for touch it will also work well for a pen. Consequently, this article is primarily focused on adding pen support to programs that have already been designed for touch.

Note: Guidelines related to mouse, accessibility, and touch are presented in separate articles.

Design concepts

Using a pen for input has the following characteristics:

- **Natural and intuitive.** Everyone knows how to point and tap with a pen. Object interactions are designed to correspond to how users interact with objects in the real world in a consistent manner.
- **Expressive.** Strokes of a pen are easy to control, making writing, drawing, sketching, painting, and annotating easier than doing so with a mouse.
- **More personal.** Just as a handwritten note or signature is more personal than a typed one, using a digitally handwritten note or signature is also more personal.
- **Less intrusive.** Using a pen is silent, and consequently much less distracting than typing or clicking, especially in social situations such as meetings.
- **Portable.** A computer with a pen capability can be more compact because most tasks can be completed without a keyboard, mouse, or touchpad. It can be more flexible because it doesn't require a work surface. It enables new places and scenarios for using a computer.
- **Direct and engaging.** Using a pen makes you feel like you are directly interacting with the objects on the screen, whereas using a mouse or touchpad always requires you to coordinate hand movements with separate on-screen pointer movements which feels indirect by comparison.

All Windows programs should have a good pen experience. Users should be able to perform your program's most important tasks efficiently using a pen. Some tasks, like typing or detailed pixel manipulation, aren't appropriate for a pen, but they should at least be possible.

Fortunately, if your program is already well designed and is touch-friendly, providing good pen support is easy to do. For this purpose, a well-designed program:

- **Has good mouse support.** The interactive controls have clear, visible affordances, and have hover states for pointer feedback. Objects have standard behaviors for the standard mouse interactions (single and double left-click, right-click, drag, and hover). The pointer shape changes as appropriate to indicate the type of direct manipulation.
- **Has good keyboard support.** The program makes users efficient by providing standard shortcut key assignments, especially for navigation and editing commands that can also be generated through gestures.
- **Has controls large enough for touch.** The controls have a minimum size of 23x23 pixels (13x13 dialog units [DLUs]), and the most commonly used controls are at least 40x40 pixels (23x22 DLUs). To avoid unresponsive behavior, there should be no small gaps between targets the UI elements should be spaced so that adjacent targets are either touching or have at least 5 pixels (3 DLUs) of space between them.
- **Is accessible.** Uses Microsoft Active Accessibility (MSAA) to provide programmatic access to the UI for assistive technologies. The program appropriately responds to theme and system metric changes.
- **Works well and looks good in 120 dpi (dots per inch),** which is the recommended default display resolution for pen-enabled computers.
- **Uses common controls.** Most common controls are designed to support a good pen experience. If necessary, the program uses well-implemented custom controls that are designed to support easy targeting and interactive manipulation.
- **Uses constrained controls.** When designed for easy targeting, constrained controls like lists and sliders can be better than unconstrained controls like text boxes, because they reduce the need for text input.
- **Provides appropriate default values.** The program selects the safest (to prevent loss of data or system access) and most secure option by default. If safety and security aren't factors, the program selects the most likely or convenient option, thereby eliminating unnecessary interaction.
- **Provides text auto completion.** Provides a list of most likely or recently input values to make text input much easier.

Unfortunately, the converse is also true if your program isn't well designed, its shortcomings are going to be especially obvious to users who use a pen.

Model for pen interaction

If you aren't experienced with using a pen, the best introduction is to learn by doing. Get a pen-enabled computer, put the mouse and keyboard aside, and do the tasks that you normally do using just a pen. Be sure to try both ink-enabled programs, such as Windows Journal, and programs that aren't ink-enabled. If you have a Tablet PC, experiment with holding it in different positions, such as on your lap, lying flat on a table, or in your arms while you're standing. Try using it in portrait and landscape orientation, and holding the pen for writing and just for pointing, in your left hand as well as your right.

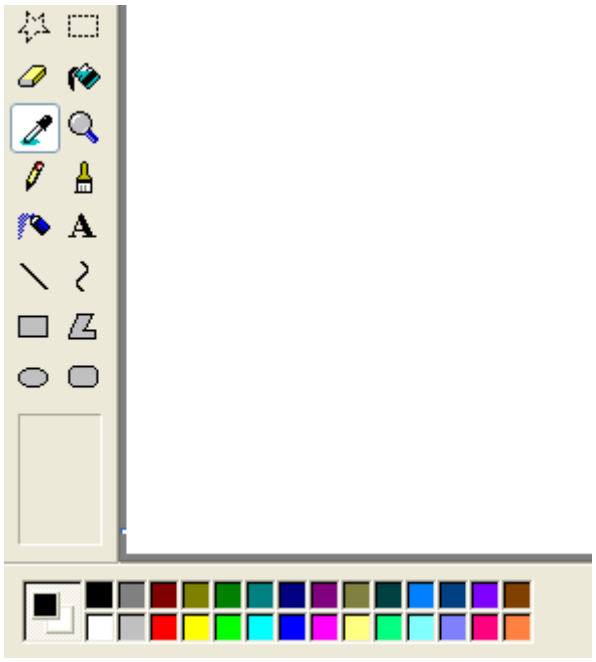
As you experiment with using a pen, you'll discover that:

- **Small controls are difficult to use.** The size of the controls greatly affects your ability to interact effectively. Controls that are 10x10 pixels work reasonably for a pen, but larger controls are even more comfortable to use. For example, spin controls (15x11 pixels) are too small to use with a pen easily.
- **Handedness is a factor.** Your hand sometimes covers things that you might want to see or interact with. For example, for right-handed users context menus are hard to use if they appear to the right of the click point, so it's better if they appear on the left. Windows allows users to indicate their handedness in the Tablet PC Settings control panel item.
- **Task locality helps.** While you can move the pointer across a 14-inch screen with a 3-inch mouse movement, using a pen requires you to move your hand the full 14 inches. Repeatedly moving between targets that are far apart can be tedious, so it's much better to keep task interactions within the range of a resting hand whenever possible. Context menus are convenient because they require no hand movement.
- **Text input and selection are difficult.** Lengthy text input is especially difficult using a pen, so auto-completion and acceptable default text values can really simplify tasks. Text selection can also be quite difficult, so tasks are easier when they don't require precise cursor placement.
- **Small targets near the edge of the display can be very difficult to tap.** Some display bezels protrude, and some touchscreen technologies are less sensitive at the edges, making controls near the edge harder to use. For example, the Minimize, Maximize/Restore, and Close buttons on the title bar can be harder to use when a window is maximized.

Control location

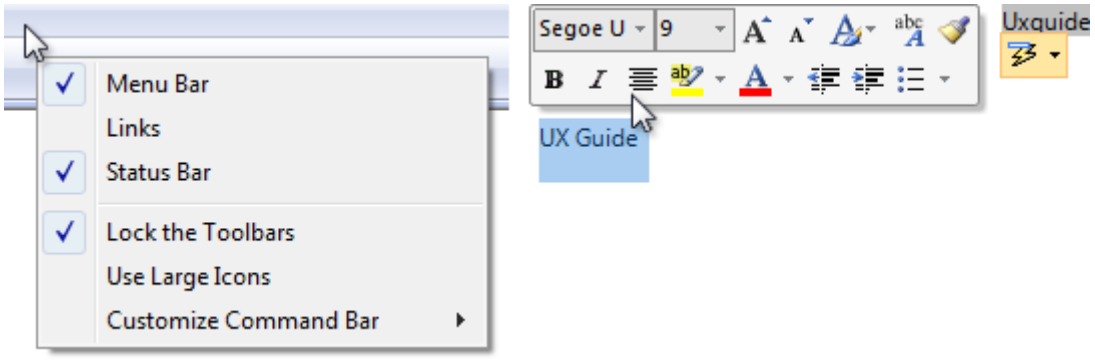
Task locality reduces tedious repeating cross-screen movements. To minimize hand movements, locate controls close to where they are most likely going to be used.

Incorrect:



In this example from Windows XP, the color palette is too far from where it is likely to be used.

Consider that the user's current location is the closest a target can be, making it trivial to acquire. Thus, context menus take full advantage of Fitts' Law, as do the mini-toolbars used by Microsoft Office.



The current pointer location is always the easiest to acquire.

Small targets near the display edge can be difficult to target, so avoid placing small controls near window edges. To ensure that controls are easy to target when a window is maximized, either make them at least 23x23 pixels (13x13 DLUs), or place them away from the window edge.

Pen interactions

System gestures

System gestures are defined and handled by Windows. As a result, all Windows programs have access to them. These gestures have equivalent mouse, keyboard, and

application command messages:

Expand table

System gesture	Synthesized equivalent message
Hover (when supported)	Mouse hover
Tap (down and up)	Mouse left-click
Double tap (down and up twice)	Mouse double left-click
Press and hold (down, pause, up)	Mouse right-click
Drag (down, move, up)	Mouse left-drag
Press, hold, and drag (down, pause, move, up)	Mouse right-drag
Select (down, move over selectable objects, up)	Mouse select

Developers: For more information, see SystemGesture Enumeration.

Flicks

Flicks are simple gestures that are roughly the equivalent of keyboard shortcuts. Navigational flicks include drag up, drag down, move back, and move forward. Editing flicks include copy, paste, undo, and delete. To use flicks, your program only needs to respond to the related keystrokes commands.

The eight flick gestures and their default assignments in Windows 7. The navigation flicks were changed to correspond to panning (where the object moves with the gesture) instead of scrolling (where the object moves in the opposite direction of the gesture).

The eight flick gestures and their default assignments in Windows Vista.

The navigational flicks have natural mapping, so they are easy to learn and remember. The editing flicks are diagonals that require more precision and their mappings are not as natural (flick towards the Recycle Bin to delete, flick in the Back arrow direction to undo), so these aren't enabled by default. All flick actions can be customized using the Pen and Input Devices control panel item.

Expand table

Flick	Synthesized equivalent message
Flick left	Forward command (Back command for Windows Vista)
Flick right	Back command (Forward command for Windows Vista)
Flick up	Keyboard Scroll Down
Flick down	Keyboard Scroll Up
Flick up-left diagonal	Keyboard Delete
Flick down-left diagonal	Keyboard Undo
Flick up-right diagonal	Keyboard Copy
Flick down-right diagonal	Keyboard Paste

Application gestures

Applications can define and handle other gestures as well. The Microsoft Gesture Recognizer can recognize over 40 gestures. To use application gestures, your program must define the gestures it recognizes, and then handle the resulting events.

Responsiveness and consistency

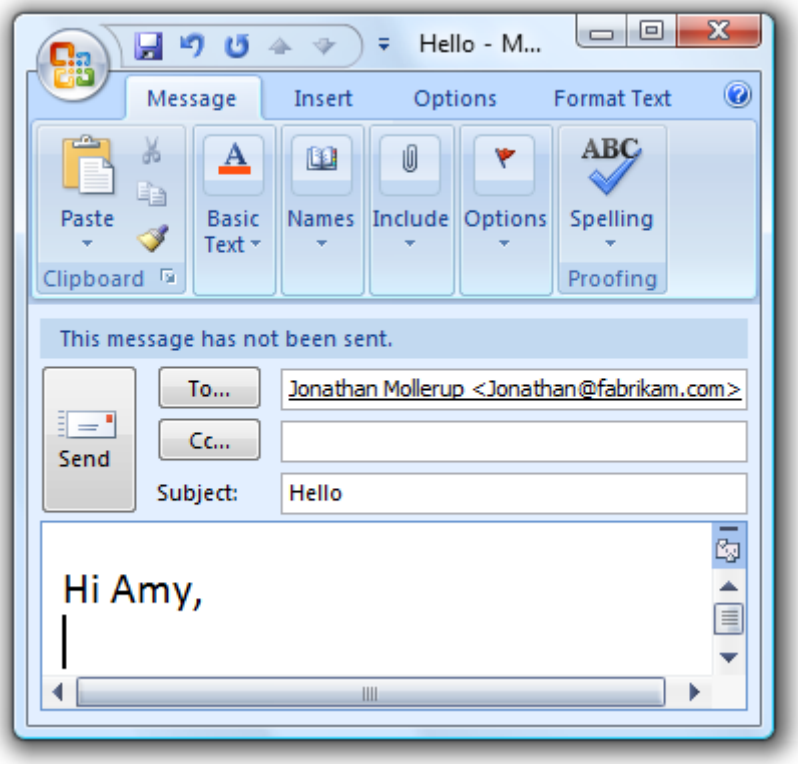
Responsiveness is essential for creating pen experiences that feel direct and engaging. To feel direct, gestures must take effect immediately, and an object's contact points must stay under the pen smoothly throughout the gesture. Any lag, choppy response, loss of contact, or inaccurate results destroys the perception of direct manipulation and also of quality.

Consistency is essential for creating pen experiences that feel natural and intuitive. Once users learn a standard gesture, they expect that gesture to have the same effect across all applicable programs. To avoid confusion and frustration, never assign non-standard meanings to standard gestures. Instead, use custom gestures for interactions unique to your program.

Editing ink and text

Editing ink and text are among the most challenging interactions when using a pen. Using constrained controls, appropriate default values, and auto-completion eliminates or reduces the need to input text. But if your program involves editing text or ink, **you can make users more productive by automatically zooming input UI up to 150 percent by default when a pen is used.**

For example, an e-mail program could display UI at normal size, but zoom the input UI to 150 percent to compose messages.



In this example, the input UI is zoomed to 150 percent.

If you do only four things...

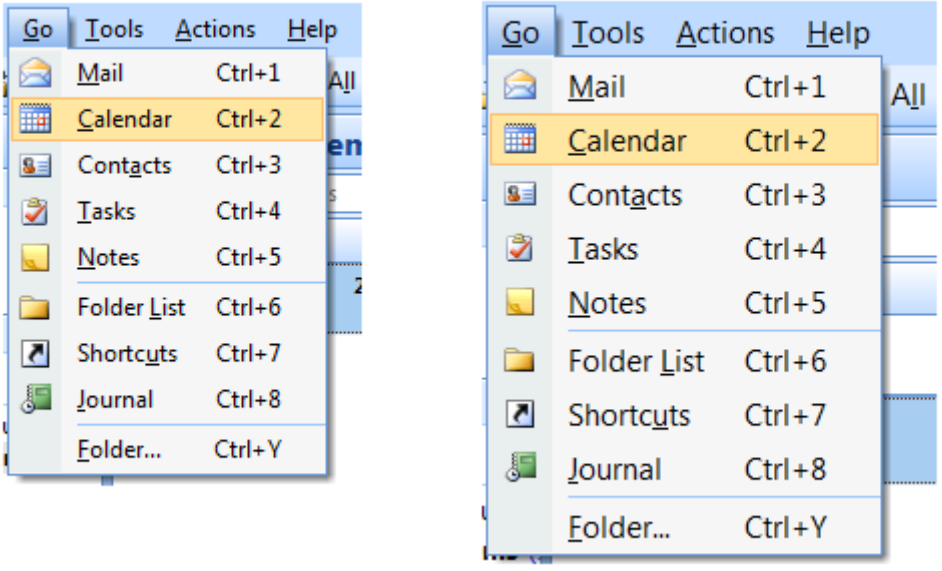
1. a. Make your Windows programs have a good pen experience! Users should be able to perform your program's most important tasks efficiently using a pen (at least those tasks that don't involve a lot of typing or detailed pixel manipulation).
2. b. Consider adding support for writing, drawing, and adding comments directly using ink in the most relevant scenarios.
3. c. To create a direct and engaging experience, have gestures take effect immediately, keep contact points under the user's pen smoothly throughout the gesture, and have the effect of the gesture map directly to the user's motion.
4. d. To create a natural and intuitive experience, support appropriate standard gestures and assign them their standard meanings. Use custom gestures for

interactions unique to your program.

Guidelines

Control usage

- **Prefer using common controls.** Most common controls are designed to support a good pen experience.
- **Prefer constrained controls.** Use constrained controls like lists and sliders whenever possible, instead of unconstrained controls like text boxes, to reduce the need for text input.
- **Provide appropriate default values.** Select the safest (to prevent loss of data or system access) and most secure option by default. If safety and security aren't factors, select the most likely or convenient option, thereby eliminating unnecessary interaction.
- **Provide text auto-completion.** Provide a list of most likely or recently input values to make text input much easier.
- **For important tasks that use multiple selection, if a standard multiple-selection list is normally used, provide an option to use a check box list instead.**
- **Respect system metrics.** Use system metrics for all sizes don't hardwire sizes. If necessary, users can change the system metrics or dpi to accommodate their needs. However, treat this as a last resort because users shouldn't normally have to adjust system settings to make UI usable.



In this example, the system metric for menu height was changed.

Control sizing, layout, and spacing

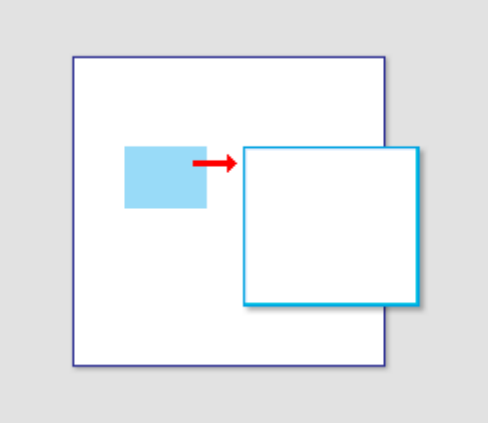
- **For common controls, use the recommended control sizes.** These are large enough for a good pen experience, except for spin controls (which aren't usable with a pen but are redundant).
- **Choose a layout that places controls close to where they are most likely going to be used.** Keep task interactions within a small area whenever possible. Avoid long distance hand movements, especially for common tasks and for drags.
- **Use the recommended spacing.** The recommended spacing is pen-friendly.
- **Interactive controls should either be touching or preferably have at least 5 pixels (3 DLUs) of space between them.** Doing so prevents confusion when users tap outside the intended target.
- **Consider adding more than the recommended vertical spacing within groups of controls,** such as command links, check boxes, and radio buttons, as well as between the groups. Doing so makes them easier to differentiate.

Interaction

- **For programs designed to accept handwriting, enable default inking.** Default inking allows users to input ink by just starting to write, without having to tap, give a command, or do anything special. Doing so enables the most natural experience with a pen. For programs not designed to accept handwriting, handle pen input in text boxes as selection.
- **Allow users to zoom the content UI** if your program has tasks that require editing text. Consider automatically zooming to 150 percent when a pen is used.
- **Because gestures are memorized, assign them meanings that are consistent across programs.** Don't give different meanings to gestures with fixed semantics. Use an appropriate program-specific gesture instead.

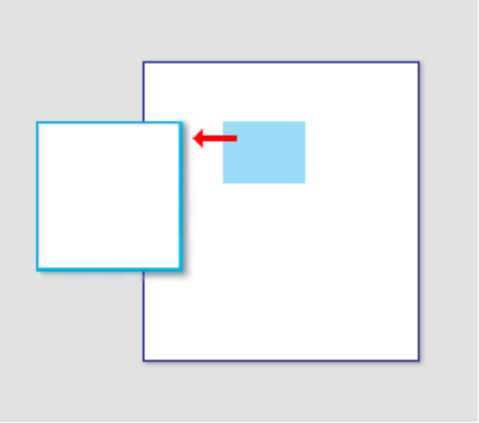
Handedness

- **If a window is contextual, always display it near the object that it was launched from.** Place it out of the way so that the source object isn't covered by the window.
    - If displayed using the mouse, when possible place the contextual window offset down and to the right.



Show contextual windows near the object that it was launched from.

- If displayed using a pen, when possible place the contextual window so as not to be covered by the user's hand. For right-handed users, display to the left; otherwise display to the right.



When using a pen, also show contextual windows so that they aren't covered by the user's hand.

- **Developers:** You can distinguish between mouse events and pen events using the GetMessageExtraInfo API. You can determine the user's handedness using the SystemParametersInfo API with SPI_GETMENUDROPALIGNMENT.

Forgiveness

- **Provide an undo command.** Ideally, you should provide undo for all commands, but your program may have some commands whose effect cannot be undone.
- **Provide good hover feedback.** Clearly indicate when the pen is over a clickable target. Such feedback is a great way to prevent accidental manipulation.
- **Whenever practical, provide good feedback on pen down, but don't take actions until a move or pen up.** Doing so allows users to correct mistakes before they make them.

- **Whenever practical, allow users to correct mistakes easily.** If an action takes effect on pen up, allow users to correct mistakes by sliding while the pen is still down.

Documentation

When referring to pen input:

- Refer to a pen-shaped stylus input device as a pen. On first mention, use tablet pen.
- Refer to the button on the side of a pen as the pen button, not the barrel button.
- Refer generically to the keyboard, mouse, trackball, pen, or finger as an input device.
- Use tap (and double-tap) instead of click when documenting procedures specific to using a pen. Tap means to press the screen and then lift before a hold time. It may or may not be used to generate a mouse click. For interactions that don't involve the pen, continue to use click.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

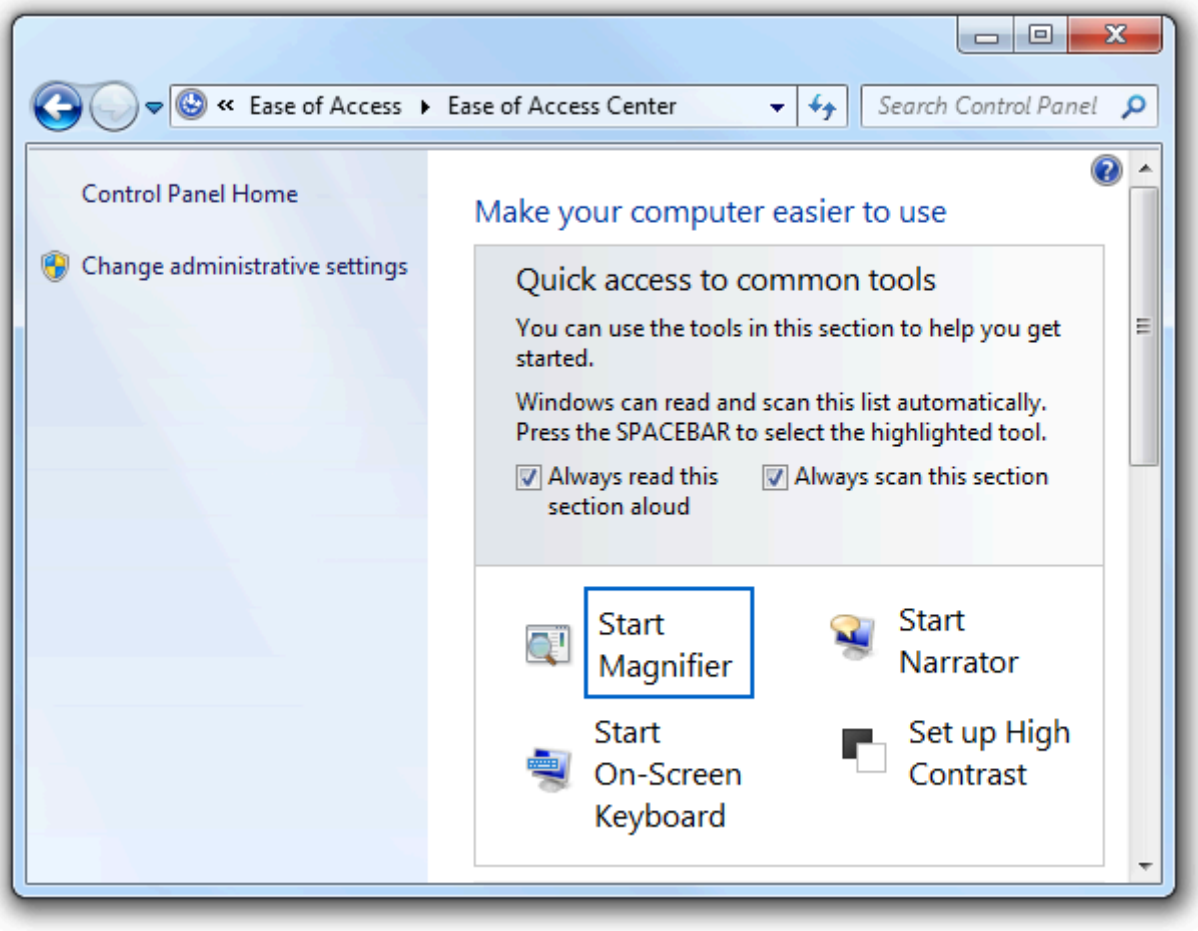
Accessibility (Design basics)

Article • 06/04/2021

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

Designing software for accessibility means ensuring that programs and functionality are easily available to the widest range of users, including those who have disabilities and impairments.

The number of users that accessibility features can help may surprise you; for example, in the United States, surveys have shown that more than half of all computer users experience difficulties or impairments related to accessibility, and are likely to benefit from the use of accessible technology. Moreover, approaching software design with the flexibility and inclusiveness that are the hallmarks of accessibility often results in overall improved usability and customer satisfaction.



The Ease of Access Center, available from Control Panel, provides a central location where users can choose and customize the accessibility features they want.

Note: Guidelines related to keyboard, mouse, color, and sound are presented in separate articles.

Design concepts

Many physical, perceptual, and cognitive factors come into play when users interact with computer hardware and software. Before considering ways to make your program's features more accessible, it helps to learn about what kinds of disabilities and impairments exist, and some of the assistive technologies these users may be working with as they interact with computers.

Types of impairments

The following table describes common user disabilities and impairments, and lists a few of the most important solutions used to make computers more accessible.

Expand table

Impairment	Description	Solutions
Visual	Ranges from mild (affecting 17 percent of users) to severe (affecting 9 percent of users).	Customizable magnification, colors, and contrast; Braille utilities; screen readers.
Hearing	Ranges from mild (affecting 18 percent of users) to severe (affecting 2 percent of users).	Information redundancy: sound used only as supplement to text or visual communication.
Dexterity	Ranges from mild (affecting 19 percent of users) to severe (affecting 5 percent of users). This impairment often involves difficulty performing certain motor skills with keyboard or mouse.	Input method redundancy: program features accessed by mouse or keyboard equivalents.
Cognitive	Includes memory impairments and perceptual differences. Affects 16 percent of users.	Highly-customizable user interface (UI); use of progressive disclosure to hide complexity; use of icons and other visual aids.
Seizure	Includes visual sensitivity to movement and flashing.	Conservative approach to modulating interfaces, such as the use of animations; avoiding screen

Impairment	Description	Solutions
		flicker in the range between 2 Hertz (Hz) and 55 Hz.
Speech or language	Includes dyslexia and oral communication difficulties.	Spell-check and grammar-check utilities; speech recognition and text-to-speech technology.

For more guidelines about helping users with these impairments, see Addressing particular impairments later in this article.

Types of assistive technologies and accessibility features

Screen readers

A screen reader enables users with visual disabilities or impairments to navigate a UI by transforming visuals to audio. Thus, UI text, controls, menus, toolbars, graphics, and other screen elements are spoken by the computerized voice of the screen reader. To create a program optimized for screen reader assistive technology, you must plan for how the screen reader will identify each UI element.

Each UI element that the user can interact with must be keyboard accessible, as well as be exposed through an accessibility application programming interface (API). We recommend using UI Automation, the new accessibility framework for all versions of Microsoft Windows that support Windows Presentation Foundation (WPF). UI Automation provides programmatic access to most elements on the desktop, enabling assistive technology products such as screen readers to provide information about the UI to users and to manipulate the UI by means other than standard input (for example, by speaking rather than or in addition to manipulating the mouse or keyboard). For more information, see the UI Automation Overview.

Be aware that although screen readers are a very important assistive technology, there are others as well. For more information about the range of technologies available, see Types of Assistive Technology Products.

Speech recognition

Speech recognition is an accessibility feature in Windows that allows users to interact with their computers by voice, reducing the need for motor interaction with the mouse or keyboard. Users can dictate documents and e-mail, use voice commands to start and switch between programs, control the operating system, and even fill out forms on the Web.

Magnifier

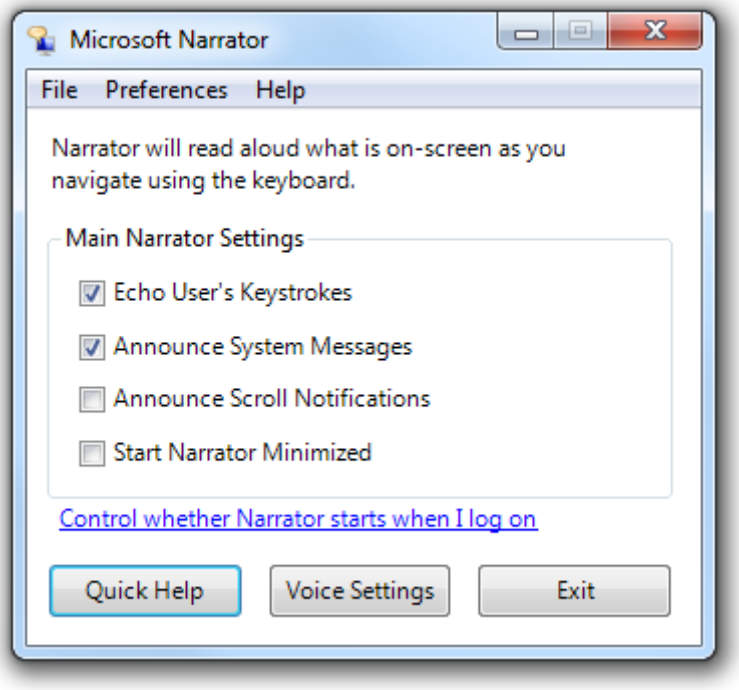
Magnification helps users with low vision by enlarging items on screen anywhere from 2 to 16 times the original. Users can set this feature to track the mouse (to see an enlarged version of what the mouse is pointing to), the keyboard (to see the area where the pointer moves when tabbing), or text editing (to see what they are typing).

Visual settings and color schemes

In addition to making things on the screen larger, users with visual impairment may benefit from system settings such as high-contrast mode or the ability to customize background and foreground color schemes.

Narrator

Narrator is a scaled-down screen reader in Windows that allows users to hear on-screen text and UI elements read aloud, even including some events (including error messages) that happen spontaneously. The user can hear the Narrator menus without leaving the active window.



Users can customize the extent to which Microsoft Narrator is used.

On-screen keyboard

For users who have difficulty with physical keyboards, and need to use an alternative input device such as a switch, on-screen keyboards are a necessity. Users can select keys using the mouse or another pointing device, a small group of keys, or just one key, depending on how you set up On-Screen Keyboard.

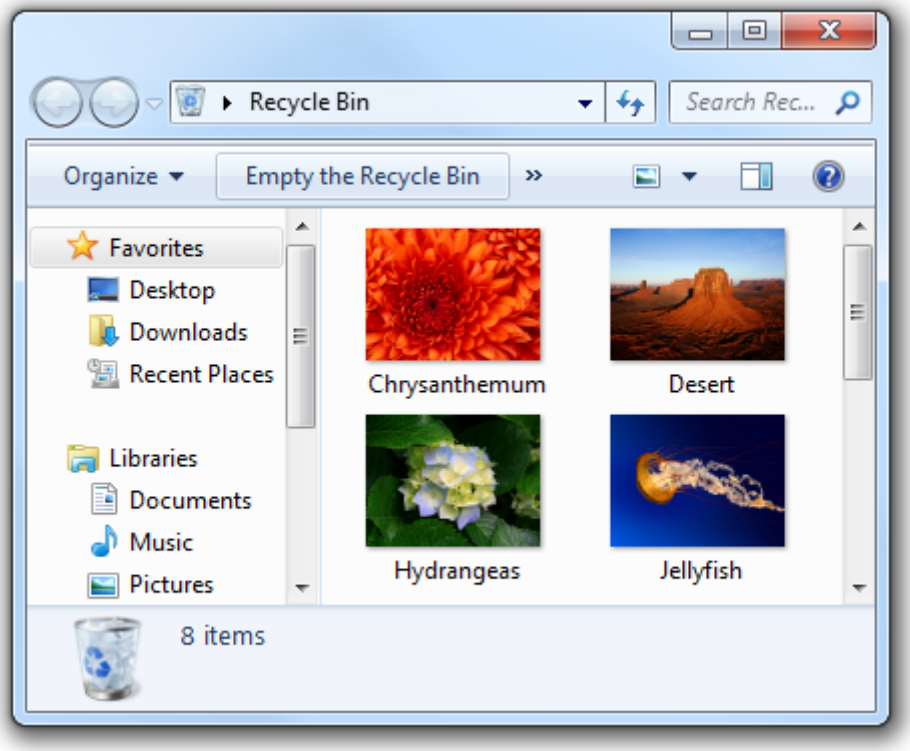
Mouse keys

With Mouse Keys enabled, users who prefer the keyboard can use the arrow keys on the numeric keypad to move the mouse pointer.

For a complete list of accessibility features, see Accessibility in Windows Vista on the Microsoft Web site.

Keyboard-based navigation

The Tab key, arrow keys, space bar, and Enter key are important for keyboard-based navigation. Pressing Tab cycles input focus through the different control groups, and pressing the arrow keys moves within a control or among controls within a group. Pressing space bar is the same as clicking the control with input focus, whereas pressing Enter is the same as clicking the default command button or command link, regardless of input focus.

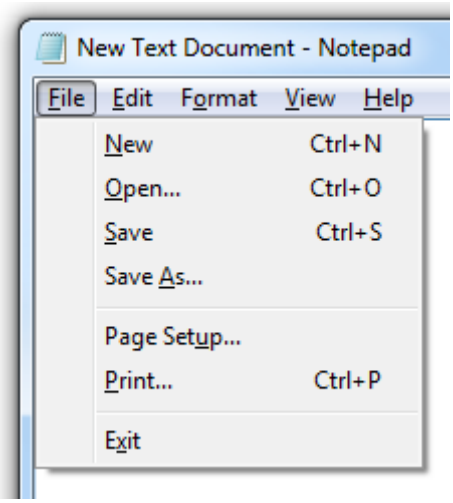


In this example, users can press Tab until the desired option has input focus, then press Enter to open the object.

Access keys

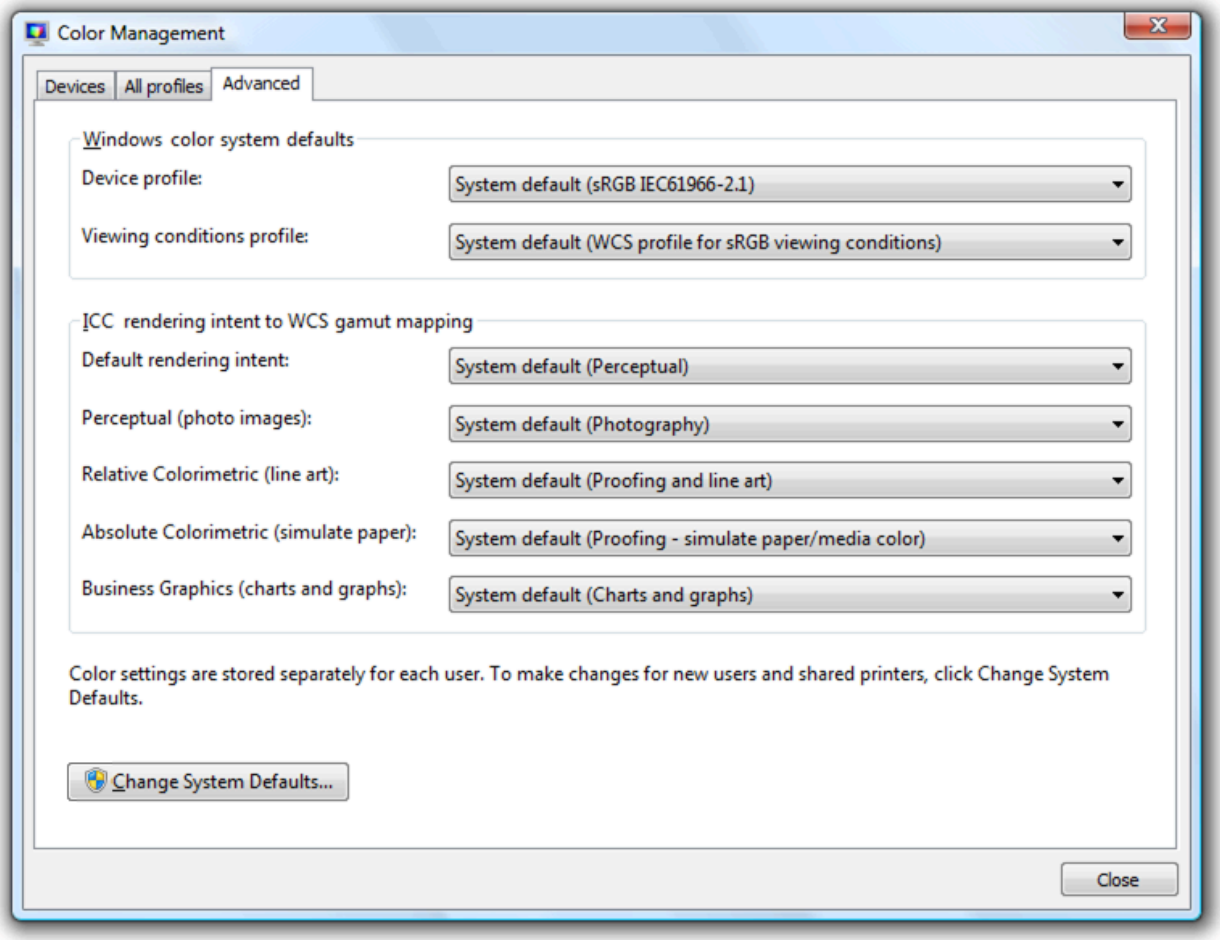
Access keys allow users to choose options and initiate commands directly without having to navigate to the control first. Access keys are indicated by underlining one of the characters in each control's label. Users then activate the option or command by

pressing the Alt key along with the underlined character. Access keys aren't case sensitive.



In this example, pressing Alt+O activates the Open command.

Choosing logical access keys for controls usually poses no difficulty; the more controls there are on a window, however, the greater the possibility you will run out of access key choices. In this case, assign access keys to control groups rather than each individual one.



In this example, access keys are assigned to control groups, rather than individual controls.

Access keys are often confused with shortcut keys, but shortcut keys are assigned differently from access keys and have different goals. For example, shortcut keys use Ctrl and Function key sequences and are intended primarily as a shortcut for advanced users instead of for accessibility.

For more information, see Keyboard.

Designing for accessibility: three fundamental practices

Accessible programs help all users in some way because the objectives of accessibility and usability overlap. For example, features designed to make advanced users as efficient as possible also benefit users who prefer using the keyboard because of dexterity impairment.

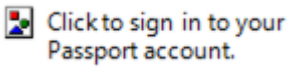
Three fundamental practices will help you with accessible design: allow for a degree of flexibility in your UI, let respect for user needs and preferences play a major role in design decisions, and provide programmatic access to your UI.

Providing flexible UI

Accessible design is, at least in part, about giving users choices. Not a frustrating, dizzying array of choices, but a limited number of choices that smartly anticipates user needs. "Don't like navigating by way of the mouse? Here, you can do the very same things using only the keyboard. Don't like physical keyboards? Here's a virtual one you can use on-screen."

For example, provide flexibility by:

- Providing user-selectable equivalents for non-text elements (for example, alt text for graphics and captions for audio).

  Users who have chosen not to render graphics should see alt text instead, describing what the control does and how to interact with it.

- Providing alternatives to color (for example, icon differentiation or the use of sounds).

  In this example, the standard icons are readily distinguishable based on their designs.

- Ensuring keyboard access (for example, a tab stop for every interactive control) so that users can accomplish the same things in your program with either the mouse or the keyboard.

- Ensuring that your program offers good color contrast options for users. Windows provides a high contrast option, but that's really designed to be a solution for severe visual impairment. Other contrast options best serve users with mild impairment, such as low vision and color blindness.

- Ensuring that users have a way to adjust the size of text in your program's UI (for example, through a slider control or drop-down box for font size). If possible, support high dots per inch (dpi) mode.

- Ensuring that your program is multimodal, meaning that if the primary mode of the program is inaccessible for some, these users have a way to work around the problem. For example, when animation is displayed, the information should be displayable in at least one non-animated presentation mode at the option of the user.

Multimodal interfaces and flexible navigation essentially offer the user the architecture of information redundancy. Redundancy sometimes has negative connotations; in user interface text, for example, we advise removing redundancy to streamline the reading experience. But in the context of accessibility, redundancy connotes positive, fail-safe mechanisms and experiences.

Respecting your users

Respect as a general, guiding principle is vital for designing accessible programs. Even as an intellectual exercise, imagine what it must be like to encounter your program as a user who is disabled. Take the time to test UI screens in high contrast mode and at various resolutions, to ensure the experience is a good one for users with visual impairments. Test keyboard accessibility by selecting the **Underline keyboard shortcuts and access keys** check box in the Ease of Access Center Control Panel item (so that access keys are always visible). You can even go beyond rigorous testing by hiring

developers and designers who have a natural aptitude for empathizing with others to begin with.

You should also demonstrate respect by:

- Using system-wide settings (for example, System Color) rather than hardwiring settings for your particular program. Respect not only the parameters that users have specifically selected for interacting with their programs, but also accessibility features built into the operating system that the user wants in effect no matter which program they are using. For more information, see About Windows Accessibility Features.
- Preferring common controls to custom controls, because common controls have already implemented the Windows accessibility APIs.
- Documenting all accessibility options and features (for example, all keyboard shortcuts). Users with impairments are highly motivated to discover accessibility features, and often expect comprehensive information to be collected in Help.
- Creating accessible documentation in accessible formats. Thus, the documentation itself should adhere to the same rules of accessibility as the primary UI, including the ability to enlarge font size, the use of alt text for graphics, and redundant information architecture (for example, using color-coding only as a supplement to text).

In software products, respect for users may manifest itself in usability and market research, in efficacious support services and documentation, and of course in design decisions. For example, thinking again in terms of design for advanced users: are you putting that cutting-edge new feature in because you want it, or because you know that your advanced users have been asking for it? The latter case indicates that your design decision-making process is well-informed by the value of respect.

Providing programmatic access

Providing programmatic access to the UI is essential so that assistive technologies (such as screen readers, alternative input devices, and speech recognition programs) interpret the screen correctly for their users. By creating a "map" of each UI screen in your program, you make it available to users of assistive technologies.

Do this well by:

- Enabling programmatic access to all UI elements and text (for example, using the Active Accessibility COM interface, **IAccessible**).
- Placing names (or titles) and descriptions on UI objects, frames, and pages (for example, using the **IAccessible** Name property).

- Ensuring programmatic events are triggered by all UI activities (for example, focus events for all UI activities involving focus movement).

If you do only four things...

1. Ensure every user can leverage the full potential of your program.
2. Think of accessibility as an opportunity for creative problem-solving and another means of increasing overall user satisfaction.
3. Respect system settings.
4. Use common controls whenever possible.

Guidelines

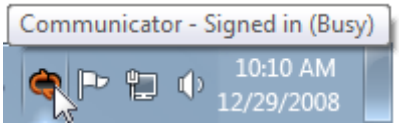
General

- **Don't disrupt or disable activated features of the operating system or other products that are identified as accessibility features.** You can identify these features by referring to the documentation of the operating system or product in question.
- **Don't force users to interact with your program as the top window on the screen.** If a function or a window is required continuously for users to perform a task, that window should always remain visible, if the user so chooses, regardless of its position relative to other windows. For example, if the user has a movable on-screen keyboard that is on top of all other windows so that it is visible at all times, your program should never obscure it by mandatory placement at the top of the Z order.
- **Use system colors, fonts, and common controls whenever possible.** By doing so, you significantly reduce the number of accessibility issues users encounter.

Addressing particular impairments

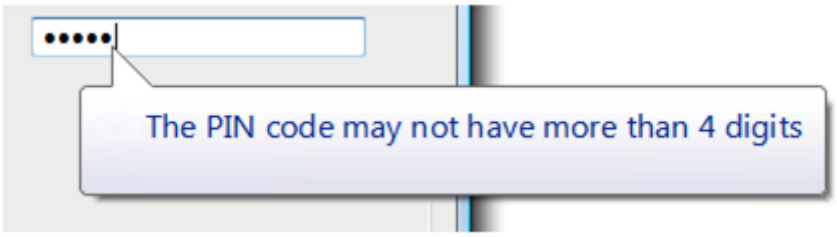
Visual

- **Never rely on color alone to convey meaning.** Use color only as a means of reinforcing the meaning provided by text, design, location, or sound.

  The primary method of communication in this example is the concise tooltip text. The use of color assists in communicating the meaning, but is secondary.

- **Use alternative (alt) text infotips to describe graphics.**
- **Don't use text in graphics.** Users with visual impairments may have graphics turned off (for example, in a Web browser), or may simply not see or look for text placed in graphics.
- **Ensure that dialog boxes and windows have meaningful names,** so that a user who is hearing rather than seeing the screen (for example, using a screen reader) gets appropriate contextual information.
- **Respect the user's settings for visual display** by always obtaining font typefaces, sizes, and colors, Windows display element sizes, and system configuration settings from the Theme and GetSystemMetrics APIs.
- **Keep balloon text concise** so that it is easier to read and minimizes disruption to screen readers.



Although balloons may use additional body text if necessary, this example shows that sometimes title text alone achieves the same goal in a more economical and accessible way.

Hearing

- **Never rely on sound alone to convey meaning.** Use sound only as a means of reinforcing the meaning provided by text, design, location, or color.
- **Enable users to control the volume of audio output.** Use the Windows Volume Mixer for this purpose. For more information, see Sound.
- **Target your program's sound to occur in a range between 500 Hz and 3000 Hz** or be easily adjustable by the user into that range. Sounds in this range are most likely to be detectable by people with hearing impairment.

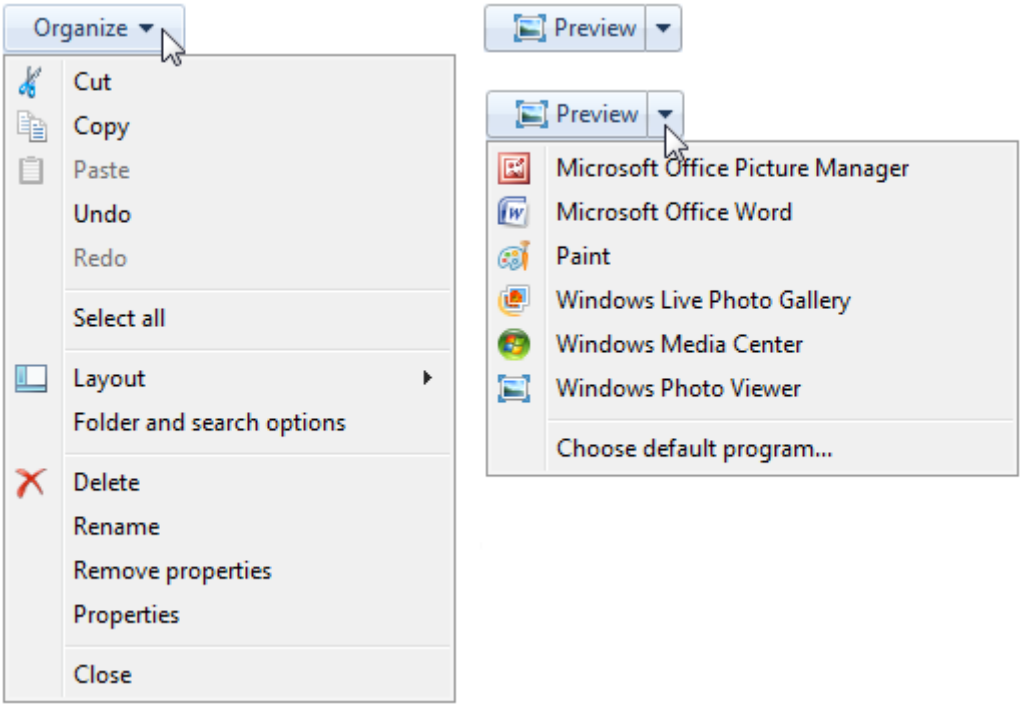
Dexterity

- **Make UI timeout values relative to GetDoubleClickTime() instead of using absolute times.** Doing so adjusts the timeouts to the speed of the user.
- **Assign access keys to all menu items** so that users who prefer working with the keyboard have the same ability to navigate your program as users who work with the mouse.

- **Don't make double-clicking and dragging the only way to perform an action.** These can be difficult movements for some users.
- **Don't remove menu bars from your program.** Menu bars are easier than toolbars for keyboard users to access. If you don't want the menu bar visible by default, hide it instead.
- **Make Help accessible from the keyboard by providing tab stops for Help buttons and links.**
- **To improve awareness of the access key assignments in your program, you can display them at all times.** In Control Panel, go to the Ease of Access Center, and click **Make the keyboard easier to use**; then select the **Underline keyboard shortcuts and access keys** check box.

Cognitive

- **Use progressive disclosure** to hide complexity.



  In these examples, options available from the command button are hidden by default, and users can choose to view the options by taking advantage of progressive disclosure controls.

- **Use icons, toolbars, and other visual aids** to reduce cognitive load of reading text.

- When possible, **provide auto-complete functionality in text boxes and editable drop-down lists**, so that users don't have to type the entire name of commands, file names, or similar choices from a limited set of options. This reduces cognitive load for all users, and reduces the amount of typing for users for whom spelling or typing is difficult, slow, or painful.

- **Demonstrate difficult concepts in Help by including tutorials and animations.** Note that animations can be difficult for users with seizure impairment, and therefore should be used only when necessary.

Seizure

- **Don't use flashing or blinking text, objects, or other elements having a flash or blink frequency in the range between 2-55 Hz.**
- **Limit use of animations.** Some users are particularly sensitive to screen movement, especially in the periphery of their visual field. If you use animation to draw attention to something, make sure that attention is deserved and worthy of interrupting the user.

Speech or language

- **Organize and write clear, concise, easily understood text.** Usability tests show that unfolding key information at the end of a phrase improves comprehension. For more guidelines, see Style and Tone.

Incorrect:

Is three the next digit?

Click OK to begin.

Correct:

Is the next digit three?

To begin, click OK.

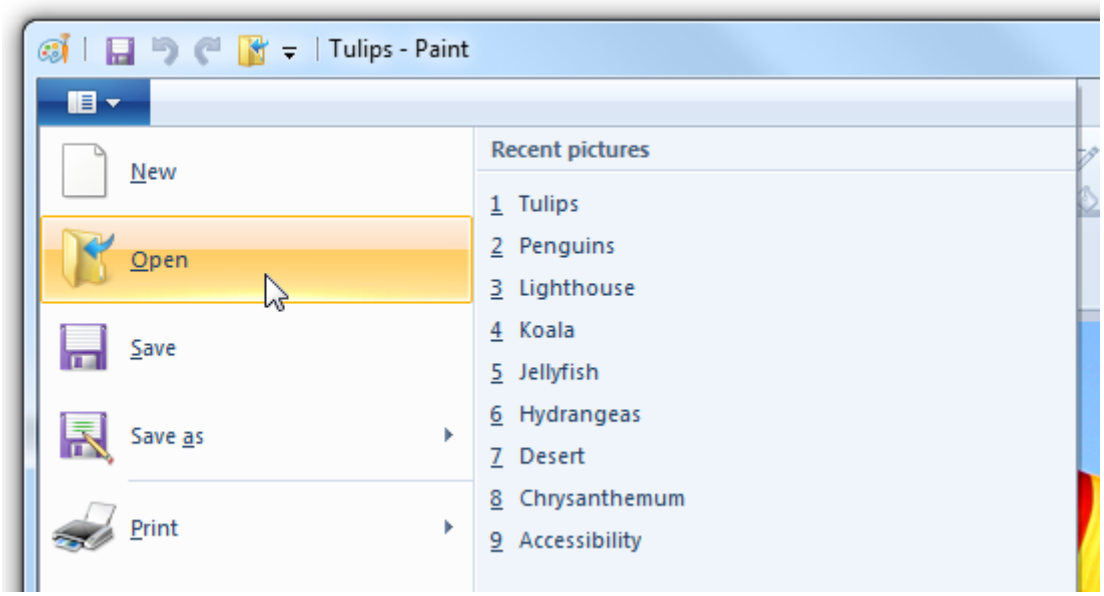
Access keys

- **Prefer characters with wide widths,** such as w, m, and capital letters.
- **Prefer a distinctive consonant or a vowel,** such as "x" in "Exit."
- **Avoid using characters that make the underline difficult to see,** such as (from most problematic to least problematic):
  - Characters that are only one pixel wide, such as i and l.
  - Characters with descenders, such as g, j, p, q, and y.
  - Characters next to a letter with a descender.

Menu access keys

- **Assign access keys to all menu items.** No exceptions.

- **For dynamic menu items (such as recently used files), assign access keys numerically.**



  In this example, the Paint program in Windows assigns numeric access keys to recently used files.

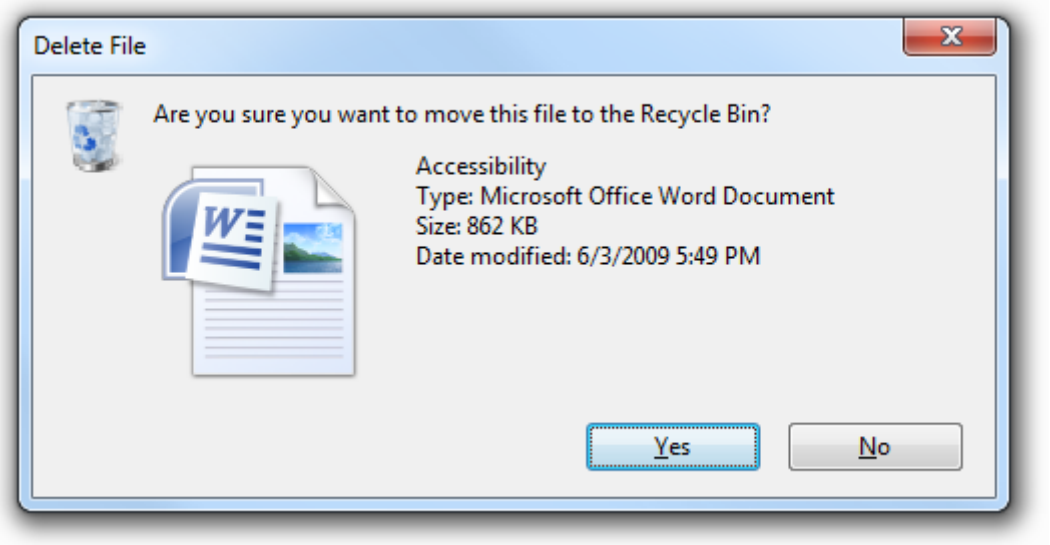
- **Assign unique access keys within a menu level.** You can reuse access keys across different menu levels.

- **Make access keys easy to find:**
  - For the most frequently used menu items, choose characters at the beginning of the first or second word of the label, preferably the first character.
  - For less frequently used menu items, choose letters that are a distinctive consonant or a vowel in the label.

Dialog box access keys

- **Whenever possible, assign unique access keys to all interactive controls or their labels.**Read-only text boxes are interactive controls (because users can scroll them and copy text), so they benefit from access keys. **Don't assign access keys to:**

  - **OK, Cancel, and Close buttons.** Enter and Esc are used for their access keys. However, always assign an access key to a control that means OK or Cancel, but has a different label.



In this example, the positive commit button has an access key assigned.

- **Group labels.** Normally, the individual controls within a group are assigned access keys, so the group label doesn't need one. However, assign an access key to the group label and not the individual controls if there is a shortage of access keys.
- **Generic Help buttons,** which are accessed with F1.
- **Link labels.** There are often too many links to assign unique access keys, and link underscores hide the access key underscores. Have users access links with the Tab key instead.
- **Tab names.** Tabs are cycled using Ctrl+Tab and Ctrl+Shift+Tab.
- **Browse buttons labeled "...".** These can't be assigned access keys uniquely.
- **Unlabeled controls,** such as spin controls, graphic command buttons, and unlabeled progressive disclosure controls.
- **Non-label static text or labels for controls that aren't interactive,** such as progress bars.
- **Assign commit button access keys first to ensure that they have the standard key assignments.** If there isn't a standard key assignment, use the first letter of the first word. For example, the access key for Yes and No commit buttons should always be "Y" and "N", regardless of the other controls in the dialog box.
- **For negative commit buttons (other than Cancel) phrased as a "Don't", assign the access key to the "n" in "Don't".** If not phrased as a "Don't", use the standard access key assignment or assign the first letter of the first word. By doing so, all Don'ts and No's have a consistent access key.

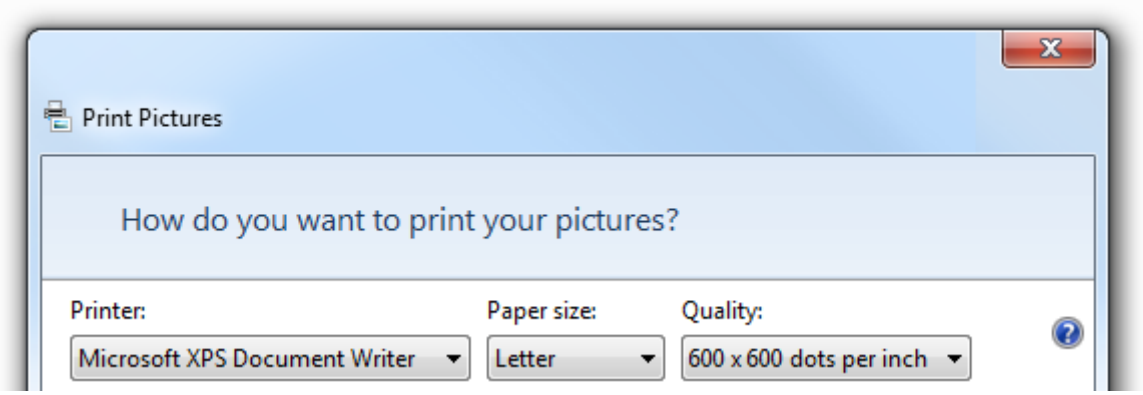
- **To make access keys easy to find, assign the access keys to a character that appears early in the label,** ideally the first character, even if there is a keyword that appears later in the label.

For more guidelines and examples, see Keyboard.

Text

- **Use colons at the end of external control labels.** Some assistive technologies look for colons to identify control labels.

- **Position labels consistently relative to the elements that they are labeling.** This helps assistive technology correctly associate the labels with their corresponding controls, and helps users of screen enlargers know where to look for a label or control.

  In this example, the labels for each of the drop-down lists are placed consistently and use colons.

- **Limit alt text to 150 characters maximum.** Describe the action to activate the control (for example, click, right-click, and so on) and then describe the control's function.

  Acceptable:

  Button.

  Blue hills.

  Better:

  Click to sign in to your account.

  Photo of distant hills showing how colors fade over distance.

- **Don't use text to draw lines, boxes or other graphical symbols.** Characters used in this way can confuse users of screen readers. For example, a box drawn with the letter "X" around an area of text is read by screen-reader software as "X X X X X X" on the first line, followed by "X" and the content and "X".

Documentation

- Document all accessibility options and features (for example, all keyboard shortcuts).
- Create accessible documentation in accessible formats. Thus, the documentation itself should adhere to the same rules of accessibility as the primary UI.
- Refer to access keys, not shortcut keys (which have a different meaning and use), mnemonic keys, or accelerators.
- In general, refer to a person with a kind of disability, not a disabled person. Consider the person first, not the label.

Expand table

Use these terms	Instead of
Has limited dexterity, has motion disabilities	Crippled, lame
Without disabilities	Normal, able-bodied, healthy
One-handed, people who type with one hand	Single-handed
People with disabilities	The disabled, disabled people, people with handicaps, the handicapped
Cognitive disabilities, developmental disabilities	

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Windows (Design basics)

Article • 08/19/2021

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

Windows are the main "canvases" or UI surfaces of your desktop app, including the main windows itself and pop-ups, dialogs, and wizards. Follow these guidelines when deciding which surface to use and how best to use them.

In this section

Expand table

Topic	Description
Window Management	This article covers default placement of windows when initially displayed on the screen, their stacking order relative to other windows (Z order), their initial size, and how their display affects input focus.
Window Frames	Most programs should use standard window frames. Immersive applications can have a full screen mode that hides the window frame. Consider using glass strategically for a simpler, lighter, more cohesive look.
Dialog Boxes	A dialog box is a secondary window that allows users to perform a command, asks users a question, or provides users with information or progress feedback.
Common Dialogs	The Microsoft Windows common dialogs consist of the Open File, Save File, Open Folder, Find and Replace, Print, Page Setup, Font, and Color dialog boxes.
Wizards	Despite that wonderful, whimsical name, wizards are not really a special form of user interface, and they have only a particular range of utility.

Topic	Description
Property Windows	Property window is the collective name for the following types of user interfaces (UIs): • Property sheet: used to **view and change properties for an object or collection of objects in a dialog box**. • Property inspector: used to **view and change properties for an object or collection of objects in a pane**. • Options dialog box: used to **view and change options for an application**.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Window Management

Article • 02/09/2021

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

This article covers default placement of windows when initially displayed on the screen, their stacking order relative to other windows (Z order), their initial size, and how their display affects input focus.

For the following guidelines:

- A top-level window has no owner window and is displayed on the taskbar. Examples: application windows. In Windows Vista and later, dialog boxes without owner windows and property sheets are also considered top-level.
- An owned window has an owner window and isn't displayed on the taskbar. Examples: modal dialog boxes, modeless dialog boxes.
- A user-initiated window is displayed as the direct result of a user's action. Otherwise, it is program initiated if initiated by a program, or system initiated if initiated by Microsoft Windows . For example, an Options dialog box is user initiated, but a meeting reminder is program initiated.
- A contextual window is a user-initiated window that has a strong relationship to the object from which it was launched. For example, windows displayed by context menus or notification area icons are contextual, but windows displayed by menu bars are not.
- The active monitor is the monitor where the active program is running.
- The default monitor is the one with the Start menu, taskbar, and notification area.

Design concepts

Window management is one of the most fundamental user activities. Before Windows Vista, windows were often given small default sizes and placed in the middle of the screen. That approach works well for older single, low-resolution monitors, but not for modern video hardware.

Windows is designed to support modern video hardware, which often runs at resolutions significantly higher than the minimum supported screen resolution and may

have multiple monitors. Doing so:

- Allows users to benefit fully from their advanced hardware.
- Requires less effort from users to move their mouse across greater distances.
- Makes window placement more predictable and therefore easier to find.

The minimum supported screen resolution

The minimum effective screen resolution supported by Windows is 800x600 pixels. This means that fixed size windows should display fully at the minimum resolution (while reserving space for the taskbar), but resizable windows can be optimized for an effective resolution of 1024x768 pixels as long as they are functional at the minimum resolution.

While currently the most common physical screen resolutions for Windows PCs are 1024x768 pixels or greater, targeting 800x600 pixels allows Windows to:

- Work well with all modern hardware, including small notebook PCs.
- Support high dpi (dots per inch) settings.
- Support larger fonts for accessibility.
- Support hardware used on a global basis.

Choosing the minimum resolution to support requires striking the right balance. Targeting a higher resolution would result in a suboptimal experience for a significant percentage of modern hardware, whereas targeting a lower resolution would prevent designers from taking full advantage of the available screen space.

If you believe that your target users are using significantly higher resolutions than the Windows minimum, you can design your program to take full advantage of the extra screen space by using resizable windows that scale well.

Guidelines

General

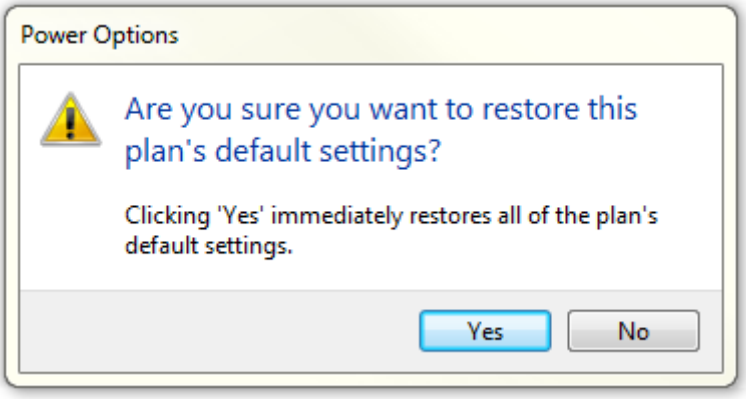
- **Support the minimum Windows effective resolution of 800x600 pixels.** For critical user interfaces (UIs) that must work in safe mode, support an effective resolution of 640x480 pixels. Be sure to account for the space used by the taskbar by reserving 48 vertical relative pixels for windows displayed with the taskbar.
- **Optimize resizable window layouts for an effective resolution of 1024x768 pixels.** Automatically resize these windows for lower screen resolutions in a way that is still functional.

- **Be sure to test your windows in 96 dpi (100 percent) at 800x600 pixels, 120 dpi (125 percent) at 1024x768 pixels, and 144 dpi (150 percent) at 1200x900 pixels.** Check for layout problems, such as clipping of controls, text, and windows, and stretching of icons and bitmaps.
- **For programs with touch and mobile use scenarios, optimize for 120 dpi.** High-dpi screens are currently prevalent on touch and mobile PCs.
- **Resizable windows no longer must show the resize glyph** in the lower-right corner, because:
  - All sides and edges of a window are resizable, not just the lower-right corner.
  - The glyph requires a status bar to display, yet many resizable windows don't provide status bars.
  - The resizable window borders and resize pointers are more effective at communicating that a window is resizable than the resize glyph.

Title bar controls

Use the title bar controls as follows:

- **Close.** All primary and secondary windows with a standard window frame should have a Close button on the title bar. Clicking Close has the effect of canceling or closing the window.

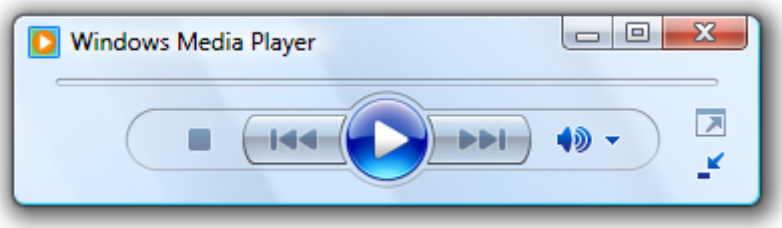


In this example, the dialog box doesn't have a Close button in the title bar.

- **Minimize.** All primary windows and long-running modeless secondary windows (such as progress dialogs) should have a Minimize button. Clicking Minimize reduces the window to its taskbar button. Consequently, windows that can be minimized require a title bar icon.
- **Maximize/Restore down.** All resizable windows should have a Maximize/Restore down button. Clicking Maximize displays the window in its largest size, which for most windows is full screen; whereas clicking Restore down displays the window in its previous size. However, some windows don't benefit from using a full screen, so these windows should maximize to their largest useful size.

Window size

- **Choose a default window size appropriate for its contents.** Don't be afraid to use larger initial window sizes if you can use the space effectively.
- **Use resizable windows whenever practical to avoid scroll bars and truncated data.** Windows with dynamic content and lists benefit the most from resizable windows.
- **For text documents, consider a maximum line length of 65 characters** to make the text easy to read. (Characters include letters, punctuation, and spaces.)
- Fixed-sized windows:
  - **Must be entirely visible and sized to fit within the work area.**
- Resizable windows:
  - **May be optimized for higher resolutions, but sized down as needed at display time to the actual screen resolution.**
  - **For progressively larger window sizes, must show progressively more information.** Make sure that at least one window portion or control has resizable content.
  - **Should avoid default restored sizes that are maximized or near maximized.** Instead, choose a default size that is typically the most useful without being full screen. Assume that users will maximize the window instead of resizing to make it full screen.
  - **Should set a minimum window size if there is a size below which the content is no longer usable.** For resizable controls, set minimum resizable element sizes to their smallest functional sizes, such as minimum functional column widths in list views.
  - **Should change the presentation if doing so makes the content usable at smaller sizes.**

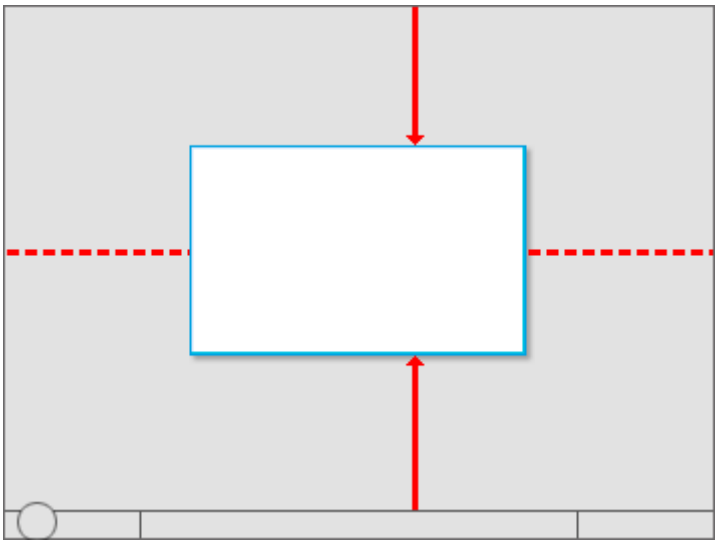


In this example, Windows Media Player changes its format when the window becomes too small for the standard format.

Window location

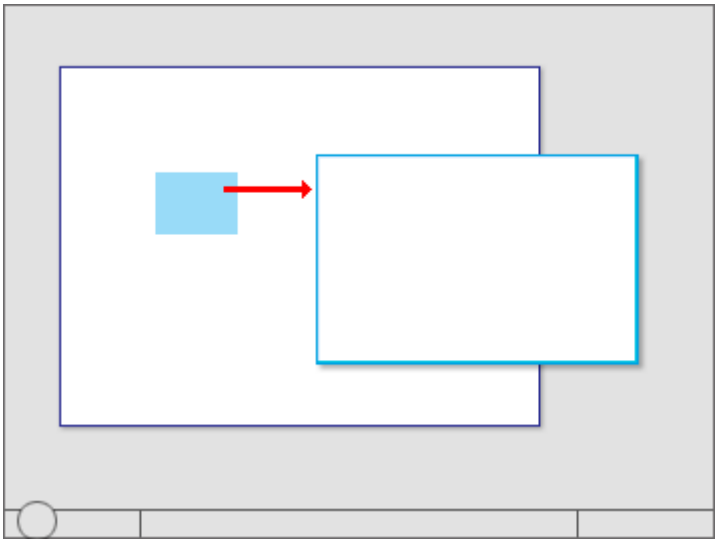
- **For the following guidelines, "centering" means to bias vertical placement slightly towards the top of the monitor, instead of placing exactly in the middle.**

Put 45 percent of the space between the top of the monitor/owner and the window top, and 55 percent between the bottom of the monitor/owner and the window bottom. Do this because the eye is naturally biased towards the top of the screen.

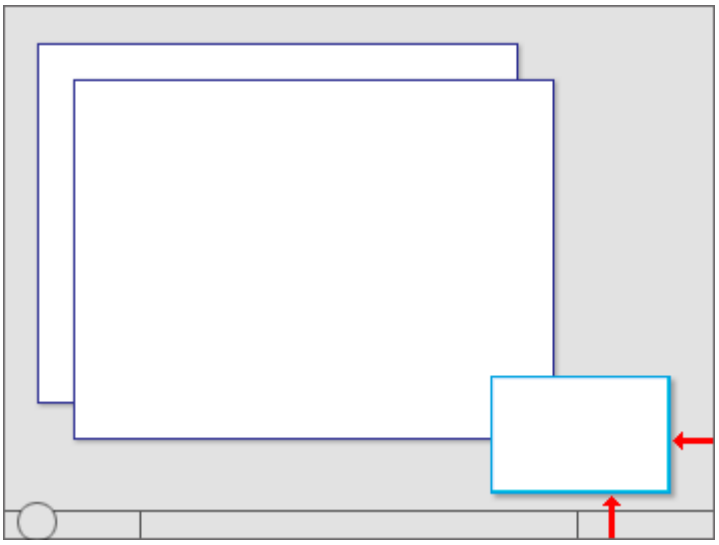


"Centering" means to bias vertical placement slightly towards the top of the monitor.

- **If a window is contextual, always display it near the object that it was launched from.** Place it out of the way so that the source object isn't covered by the window.
  - If displayed using the mouse, when possible place it offset down and to the right.

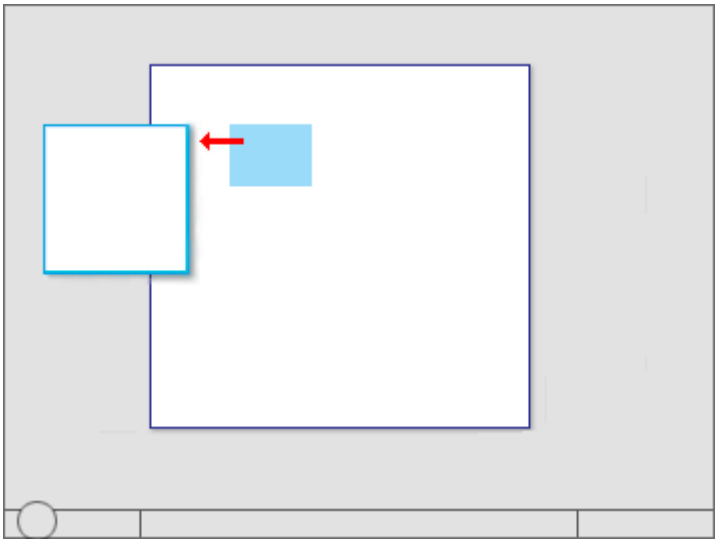


Show contextual windows near the object that it was launched from.



Windows launched from notification area icons are displayed near the notification area.

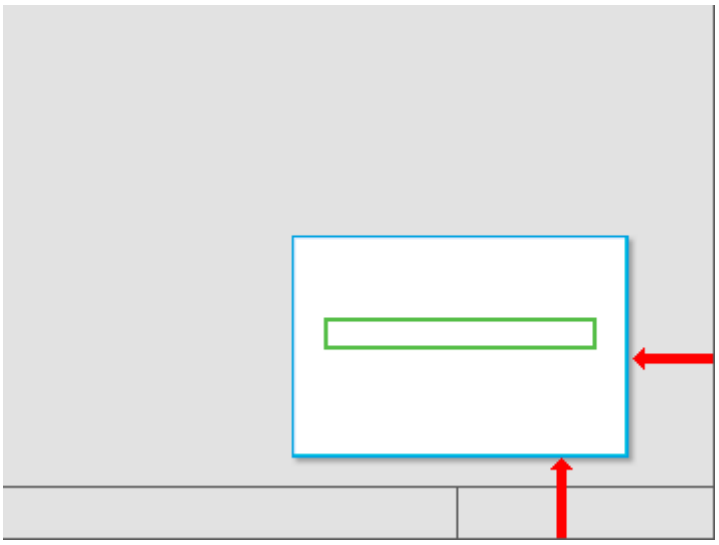
- If displayed using a pen, when possible place it so as not to be covered by the user's hand. For right-handed users, display to the left; otherwise display to the right.



When using a pen, also show contextual windows so that they aren't covered by the user's hand.

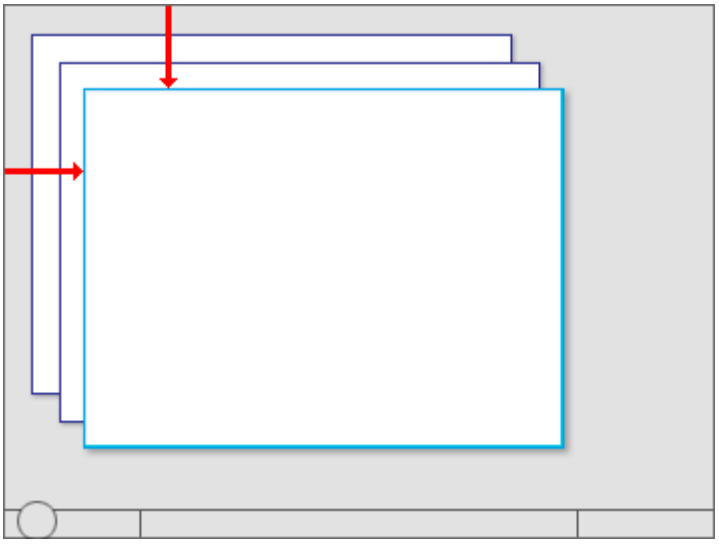
- **Developers:** You can distinguish between mouse events and pen events using the GetMessageExtraInfo API. You can determine the user's handedness using the SystemParametersInfo API with SPI_GETMENUDROPALIGNMENT.

- **Place progress dialogs out of the way in the lower-right corner of the active monitor.**



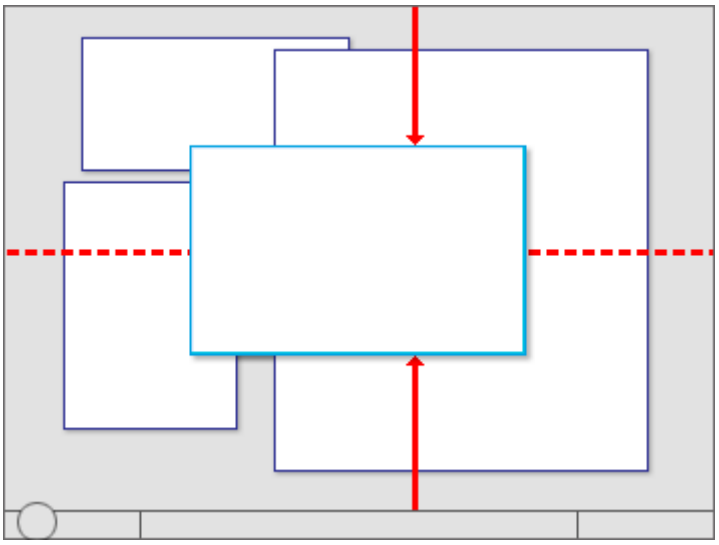
Place progress dialogs in the lower-right corner.

- **If a window isn't related to the current context or user action, place it away from the current pointer location.** Doing so prevents accidental interaction.
- **If a window is a top-level application or document, always cascade its origin off the upper-left corner of the monitor.** If created by the active program, use the active monitor; otherwise, use the default monitor.



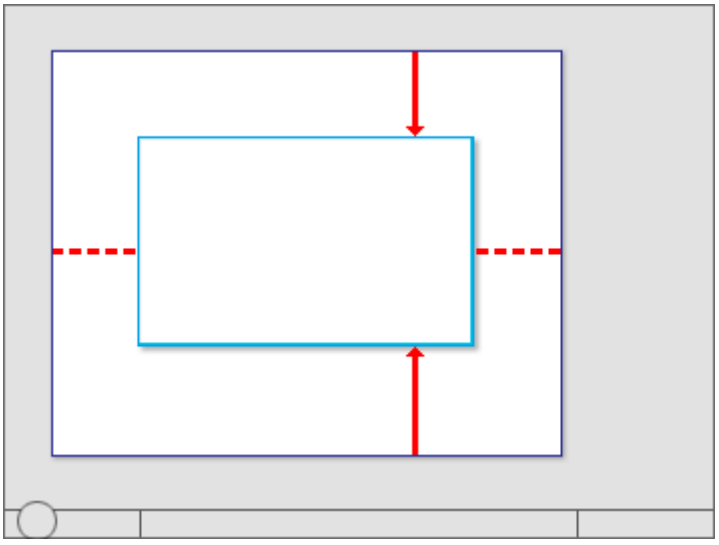
Cascade top-level application or document windows off the upper-left corner of the monitor.

- **If a window is a top-level utility, always display it "centered" in the monitor.** If created by the active program, use the active monitor; otherwise, use the default monitor.



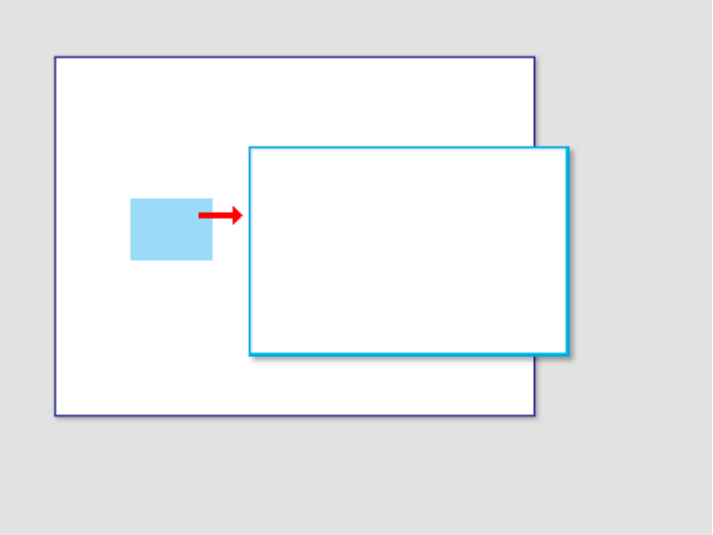
Center top-level utility windows.

- **If a window is an owned window, initially display it "centered" on top of the owner window.** For subsequent display, consider displaying it in its last location (relative to the owner window) if doing so is likely to be more convenient.



Initially center owned windows on top of the owner window.

- **For modeless dialogs, always display initially on top of the owner window to make them easy to find.** However, if the user activates the owner window, that may obscure the modeless dialog.



Display modeless dialogs initially on top of the owner window to make them easy to find.

- **If necessary, adjust the initial location so that the entire window is visible within the target monitor.** If a resizable window is larger than the target monitor, reduce it to fit.

Window order (Z order)

- **Always place owned windows on top of their owner window.** Never place owned windows under their owner windows, because most likely users won't see them.
- **Respect users' Z order selection. When users select a window, bring only the windows associated with that instance of the program (the window plus any owner or owned windows) to top of the Z order.** Don't change the order of any other windows, such as independent instances of same program.

Window activation

- **Respect users' window state selection. If an existing window needs attention, flash the taskbar button three times to draw attention and leave it highlighted, but don't do anything else.** Don't restore or activate the window. Don't use any sound effects. Instead, let users activate the window when they are ready.
  - **Exception:** If the window doesn't appear on the taskbar, bring it to the top of all the other windows and flash its title bar instead.
- **Restoring a primary window should also restore all its secondary windows**, even if those secondary windows have their own taskbar button. When restoring, place secondary windows on top of the primary window.

Input focus

- **Windows displayed by user-initiated actions should take input focus, but only if the window is rendered immediately** (within 5 seconds). Once the window is rendered, it can take input focus once.
  - If a window renders slowly (more than 5 seconds), users are likely to perform another task while they wait. Taking focus at this point would be an annoyance, especially if done more than once.
- **Windows that aren't immediately displayed or displayed by a system-initiated action shouldn't take input focus.** Instead, display on top without focus, and let users activate them when they are ready.
  - **Exception:** Credential Manager.

Persistence

- **When a window is redisplayed, consider displaying it in the same state as last accessed.** When closing, save the monitor used, window size, location, and state (maximized vs. restore). When redisplaying, restore the saved window size, location, and state using the appropriate monitor. Also, consider making these attributes persist across program instances on a per-user basis. **Exceptions:**
  - Don't save or make these attributes persist for windows when their usage is such that users are far more likely to want to start completely over.
  - For programs likely to be used on Windows Tablet and Touch Technology computers, save two windows states for landscape and portrait modes. For more information, see Designing for Varying Display Sizes.
- **If the current monitor configuration prevents displaying a window using its last state:**
  - Try to display the window using its last monitor.
  - If the window is larger than the monitor, resize the window as necessary.
  - Move the location toward the upper-left corner to fit within the monitor as necessary.
  - If the above steps don't solve the problem, revert to the default window placement guidelines. Consider restoring the previous size, if possible.

Feedback

Was this page helpful? Yes No

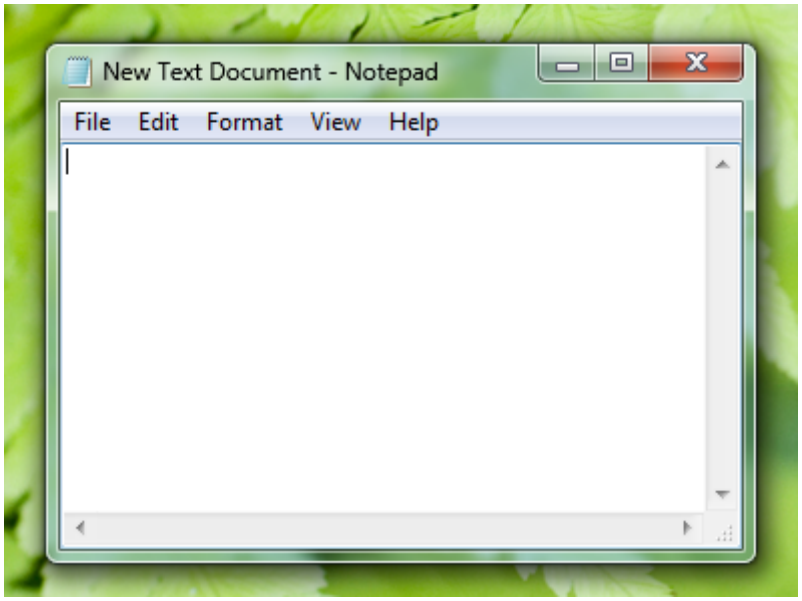
Window Frames

Article • 08/09/2021

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

Most programs should use standard window frames. Immersive applications can have a full screen mode that hides the window frame. Consider using glass strategically for a simpler, lighter, more cohesive look.

With a window frame, users can manipulate a window and view the title and icon to identify its contents.



A typical window frame.

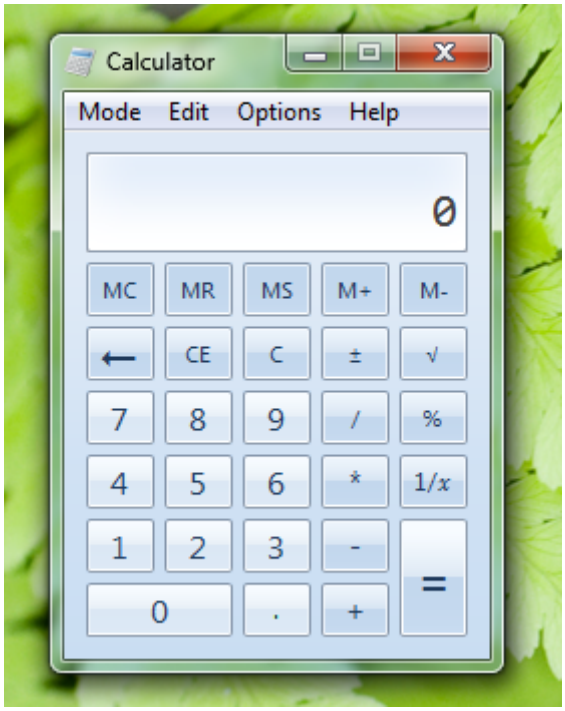
Note: Guidelines related to window management and branding are presented in separate articles.

Design concepts

Glass window frames

The glass window frames are a striking new aspect of the Microsoft Windows aesthetic, aiming to be both attractive and lightweight. These translucent frames give windows an

open, less intrusive appearance, helping users focus on content and functionality rather than the interface surrounding it.



Glass window frames.

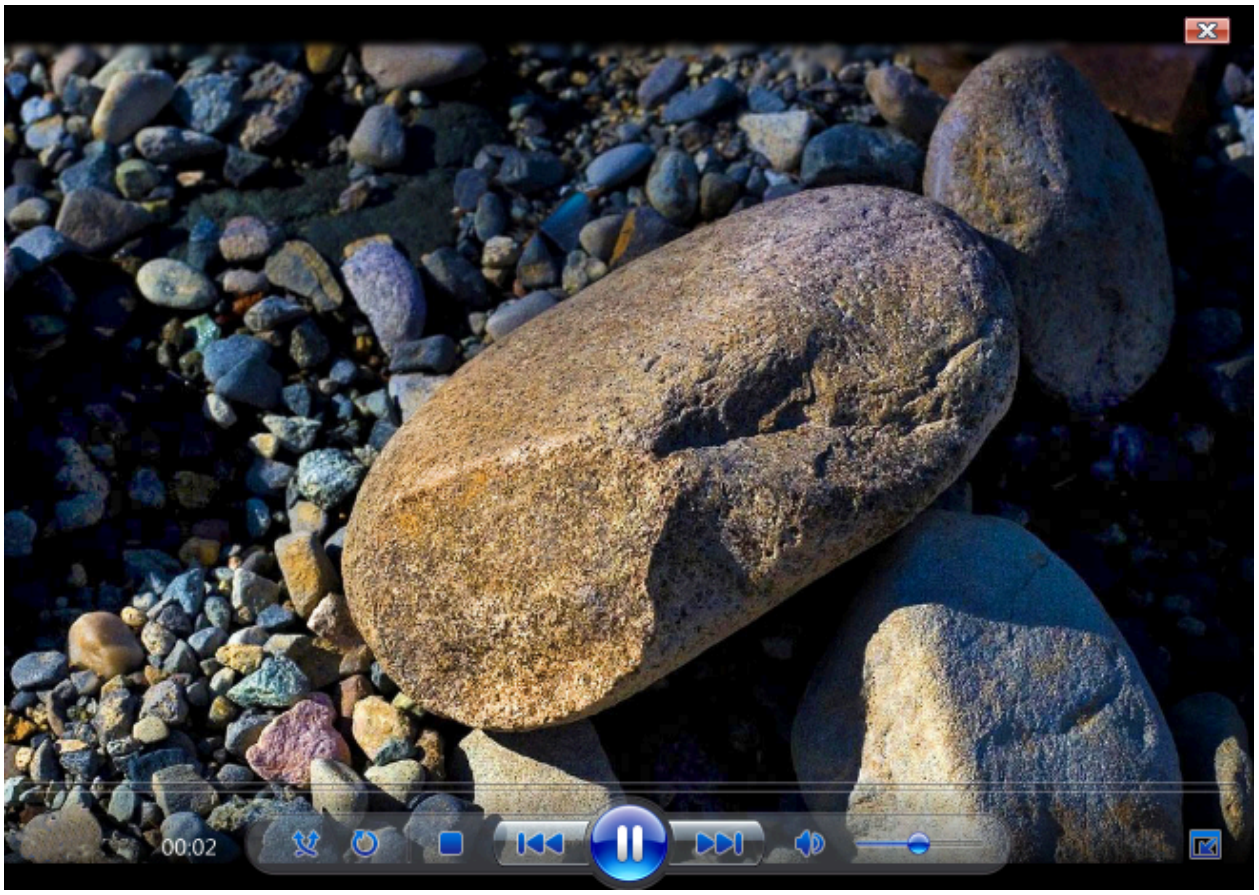
You can use glass strategically in small regions within a window that touch the window frame. Such regions appear to be part of the window frame, even though technically they are part of the window's client area.

In this example, glass is used in the client area to make it look like part of the frame.

Hidden frames

Sometimes the best window frame is no frame at all. This is often the case for the primary window of immersive full screen applications that aren't used in conjunction with other programs, such as media players, games, and kiosk applications.

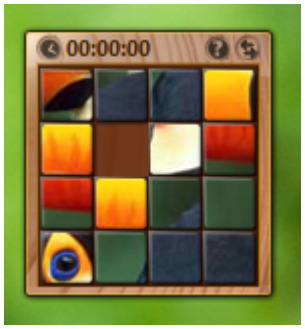
Content viewers often benefit from having the option to show content full screen. Examples include Windows Internet Explorer , Windows Live Photo Gallery, Windows Movie Maker HD, Microsoft PowerPoint, and Microsoft Word.



In this example, Windows Media Player can display its content full screen.

Custom frames

Most Windows applications should use the standard window frames. However, for immersive, full screen, stand-alone applications like games and kiosk applications, it may be appropriate to use custom frames for any windows that aren't shown full screen. The motivation to use custom frames should be to give the overall experience a unique feel, not just for branding.



Custom frames are appropriate for immersive, full screen, stand-alone applications such as games.

Guidelines

Window frames

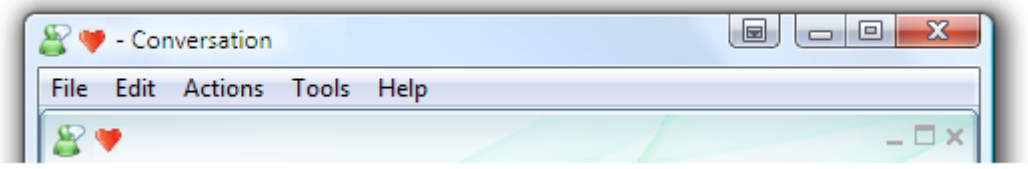
- Use standard window frames.
  - **Exception:** To give immersive full screen, stand-alone applications a unique feel:
    - Consider hiding the window frame of the primary window.
    - Consider using custom frames for secondary window.
    - If a custom frame is appropriate, choose a design that is lightweight and doesn't draw too much attention to itself.

      Incorrect:

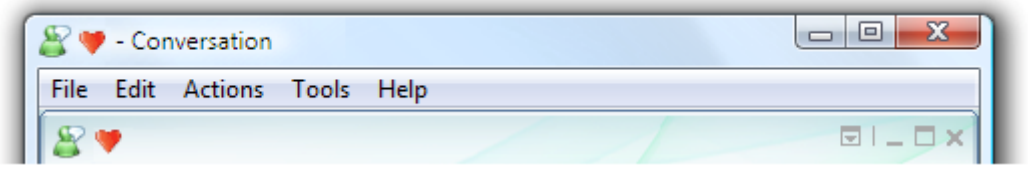
      In this example, the custom frame draws too much attention to itself.

- Don't add controls to a window frame. Put the controls within the window instead.

  Incorrect:

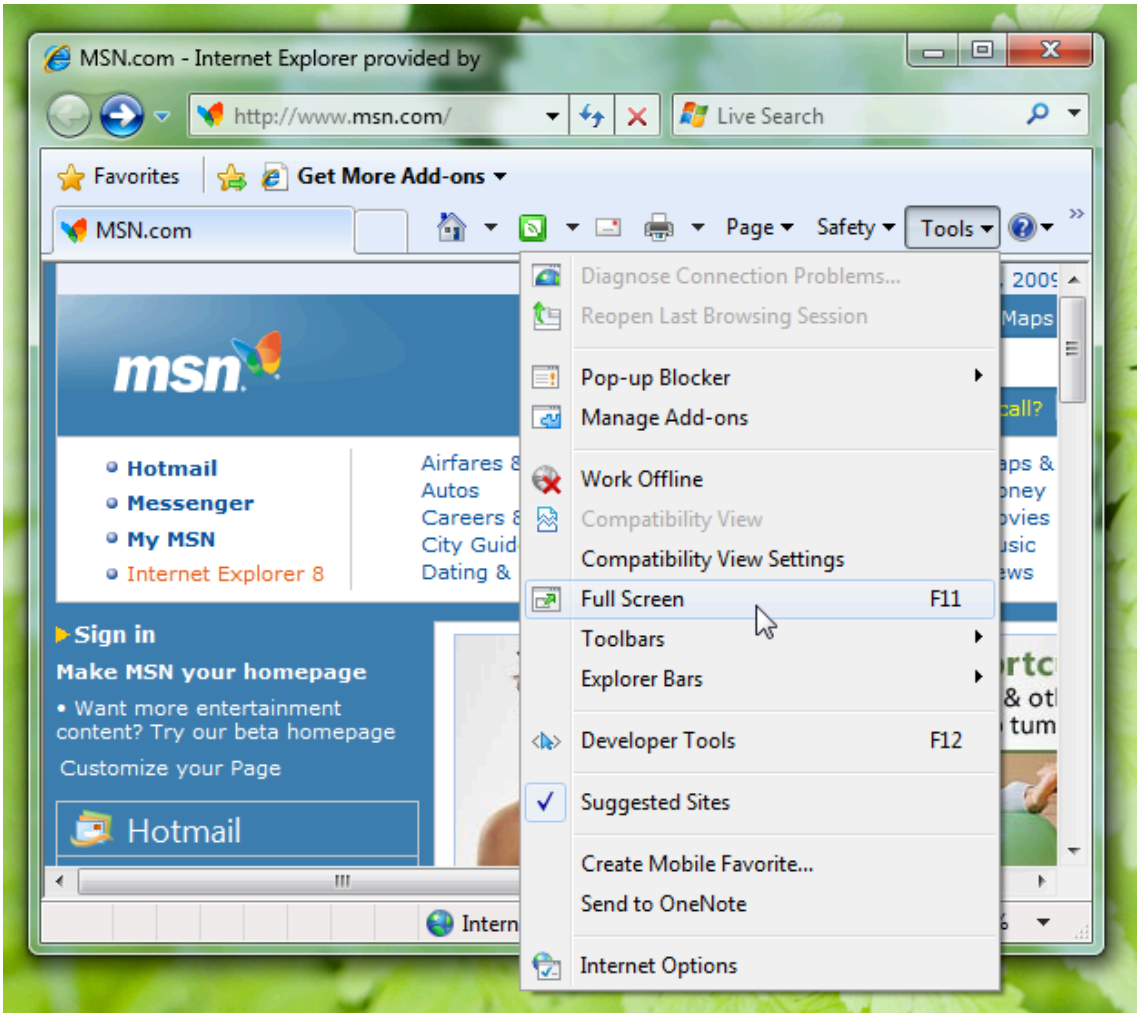
  Correct:

  In the correct example, the control is within the client area instead of the window frame.

Full screen mode

- For programs that have an optional full screen mode, to enable full screen mode:
  - Have a modal full screen command in the menu bar or toolbar. When the user clicks the command, show the command in its selected state.



    This example shows the full screen command along with its standard shortcut key.

- Use F11 for the full screen shortcut key.
- If there is a toolbar and full screen mode is commonly used, also have a graphic toolbar button with a Full screen tooltip.

    Examples of full screen toolbar buttons.

- To revert back from full screen mode:
  - Have a modal full screen command in the menu bar or toolbar. When the user clicks the command, show the command in its cleared state.

- Use F11 for the full screen shortcut key. If not already assigned, Esc can also be used for this purpose.

Glass

Standard window frames use glass automatically in Windows, but you can also use glass in regions that touch the window frame.

- **Consider using glass strategically in small regions touching the window frame without text.** Doing so can give a program a simpler, lighter, more cohesive look by making the region appear to be part of the frame.
-

- In this example, glass focuses the user's attention on the content instead of the controls.
- **Don't use glass in situations where a plain window background would be more attractive or easier to use.**

Correct:

In this example, glass is used to give the Alt+Tab window a lightweight appearance. Glass works for this window because it consists of graphics and a single, strong text label.

Incorrect:

In this incorrect example, the use of glass is distracting. A plain window background would be a better choice.

Feedback

Was this page helpful? Yes No

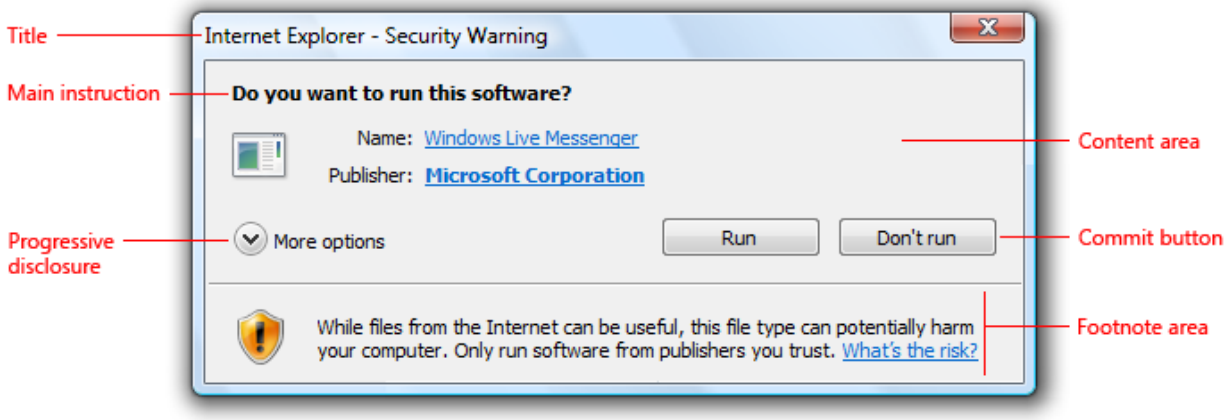
Provide product feedback | Get help at Microsoft Q&A

Windows 7 Dialog Boxes (Design basics)

Article • 02/08/2022

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

A dialog box is a secondary window that allows users to perform a command, asks users a question, or provides users with information or progress feedback.



A typical dialog box.

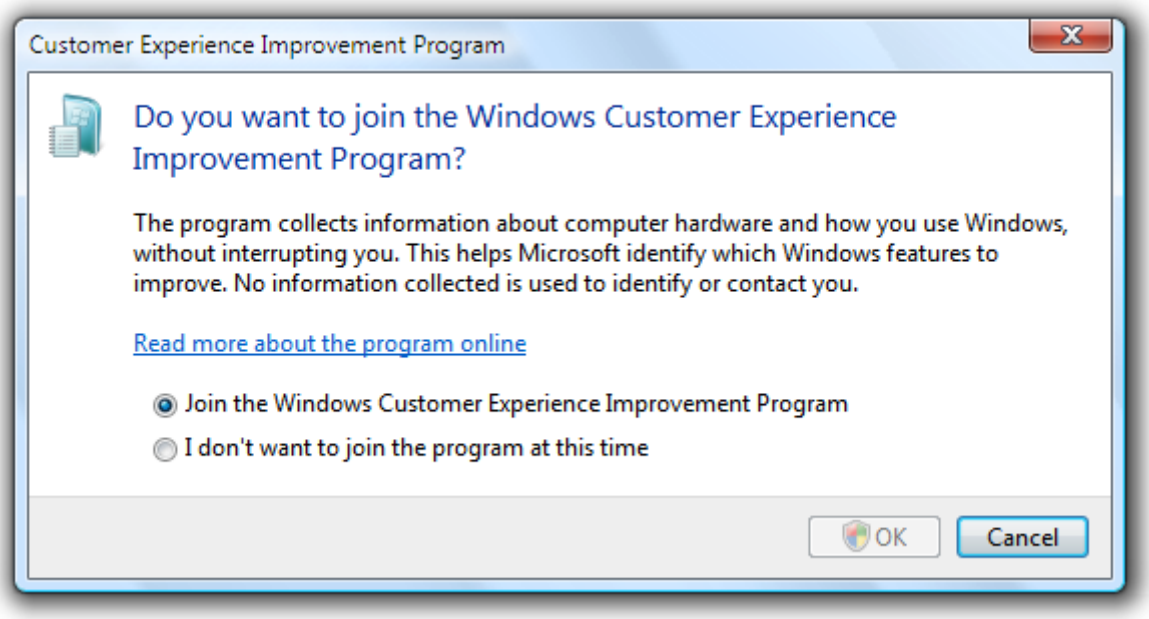
Dialog boxes consist of a title bar (to identify the command, feature, or program where a dialog box came from), an optional main instruction (to explain the user's objective with the dialog box), various controls in the content area (to present options), and commit buttons (to indicate how the user wants to commit to the task).

Dialog boxes have two fundamental types:

- **Modal dialog boxes** require users to complete and close before continuing with the owner window. These dialog boxes are best used for critical or infrequent, one-off tasks that require completion before continuing.
- **Modeless dialog boxes** allow users to switch between the dialog box and the owner window as desired. These dialog boxes are best used for frequent, repetitive, on-going tasks.

A task dialog is a dialog box implemented using the task dialog application programming interface (API). They consist of the following parts, which can be assembled in a variety of combinations:

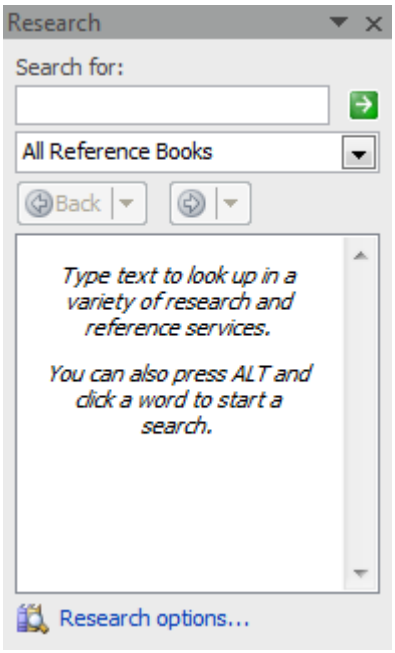
- A **title bar** to identify the application or system feature where the dialog box came from.
- A **main instruction**, with an optional icon, to identify the user's objective with the dialog.
- A **content area** for descriptive information and controls.
- A **command area** for commit buttons, including a Cancel button, and optional More options and Don't show this <item> again controls.
- A **footnote area** for optional additional explanations and help, typically targeted at less experienced users.



A typical task dialog.

Task dialogs are recommended whenever appropriate because they are easy to create and they achieve a consistent look. Task dialogs do require Windows Vista or later, so they aren't suitable for earlier versions of Microsoft Windows.

A task pane is like a dialog box, except that it is presented within a window pane instead of a separate window. As a result, task panes have a more direct, contextual feel than dialog boxes. Even though technically they are not the same, **task panes are so similar to dialog boxes that their guidelines are presented in this article**.



A typical task pane.

Property windows are a specialized type of dialog box used to view and change properties for an object, collection of objects, or a program. Additionally, property windows typically support several tasks, whereas dialog boxes typically support a single task or step in a task. Because their usage is specialized, **property windows are covered in a different set of guidelines**.

Dialog boxes can have tabs, and if so they are called tabbed dialog boxes. Property windows are determined by their presentation of properties, not by the use of tabs.

Note: Guidelines related to layout, window management, common dialog boxes, property windows, wizards, confirmations, error messages, and warning messages are presented in separate articles.

Is this the right user interface?

To decide, consider these questions:

- **Is the purpose to provide users with information, ask users a question, or allow users to select options to perform a command or task?** If not, use another user interface (UI).
- **Is the purpose to view and change properties for an object, collection of objects, or a program?** If so, use a property window or toolbar instead.
- **Is the purpose to present a collection of commands or tools?** If so, use a toolbar or palette window.
- **Is the purpose to verify that the user wants to proceed with an action?** Is there a clear reason not to proceed and a reasonable chance that sometimes users won't?

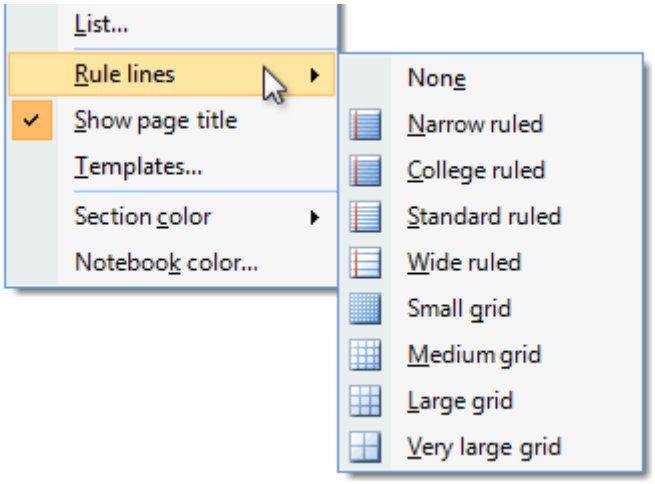
If so, use a confirmation.

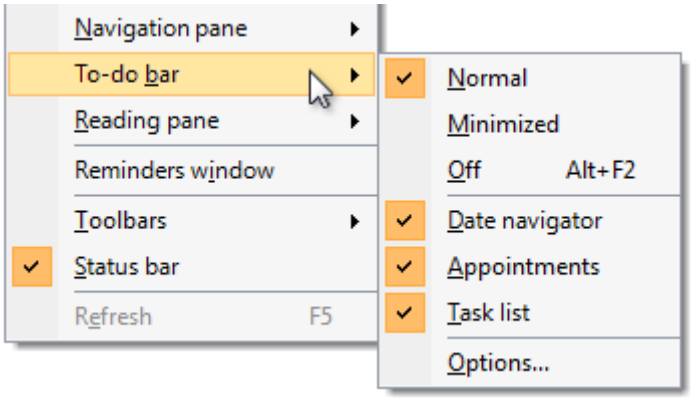
- **Is the purpose to give an error or warning message?** If so, use an error message or warning message.
- Is the purpose to:
  - Open files
  - Save files
  - Open folders
  - Find or replace text
  - Print a document
  - Select attributes of a printed page
  - Select a font
  - Choose a color
  - Browse for a file, folder, computer, or printer
  - Search for users, computers, or groups in Microsoft Active Directory
  - Prompt for a user name and password?

If so, use the appropriate common dialog instead. Many of these common dialogs are extensible.

- **Is the purpose to perform a multi-step task that requires more than a single window?** If so, use a task flow or wizard instead.
- **Is the purpose to inform users of a system or program event that isn't related to the current user activity, that doesn't require immediate user action, and users can freely ignore?** If so, use a notification instead.
- **Is the purpose to show program status?** If so, use a status bar instead.
- **Would it be preferable to use in-place UI?** Dialog boxes can break the user's flow by demanding attention. Sometimes that break in flow is justified, such as when the user must perform an action that is outside the current context. In other cases, a better approach is to present the UI in context, either directly with in-place UI (such as a task pane), or on demand using progressive disclosure.
- **Is the purpose to display a non-critical user input problem or special condition?** If so, use a balloon instead.
- **For task flows, would it be preferable to use another page?** Generally you want a task to flow from page to page within a single window. Use dialog boxes to confirm in-place commands, to get input for in-place commands, and to perform secondary, stand-alone tasks that are best done independently and outside the main task flow.
- **For selecting options, are users likely to change the options?** If not, consider alternatives, such as:
  - Using the default options without asking, but allowing users to make changes later.

    - Providing a version with options (for example, **Print...** in a menu) as well as a version without options (for example, **Print** in the toolbar). Generally, toolbar commands should be immediate and avoid displaying dialog boxes.
- **For selecting options, is there a simpler, more direct way to present the options?** If so, consider alternatives, such as:
    - Using a split button to select variations of a command.
    - Using a submenu for commands, check boxes, radio buttons and simple lists.





In these examples, submenus are used instead of dialog boxes for simple selections.

Design concepts

When properly used, dialog boxes are a great way to give power and flexibility to your program. When misused, dialog boxes are an easy way to annoy users, interrupt their flow, and make the program feel indirect and tedious to use. **Modal dialog boxes demand users' attention.** Dialog boxes are often easier to implement than alternative UIs, so they tend to be overused.

A dialog box is most effective when its design characteristics match its usage. A dialog box's design is largely determined by its purpose (to offer options, ask questions, provide information or feedback), type (modal or modeless), and user interaction (required, optional response, or acknowledgment), whereas its usage is largely

determined by its context (user or program initiated), probability of user action, and frequency of display.

To design effective dialog boxes, use the following elements effectively:

- Dialog box text
- Main instructions
- Don't show this <item> again option

If you do only one thing...

Make sure that your dialog box design (determined by its purpose, type, and user interaction) matches its usage (determined by its context, probability of user action, and frequency of display).

Usage patterns

Dialog boxes have several usage patterns:

- Question dialogs (using buttons) ask users a single question or to confirm a command, and use simple responses in horizontally arranged command buttons.
- Question dialogs (using command links) ask users a single question or to select a task to perform, and use detailed responses in vertically arranged command links.
- Choice dialogs present users with a set of choices, usually to specify a command more completely. Unlike question dialogs, choice dialogs can ask multiple questions.
- Progress dialogs present users with progress feedback during a lengthy operation (longer than five seconds), along with a command to cancel or stop the operation.
- Informational dialogs display information requested by the user.

Guidelines

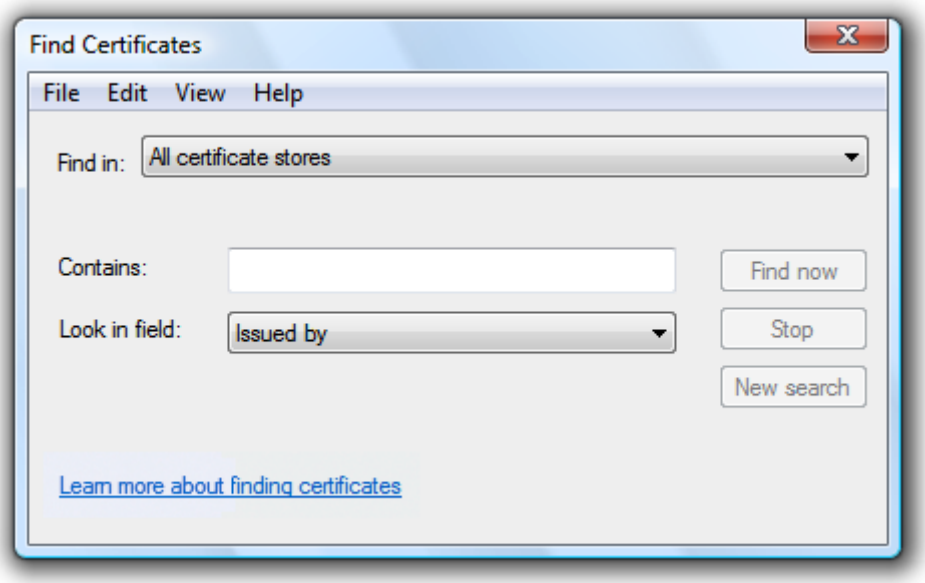
General

- **Don't use scrollable dialog boxes.** Don't use dialog boxes that require the use of a scroll bar to be viewed completely during normal usage. Redesign the dialog box instead. Consider using progressive disclosure or tabs.

- **Don't have a menu bar or status bar.** Instead, provide access to commands and status directly on the dialog box itself, or by using context menus on the relevant controls.

- **Exception:** Menu bars are acceptable when a dialog box is used to implement a primary window (such as a utility).

Incorrect:



In this example, Find Certificates is a modeless dialog box with a menu bar.

- If a dialog box requires immediate attention and the program isn't active, **flash its taskbar button three times to draw attention, and leave it highlighted.** Don't do anything else: don't restore or activate the window and don't play any sound effects. Instead, respect the user's window state selection and let the user activate the window when ready.

- For more guidelines and examples, see Taskbar.

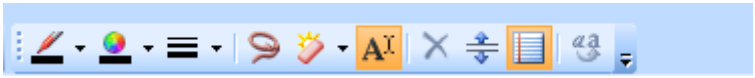
Modal dialog boxes

- **Use for critical or infrequent, one-off tasks that require completion before continuing.**
- Use a delayed commit model so that changes don't take effect until explicitly committed.
- **Implement using a task dialog whenever appropriate to achieve a consistent look.** Task dialogs do require Windows Vista or later, so they aren't suitable for earlier versions of Windows.

Modeless dialog boxes

- **Use for frequent, repetitive, on-going tasks.**
- Use an immediate commit model so that changes take effect immediately.

- For modeless dialogs, use an explicit Close command button in the dialog to close the window. For both, use a Close button on the title bar to close the window.
- **Consider making modeless dialog boxes dockable.** Dockable modeless dialogs allow for more flexible placement.



Some modeless dialog boxes used in Microsoft Office are dockable.

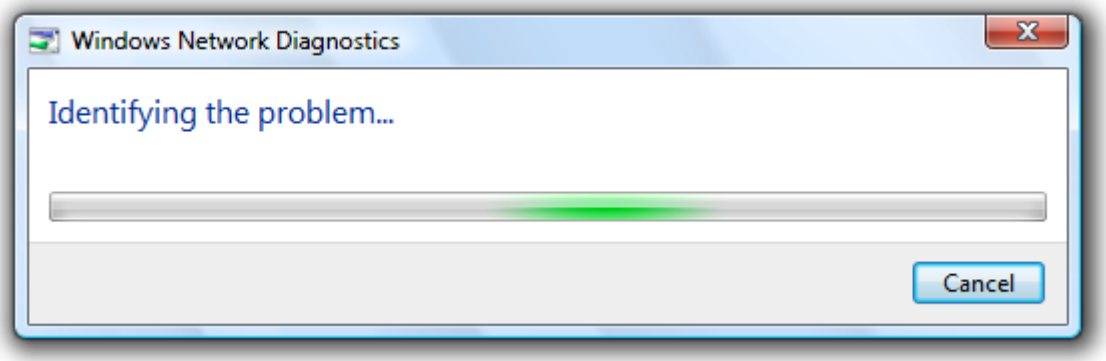
Multiple dialog boxes

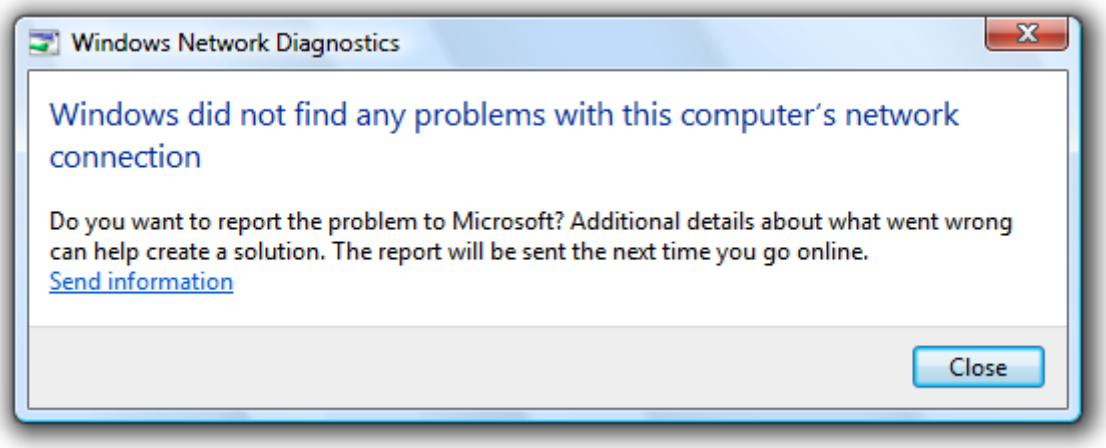
- **Don't display more than one owned choice dialog at a time from an owner choice dialog.** Displaying more than one makes the meaning of the commit buttons difficult for users to understand. You may display other types of dialog boxes (such question dialogs) as needed.
- **For a sequence of related dialogs, consider using a multi-page dialog if possible.** Use individual dialogs if they aren't clearly related.

Multi-page dialog boxes

- Use a multi-page dialog box instead of individual dialog boxes when you have the following sequence of related pages:
  - A single input page (optional)
  - A progress page
  - A single results page

The input page is optional because the task may have been initiated somewhere else. **Doing so gives the resulting experience a stable, simple, lightweight feel.**





In this example, Windows Network Diagnostics consists of progress and results pages.

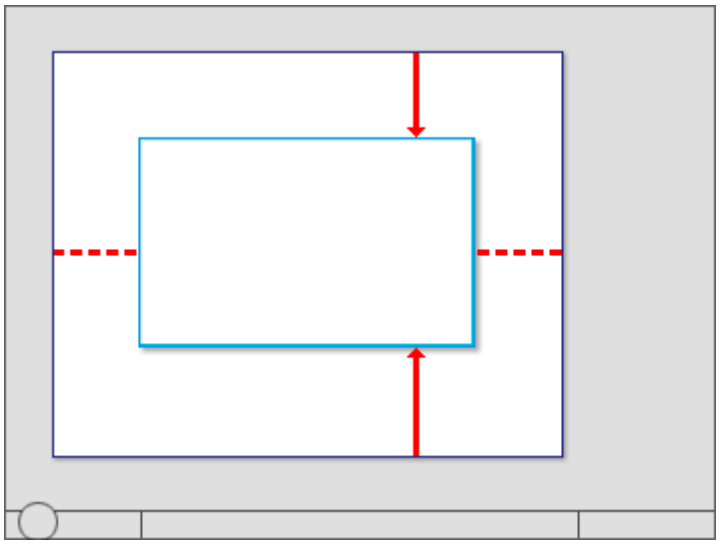
- **Don't use a multi-page dialog if the input page is a standard dialog.** In this case the consistency of using a standard dialog is more important.
- **Don't use Next or Back buttons and don't have more than three pages.** Multi-page dialog boxes are for single-step tasks with feedback. They aren't wizards, which are used for multi-step tasks. Wizards have a heavy, indirect feel compared to multi-page dialog boxes.
- **On the input page, use specific command buttons or command links to initiate the task.**
- **Use a Cancel button on the input and progress pages, and a Close button on the results page.**

Developers: You can create multi-page task dialogs using the TDM_NAVIGATE_PAGE message.

Presentation

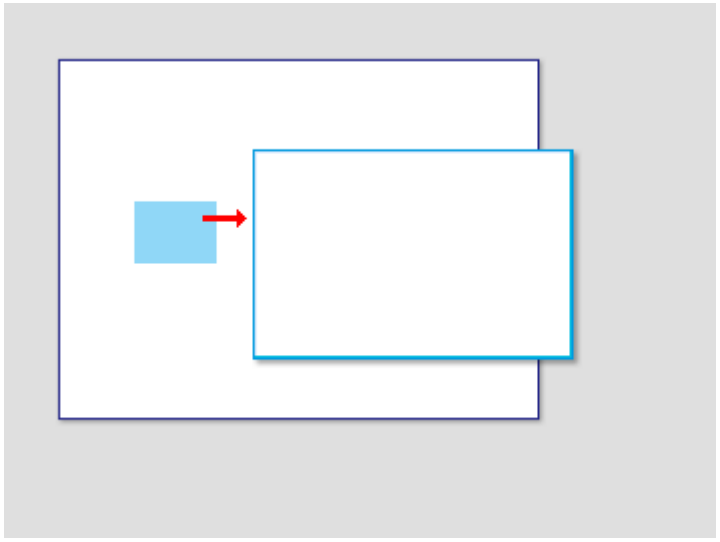
To make dialog boxes easy to find and access, clearly associate the dialog with its source, and work well with multiple monitors:

- **Initially display dialogs "centered" on top of the owner window.** For subsequent display, consider displaying it in its last location (relative to the owner window) if doing so is likely to be more convenient.



Initially center dialogs on top of the owner window.

- **If a dialog is contextual, display it near the object from which it was launched.** However, place it out of the way (preferably offset down and to the right) so that the object isn't covered by the dialog.



An object's properties are displayed near to the object.

- **For modeless dialogs, display initially on top of the owner window to make it easy to find.** If the user activates the owner window, that may obscure the modeless dialog.
- **If necessary, adjust the initial location so that the entire dialog is visible within the target monitor.** If a resizable window is larger than the target monitor, reduce it to fit.
- **When a dialog is redisplayed, consider displaying it in the same state as last accessed.** On close, save the monitor used, window size, location, and state (maximized vs. restore). On redisplay, restore the saved dialog size, location, and state using the appropriate monitor. Also, consider making these attributes persist across program instances on a per-user basis.

- **For resizable windows, set a minimum window size if there is a size below which the content is no longer usable.** Consider altering the presentation to make the content usable at smaller sizes.

In this example, Windows Media Player changes its format when the window becomes too small for the standard format.

- **Don't use the Always on Top attribute.**
  - **Exception:** Use only when a dialog box implements an essentially modal operation, but it needs to be suspended briefly to access the owner window. For example, when spell-checking a document, users may occasionally leave the spell check dialog box and access the document to correct errors.

For more information and examples, see Window Management.

Title bars

- **Dialog boxes don't have title bar icons.** Title bar icons are used as a visual distinction between primary windows and secondary windows.
  - **Exception:** If a dialog box is used to implement a primary window (such as a utility) and therefore appears on the taskbar, it does have a title bar icon. In this case, optimize the title for display on the taskbar by concisely placing the distinguishing information first.
- **Dialog boxes always have a Close button.** Modeless dialogs can also have a Minimize button. Resizable dialogs can have a Maximize button.
- **Don't disable the Close button.** Having a Close button helps users stay in control by allowing them to close windows they don't want.
  - **Exception:** For progress dialogs, you may disable the Close button if the task must run to completion to achieve a valid state or prevent data loss.
- **The Close button on the title bar should have the same effect as the Cancel or Close button** within the dialog box. Never give it the same effect as OK.
- If the title bar caption and icon are already displayed in a prominent way near the top of the window, you can hide the title bar caption and icon to avoid redundancy. However, you still have to set a suitable title internally for use by Windows.

Interaction

- **When displayed, user initiated dialog boxes should always take input focus.** Program initiated dialog boxes shouldn't take input focus because the user may be interacting with another window. Such interaction misdirected at the dialog box may have unintended consequences.

- **Assign initial input focus to the control that users are most likely to interact with first**, which is usually (but not always) the first interactive control. Avoid assigning initial input focus to a Help link.

- **For keyboard navigation, tab order should flow in a logical order, generally from left to right, top to bottom.** Usually tab order follows reading order, but consider making these exceptions:
  - Put the most commonly used controls earlier in tab order.
  - Put Help links at the bottom of a dialog box, after the commit buttons in tab order.

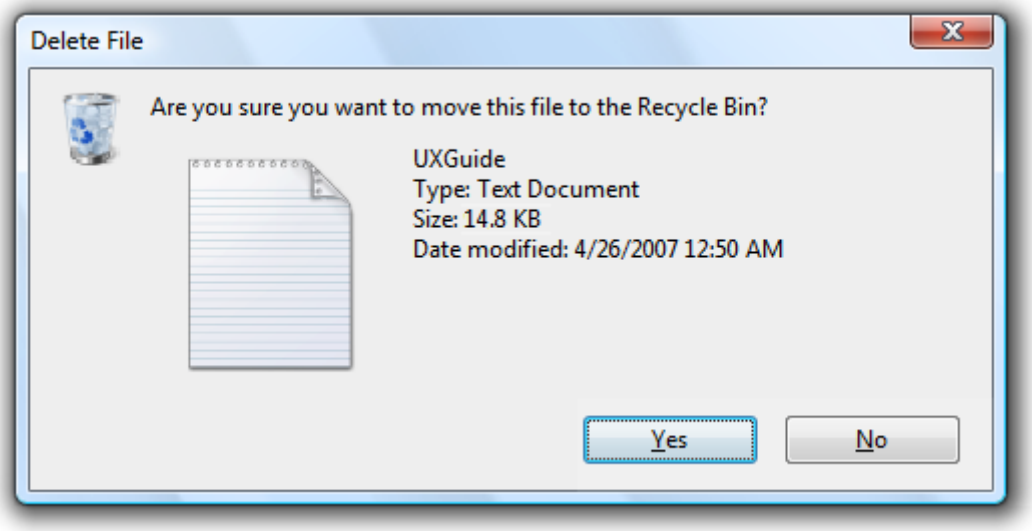
  When assigning order, assume that users display dialog boxes for their intended purpose; so, for example, users display choice dialogs to make choices, not to review and click Cancel.

- **Pressing the Esc key always closes an active dialog box.** This is true for dialog boxes with Cancel or Close, and even if Cancel has been renamed to Close because the results can no longer be undone.

Access keys

- **Whenever possible, assign unique access keys to all interactive controls or their labels.**Read-only text boxes are interactive controls (because users can scroll them and copy text) so they benefit from access keys. **Don't assign access keys to:**

  - **OK, Cancel, and Close buttons.** Enter and Esc are used for their access keys. However, always assign an access key to a control that means OK or Cancel, but has a different label.



In this example, the positive commit button has an access key assigned.

- **Group labels.** Normally, the individual controls within a group are assigned access keys, so the group label doesn't need one. However, if there is a shortage of access keys, assign an access key to the group label and not the individual controls.
- **Generic Help buttons,** which are accessed with F1.
- **Link labels.** There are often too many links to assign unique access keys, and the underscores often used to signify links hide the access key underscores. Access links with the Tab key instead.
- **Tab names.** Tabs are cycled using Ctrl+Tab and Ctrl+Shift+Tab.
- **Browse buttons labeled "...".** These Browse buttons can't be assigned access keys uniquely.
- **Unlabeled controls,** such as spin controls, graphic command buttons, and unlabeled progressive disclosure controls.
- **Non-label static text or labels for controls that aren't interactive,** such as progress bars.

- **Whenever possible, assign access keys for commonly used commands according to the Standard Access Key Assignments**. While consistent access key assignments aren't always possible, they are certainly preferred especially for frequently used dialog boxes.
- **Assign commit button access keys first to ensure that they have the standard key assignments.** If there isn't a standard key assignment, use the first letter of the

first word. For example, the access key for Yes and No commit buttons should always be "Y" and "N", regardless of the other controls in the dialog box.

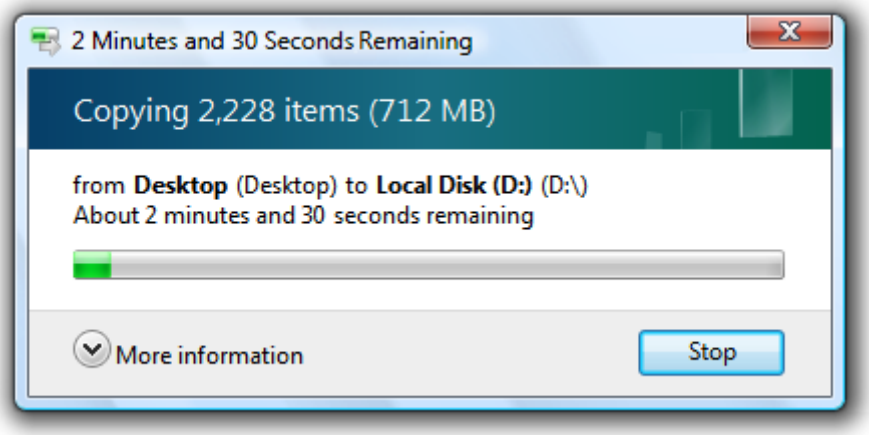
- **To make access keys easy to find, assign the access keys to a character that appears early in the label,** ideally the first character, even if there is a keyword that appears later in the label.
- **Prefer characters with wide widths,** such as w, m, and capital letters.
- **Prefer a distinctive consonant or a vowel,** such as "x" in Exit.
- **Avoid using characters that make the underline difficult to see,** such as (from most problematic to least problematic):
  - Letters that are only one pixel wide, such as i and l.
  - Letters with descenders, such as g, j, p, q, and y.
  - Letters next to a letter with a descender.

For more guidelines and examples, see Keyboard.

Progress dialogs

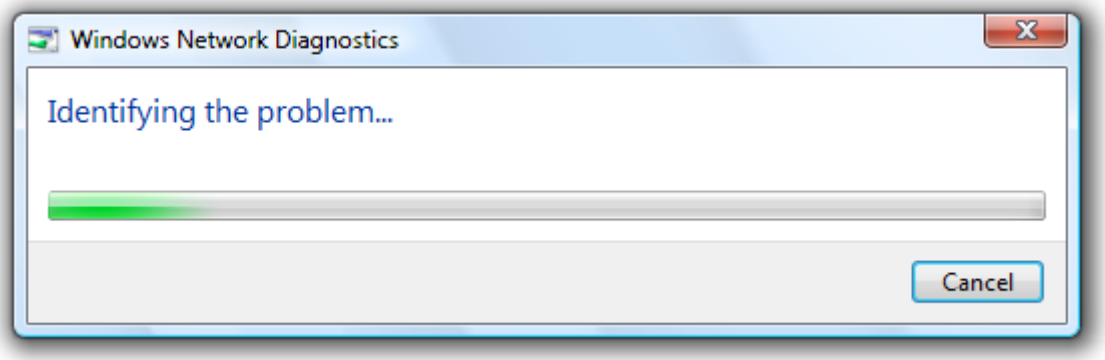
For long-running tasks, **assume that users will do something else while the task is completing**. Design the task to run unattended.

- **Present users with progress feedback dialog box if an operation takes longer than five seconds to complete**, along with a command to cancel or stop the operation.
  - **Exception:** For wizards and task flows, use a modal dialog for progress only if the task stays on the same page (as opposed to advancing to another page) and users can't do anything while waiting. Otherwise, use a progress page or in-place progress.
- If the operation is a long-running task (over 30 seconds) and can be performed in the background, use a modeless progress dialog so that users can continue to use your program while waiting.
- Modeless progress dialogs:
  - Have a Minimize button on the title bar.
  - Are displayed on the taskbar.
- Implement modeless progress dialogs so that they continue to run even if the owner window is closed.



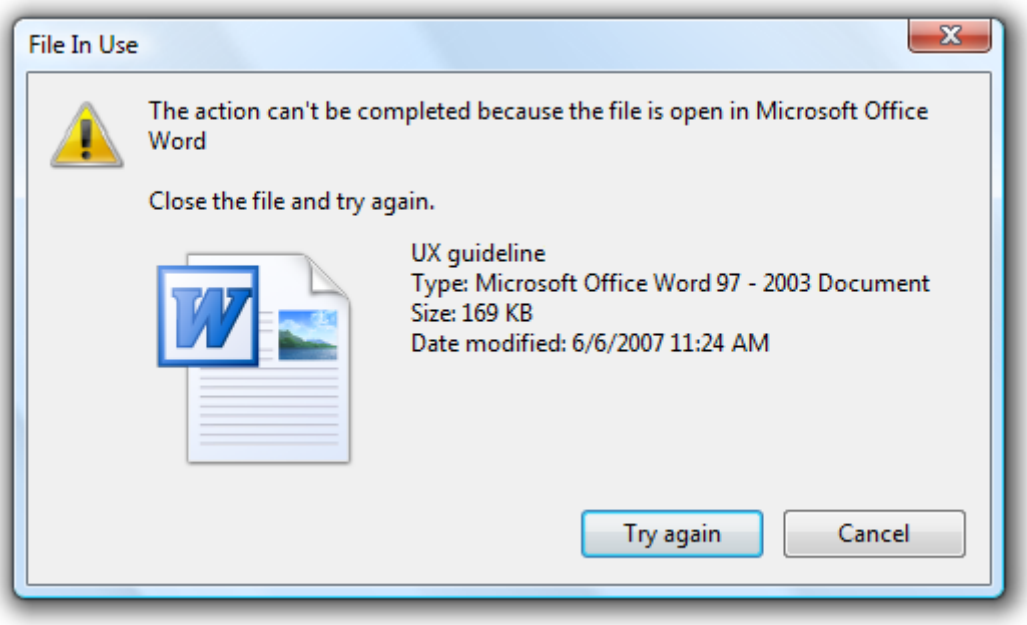
In this example, the file copy continues even if the owner window is closed.

- **Provide a command button to halt the operation if it takes more than a few seconds to complete, or has the potential never to complete.** Label the button Cancel if canceling returns the environment to its previous state (leaving no side effects); otherwise, label the button Stop to indicate that it leaves the partially completed operation intact. You can change the button label from Cancel to Stop in the middle of the operation, if at some point it isn't possible to return the environment to its previous state.



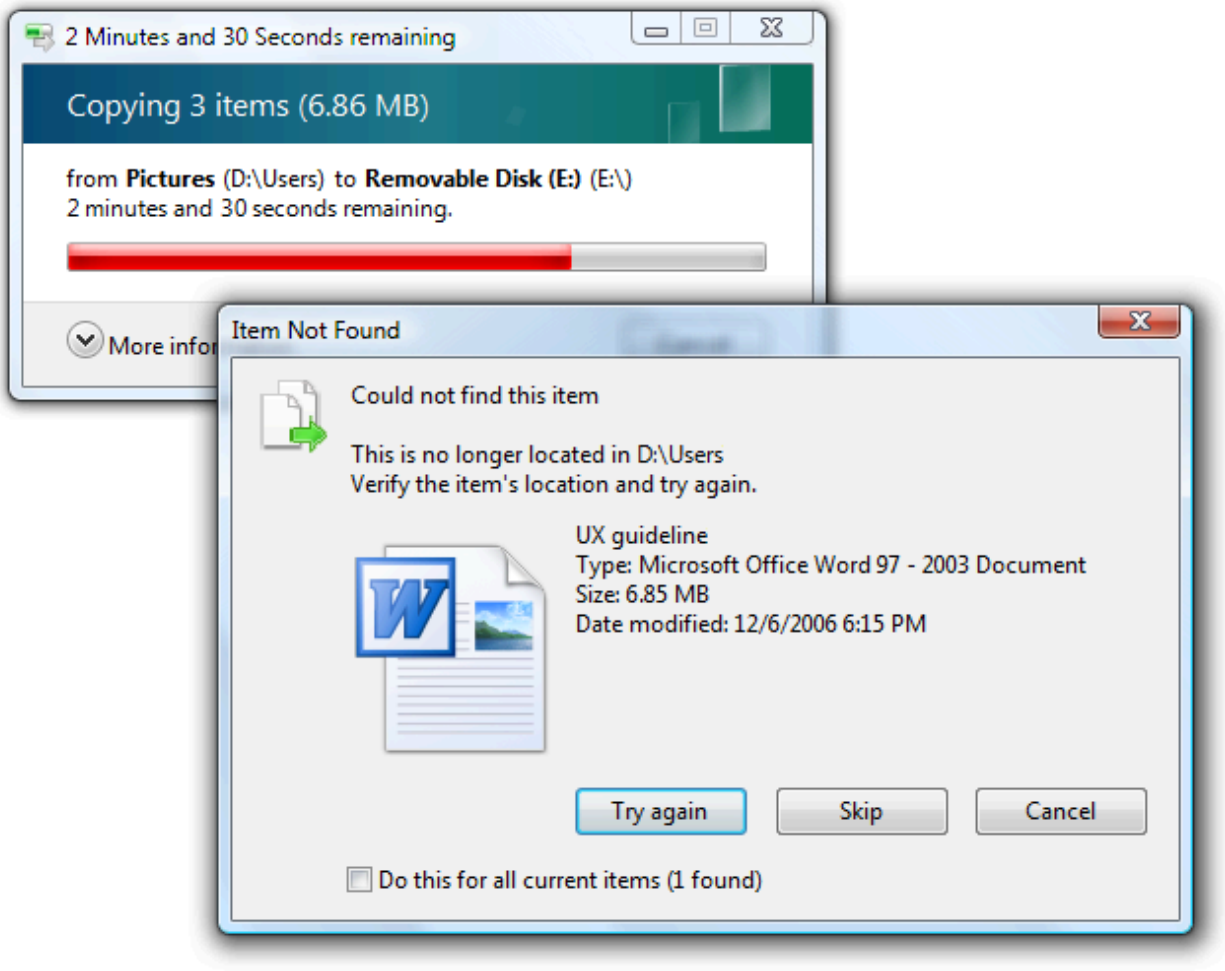
In this example, halting the problem diagnosis has no side effect.

- **Provide a command button to pause the operation if it takes more than several minutes to complete, and it impairs users' ability to get work done.** Doing so doesn't force the user to choose between completing the task and getting their work done.
- **Gather as much information as you can before starting the task.**
- **If recoverable problems are detected, have users deal with all problems found at the end of the task.** If that isn't practical, have users deal with problems as they happen.
- **Don't abandon tasks as the result of recoverable errors.**



In this example, Windows Explorer allows users to continue with the task after a recoverable error.

- **Indicate problems by turning the progress bar red.**



In this example, a removable disk was removed during a file copy.

- **If the results are clearly apparent to users, close the progress dialog automatically on successful completion.** Otherwise, use feedback only to report problems:
  - To display simple feedback, display the feedback in the progress dialog, and change the Cancel button to Close.
  - To display detailed feedback, close the progress dialog box and display an informational dialog.

Don't use a notification for completion feedback. Use either a progress dialog or an action success notification, but not both.

Time remaining

- **Use the following time formats.** Start with the first of the following formats where the largest time unit isn't zero, then change to the next format once the largest time unit becomes zero.

For progress bars:

If related information is shown in a colon format:

Time remaining: h hours, m minutes

Time remaining: m minutes, s seconds

Time remaining: s seconds

If screen space is at a premium:

h hrs, m mins remaining

m mins, s secs remaining

s seconds remaining

Otherwise:

h hours, m minutes remaining

m minutes, s seconds remaining

s seconds remaining

For title bars:

hh:mm remaining

mm:ss remaining

0:ss remaining

This compact format shows the most important information first so that it isn't truncated on the taskbar.

- **Make estimates accurate, but don't give false precision.** If largest unit is hours, give minutes (if meaningful) but not seconds.

Incorrect:

hh hours, mm minutes, ss seconds

- **Keep the estimate up-to-date.** Update time remaining estimates at least every 5 seconds.
- **Focus on the time remaining** because that is the information users care about most. Give total elapsed time only when there are scenarios where elapsed time is helpful (such as when the task is likely to be repeated). If the time remaining estimate is associated with a progress bar, don't have percent complete text because that information is conveyed by the progress bar itself.
- **Be grammatically correct.** Use singular units when the number is one.

Incorrect:

1 minutes, 1 seconds

- **Use sentence-style capitalization.**

For more information and examples, see Progress Bars.

Icons and graphics

Graphics

- **Don't use large graphics that serve no purpose beyond filling space with eye candy.** Keep the appearance simple instead.

Incorrect:

In this example, the large graphic serves no purpose.

Title bar icons

- **Dialog boxes don't have title bar icons.**
  - **Exception:** If a dialog box is used to implement a primary window (such as a utility) and therefore appears on the taskbar, it does have a title bar icon.

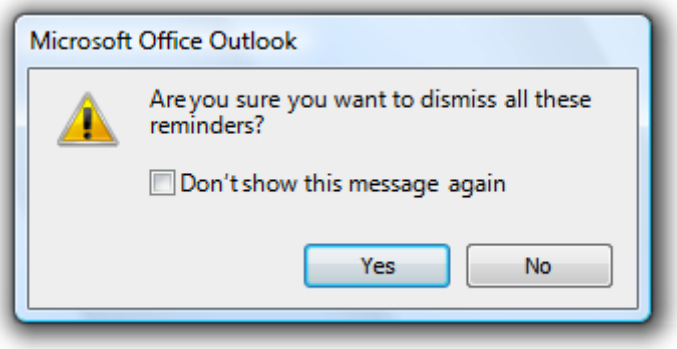
Body icons

- **Choose the body icon based on the design pattern:**

Expand table

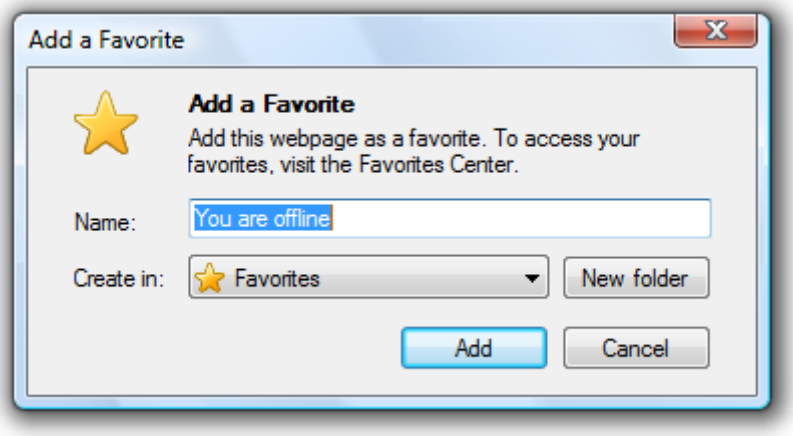
Pattern	Body icon
Question dialogs	Program, feature, object, warning icon (if potential loss of data or system access), security warning, or none.
Choice dialogs	None.
Progress dialogs	None (but may have an animation).
Informational dialogs	None.

- **Incorrect:**



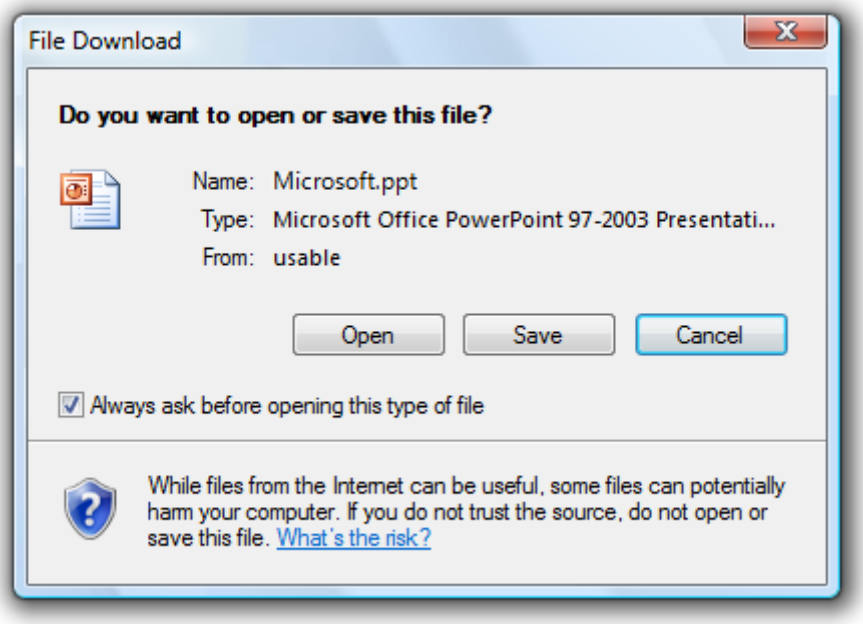
In this example, a warning icon is incorrectly used for a question that doesn't involve potential loss of data or system access.

- **Consider using icons to help users visually recognize your program's features.** This technique is most effective when the icons are easily recognizable and used in several locations within your program.



In this example, the yellow star icon represents Favorites. The icon is easily recognizable and is used consistently throughout Windows to represent Favorites.

- **Use icons to help users recognize the object in question.**



In this example, the object's icon helps users recognize the type of file being opened or saved.

- **Consider using icons to help make features self-explanatory.**

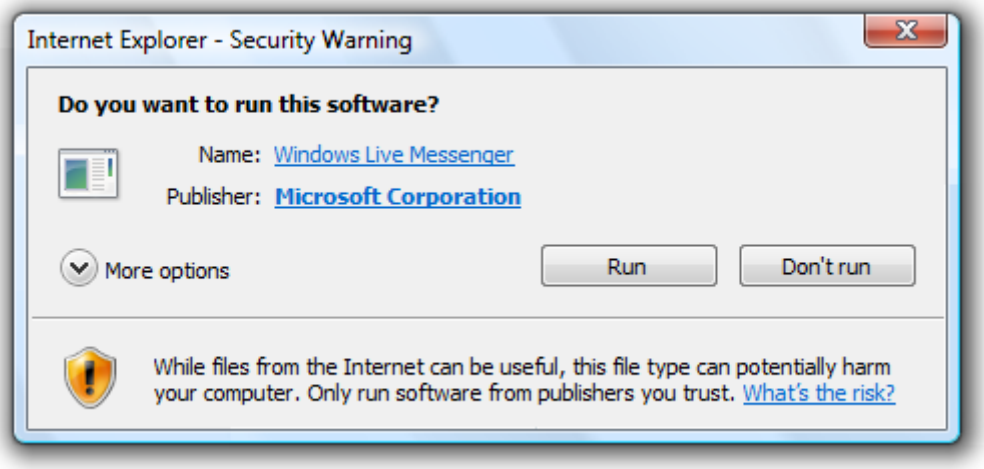
In this example, these icons help users visualize the effect of their features.

- **Use an icon in About Box dialogs for application branding.**

In this example, a bitmap is used in the About Box to identify and brand the application.

Footnote icons

- **If you have a footnote, consider using a footnote icon to summarize the footnote's subject.**



In this example, the footnote icon indicates that the question has security implications.

- **Don't use a footnote icon that repeats the body icon.**
- **Don't use the error or information standard icons.** Error conditions must be conveyed through the body icon and footnotes are always for information, making the information icon redundant. However, you can use the standard warning icon and the yellow security shield to alert users of risky consequences.

For more information and examples, see Icons.

Commit buttons

Notes:

- These guidelines don't apply to question dialogs using command links, because that pattern uses command links instead of buttons.
- [Do it] and [Don't do it] are affirmative and negative responses, respectively, to the main instruction.

General

- **Choose the commit buttons based on the design pattern:**

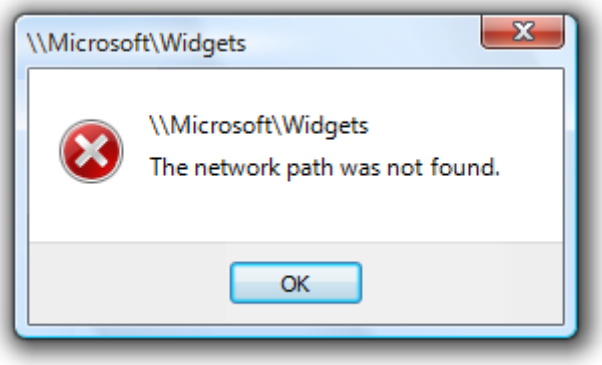
Expand table

Label	Value
Pattern	**Commit buttons**
Question dialogs (using buttons)	One of the following sets of concise commands: Yes/No, Yes/No/Cancel, [Do it]/Cancel, [Do it]/[Don't do it], [Do it]/[Don't do it]/Cancel.
Question dialogs (using links)	Cancel.
Choice dialogs	• Modal dialogs: OK/Cancel or [Do it]/Cancel • Modeless dialogs: Close button on dialog box and title bar • Task pane: Close button on title bar
Progress dialogs	Use Cancel if returns the environment to its previous state (leaving no side effect); otherwise, use Stop.
Informational dialogs	Close.

- **All commit buttons except Apply result in closing the dialog box window.**
- **Don't confirm commit buttons.** Doing so unnecessarily can be very annoying. **Exceptions:**
  - The action is potentially catastrophic.
  - The action is clearly inconsistent with other actions.
  - If incorrect, the action may result in a significant loss of data, time, or effort on behalf of the user.

  For more guidelines and examples, see Confirmations.
- **Don't disable commit buttons. Exceptions:**
  - **If users must elevate to make a change, disable the positive commit buttons until the user makes a change.** Doing so prevents users from elevating just to close a window by forcing them to click Cancel.
  - For more exceptions, see Disabling or removing controls vs. giving error messages.
- **Right-align commit buttons in a single row across the bottom of the dialog box,** but above the footnote area. Do this even if there is a single commit button (such as OK).

  Incorrect:



In this example, the OK button is incorrectly centered.

- **Present the commit buttons in the following order:**

  1. OK/[Do it]/Yes
  2. [Don't do it]/No
  3. Cancel
  4. Apply (if present)
  5. Help (if present)

- **If you have many related commit buttons, consolidate them using split buttons**.

- **Have a clear separation from commit buttons (which close the window) and all other command buttons (such as Advanced).**

Responding to main instructions

- **Use positive commit buttons that are specific responses to the main instruction, instead of generic labels such as OK or Yes/No.** Users should be able to understand the options by reading the button text alone. **Exceptions:**

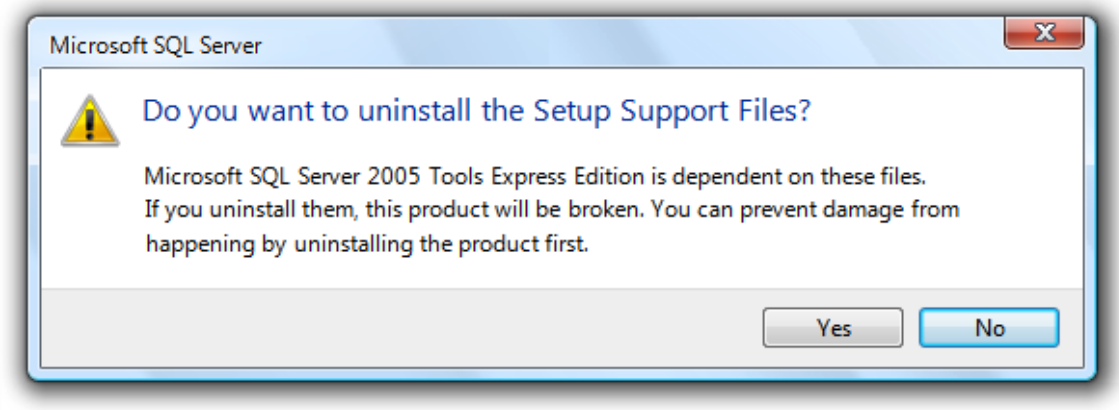
  - Use Close for dialogs that don't have settings, such as informational dialogs. Never use Close for dialogs that have settings.

  - Use OK when the "specific" responses are still generic, such as Save, Select, or Choose. Use OK when changing a specific setting or a collection of settings.

  - **For legacy dialog boxes without a main instruction, you can use generic labels such as OK.** Often such dialog boxes aren't designed to perform a specific task, preventing more specific responses.

  - Certain tasks require more thought and careful reading for users to make informed decisions. This is usually the case with confirmations. **In such cases, you can purposely use generic commit button labels to force users to read the main instructions and prevent hasty decisions.**

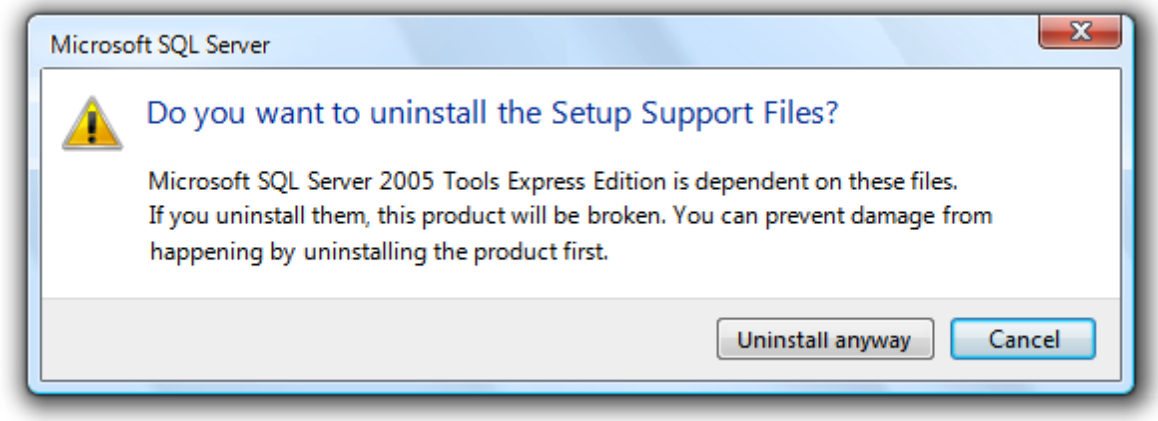
Correct:



In this example, using Yes/No commit buttons forces users to at least read the main instruction.

- **Alternatively, you can add the word "anyway" to the positive commit button label to indicate that the dialog box presents a reason not to proceed** and that users should read the dialog carefully before proceeding.

  Correct:

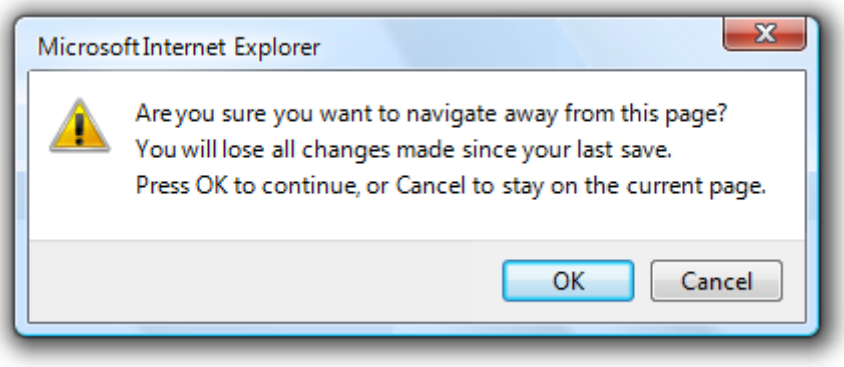


  In this example, "anyway" is added to the commit button label to indicate that users should proceed carefully.

- **Use Cancel or Close for negative commit buttons instead of specific responses to the main instruction.** Quite often users realize that they don't want to perform a task once they see a dialog box. If Cancel or Close were relabeled to specific responses, users would have to carefully read all the commit buttons to determine how to cancel. **Labeling Cancel and Close consistently makes them easy to find.Exceptions:**
  - **Don't use Yes/Cancel.** Always use Yes/No as a pair.
  - **Use a specific response when Cancel is ambiguous.**

- **Don't map generic labels to their specific meaning with text in the content area.** Instead, use specific commit button labels, or a question dialog using links if the labels are lengthy.

  Incorrect:

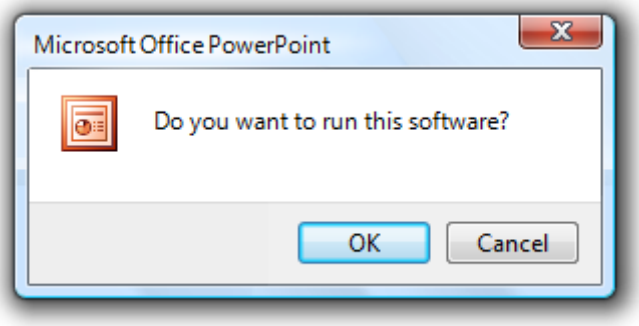
  In this example, OK is mapped to Continue, Cancel is mapped to Remain on Page.

Yes and No buttons

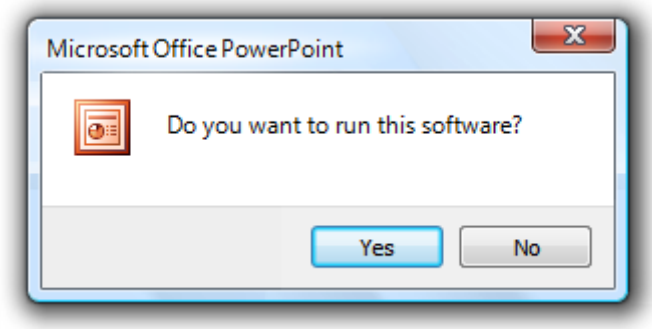
- **Prefer specific responses to Yes and No buttons.** While there's nothing wrong with using Yes and No, specific responses can be understood more quickly, resulting in efficient decision making. However, confirmations usually have Yes and No buttons to make users give the confirmation some thought before responding.

- **Use Yes and No buttons only to respond to yes or no questions.** The main instruction should be naturally expressed as a yes or no question. Never use OK and Cancel for yes or no questions.

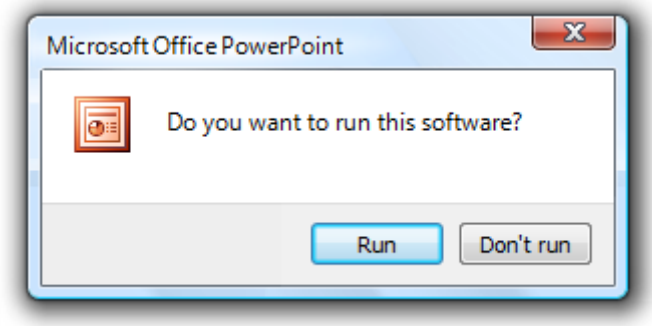
  Incorrect:

  Correct:



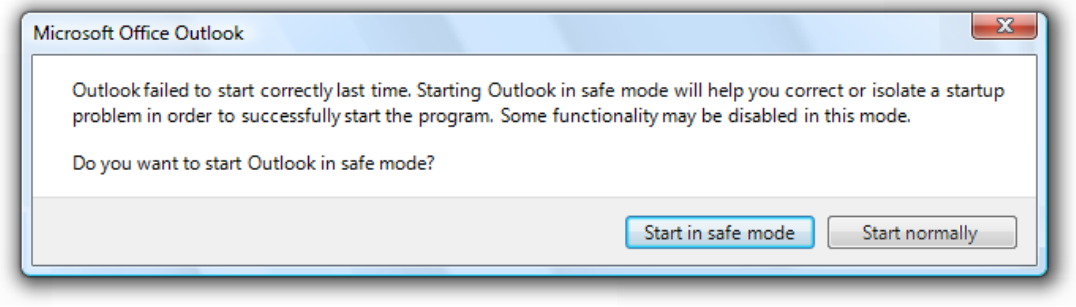
Better:



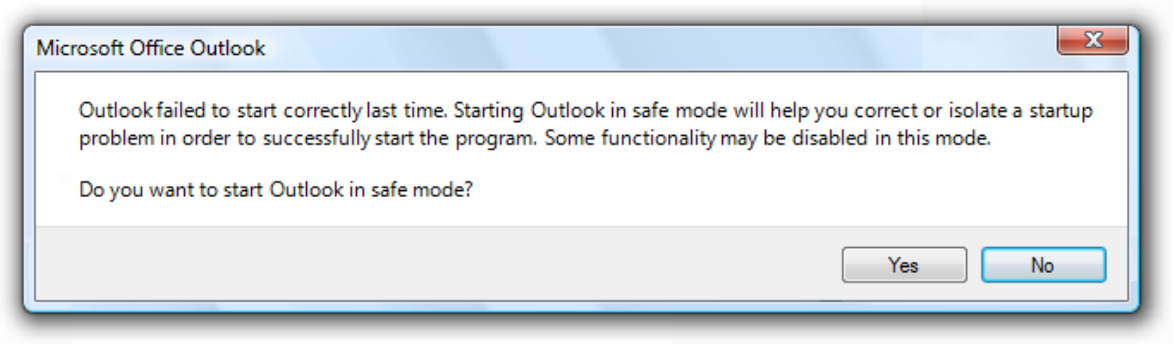
In these examples, Yes and No are good responses to yes and no questions, but specific responses are even better.

- **Consider phrasing the main instruction as a yes or no question if commit buttons with specific phrasing turn out to be long or awkward.** Alternatively, you can use command links for longer responses (five words or more) to the main instruction.

  Incorrect:



  Correct:



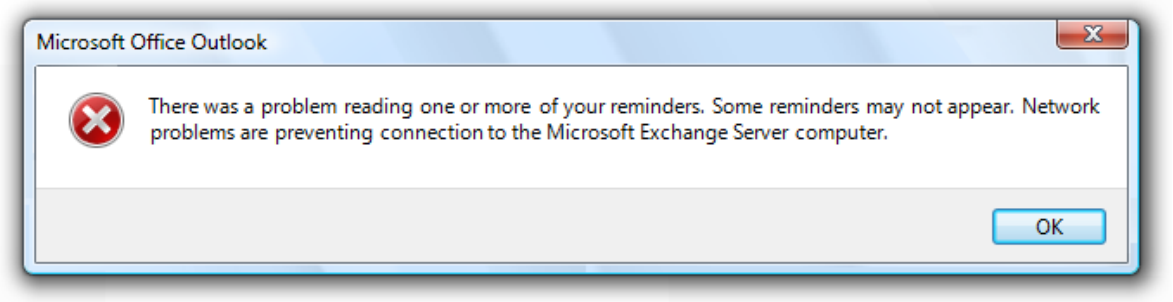
The specific phrasing in the incorrect example is too long, so the correct example uses Yes and No.

- **Don't use Yes and No buttons if the meaning of the No response is unclear.** If so, use specific responses instead.

OK buttons

- **In modal dialogs, clicking OK means apply the values, perform the task, and close the window.**
- **Don't use OK buttons to respond to questions.**
- **Don't assign access keys to OK, because Enter is the access key for the default button.** Doing so makes the other access keys easier to assign.
- **Label OK buttons correctly.** The OK button should be labeled OK, not Ok or Okay.
- **Don't use OK buttons for errors or warnings.** Problems are never OK. Use Close instead.

  Incorrect:



  In this example, Close should be used instead of OK.

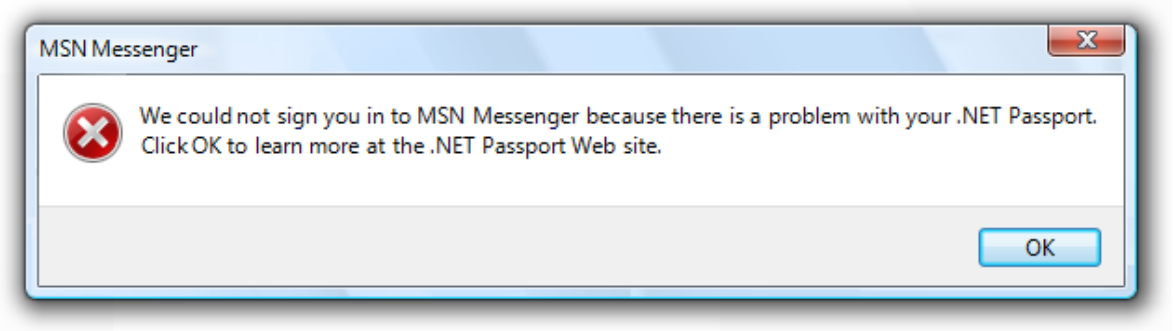
- **Don't use OK buttons in modeless dialog boxes.** Rather, modeless dialogs should use task-specific commit buttons (for example, Find). However, some modeless dialog boxes require only a Close button.

Cancel buttons

- **Clicking Cancel means abandon all changes, cancel the task, close the window, and return the environment to its previous state, leaving no side effect.** For nested choice dialog boxes, clicking Cancel in the owner choice dialog means any changes made by owned choice dialogs are also abandoned.

- **Provide a Cancel button to let users explicitly abandon changes.** Dialog boxes need a clear exit point. Don't depend on users finding the Close button on the title bar.
  - **Exception:** Don't provide a Cancel button for dialog boxes without settings. The OK and Close buttons have the same effect as Cancel in this case.

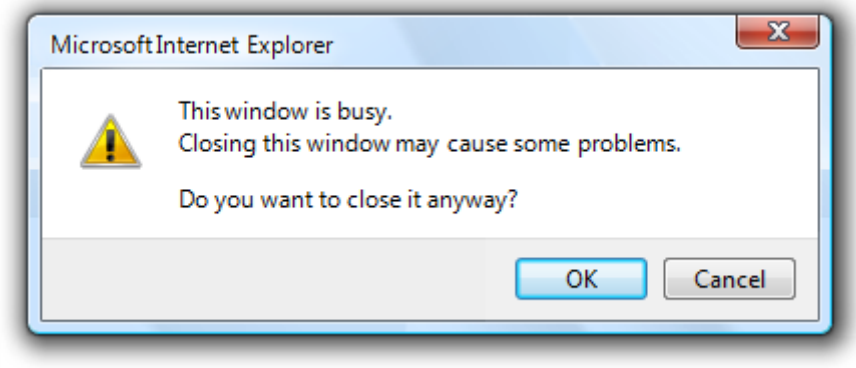
  Incorrect:

  In this example, having only a Close button on the title bar makes it appear as though users don't have a choice.

- **Don't use Cancel buttons to respond to questions.**

  Incorrect:

  In this example, OK and Cancel are incorrectly used to respond to a Yes or No question.

- **Don't assign access keys to Cancel, because Esc is the access key.** Doing so makes the other access keys easier to assign.

- **Don't use Cancel buttons in modeless dialog boxes.** Rather, use Close instead.

- **Don't disable the Cancel button.** Users should always be able to cancel dialog boxes.
  - **Exception:** You may disable the Cancel button in a progress dialog if there is a period during which the operation can't be cancelled. However, a better solution is to design such operations to always be cancelable.

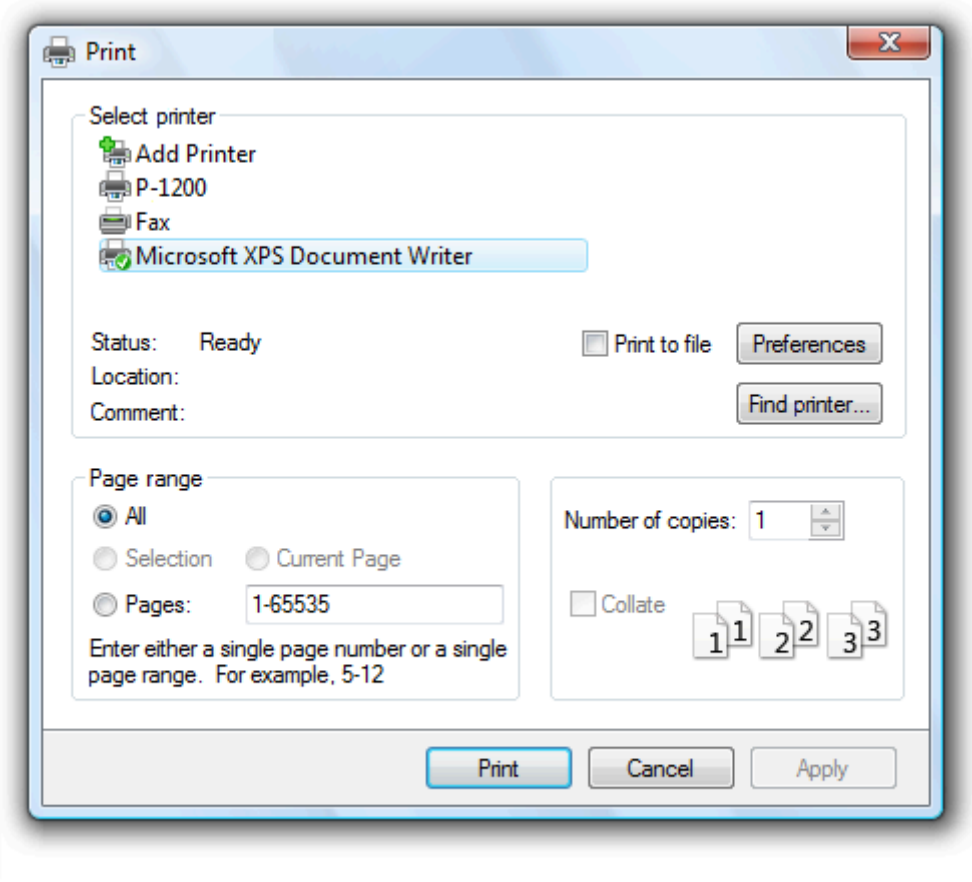
Close buttons

- **Use Close buttons for modeless dialog boxes, as well as modal dialogs that cannot be cancelled.**
- **Clicking Close means close the dialog box window, leaving any existing side effects.** Don't use Done, because it isn't an imperative construction. For nested choice dialog boxes, clicking Close in the owner choice dialog means any changes made by owned choice dialogs are preserved.
- **Put an explicit Close button in the dialog box body.** Dialog boxes need a clear exit point. Don't depend on users finding the Close button on the title bar.
- **Make sure the Close button on the title bar has the same effect as Cancel or Close.**
- **Don't assign access keys to Close, because Esc is the access key.** Doing so makes the other access keys easier to assign.

Apply buttons

- **Don't use Apply buttons in dialog boxes that aren't property sheets or control panels.** The Apply button means apply the pending changes, but leave the window open. Doing so allows users to evaluate the changes before closing the window. However, only property sheet and control panels have this need.

  Incorrect:



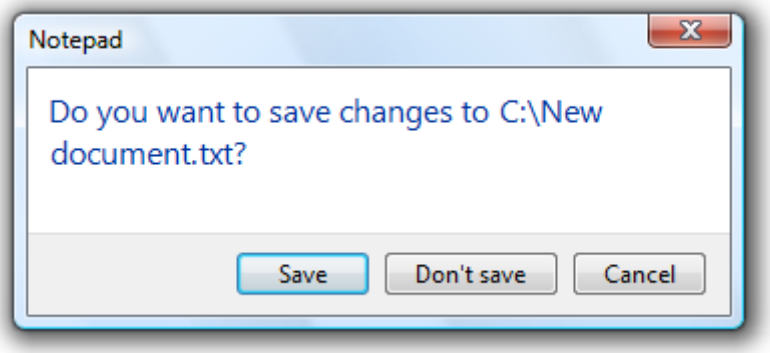
In this example, a choice dialog needlessly has an Apply button.

Commit buttons for indirect dialog boxes

Note: Indirect dialog boxes are displayed out of context, either as an indirect result of a task or the result of a problem with a system or background process. For indirect dialogs, the Cancel button is ambiguous because it could mean cancel the dialog or cancel the entire task.

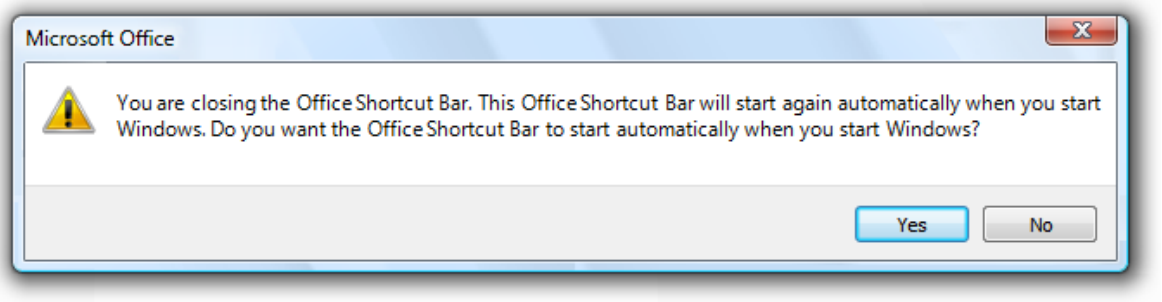
- **If users need to both cancel the dialog box and the task, give commit buttons to do both.** Label the button that cancels the dialog box with a negative response to the main instruction. Label the button that cancels the entire task with Cancel. Using Cancel allows the dialog box to be used in many contexts.

  Correct:



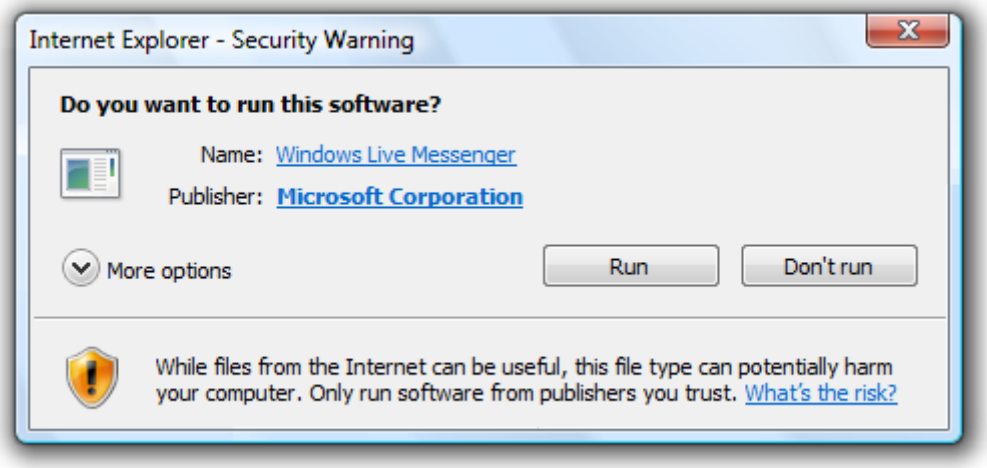
In this example, this dialog box is displayed by Windows Paint as the result of a New or Exit command when the graphic hasn't been saved. Don't Save closes the dialog without saving, whereas Cancel cancels the New or Exit command.

Incorrect:



In this example, there is no way to cancel the task (closing Office Shortcut Bar) that led to displaying this dialog box. This dialog box needs a Cancel button.

- **If users just need to cancel the dialog but not the task, use a button with a specific, negative response to the main instruction,** and don't have a Cancel button.



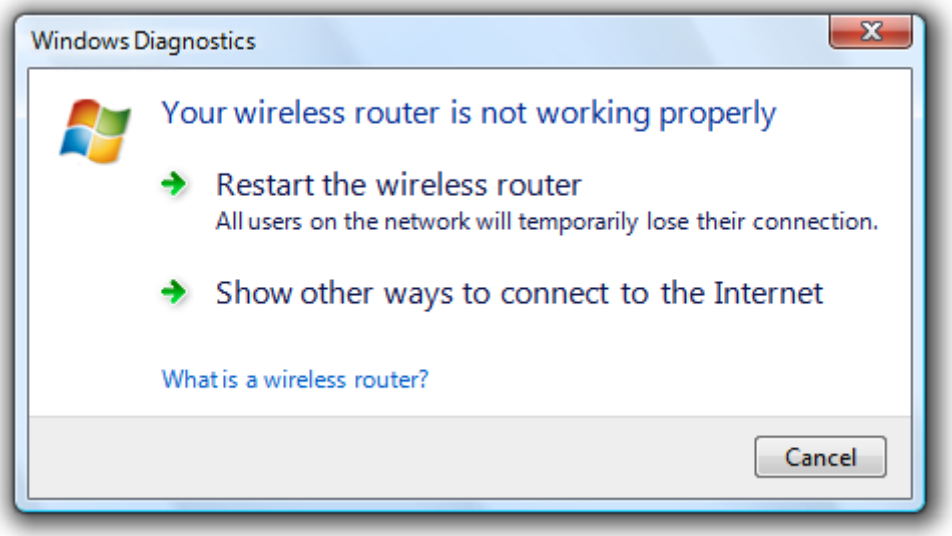
In this example, this dialog box is displayed indirectly as the result of navigating to a Web page that installs an ActiveX control. Using Cancel would be ambiguous here, so Don't run is used instead.

For more information and examples, see Command Buttons.

Command links

- **Present a set of lengthy commands using command links, instead of command buttons or a combination of radio buttons and an OK button.** Doing so allows users to respond with a single click. However, this approach works only for a single question.

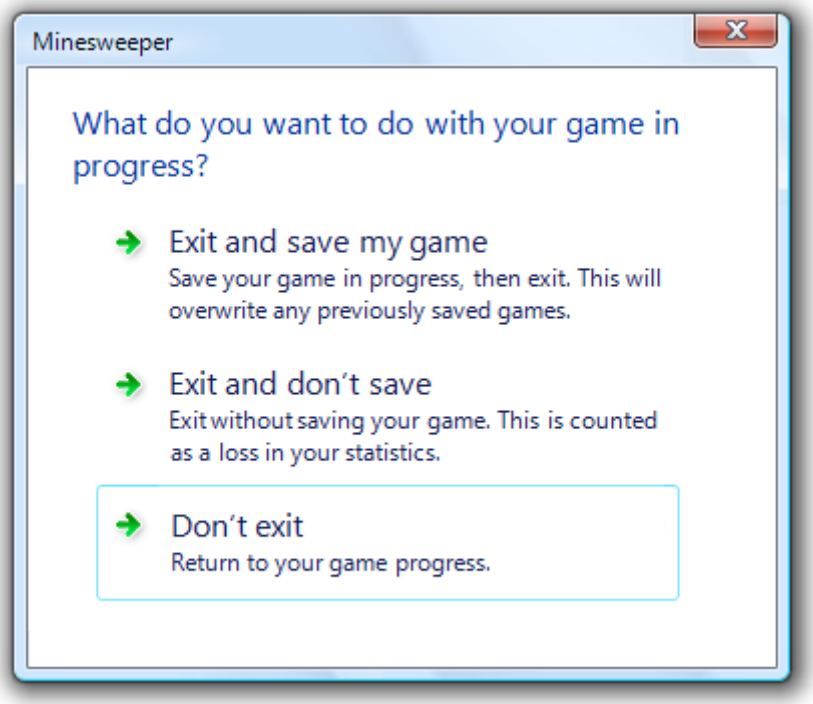
- **Present the most commonly used command links first.** The resulting order should roughly follow the likelihood of use, but also have a logical flow.
  - **Exception:** Command links that result in doing everything should be placed first.
- If a command link requires further explanation, **provide a supplemental explanation.** Supplemental explanations describe why users might want to choose the command, or what happens if the command is chosen.
- **Don't use supplemental explanations that are wordy restatements of the command link.** Use a supplemental explanation only when you can't make a command link self-explanatory. Providing a supplemental explanation for one command link doesn't mean that you have to provide them for all commands.



In this example, the supplemental explanation describes the implications of one of the options.

- **Use phrases that start with a verb, without ending punctuation.**
- **If a command is strongly recommended, consider adding "(recommended)" to the label.** Be sure to add to the link label, not the supplemental explanation.
- **If a command is intended only for advanced users, consider adding "(advanced)" to the label.** Be sure to add to the link label, not the supplemental explanation.
- **Always provide an explicit Cancel button**. Don't use a command link for this purpose.

Incorrect:



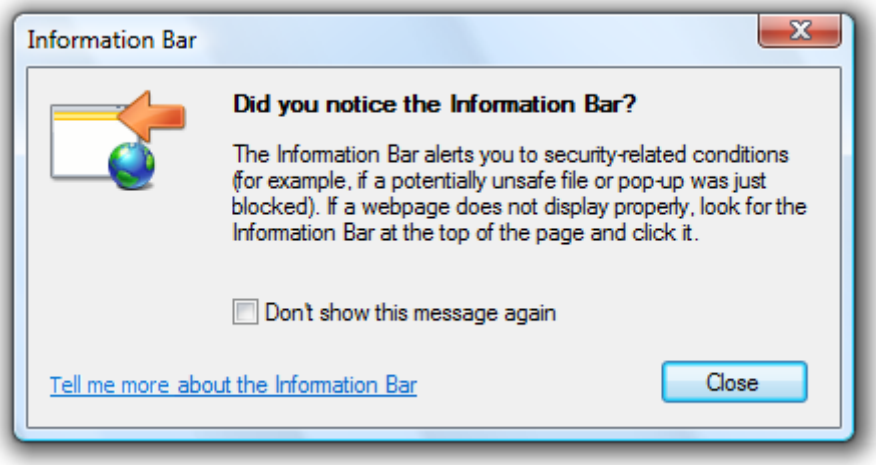
In this example, the dialog box uses a command link instead of a Cancel button.

For more information and examples, see Command Links.

Don't show this <item> again

- **Consider using a Don't show this <item> again option to allow users to suppress a recurring dialog box, only if there isn't a better alternative.** It is better always to show the dialog if users really need it, or simply eliminate it if they don't.
- **Use this specific phrasing replace <item> with the specific item.** For example, Don't show this reminder again. When referring to a dialog box in general, use Don't show this message again.
- **Clearly indicate when user input will be used for future default values** by adding the following sentence under the option: Your selections will be used by default in the future.
- **Don't select the option by default. If the dialog box really should be displayed only once, do so without asking.** Don't use this option as an excuse to annoy users make sure the default behavior isn't annoying.

Incorrect:

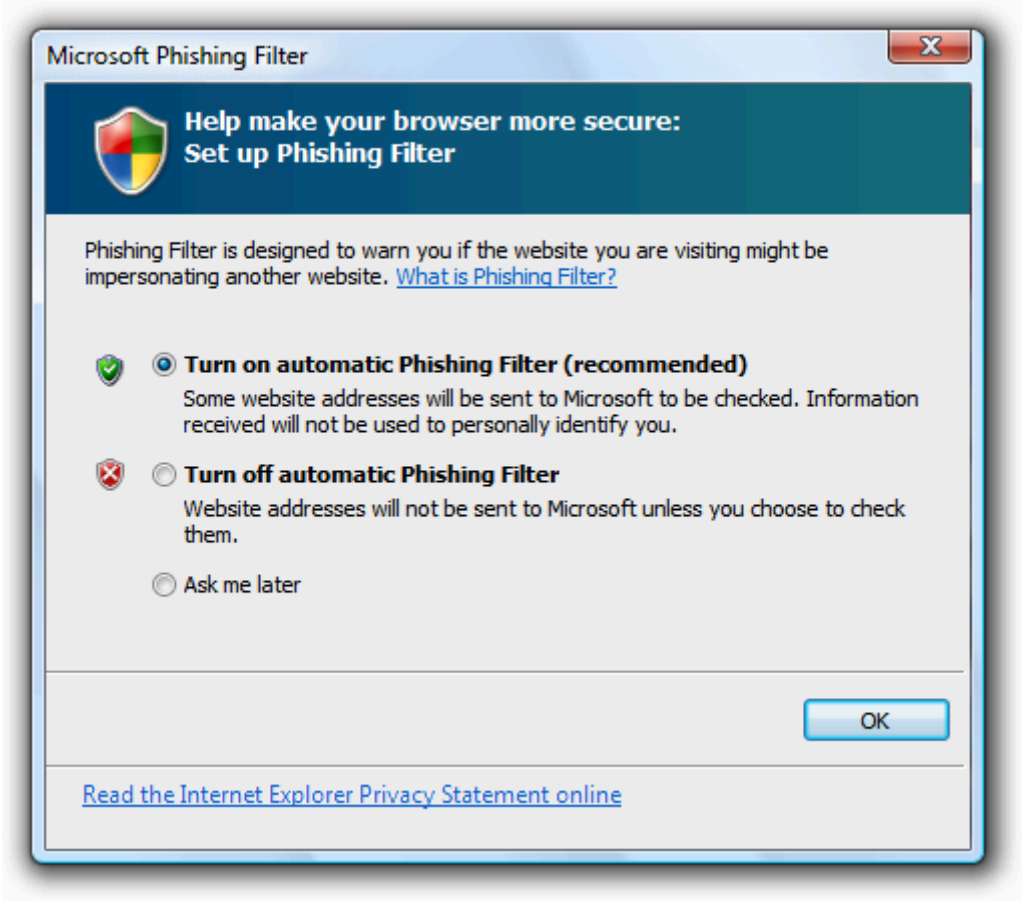


In this example, the message should just be displayed once. No need to ask.

- **Make the setting persist on a per-user basis.**
- **If users select the option and click Cancel, this option does take effect.** This setting is a meta-option, so it doesn't follow the standard Cancel behavior of leaving no side effect. Note that if users don't want to see the dialog in the future, most likely they want to cancel it as well.
- If users may need to restore these dialog boxes, provide a **Restore messages** command in the program's Options dialog box.

Ask me later

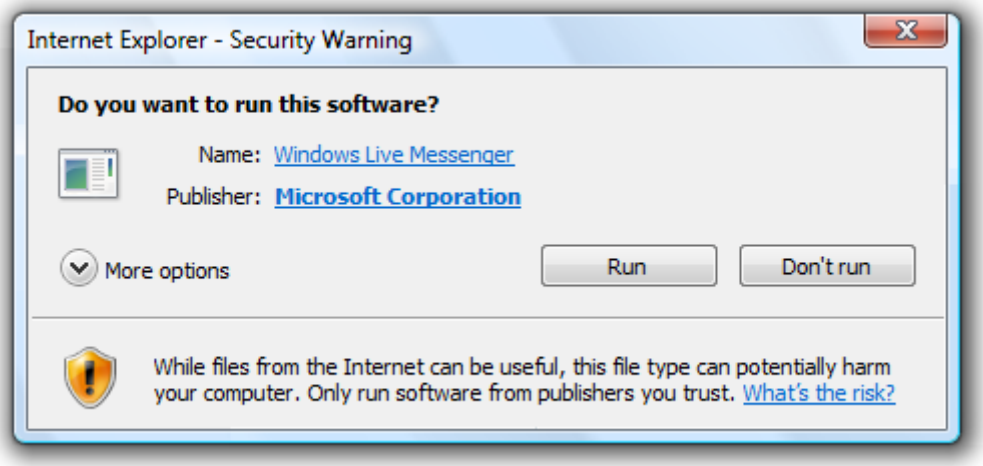
- Provide this option to dismiss a dialog box only when:
  - **The dialog box is indirect**, so users are likely to be focused on another task.
  - **Users must respond but not immediately**, so they can continue with their work.
  - **The question requires sufficient thought or effort** such that users might make better decisions if given more time.
  - **The dialog box or option will be presented automatically later** (so that users really are asked later).
- **Incorrect:**



- In this example, the question is simple enough that adding an Ask me later option only complicates it.
- Otherwise, expect users to respond now, but allow them to close the dialog box normally with either Cancel or Close. When used properly, this option should be rare.

More/Fewer

- **Use More/Fewer progressive disclosure buttons to show or hide advanced or rarely used options, commands, or details that target users typically don't need.** Doing so simplifies the dialog box for typical usage. Don't hide commonly used options, commands, or information because users might not find them.

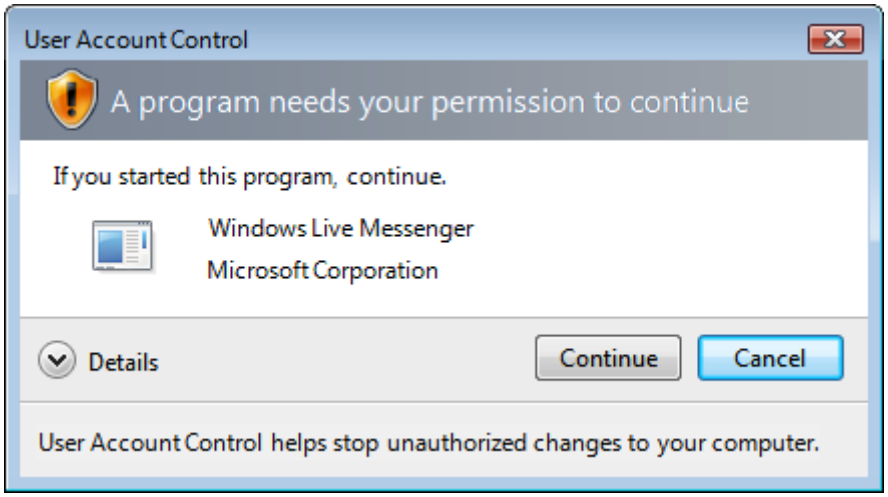


In this example, rarely used options are hidden by default.

- **Don't use More/Fewer controls unless there really is more detail to show.** Don't just restate the same information in a different format.
- **Don't use More/Fewer controls to show Help.** Use Help links or footnotes instead.
- **With task dialogs, avoid combining More/Fewer controls with Don't show this <item> again.** This combination has an awkward appearance.
- For labeling guidelines, see Progressive Disclosure.

Footnotes

- **Use footnotes for information that's not essential to a dialog box's purpose, but that users may find helpful in making decisions.** Most users should be able to skip footnotes and still make informed decisions in their response to the dialog box.



In this example, the footnote information is supplemental, not essential.

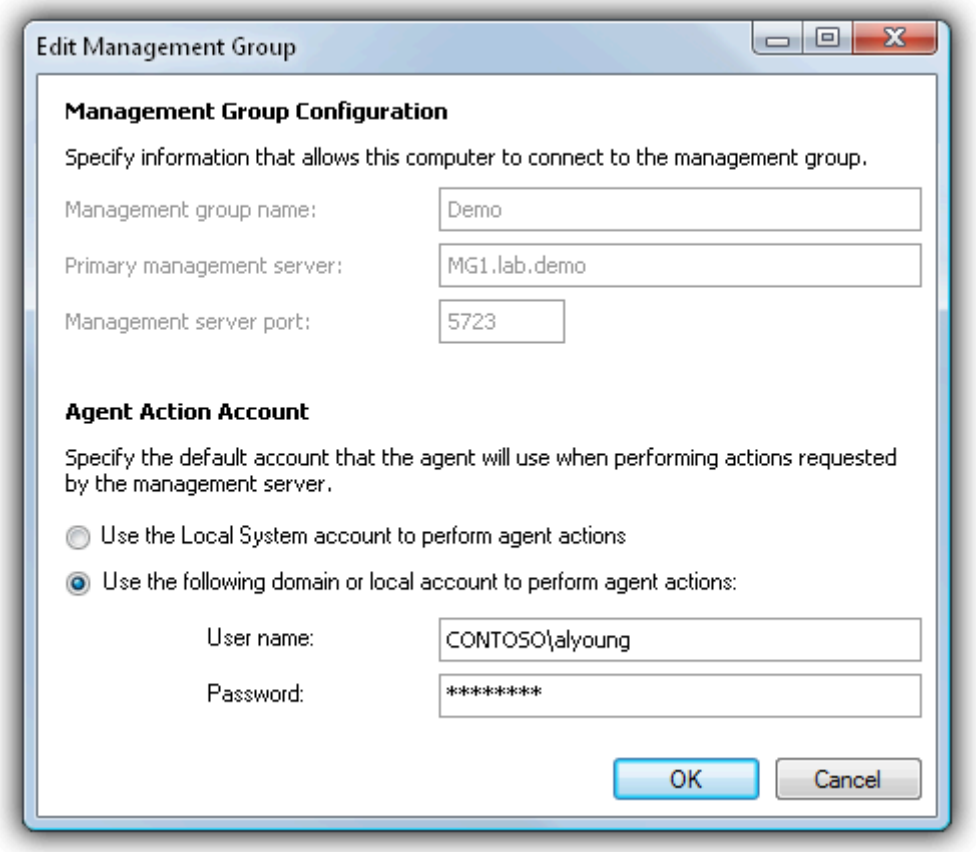
Disabling or removing controls vs. giving error messages

- When a control doesn't apply in the current context, consider the following options:
  - **Remove the control when there is no way for users to enable it, or users don't expect it to apply and its state doesn't change frequently.** Doing so simplifies the dialog box, and users won't miss it. Having a control appear and disappear frequently is annoying.
  - **Disable the control when users expect it to apply or its state changes frequently, and users can easily deduce why the control is disabled.** An example of easy deduction is disabling a commit button when there is a single, empty text box that requires any input. You can use balloons to display non-critical user input problems with text boxes and editable drop-down lists.

However, if the problem can't be explained with a balloon or involves multiple controls, the deduction would no longer be easy.

    - **Otherwise, leave the control enabled, but give an error message when it is used incorrectly.** Disabling in this case would make it difficult for users to understand why the control is disabled. Users would be forced to determine the problem through experimentation and deductive logic. It's better just to provide a helpful error message to explain the problem explicitly.

- **Tip:** If you aren't sure whether you should disable a control or give an error message, start by composing the error message that you might give. If the error message contains helpful information that target users aren't likely to quickly deduce, leave the control enabled and give the error. Otherwise, disable the control.
- **If you disable a control, also disable all associated controls**, such as its label, supplemental explanations, or command buttons. However, don't disable its group boxes, group label, or group explanation if there are any.



In this example, the disabled text box labels are also disabled, but their group label and group explanation are not.

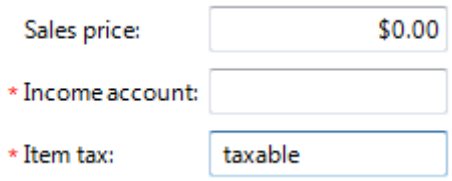
Required input

- To indicate that users must provide information in a control, consider the following options:

- **Don't indicate anything, but handle missing required input with error messages.** This approach reduces clutter and works well if most input is optional or users aren't likely to skip controls, thus keeping the number of error messages low.

- **Indicate required input using an asterisk at the beginning of the label.** Explain the asterisk using either:
  - A footnote at the bottom of the content area that says * Required input.
  - A tooltip on the asterisk that says Required input.

  This approach works well if there aren't many required controls, but poorly if most controls are required.



  In this example, asterisks are used to indicate required input.

- **If all controls require input, state "All input required" at an appropriate place at the top of the content area.** This approach reduces clutter for this specific case.

- **Indicate optional inputs with "(optional)" after the label.** This approach works well if most input is required, but poorly otherwise.

- **For consistency, try to use the same method to indicate required input throughout your program.** Specifically, indicate either required or optional input as needed, but avoid using both within the same program.

Error handling

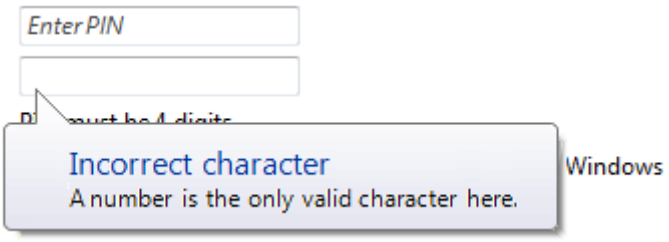
- Prevent errors by using controls that are constrained to valid user input. You can also help reduce the number of errors by providing reasonable default values.

- Validate user input as soon as possible, and show errors as closely to the point of input as possible.

- **Use modeless error handling (in-place errors or balloons) for user input problems.**

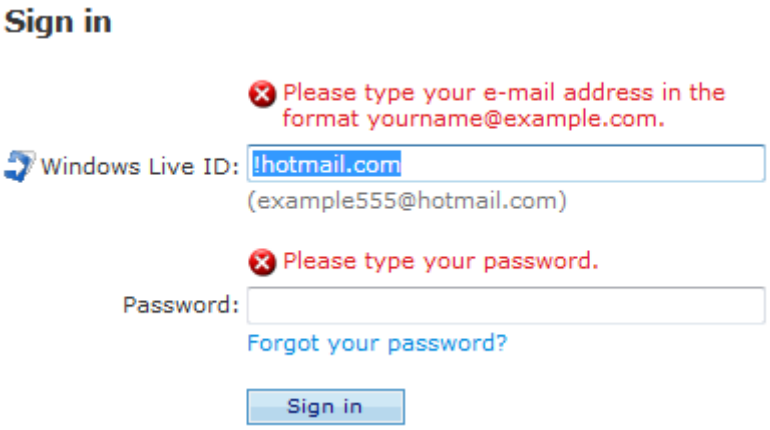
  - **Use balloons for non-critical, single-point user input problems detected while in a text box or immediately after a text box loses focus.** Balloons don't

require available screen space or the dynamic layout that is required to display in-place messages. Display only a single balloon at a time. Because the problem isn't critical, no error icon is necessary. Balloons go away when clicked, when the problem is resolved, or after a timeout.



In this example, a balloon indicates an input problem while still in the control.

- **Use in-place errors for delayed error detection**, usually errors found by clicking a commit button. (Don't use in-place errors for settings that are immediately committed.) There can be multiple in-place errors at a time. Use normal text and a 16x16 pixel error icon, placing them directly next to the problem whenever possible. In-place errors don't go away unless the user commits and no other errors are found.



In this example, an in-place error is used for an error found by clicking the commit button.

- **Use modal error handling (task dialogs or message boxes) for all other problems,** including errors that involve multiple controls, or are non-contextual or non-input errors found by clicking a commit button.

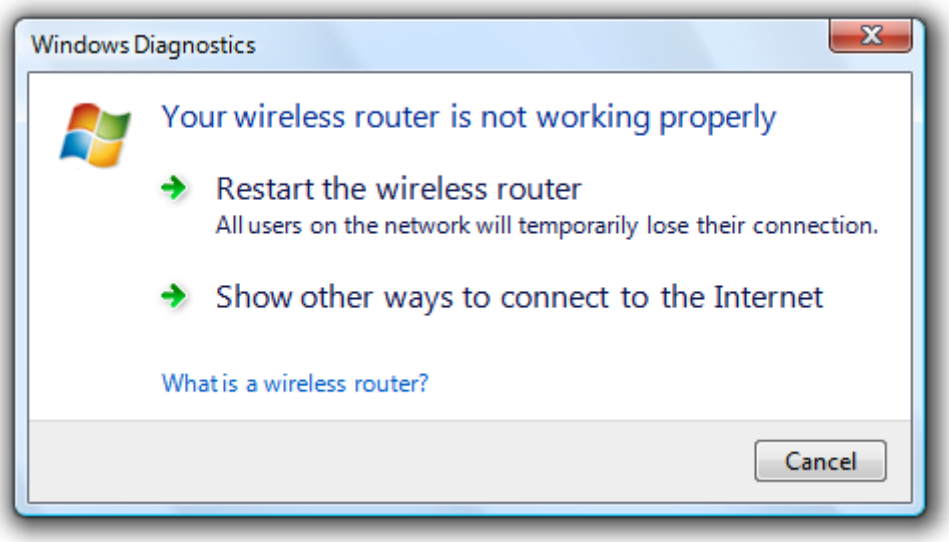
- **When an input problem is found and reported, set input focus to the first control with the incorrect data.** Scroll the control into view if necessary.

For more information and examples, see Error Messages and Balloons.

Help

- When providing user assistance, consider the following options (listed in their order of preference):
  - Give interactive controls self-explanatory labels. Users are more likely to read the labels on interactive controls than any other text.
  - Provide in-context explanations using static text labels.
  - Provide a specific Help link to a relevant Help topic.

- **Locate Help links at the bottom of the content area of the dialog box.** If the dialog box has a footnote and the Help link is related to it, place the Help link within the footnote.



  In this example, the Help link applies to the entire dialog.
  - **Exception:** If a dialog box has several distinct groups of settings that have separate Help topics (perhaps within group boxes), locate the Help links at the bottom of the groups.

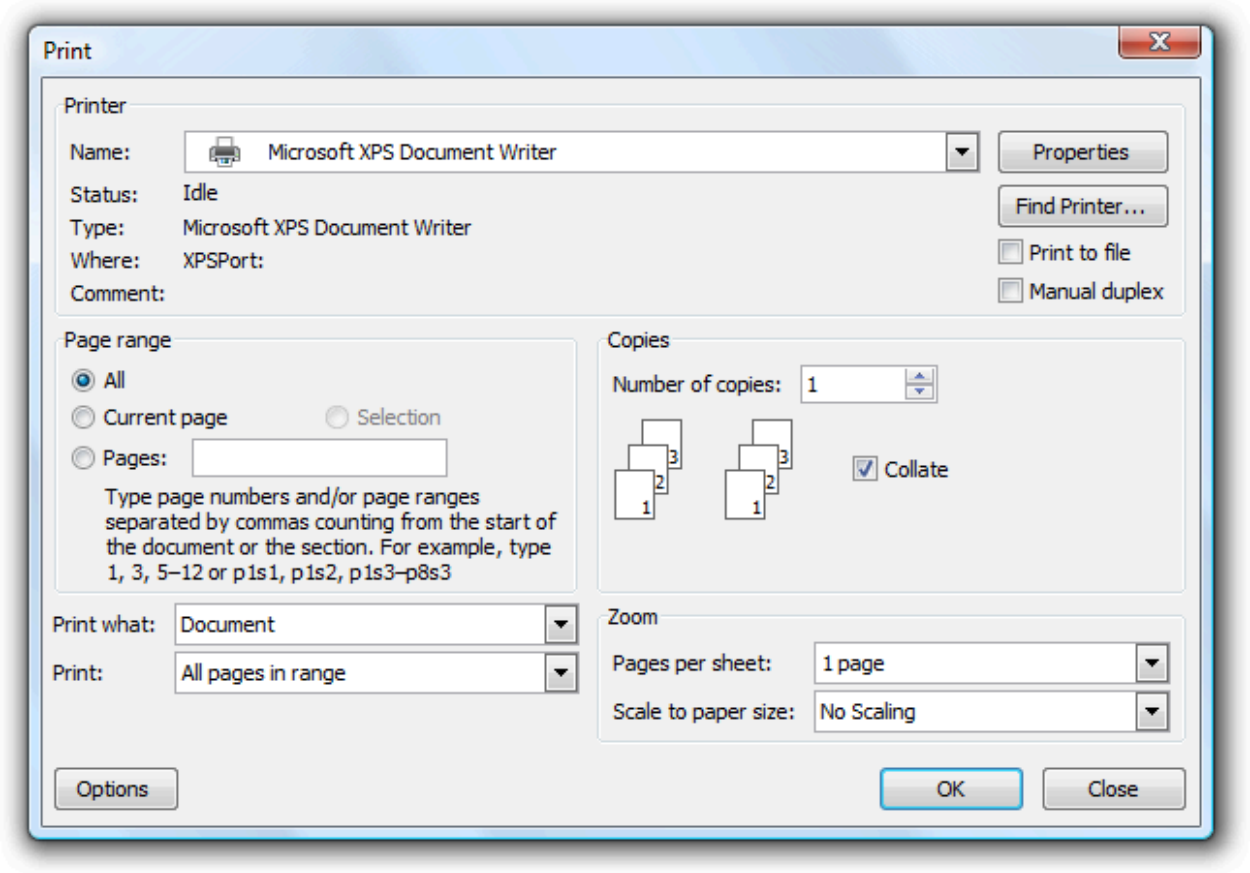
- **Don't use general or vague Help topic links or generic Help buttons.** Users often ignore generic Help.

For more information and examples, see Help.

Default values

- Include a default commit button on every dialog box.
- For question dialogs:
  - **Select the safest (to prevent loss of data or system access), most secure response to be the default.** If safety and security aren't factors, select the most likely or convenient response.
    - **Exception:** Don't make a destructive response the default unless there is an easy, obvious way to undo the command.

- For choice dialogs:
  - For the initial default values, **select the safest (to prevent loss of data or system access) and most secure values for each control.** If safety and security aren't factors, select the most likely or convenient options.
  - For the subsequent default values, **reselect the previously selected options if those values are likely to be repeated, and doing so is safe and secure.** Otherwise, select the initial default values.



In this example, users are most likely to choose the same printing settings as they did last time. However, the number of copies desired is likely to change, so this setting isn't reselected.

Recommended sizing and spacing

- **Support the minimum Windows Vista screen resolution of 800 x 600 pixels.** Layouts may be optimized for resizable windows using a screen resolution of 1024 x 768 pixels.
- **Use resizable windows whenever practical to avoid scroll bars and truncated data.** Windows with dynamic content and lists benefit the most from resizable windows.
- **Fixed-sized windows must be entirely visible and sized to fit within the work area.**

- **Resizable windows may be optimized for higher resolutions, but sized down as needed at display time to the actual screen resolution.**
- **Choose a default window size appropriate for its contents.** Don't be afraid to use larger initial window sizes if you can use the space effectively.

Text

General

- **Remove redundant text.** Look for redundant text in titles, main instructions, supplemental instructions, content areas, command links, and commit buttons. Generally, leave full text in instructions and interactive controls, and remove any redundancy from the other places.
- **Use positive phrasing.** Positive phrasing is easier for users to understand.

Correct:

Do you want to enable file and printer sharing?

Incorrect:

Do you want to disable file and printer sharing?

However, phrasing must match the associated command, even if the command is negatively phrased; so, for example, use disable to confirm a Disable command.

- **If necessary, use the word "window" to refer to the dialog box itself.**
- **Use the second person ("you/your") to tell users what to do** in the main instruction and content area. Often the second person is implied.

Examples:

Choose the pictures you want to print

Choose an account

- **Use the first person ("I/me/my") to let users tell the program what to do** in controls in the content area that respond to the main instruction.

Example: Print the photos on my camera.

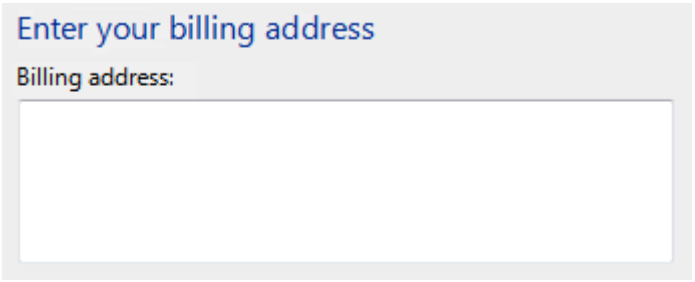
Dialog box titles

- **Use the title to identify the command, feature, or program where a dialog box came from.**
  - If dialog is user initiated, identify it using the command or feature name.
    Exceptions:
    - If a dialog box is displayed by many different commands, consider using the program name instead.
    - If that title would be redundant with the main instruction, use the program name instead.
  - If it is program or system initiated (and therefore out of context), identify it using the program or feature name to give context.
  - Don't use the title to explain what to do in the dialog that's the purpose of the main instruction.
- Use the exact command name for command-based names, but don't include the ellipsis if there is one. You can include the command's menu title if necessary to compose a good title. Example: for an Object... command in an Insert menu, use the title Insert Object.
- **If a modeless dialog box appears on the taskbar, optimize the title for display on the taskbar** by concisely placing the distinguishing information first. Examples: "66% Complete," and "3 Reminders."
- **Don't include the words "dialog" or "progress" in the title.** This is implied, and leaving it off makes it easier for users to scan.
- Use title-style capitalization, without ending punctuation.

Main instructions

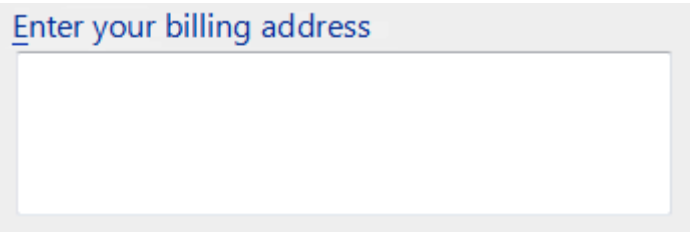
- **Use the main instruction to explain concisely what to do in the dialog.** The instruction should be a specific statement, imperative direction, or question. Good instructions communicate the user's objective with the dialog rather than focusing purely on the mechanics of manipulating it.
- **Omit the main instruction when the only thing you can say is obvious.** In such cases, the content of the dialog box is self-explanatory. For example, the File Open and File Save common dialogs don't need a main instruction because their context and design make their purpose obvious.
- **Omit control labels that restate the main instruction.** In this case, the main instruction takes the access key.

Acceptable:



In this example, the text box label is just a restatement of the main instruction.

Better:



In this example, the redundant label is removed, so the main instruction takes the access key.

- **Be concise use only a single, complete sentence.** Pare the main instruction down to the essential information. If you must explain anything more, use supplemental instruction.
- **Use specific verbs whenever possible.** Specific verbs (examples: connect, save, install) are more meaningful to users than generic ones (examples: configure, manage, set).
- Use sentence-style capitalization.
- **Don't include final periods if the instruction is a statement.** If the instruction is a question, include a final question mark.
- **For progress dialogs, use a gerund phrase briefly explaining the operation in progress,** ending with an ellipsis. Example: Printing your pictures...
- **Tip:** You can evaluate a main instruction by imagining what you would say to a friend. If responding with the main instruction would be unnatural, unhelpful, or awkward, rework the instruction.

Supplemental instructions

- **When necessary, use an optional supplemental instruction to present additional information helpful to understanding or using the page.** You can provide more detailed information and define terminology.
- **If the appearance of the dialog box is program or system initiated (and therefore out of context), use the supplemental instruction to explain why the dialog has appeared.** For such dialogs, the context is usually not obvious.

- **Don't repeat the main instruction with slightly different wording.** Instead, omit the supplemental instruction if there is not more to add.
- Use complete sentences, sentence-style capitalization, and ending punctuation.

Command links

- **Choose concise link text that clearly communicates and differentiates what the command link does.** It should be self-explanatory and correspond to the main instruction. Users shouldn't have to figure out what the link really means or how it differs from other links.
- **Always start command links with a verb.**
- Use sentence-style capitalization.
- Don't use ending punctuation.
- **If necessary, provide any further explanation using complete sentences and ending punctuation.** However, add such explanations only when needed don't add explanations to all command links just because one command link needs one.

For more information and examples, see Command Link guidelines.

Commit buttons

- **Use specific commit button labels that make sense on their own and are a response to the main instruction.** Ideally users shouldn't have to read anything else to understand the label. Users are far more likely to read command button labels than static text.
- **Start commit button labels with a verb. Exceptions are OK, Yes, and No.**
- Use sentence-style capitalization.
- Don't use ending punctuation.
- Assign a unique access key.
  - **Exception:** Don't assign access keys to OK and Cancel buttons because Enter and Esc are their access keys. Doing so makes the other access keys easier to assign.

Documentation

When referring to dialog boxes:

- In programming and other technical documentation, refer to dialog boxes as dialog boxes. Everywhere else, refer to dialog boxes by their title. If the title bar is hidden, refer to the dialog using the main instruction.

- If you must refer to a dialog box in general, use window in user documentation. You can refer to a simple question dialog or confirmation as a message.
- Use the exact title or main instruction text, including its capitalization.
- When possible, format the title using bold text. Otherwise, put the title in quotation marks only if required to prevent confusion.

Example: In **Windows Security**, click **More Options**.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

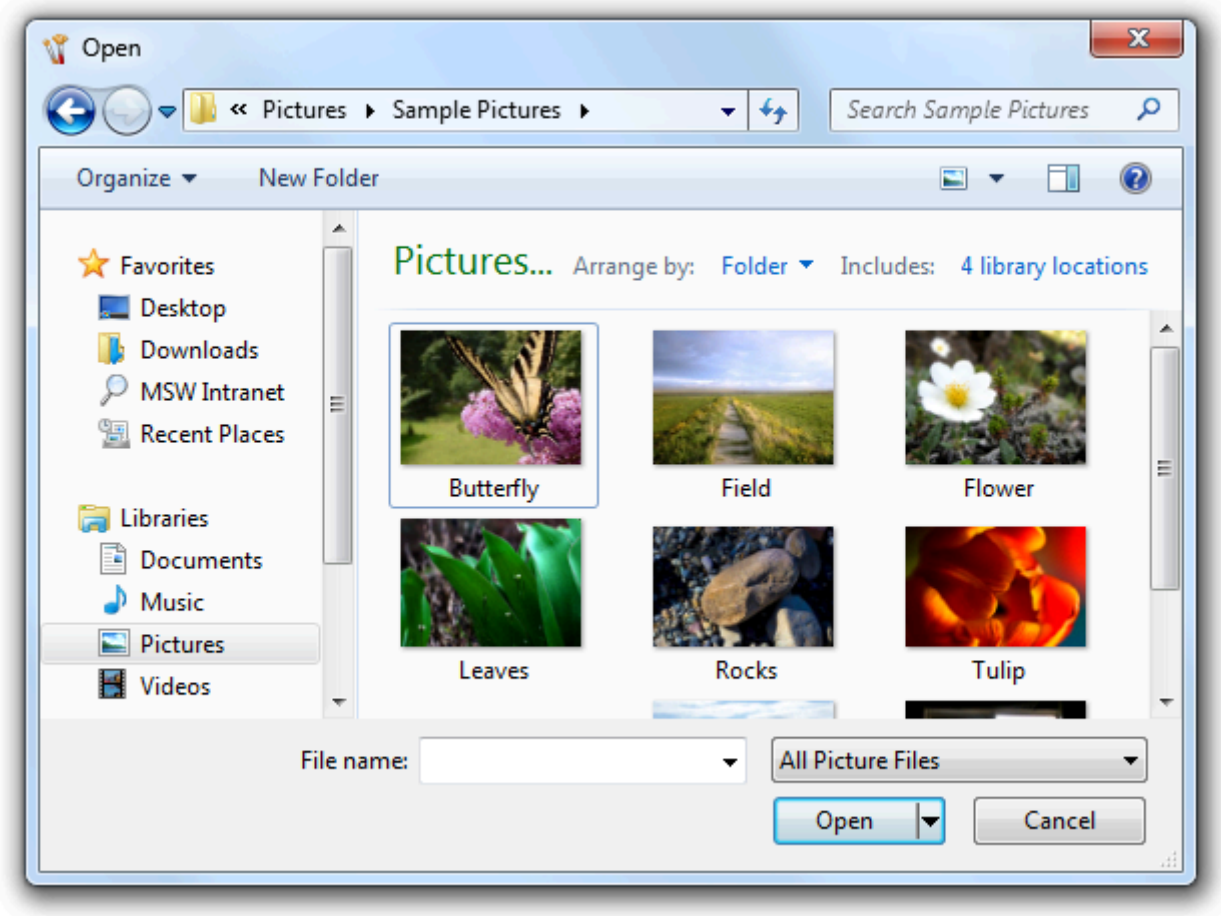
Common Dialogs

Article • 06/03/2021

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

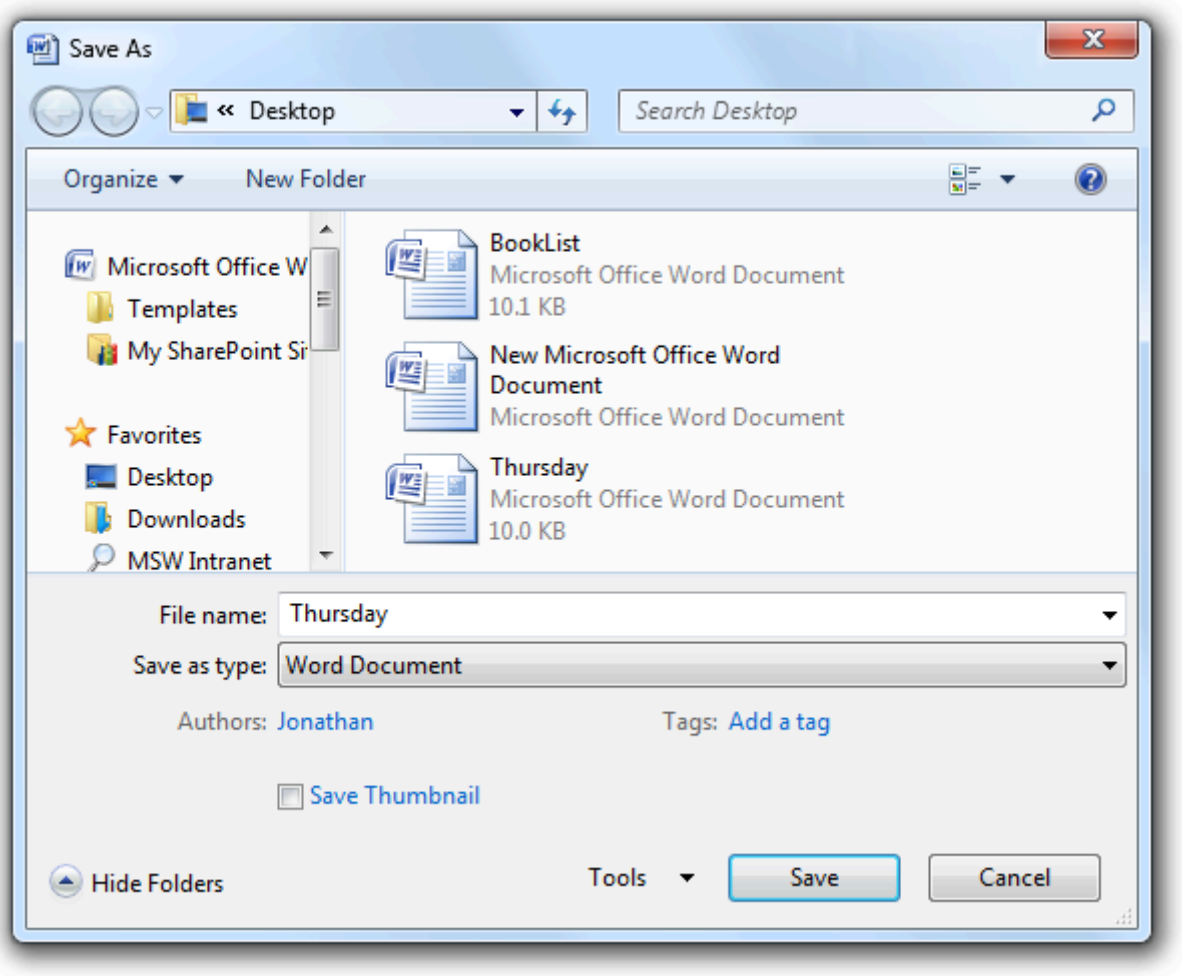
The Microsoft Windows common dialogs consist of the Open File, Save File, Open Folder, Find and Replace, Print, Page Setup, Font, and Color dialog boxes.

Open File



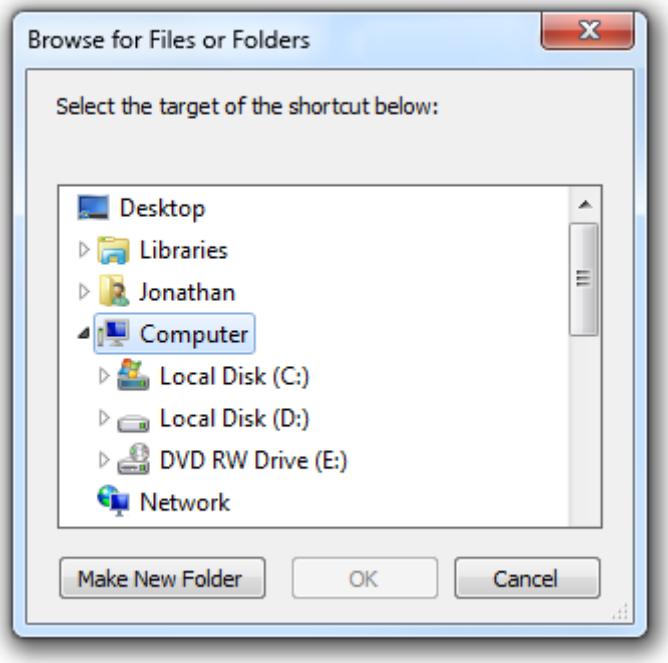
Open File is optimized for quickly finding items to use with a program.

Save File



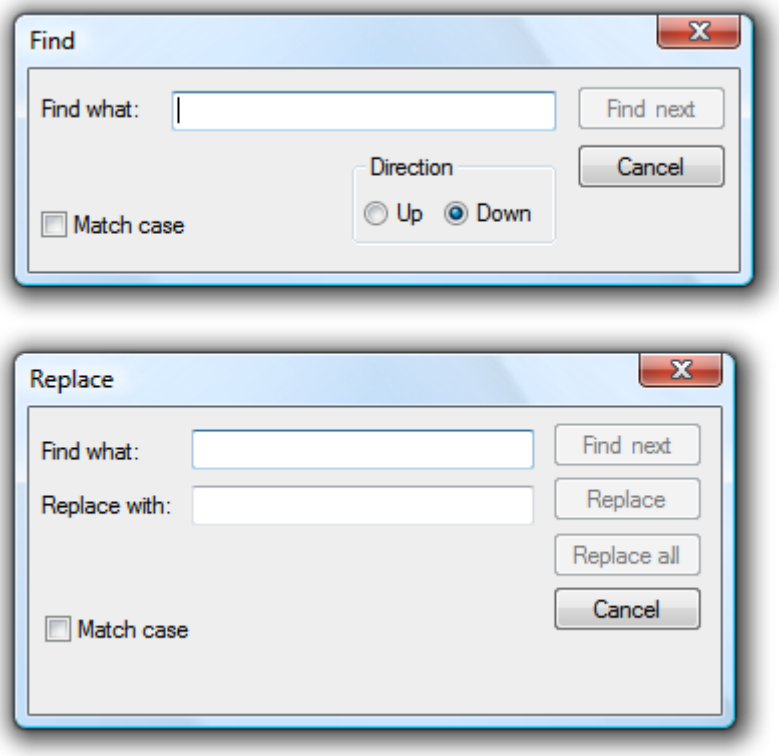
Save File closes the loop by saving a file with its metadata.

Open Folder



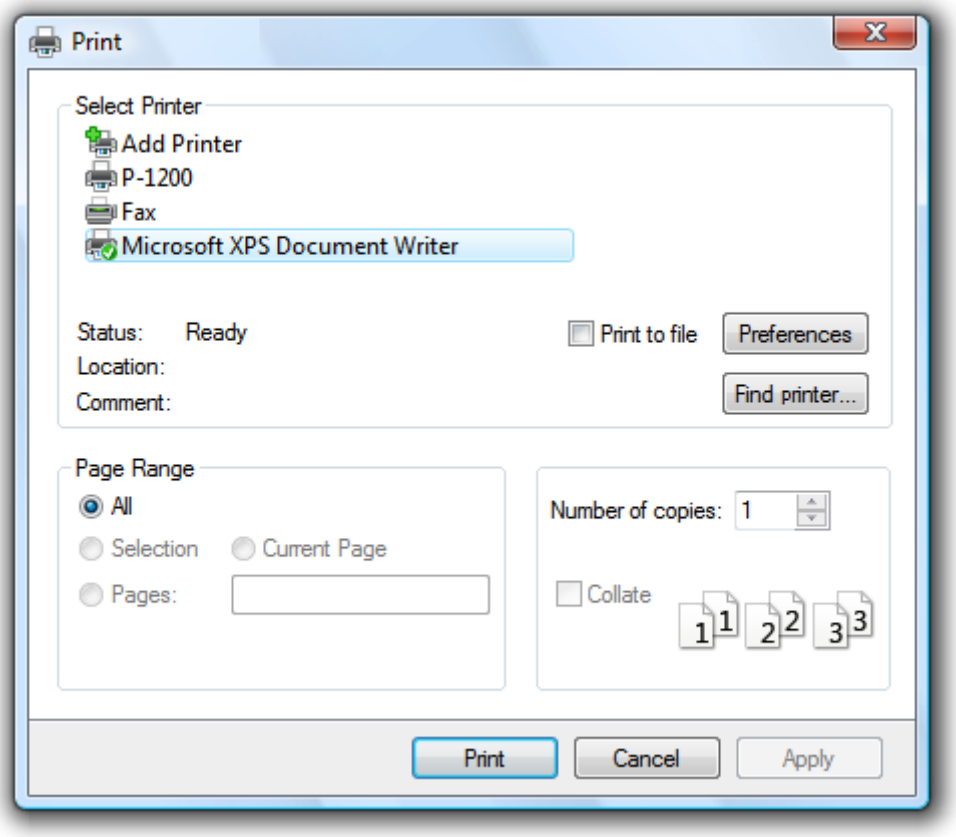
Open Folder is specifically for choosing folders.

Find and replace



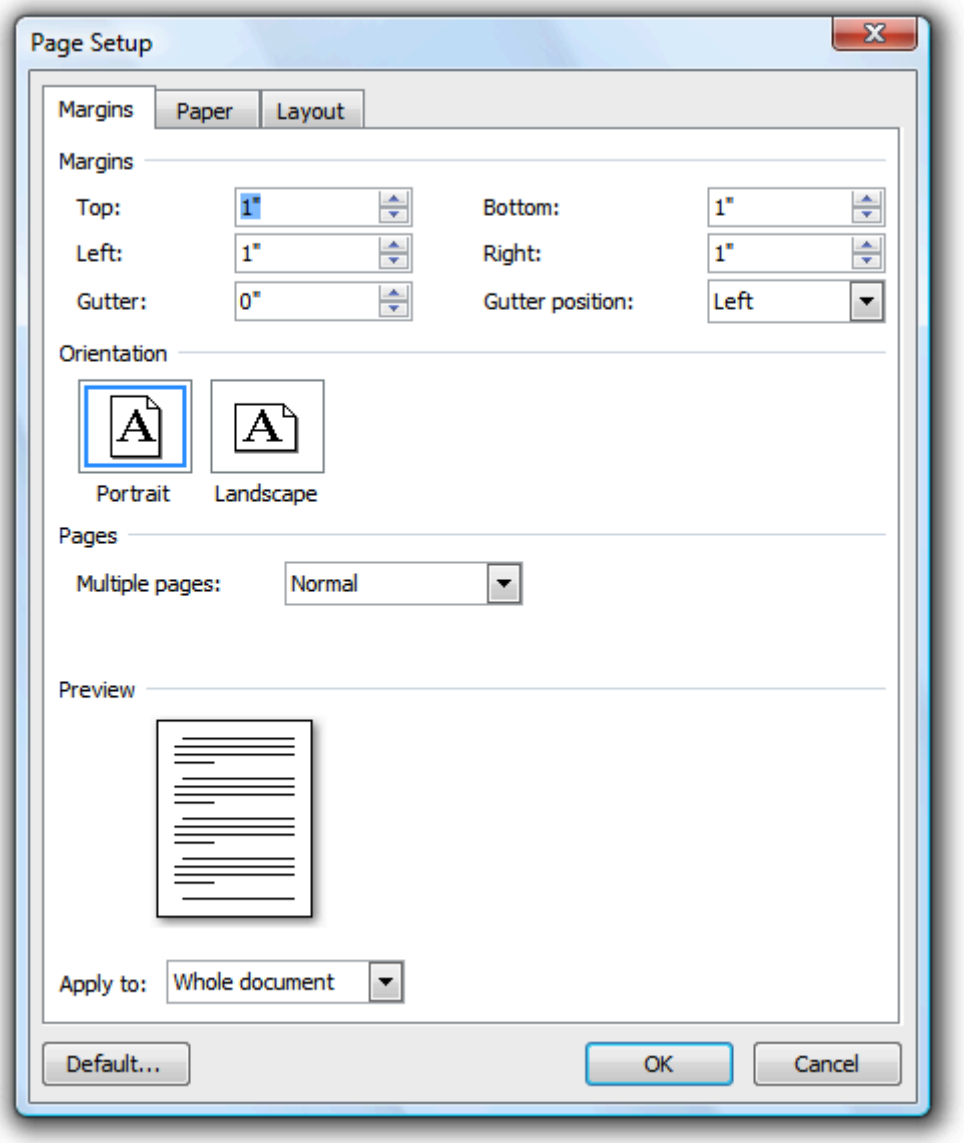
Find allows users to search for text strings, whereas the Replace version optionally allows users to replace matches with another string.

Print



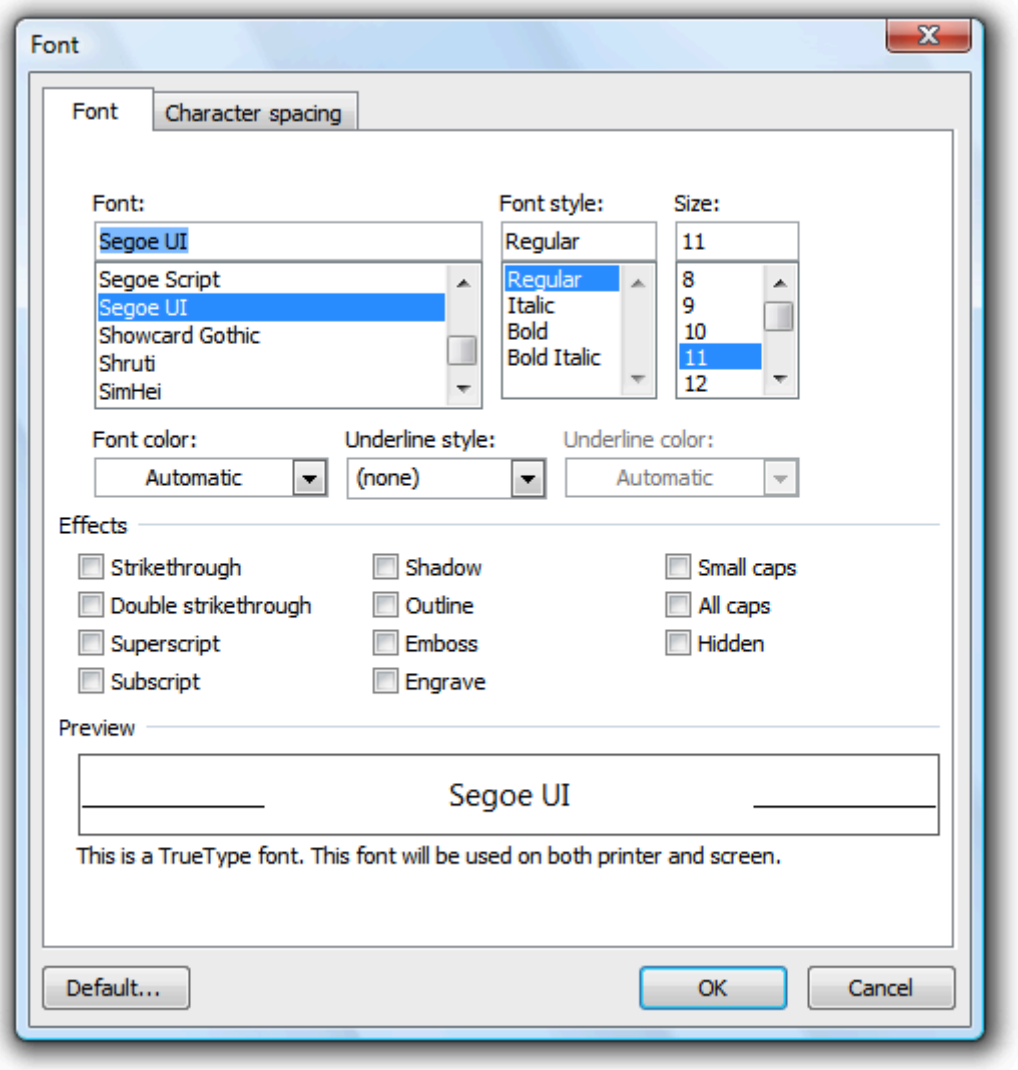
Print allows users to select what to print, the number of copies to print, and the collation sequence, along with the ability to choose and configure printers.

Page setup



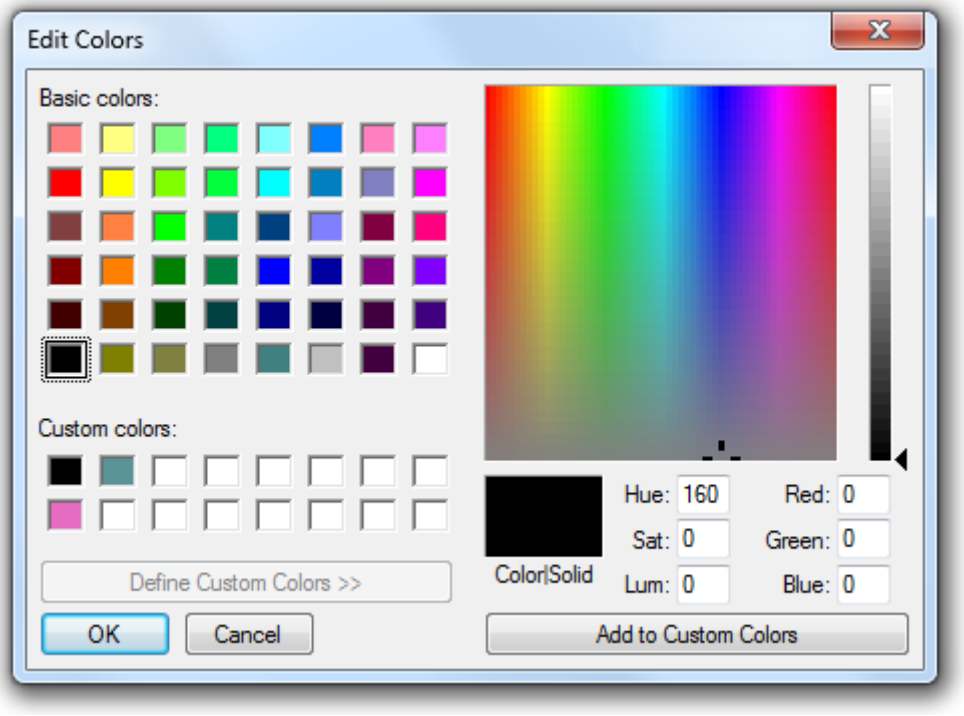
Page setup allows users to select the paper size and source, page orientation, and margins.

Font



Font displays the fonts and point sizes of the available installed fonts.

Color



Color allows users to select a color, either through a predefined set of colors or by choosing a "custom" color.

Design concepts

By using the common dialogs, you help give users a consistent experience across different programs. And by using the common dialogs well, you also help give users an efficient, enjoyable experience.

You can significantly improve users' experience with these dialogs by choosing the most appropriate defaults for:

- Input values (examples: default folders, default file names).
- Selected options (examples: selected printer, printing options).
- Views (examples: showing pictures in thumbnail view, showing pictures without file names, sorting by date, column widths).
- Presentation (examples: window size, location, and contents).

You must determine both the initial defaults and subsequent defaults. Initial default values are determined by your program and based on the target user's expected usage, whereas subsequent defaults are based on the actual usage. Past usage is the best indicator of future usage.

Are your program's defaults efficient? Monitor the number of steps users have to take to perform the most common tasks. If users have to repeat the same, potentially unnecessary steps every time they perform a task, your default values can be improved.

If you do only one thing...

Give users an efficient, enjoyable experience by selecting appropriate initial and subsequent defaults.

Is this the right user interface?

Yes! Use the common dialogs for a consistent user experience. Don't create your own. It is especially difficult to create custom UIs that navigate the namespace correctly and securely. Note that you can customize the common dialogs if necessary.

For Windows Vista, the Open File and Save File have a new extensible architecture to make it easier to expose additional functionality. This mechanism is flexible enough to meet the minimum requirements of major independent software vendors (ISVs), but not be broken by future releases of Windows.

Guidelines

General

- When appropriate, provide more direct or modeless alternatives. Allow users to:
  - Open files by dropping them on your program.
  - Save files using their current name and location with a Save command.
  - Find the next occurrence of a string using the F3 key.
  - Print one copy of an entire document to the default printer with a Print command.
  - Change fonts and font attributes using a toolbar or palette window.
  - Change colors using a toolbar or palette window.
- Use the following commands to display common dialogs (given along with their preferred access keys):

Expand table

Common dialog	Command
Open File	Open...
Save File	Save as...
Open Folder	Open folder... or Choose folder...
Find and Replace	Find... or Replace...
Print	Print...
Page Setup	Page setup...
Font	Font... or Choose font...
Color	Color... or Choose color...

- You can use more specific commands, as appropriate. Example: for exporting a file, use the command Export file instead of Save as.
- Set the dialog box title to reflect the command that launched it. Example: If Save File is launched from an Export file command, rename the dialog box to Export File.

Open File

- For the initial default folder, use a specialized folder (Pictures, Music, Videos) as appropriate, otherwise use Documents.

- For subsequent default folders, use the last folder opened by the user using the program.
- When opening photo files, suppress file names by default. Photos are usually identified by their thumbnails and their names typically aren't meaningful.

Save File

- For the initial default folder (if a new file is being saved for the first time), use the specialized folder (Pictures, Music, Videos) as appropriate, otherwise use Documents.
- For temporary files, use the current user's temporary folder. Choose plain, but unique file names. Example: Use File0001.tmp instead of ~DF1A92.tmp.
  - **Developers:** You can get the current user's temporary folder using the GetTempPath API function.
- For the initial default file name, use a unique default name based on:
  - The file's contents, if known. Example: The first words in a document.
  - A pattern chosen by the user. Example: If the previous file was named "Hawaii 1.jpg", choose "Hawaii 2.jpg" as the next file.
  - A generic pattern based on the file type. Example: "Photo1.jpg".
- For subsequent defaults (if the file already exists), use the file's current folder and name.
- When saving a file, preserve its creation date. If your program saves files by creating a temporary file, deletes the original, and renames the temporary file to the original file name, be sure to copy the creation date from the original file.
- Use Save File if the user selects the Save command without specifying a file name.

File types lists

Note: File types lists are used by Open File and Save File to determine the types of files displayed and the default file extension.

- If the file types list is short (five or fewer), order the list by likelihood of usage. If the list is long (six or more), use an alphabetical order to make the types easy to find.
- For Save File, include all variations of the supported file extensions, even if uncommon, and put the most common extension first. The file handling logic looks at this list to determine if the user supplied a supported file extension. Example: If a JPEG file types list includes only .jpg and .jpeg, the file test.jpe might be saved as test.jpe.jpg.
- For Save File, the initial default file type is the most likely chosen by the target user. The subsequent default is the file's current type.

- For Open File, the initial default file type is the most likely chosen by the target user. The subsequent default should be the last file type used.
- For Open File, include an "All files" entry as the first item if users can open any file type, or may need to see all files in a folder at the same time. Consider providing other meta filters, such as "All pictures," "All music," and "All videos." Place these immediately after "All files."
- Use the format "File type name (*.ext1; *.ext2)." The file type name should be the registered file type name, which you can view in the Folder Options control panel item. Example: "HTML document (*.htm; *.html)."
  - **Exception:** For meta-filters, remove the file extension list to eliminate clutter. Examples: "All files," "All pictures," "All music," and "All videos."
- Use sentence-style capitalization for the file type names, and lowercase for the file type extensions.

Open Folder

- **For new programs, use the Open Files dialog in the "pick folders" mode.** Doing so requires Windows Vista or later, so use the Open Folder dialog for programs that run in earlier versions of Windows.
  - **Developers:** You can use the Open Files dialog in the "pick folders" mode by using the FOS_PICKFOLDERS flag.

Font

- If necessary, you can filter the font list to show only the fonts available to your program.

Persistence

- Consider making the following values persistent to use as subsequent defaults:
  - Input values (examples: default folders, default file names).
  - Selected options (examples: selected printer, printing options).
  - Views (examples: showing pictures in thumbnail view, showing pictures without file names, sorting by date, column widths).
  - Presentation (examples: window size, location, and contents).

Exception: Don't make these values persist for common dialogs when their usage is such that users are far more likely to want to start completely over.

- When determining default values, consider what target users are most likely to want based on the important scenarios. Also, consider scenarios within a program

instance, across multiple instances (both consecutive or concurrent), and across multiple documents. Don't make values persist in circumstances that aren't likely to be helpful.
    - **Example:** For a typical document-based application, it's helpful to use persistent Open File and Save File settings within a program instance and across consecutive instances, but keep concurrent instances independent. That way, users can work efficiently with several documents at a time.
- Make the settings persist on a per-program, per-user basis.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

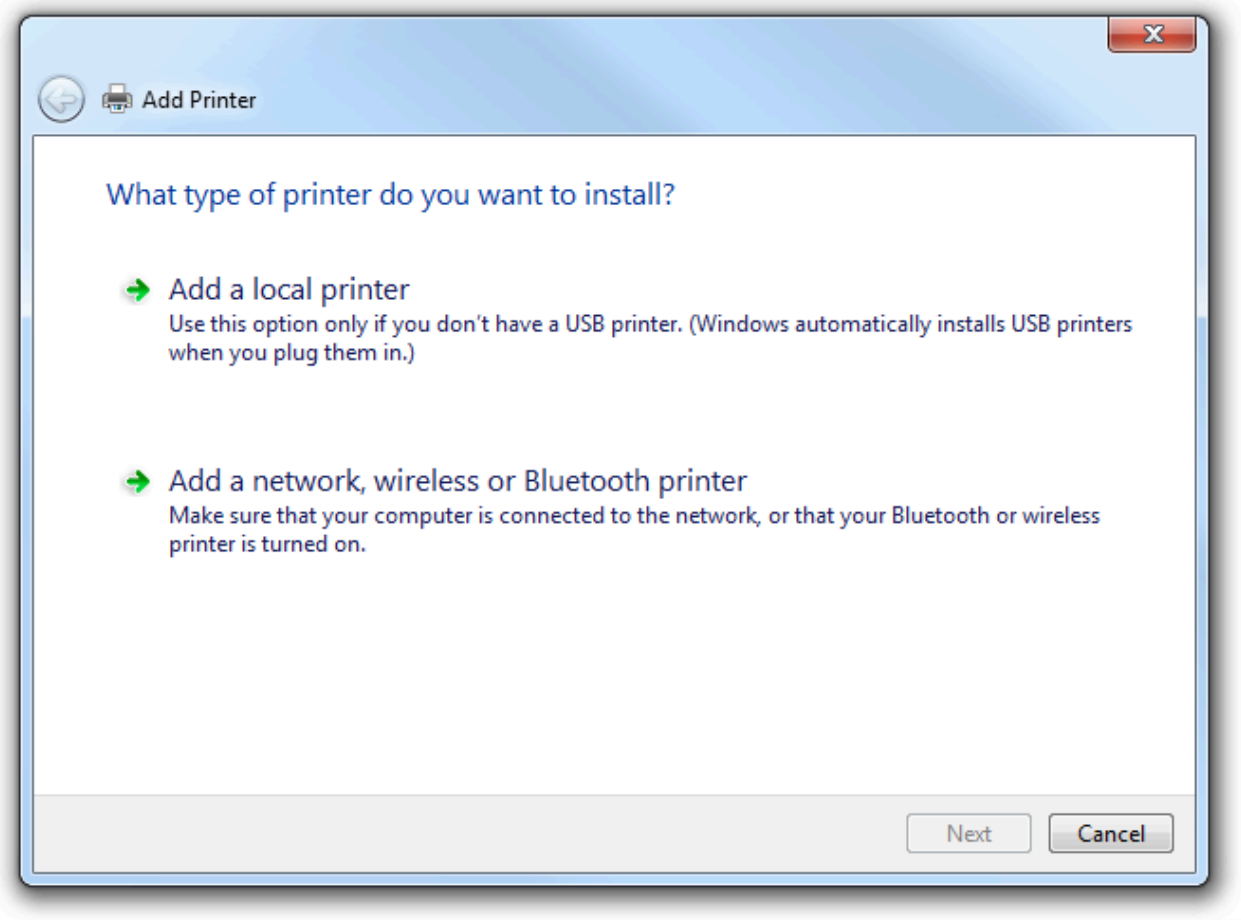
Wizards

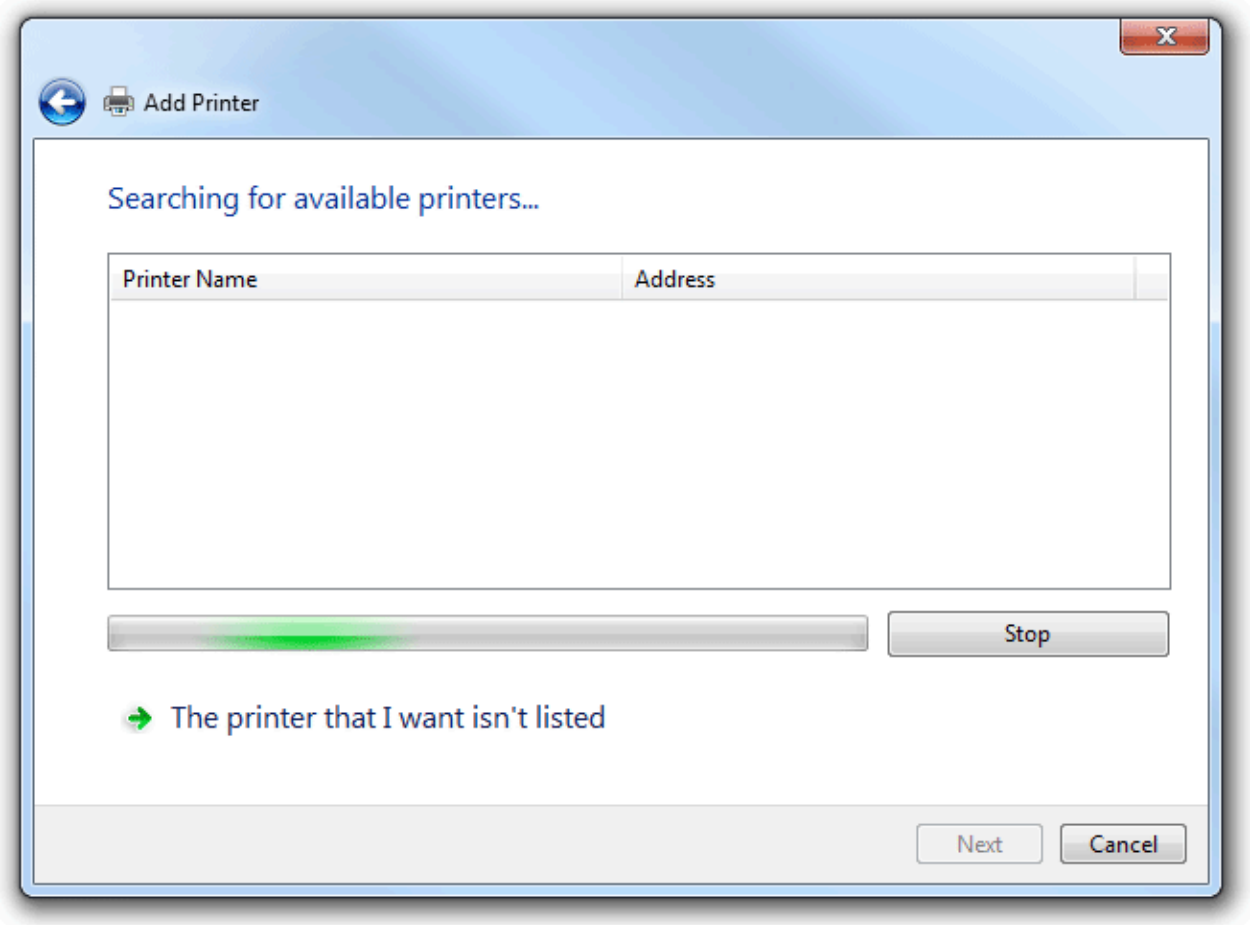
Article • 02/09/2021

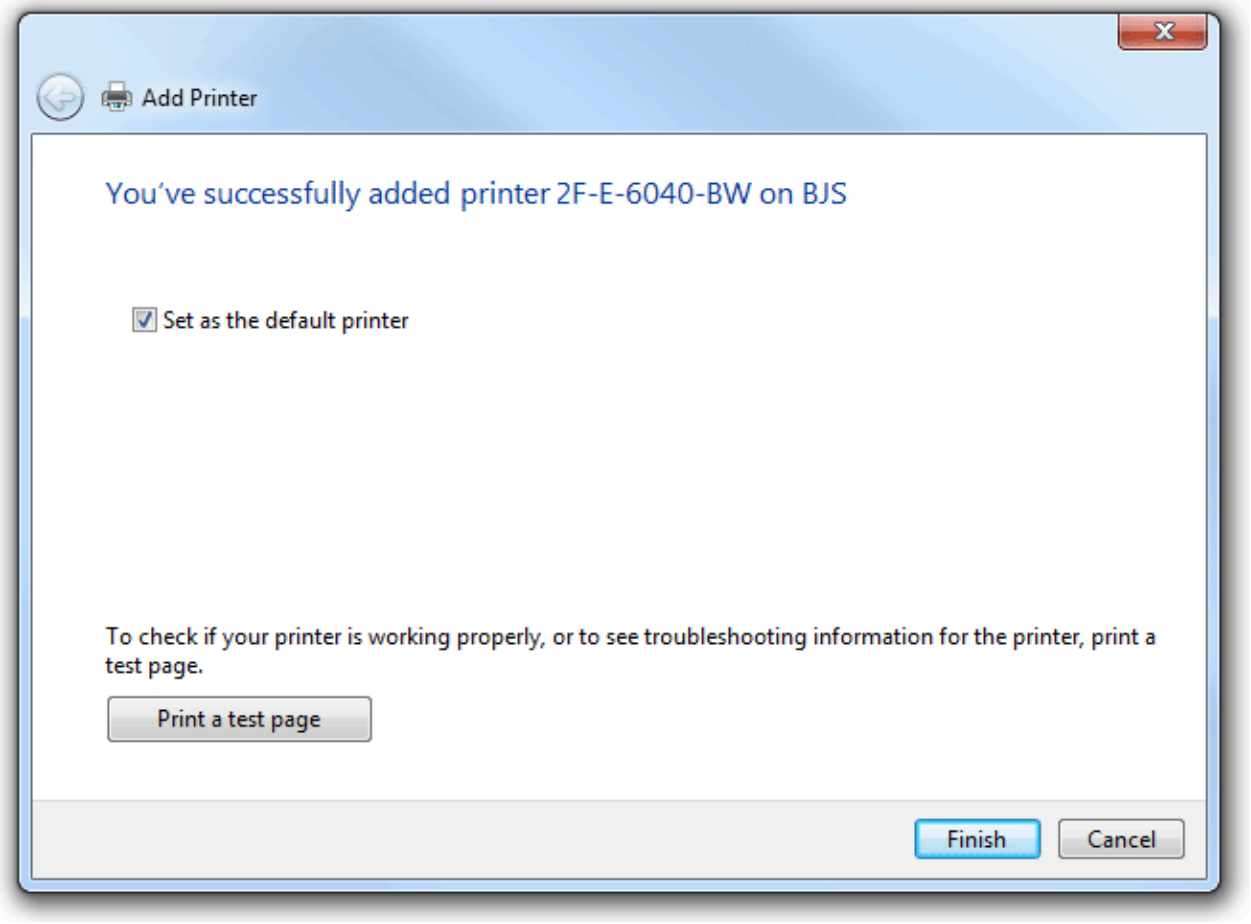
> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

Despite that wonderful, whimsical name, wizards are not really a special form of user interface, and they have only a particular range of utility.

Wizards are used to perform multi-step tasks.







Multiple steps of a wizard are presented as a sequence of pages.

Wizards typically include the following types of pages:

- Choice pages are used to gather information and allow users to make choices.
- The Commit page is used to perform an action that cannot be undone by clicking Back or Cancel.
- The Progress page is used to show the progress of a lengthy operation.

Modern wizard design places a premium on efficiency, making the Progress page optional for shorter operations, and often dispensing with the traditional Welcome page and Congratulations page at the beginning and end.

All wizard pages have these components:

- A title bar to identify the name of the wizard, with a Back button in the upper-left corner, and a Close button with optional Minimize/Maximize and Restore buttons. Note that the title bar also includes an icon to identify it on the taskbar.
- A main instruction to explain the user's objective with the page.
- A content area with optional text and possibly other controls.
- A command area with at least one commit button to commit to the task or proceed to the next step.

Although a wizard has multiple steps, these steps must all add up to a single task, from the user's point of view. This is the fundamental wizard design principle of "one wizard, one task."

Thus in this article, a task is the basic function of a wizard (for example, the task of a setup wizard is to install a program). Sub-tasks are aspects of the larger task (for example, a sub-task of a setup wizard may be to configure the program to be installed). Finally, each wizard page is considered a step in a given sub-task or task (for example, there may be two or three steps involved in configuring the program).

Note: Guidelines related to setup, dialog boxes, and progress bars are presented in separate articles.

Is this the right user interface?

A wizard can be used for any task that requires multiple input steps. However, effective wizards have additional requirements:

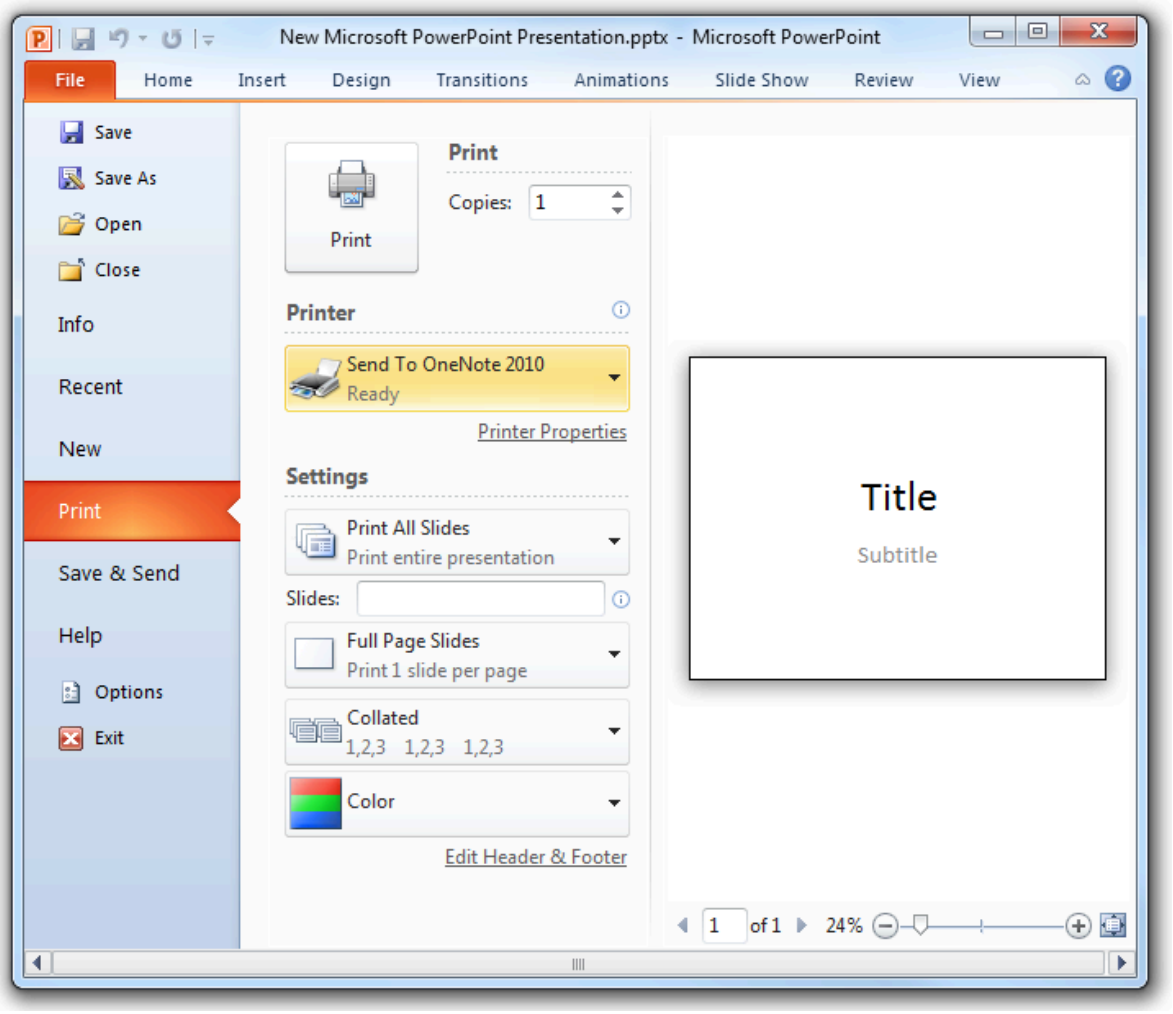
- **Does the wizard perform a single, atomic task?** Don't use interactions that aren't single tasks (a whole program should never be a wizard unless it performs a single task). Don't use wizards to combine independent tasks or largely unrelated steps.

- **Can the number of required questions be reduced? Are there acceptable defaults that either work well for most cases or can be adjusted as needed later?**

Consequently, can the number of pages be reduced? If so, try to simplify the task so that it can be presented on a single page (such as a dialog box), or eliminate the need for input completely (allowing for the task to be performed directly).

- **Must the required questions be provided sequentially? Are there several probable, but optional questions?** If so, consider a dialog box or tabbed dialog box.

  Correct:



  The Microsoft PowerPoint Print options dialog box contains many user input options, so you could present them in a wizard. However, there is no need to provide them sequentially, so a dialog box is a better choice.

Wizards are a relatively heavy form of user interface; if there is a suitable, lighter-weight solution available, use it!

Design concepts

Overuse of wizards

Historically, wizards differed from ordinary UI in that they were designed to help users perform especially complex tasks (with steps residing in disparate locations) and often had built-in intelligence to help users succeed. Today, all UI should be designed to make tasks as simple as possible, so there is no need for a special UI just for this purpose.

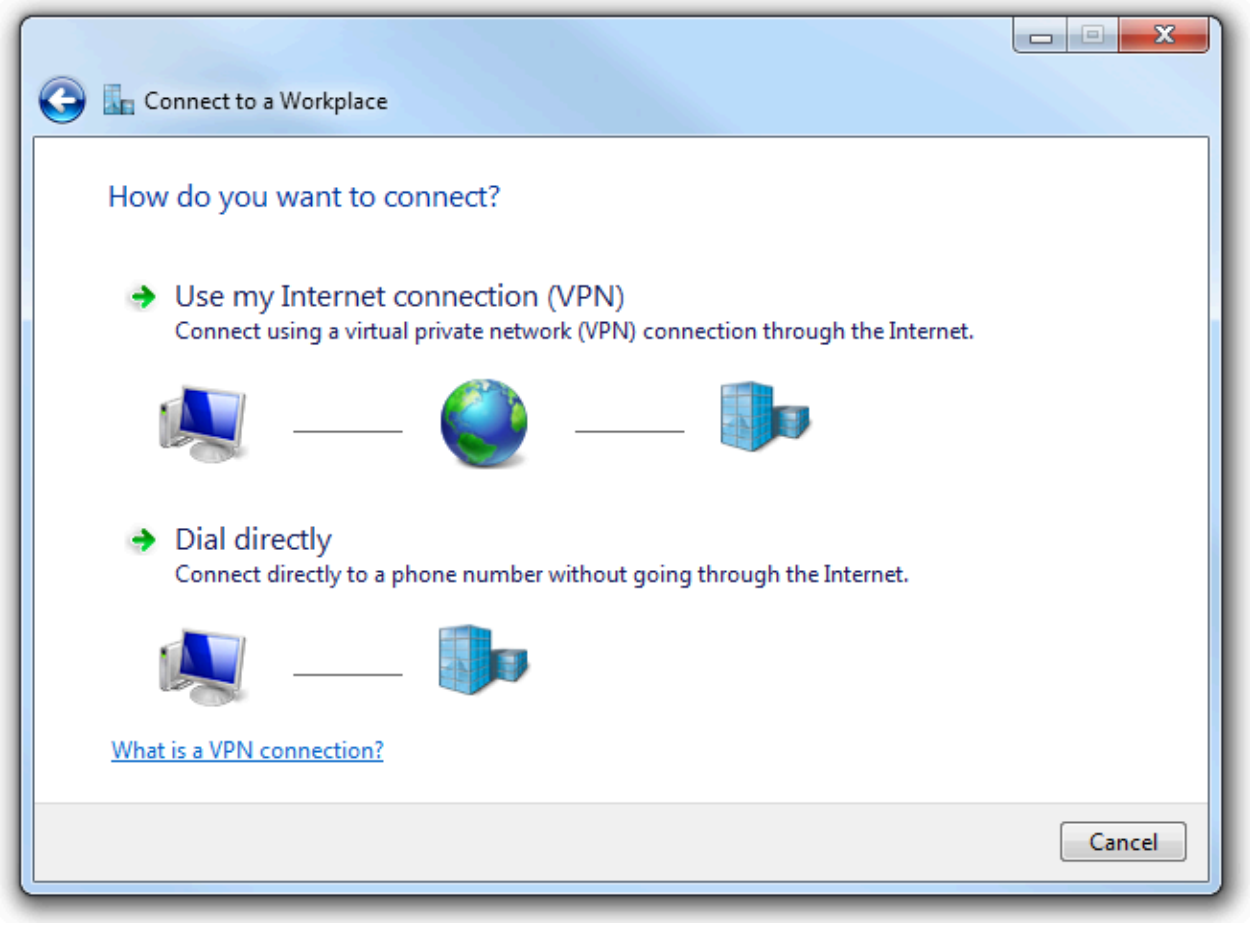
Yet the belief persists that wizards are a special UI—largely because they are referred to as "wizards" (much more creative than, say, "dialogs" and "property windows"). Instead, it's better to consider them to be multi-step tasks and not draw special attention to that fact.

Before creating a wizard, consider whether users really must be interrupted from the main flow of the program. There may be a lighter, inline, contextual solution that will ultimately feel more helpful and efficient to users. For example, a badly designed feature in a program doesn't warrant a wizard to explain and simplify it; it warrants redesign of the feature itself. A wizard should not be used as a band-aid to fix a more basic problem with the program.

Wizards do have appropriate functions

Wizards are one of the keys to simplifying the user experience. They allow you to take a complex operation, such as configuration of a program, and break it into a series of simple steps. At each point in the process, you can provide an explanation of what is needed, and display controls that allow the user to make selections and enter text.

Certain types of multi-step tasks lend themselves to the wizard form. For example, in Windows, several wizards involve connectivity functions (to the Internet or corporate network, or to peripheral devices such as printers and fax machines).



Connecting to a network is a typical task in Windows appropriate for a wizard.

Here the function of the wizard is to mediate between something known and stable (the out-of-box operating system) and something unknown and variable (connectivity arrangements with a phone company or Internet service provider). The complexity of computing ecosystems is significant enough now that it is genuinely helpful to use wizards to reduce that complexity.

Other types of tasks that work well as Windows wizards include high-end functionality (such as speech and handwriting recognition) and rich media experiences (such as configuring options for making and publishing movies). Wizards can also be deployed for more basic multi-step tasks, such as troubleshooting. In short, if different users are likely to want to experience your program in widely different ways, this can indicate the need for a wizard and its capacity for multiple user input points.

For your program, it is worth a little design time up-front to determine what function your wizard is serving, and whether that function really does rise to the level of deploying a wizard.

Wizard length

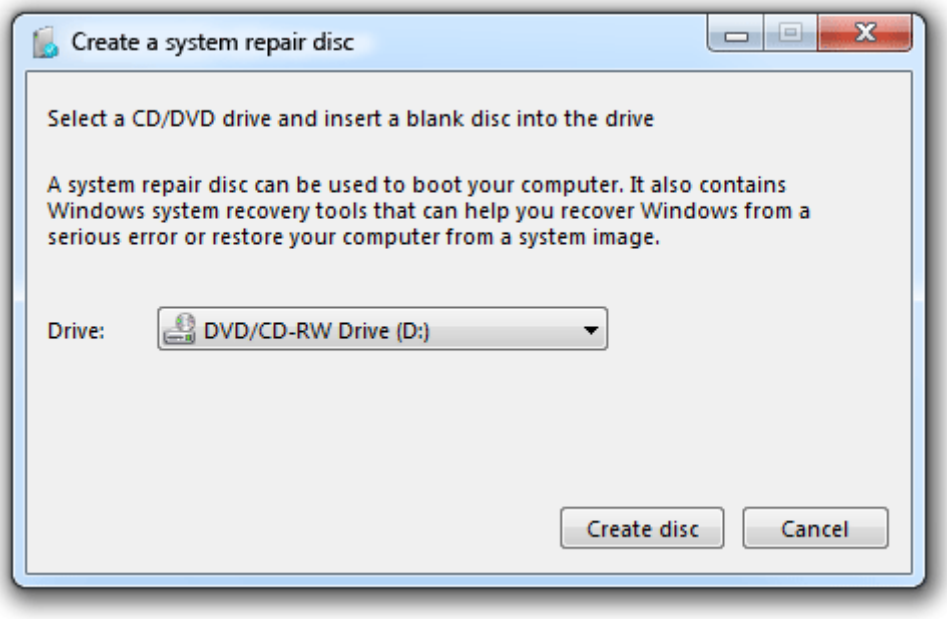
Design questions naturally arise around the number and organization of pages and options. For example:

- Is there an optimal number of pages for a wizard? Or at least a desirable range?
- Should the wizard be concise and streamlined, so users can complete it as quickly as possible?
- Should there be more pages that require fewer choices? Or fewer pages with more complexity? Which design is considered more usable?
- Can you engineer faster wizard experiences by applying UI conventions such as tabbed pages?

Microsoft used to advise that wizards of three pages or fewer be designed as simple wizards, and those of four or more pages use an advanced wizard design (see the Windows User Experience guidelines from 1999). But current wizard design standards dispense with what had been one of the key differences between the simple and advanced forms (the use of the Welcome and Congratulations pages), so these categories now feel inadequate, and the number of pages determining the design choice seems arbitrary.

Your wizard should be as long or short as the task requires; there is no fixed guideline for its length. A one-page wizard should really be presented as a dialog box, so two pages is probably the most condensed form possible for a wizard.

Correct:



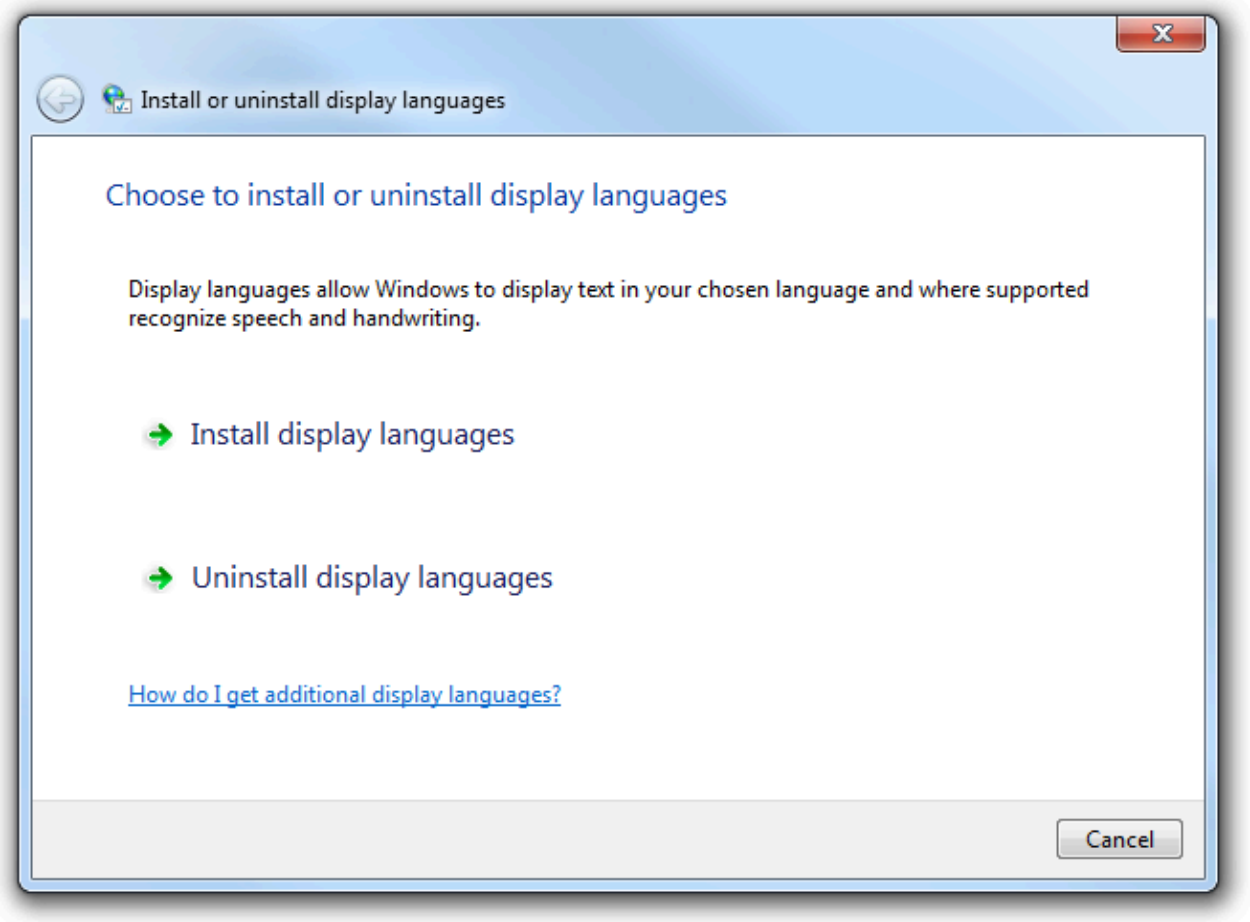
This task has so few options that presenting it as a wizard would be wasteful. A dialog box is the appropriate form for this user interface.

At the other end of the spectrum, if you have a wizard that includes multiple decision points and branches, and frequently results in users losing track of their navigation path, you have exceeded a practical limit and should reduce the length of the wizard. Alternatively, you may be able to break the wizard up into several distinct tasks.

As you determine the most appropriate length for your wizard, pay particular attention to your target users. Programs for end users such as home consumers and office workers tend to use wizards to hide complexity; the wizards are as short as possible, with clean, simple page design, and pre-selected defaults for as many options as possible. By contrast, server wizards or programs intended for IT professionals tend to be longer and more complex. This group of target users has a much higher tolerance for making configuration decisions, and may in fact become suspicious if too much complexity is hidden.

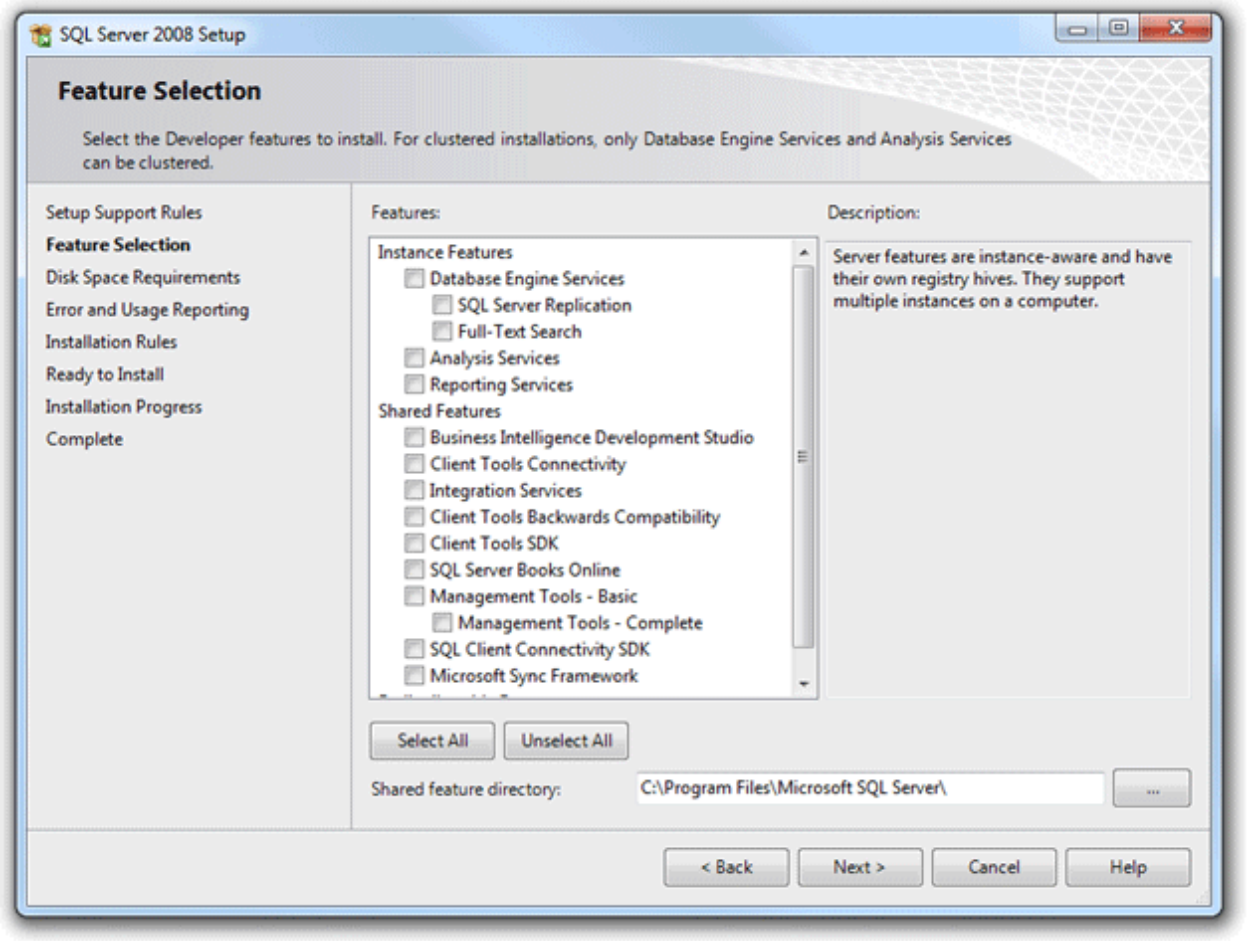
If a wizard by nature simplifies a complex task, it should do so relatively minimally for a technically sophisticated audience, and relatively aggressively for a novice user base.

Correct:



This wizard page is well-designed for end users because it reduces a potentially complex subject to a simple, logical binary choice: either install or uninstall.

Correct:



In the setup wizard for Microsoft SQL Server 2008, page design is busier and the numerous choices require more thought, but the target audience is database administrators who expect tight control of feature selection.

Finally, pay attention to how frequently the particular task might be performed. An infrequent task may deploy a longer wizard, whereas frequent tasks should definitely favor brevity.

Branching

For longer wizards, you may need to create branches of the task flow in which the sequence of pages may differ according to the user input provided "upstream." Branching is inherently dislocating for users, so you must design the user experience to convey stability. We recommend no more than two decision points that will cause branching in the entire wizard, and no more than one nested branch within a single branch.

For guidelines about creating a stable user experience within a branching wizard, see Branching in the Guidelines section of this article.

Providing a navigation guide

Navigation guides can be useful when there are many steps in the task, and users may lose their place in the sequence, or simply want to know how much longer it will take to complete.

Navigation guides often appear as a list of pages or sections of the wizard, looking a bit like a table of contents, in a column or pane on the left side of each page. Although the list persists throughout the wizard (the same list of pages appears on each page), there is some visual means of indicating where the user currently is in the sequence (for example, using bold to distinguish the active page or section).

Navigation guides can be sequential or non-sequential. The sequential type presents the past pages along with the known future pages. You can present the future in terms of steps instead of pages if the steps are known and pages are dependent. You can then populate pages dynamically as they become known. Because the navigation sequence is fixed, the navigation guide isn't interactive.

Non-sequential navigation guides are interactive, so users can revisit previously viewed pages directly. They can also skip ahead of the navigation sequence for pages that are designed to be optional. Optional pages must have defaults that are acceptable in most circumstances. With this type of guide:

- Previously viewed pages can always be viewed directly.
- Future pages may not be viewed if they have prerequisites.
- Pages that can be visited should be visibly distinguished from those that can't (such as by using links that are active or disabled), along with pages that are required or optional.

Users can become confused about the meaning of the Back button in this scenario. Does clicking Back lead you to the previous page or section in the navigation guide, or the last page or section viewed? Because Windows wizards now place the Back button in the upper-left corner of wizard pages, rather than in the lower-right corner with the other commit buttons, users think of Back functionality as they do on the Web. So the best solution is to give your Back button the Web navigation meaning (clicking Back should lead to the last page or section viewed), and use the wizard navigational guide for sequential navigation.

Page integrity

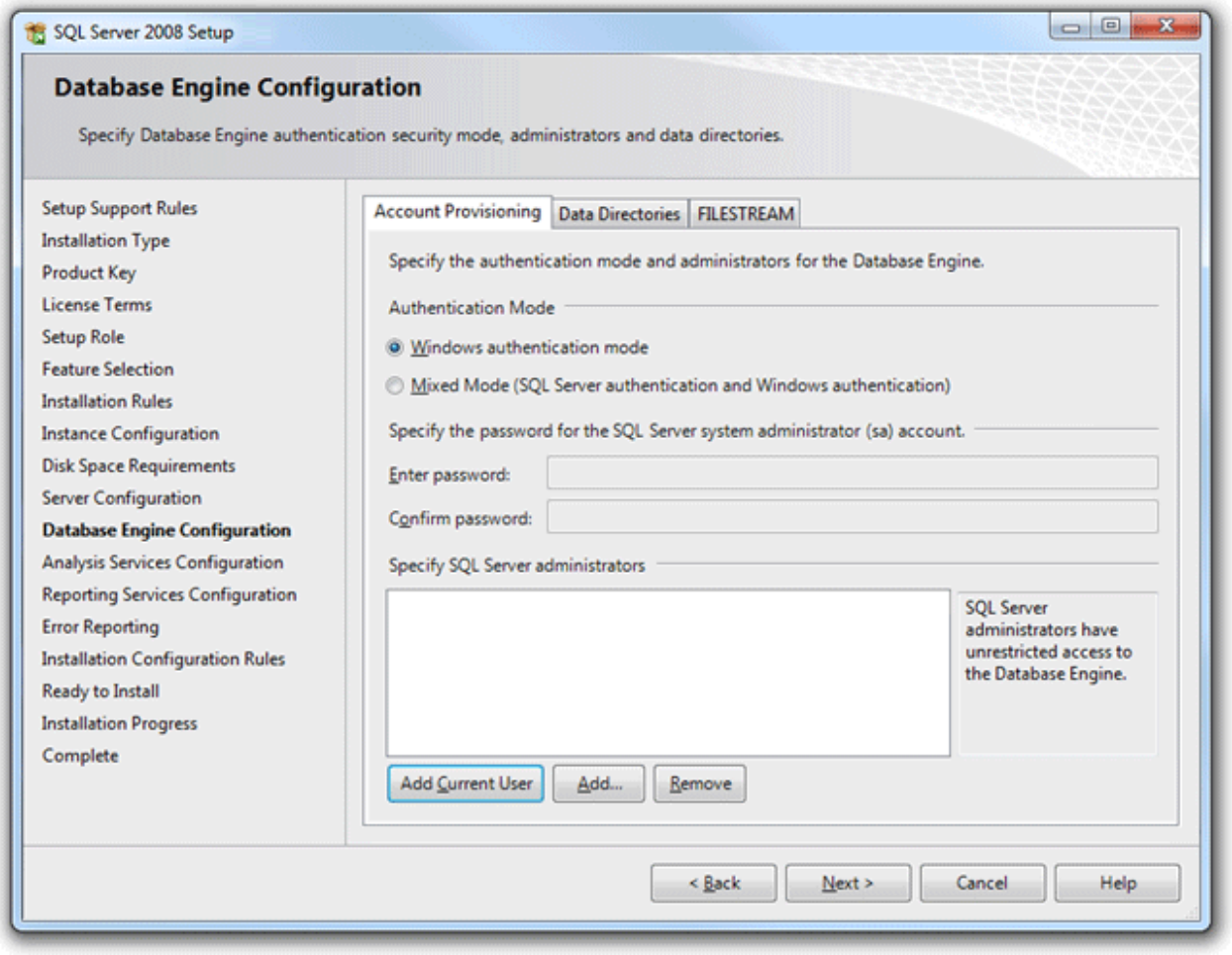
Wizard design involves not only decisions pertaining to the entire task flow, like how to handle navigation and the branching experience, but also those pertaining to the individual pages that make up the wizard. **The most important principle for designing good wizard pages is that of integrity: the contents of a page should belong together.**

Wizard pages are significantly more usable if each one hangs together conceptually, dealing with only one aspect of the overall task. The main instruction is the primary means of achieving this. Clearly identify the goal or purpose of the page to users. Supplemental instructions, and any controls on the page, all pertain directly to the main instruction. Although wizard pages should present users with options for which some thought is required, that effort doesn't feel like work because it is tightly focused by the integrity of the page itself.

Unfortunately wizard designers often mistake users' rapid clicking of the Next button as evidence of the usability, simplicity, and integrity of their pages. The ultimate wizard experience isn't Next, Next, Next, Next, Finish. While such an experience suggests that the defaults were well chosen, it also suggests that the wizard wasn't really necessary because all the choices are optional.

In terms of visuals and text, pare down these elements to the bare essentials. Resist the urge to bundle up multiple sub-tasks on a single page (the "burrito wizard") or to resort to tabs for presenting complex input requirements. A single page should cover a single sub-task of the overall task of the wizard.

Incorrect:



With three tabs of fairly dense user input required, this wizard page is trying to accomplish too much.

In most cases, maintain the size of each page throughout the wizard to foster a consistent look and feel. Although Windows wizards allow for resizable pages so that the size of a page matches the amount of content, only a few make use of this option.

And finally, maintain structural elements of each wizard page through the sequence. For example, don't move the Back button from the upper-left corner back down into the commit buttons area for a page or two. This level of layout consistency helps users feel stable within the wizard. Think of this as a baseline for the visual integrity of a page.

Finding the right level of communication

Users have a low tolerance for reading big blocks of text on screen, and even less so within a UI surface whose express purpose is to move expeditiously through a task.

Wizards have a tendency to over-communicate. They take up a lot of space on the screen, which seems to encourage a drive to fill the space. It's like a variation on Parkinson's Law: UI text will expand to fill the space available.

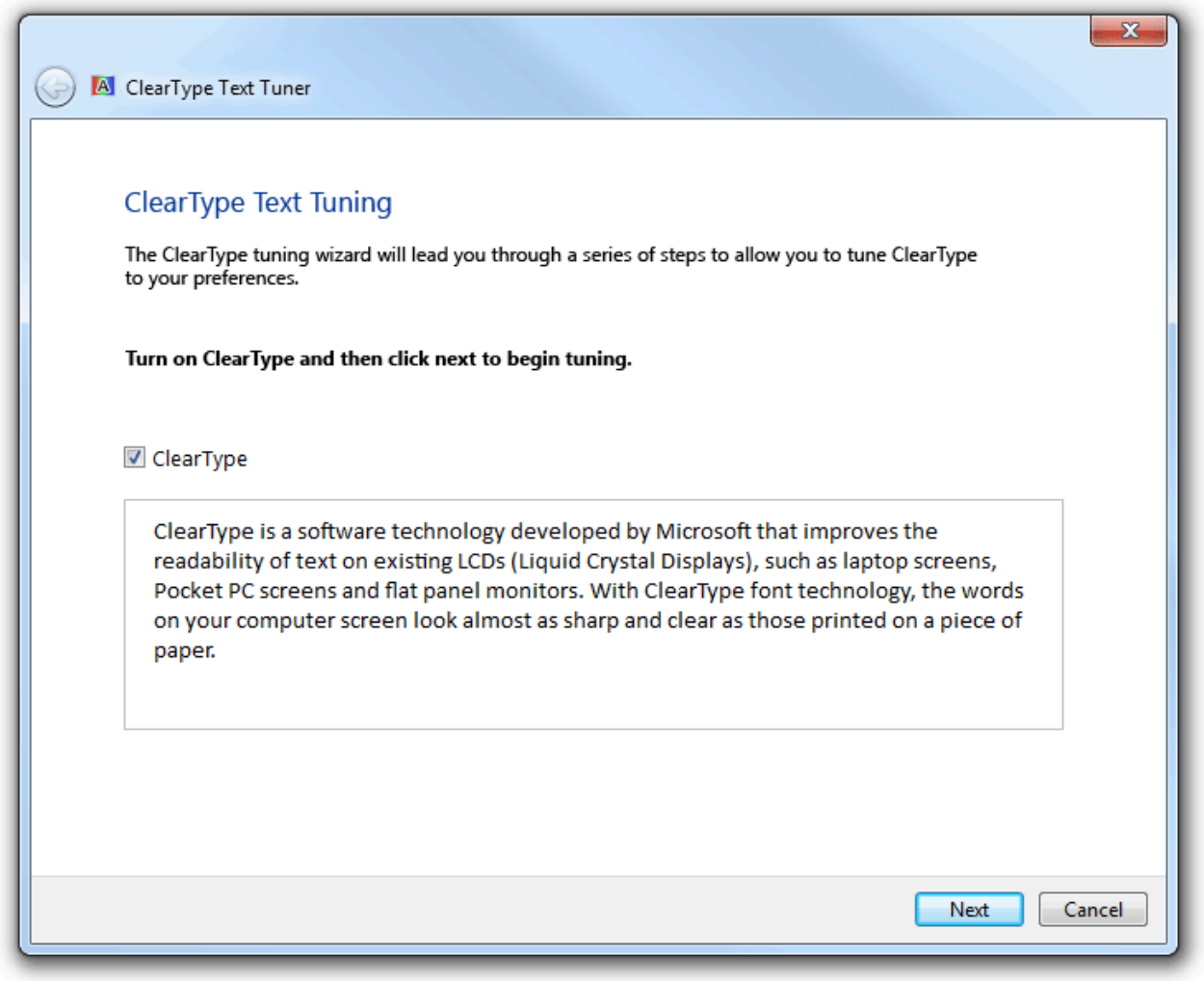
One culprit in this excess is redundancy. Because of templates used in early wizard design, the same language might appear in multiple locations on a page, such as in the title bar, headings, body text, control labels, and so on.

It's worth it to hire a professional editor to prune your wizard text ruthlessly. Eliminate unnecessary questions and options on individual pages, and eliminate entire pages from the wizard as a whole (for example, the traditional Welcome and Congratulations pages). Get right to the point of the page with a concisely written main instruction, using language your target audience uses to describe the task, not the jargon of the technology or feature that you or your team uses internally. This user-centric approach is vital to improving the communication of your program's wizards.

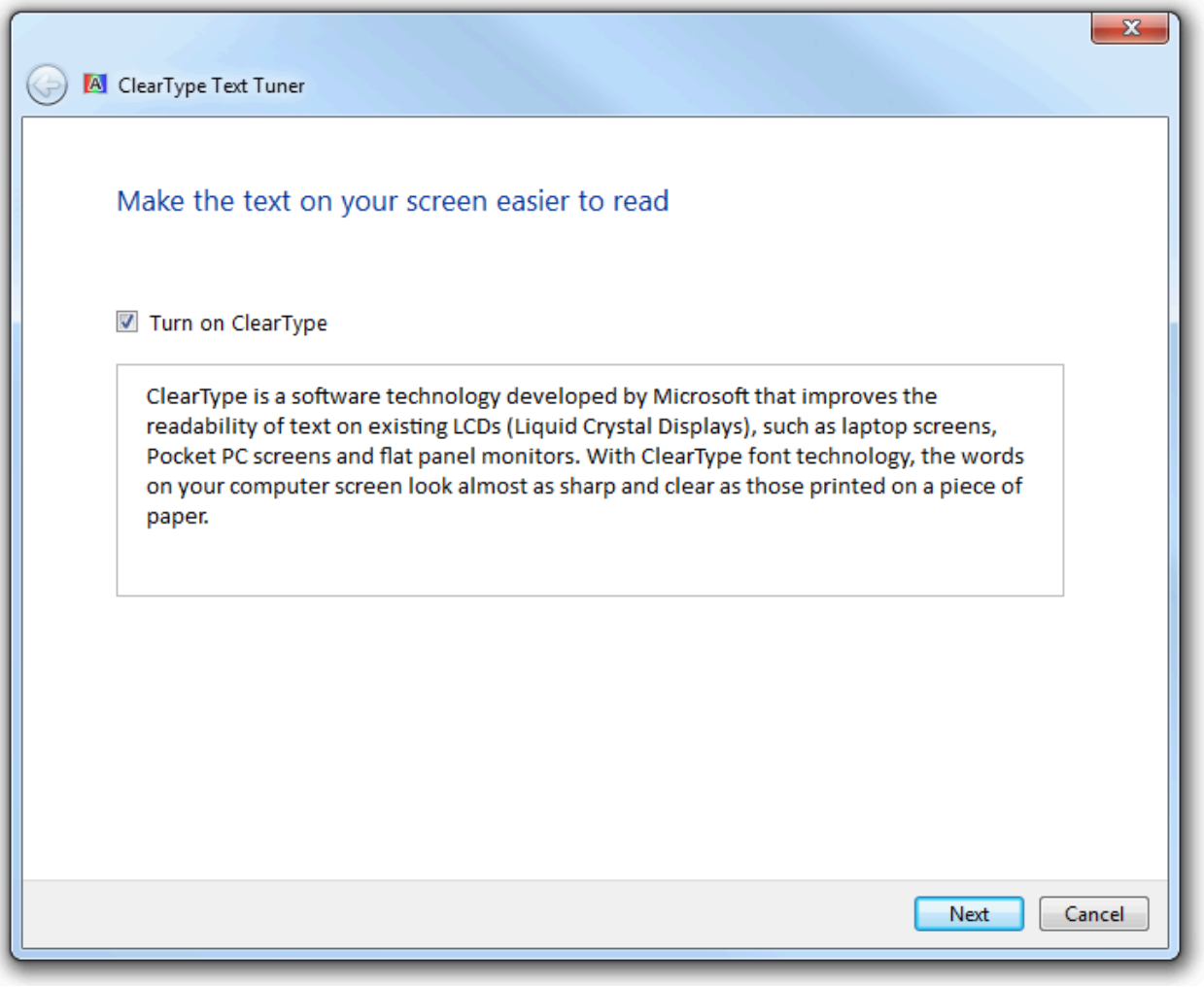
Pay special attention to the tone of your wizard: sometimes the most lasting impressions of your program are the result not of what you say but how you say it! In wizards, users are comfortable with a friendly, conversational tone, with liberal use of the second-person pronoun ("you") when the program is asking for input. For more guidelines, see Style and Tone.

Reducing word count on the wizard page is generally commendable, but be careful not to go too far. If the task is important and warrants a wizard, users do appreciate having enough information to make wise choices. The following example shows how wizard text can be condensed without sacrificing meaning.

Before:



After:



The edited version of this wizard page provides a task-oriented main instruction, removes the unnecessary explanatory paragraph beneath the main instruction, and revises the check box label to clarify the check box's purpose.

If you do only three things...

1. Map the task you are trying to accomplish with the appropriate UI to do the job; don't simply default to a wizard when you think you need to collect a lot of input from users.

2. Think carefully about the length and structure of your wizard; prefer short, non-branching wizards to keep the experience as simple as possible, so users can get back to their primary task or interest in your program.

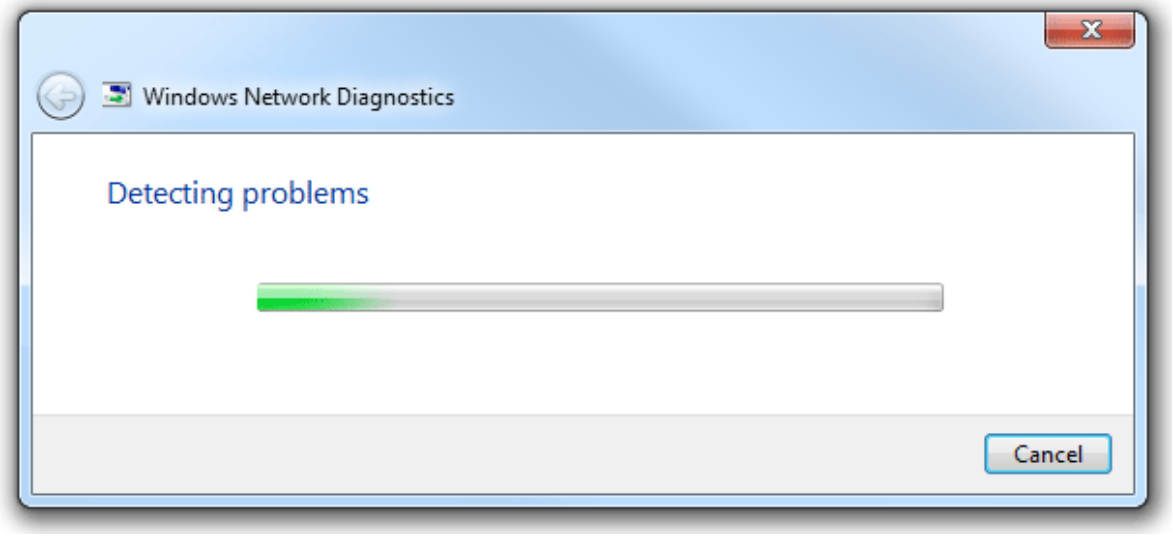
3. Ensure the integrity of each page in your wizard: the contents of a page should clearly belong together.

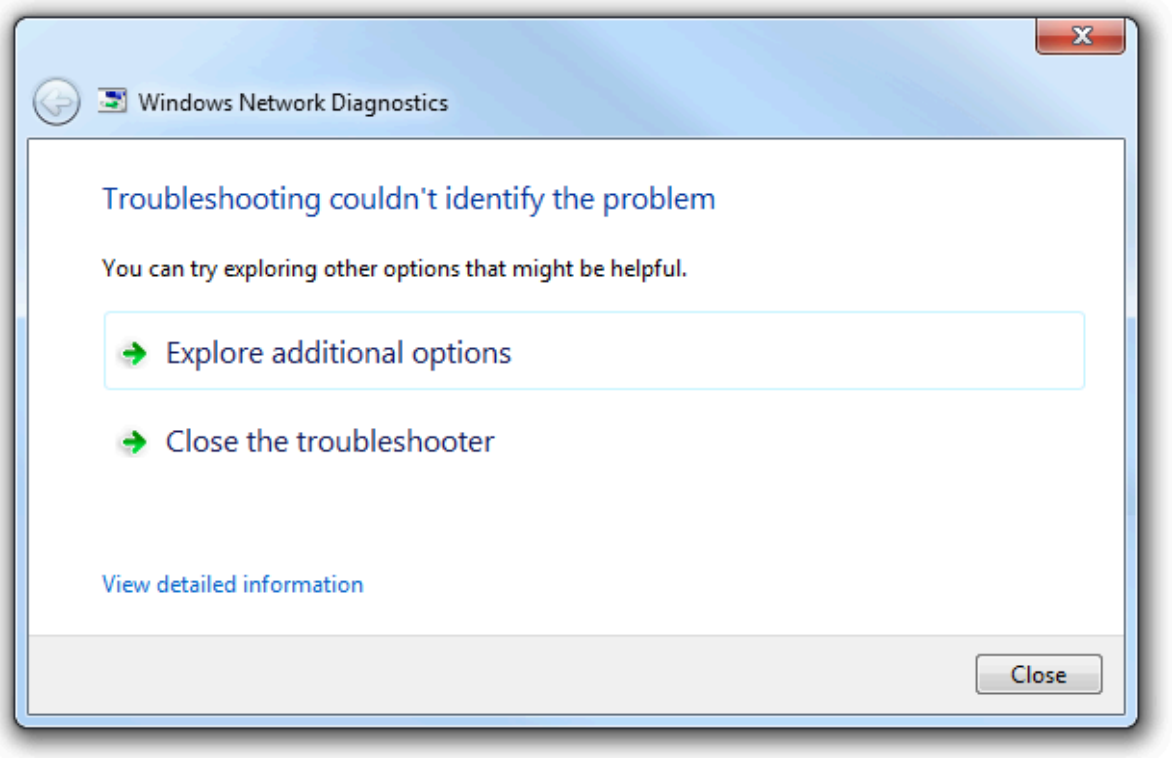
Guidelines

General

- **Consider lightweight alternatives first, such as dialog boxes, task panes, or single pages.** You don't have to use wizards—you can provide helpful information and assistance in any UI.
- **Use wizards for multi-step tasks.** Use multi-page dialog boxes for single-step tasks with feedback. For more guidelines, see Dialog Boxes.

  Correct:



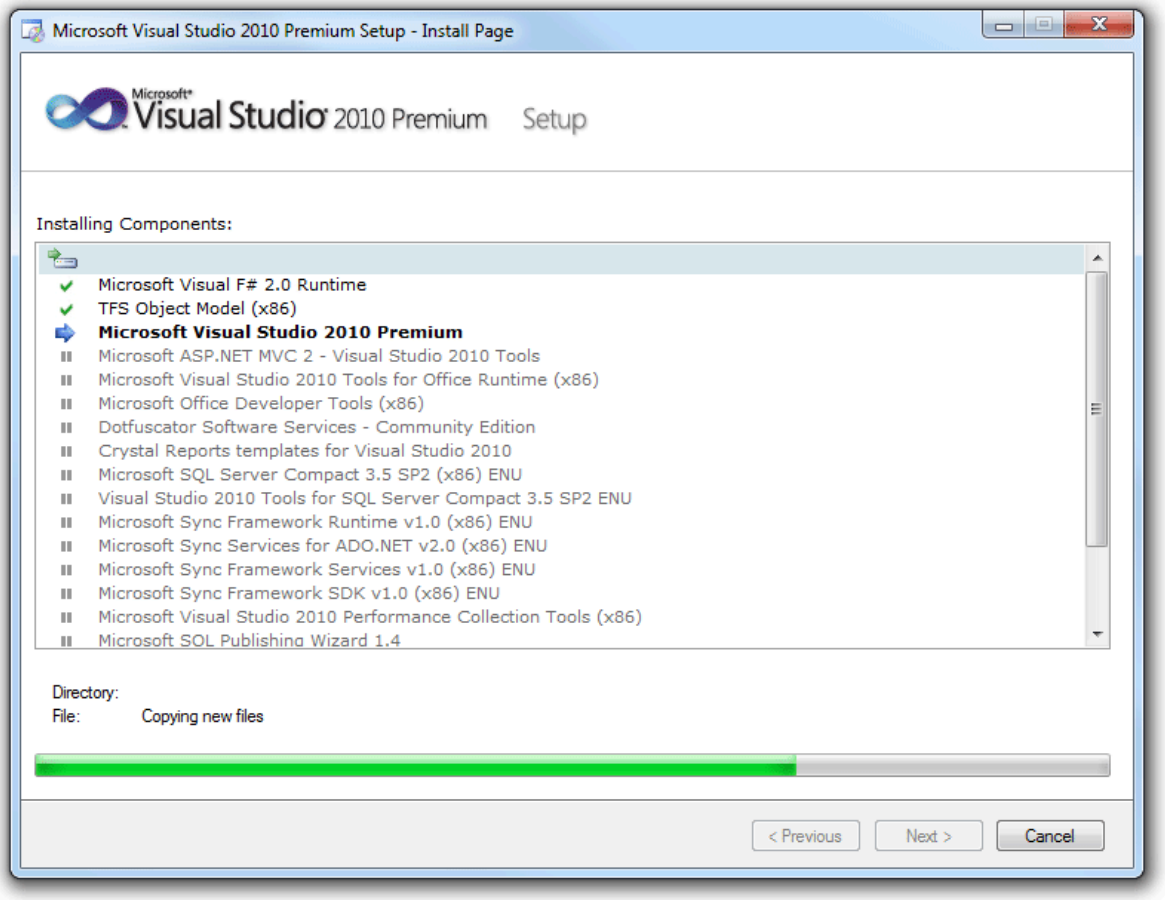


  In this example, Windows Network Diagnostics consists of progress and results pages. Because the task is only a single step, it does not require the navigational buttons that users need in a wizard. It is effectively presented as a multi-page dialog box.

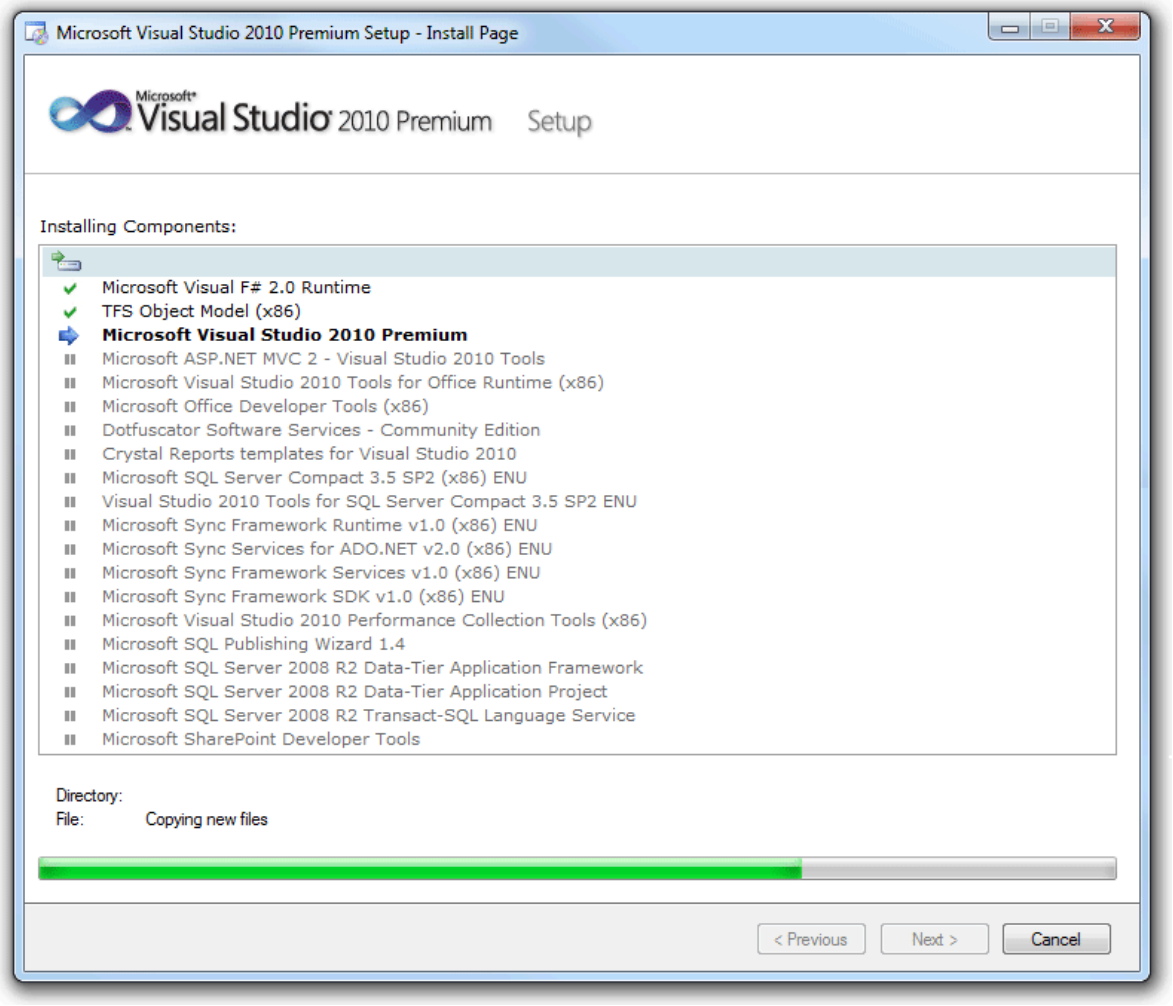
Window size

- **Choose a window size that can display all the wizard pages without vertical or horizontal page scrolling.** While the controls on the page may require scrolling, the wizard pages themselves must not.
- **Size windows large enough to perform their tasks comfortably.** Page layout shouldn't be cramped or require users to scroll or resize excessively.
- **But don't make windows excessively large.** Larger windows make the task feel more complex and require additional movement for interaction.
- **Use resizable windows for a wizard that can benefit from more screen space but doesn't require it.** Assign an appropriate minimum size. Resizable windows are helpful when pages require interacting with resizable content such as large list views.

  Correct:



  Better:



In this example, resizing the window helps users see the full list.

- **Consider using dynamically sized wizards whose page size changes as needed for its content.** Doing so allows a wizard to accommodate page layouts with a wide range of content.

- **Prefer static sizing over dynamic if users may perceive the changes as a lack of stability in their experience of the wizard.** Visual stability often trumps accommodation of content. Most wizards should adopt standard, static window sizes, with dynamic sizing reserved for special cases.

Wizard length

- **Make your wizard as concise and streamlined as possible.** Get rid of unnecessary options and questions, and use smart defaults, to reduce the number of pages required for user input.
  - **Exception:** IT professionals and other technical users have a higher tolerance for longer wizards and detailed input requirements.
- **Make your wizard a minimum of two pages.** A one-page wizard should be redesigned as a dialog box instead.

- **Don't reduce the wizard's page count simply by increasing the complexity of each page.** For example, a wizard page that includes three tabs requiring user input should be redesigned as three separate pages.
- **Don't increase the wizard's page count by making each page so simple that users mindlessly click Next through the whole sequence.** This is a common wizard design flaw. If a wizard page does not require at least some degree of thought, it probably doesn't need to be in the wizard at all.

Branching

- **Prefer non-branching wizard design over branching.** Non-branching wizards tend to be simpler, shorter, and easy to navigate. Branching wizards make it more difficult for users to determine how many steps in the task, and where they are in the sequence.

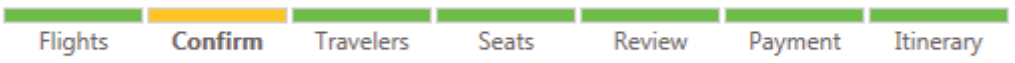
- **If you must branch, help users orient themselves by using one of the following techniques:**

  - **Enumerate pages.** A common technique is to indicate the user's location in the sequence on each page, such as with the phrase Step X of Y. Ensure that the endpoint (Y) is stable. If it changes value, this undermines users' confidence.

  - **Include the notion of sub-steps** (such as Step 2a of 6).

  - **Make steps independent of pages, where each step may involve several pages.** For example, a travel service might employ wizard organization based on well-established e-commerce conventions for the industry.

    Correct:

    Logical labels can provide adequate orientation for users of a branching wizard.

  - **Treat optional steps as persistent in the enumeration sequence.** For example, if a branch is just skipping a few optional steps, just skip the steps in the feedback as well, rather than renumbering. Thus if a user makes a choice on page 2 that results in making pages 3 and 4 optional, show steps 1, 2, 5, and 6 of 6. Don't renumber steps 5 and 6.

  - **If the wizard employs a single branch, and the branch happens early in the task, start the sequence at that point, and then simply use the non-branching**

approach. That is, beginning at the point of the branch, progress in sequence to the end of the branch.

- **If you must branch, limit the number of branches to one or two within a single wizard.** Never include more than one branch within a branch (a "nested" branch).

Commit buttons

- **When users are committing to a task, use a commit button that is a specific response to the main instruction** (for example, Print, Connect, or Start). Don't use generic labels like Next (which doesn't imply commitment) or Finish (which isn't specific) for committing a task. The labels on these commit buttons should make sense on their own. Always start commit button labels with a verb. **Exceptions:**
  - Use Finish when the specific responses are still generic, such as Save, Select, Choose, or Get.
  - Use Finish to change a specific setting or a collection of settings.
- **A single wizard can have multiple commit points, but a single point is preferred.**
- **If necessary, you can rename or hide commit buttons on a page.** This flexibility is one advantage of new wizard design in Windows that was not available in older wizards. Note that hiding a commit button is different from disabling it.
- **Avoid disabling a positive commit button.** Otherwise users have to deduce why the commit buttons are disabled. It's better to leave commit buttons enabled, and give a helpful error message whenever a problem arises. Disabling the button is only acceptable if the reason for doing so is obvious and unambiguous.
- **Don't confuse navigational buttons (Next and Back) with commit buttons.** Next means to progress in the wizard without commitment; Back should always be available on the next page, and clicking Back should undo the effect of the last Next button. If that's not possible, users are making a commitment, and that is indicated through a specific label on the commit button. For more guidelines about the Next and Back buttons, see Navigation.

Cancel buttons

- **Don't ask users to confirm whether they really intend to cancel.** Doing so can be annoying. **Exceptions:**
  - The action has significant consequences and, if incorrect, is not readily fixable.
  - The action may result in a significant loss of the user's time or effort.
  - The action is clearly inconsistent with other actions.
- **Allow users to restart wizards in case they have cancelled by mistake.**
- **Don't disable the Cancel button. Exceptions:**

    - If cancelling is harmful, which might be the case when doing a task in self-contained wizards.
    - If cancelling is impossible, which might be the case when the wizard doesn't have control over all steps.

Close buttons

- **Use Close for Follow-Up and Completion pages.** Don't use Cancel, because closing the window won't abandon any changes or actions done at this point. Don't use Done, because it isn't an imperative verb.
- **Once the task has been performed, Cancel should become Close (for self-contained wizards).** The effect of Close is simply to close the window.

Other controls

- **Use command links only for choices, not commitments.** Specific commit buttons indicate commitment far better than command links in a wizard.
- **When using command links, hide the Next button, but leave the Cancel button.**

Using pages (vs. dialog boxes or inline UI)

- **In general, prefer pages to dialog boxes.** Users expect wizards to be page-based.
- **Use dialog boxes to assist in completing pages,** such as with object pickers and browsers.
- **Use dialog boxes to give error messages that apply to the whole page, and result from clicking a commit button.**
- **Use inline presentation for simple dynamic behaviors,** such as progressive disclosure and contextual UI.
- **Use inline presentation for error messages that apply to specific controls.**

Wizard pages

- **Focus on efficient decision making.** Reduce the number of pages to focus on essentials. Consolidate related pages, and take optional pages out of the main flow. Having users click Next completely through your wizard may seem like a good experience at first, but if users never need to change the defaults, the pages are probably unnecessary.

- **Design each page to have a single purpose and visual consistency.** For more information, see Page integrity.

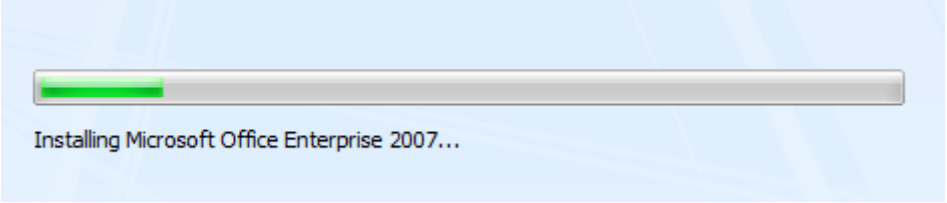
- **Don't use Welcome pages—make the first page functional whenever possible.** Use an optional Getting Started page only when:
  - The wizard has prerequisites that are necessary to complete the wizard successfully.
  - Users may not understand the purpose of the wizard based on its first Choice page, and there isn't room for further explanation.
  - The main instruction for Getting Started pages is "Before you begin:".

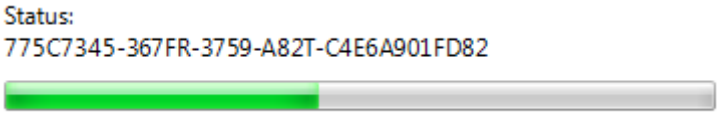
  Incorrect:

- Modern wizards opt for functional first pages. Here there is nothing to do but click Next. Why force users to pay this token tax on their valuable time?

- **On pages in which users are asked to make choices, optimize for the most likely cases.** These types of pages should present actual choices, not just instructions.
  - If you don't use a Getting Started page, explain the purpose of the wizard at the top of the first page of choices.

- **Use Commit pages to make it clear when users are committing to the task.** Usually the Commit page is the last page of choices, and the Next button is relabeled to indicate the task being committed.
  - Don't use Summary pages that merely summarize the user's previous selections, unless the task is risky (involving security, or loss of time or money) or there is a good chance that users need to review their selections.

- **Use Progress pages to show the status of a lengthy operation.** Upon successful completion, the progress page should advance to the next step automatically. It should stay on the progress page only if there is a problem the user needs to see. Clicking Back to a progress page should have no side effect.
  - **Use a single, determinate progress bar.** Follow the determinate progress bar guidelines, including:
    - Clearly indicate completion. Don't let a progress bar go to 100 percent unless the operation has completed.
    - Don't restart progress. A progress bar loses its value if it restarts (perhaps because a step in the operation completes) because users have no way of knowing when the operation will complete. Instead, have all the steps in the operation share a portion of the progress and have the progress bar go to completion once.
  - **Provide a concise description of the current step above the progress bar.** For quick operations, such text is unnecessary; the progress bar alone is sufficient. For operations requiring a minute or longer, text can be helpful.
    - Use sentence fragments, typically beginning with a verb, and ending with an ellipsis. Examples: Copying files..., Installing required components....
    - Place text above the bar, not below.
    - **Incorrect:**
    - 

    - In this example, the explanatory text should appear above the progress bar.
    - Refrain from cluttering the progress page with unnecessary details. This page isn't for technical support; it's for users.
    - **Incorrect:**
    - 

    - In this example, technical details such as GUIDs are meaningless to users.

- **Don't use Congratulations pages that do nothing but end the wizard.** If the wizard results are clearly apparent to users, just close the wizard on the final commit button.
  - Use Follow-Up pages when there are related tasks that users are likely to perform as follow-up. Avoid familiar follow-up tasks, such as "Send an e-mail message."

    - Use Completion pages only when the results aren't visible and there's no better way to provide feedback for task completion.
    - Wizards that have Progress pages must use a Completion page or Follow-Up page to indicate task completion. For long-running tasks, close the wizard on the Commit page and use notifications to give feedback.

- **Use Summary pages only if input is complex and users need to review, if the task involves significant risk (such as a financial transition), or if the wizard will take action based on user input that isn't obvious (to build trust through transparency).** Often, summary pages don't meet this relevancy bar, and can be omitted.

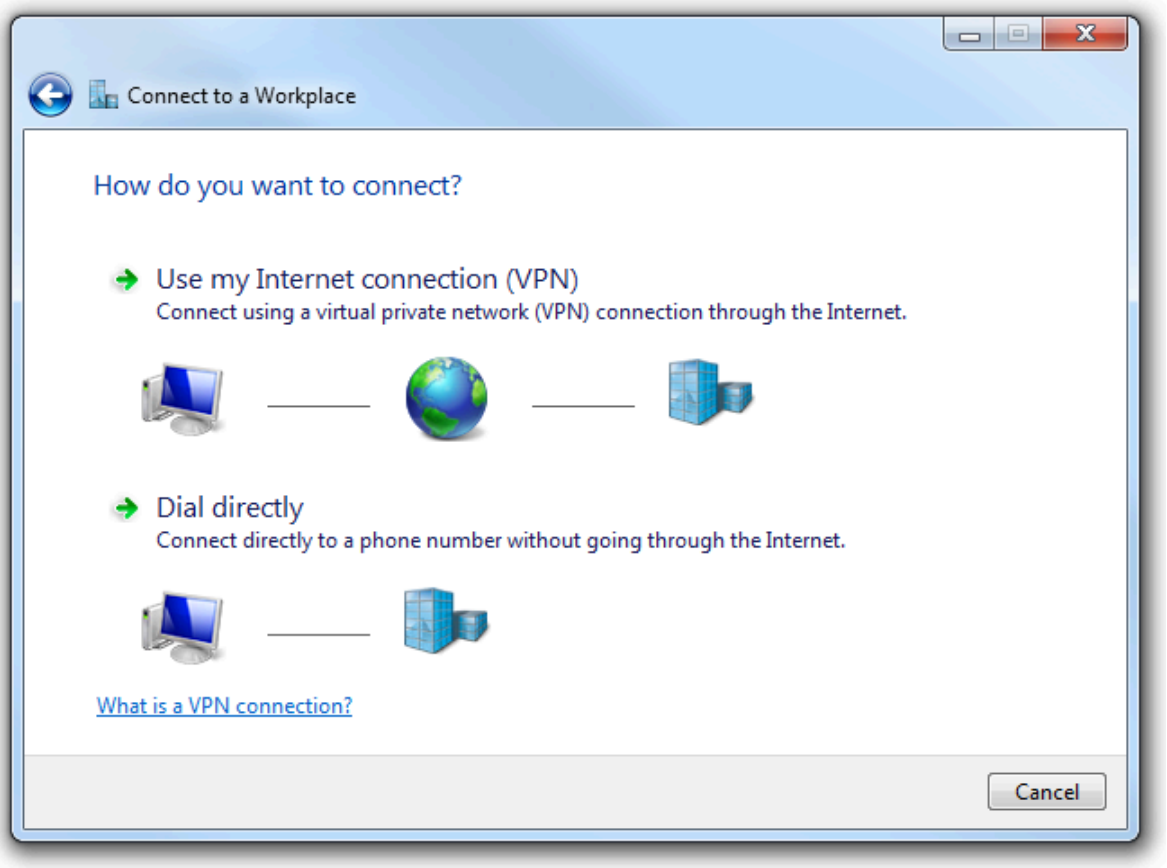
- **Use error pages if the wizard can't be completed due to a problem from which recovery isn't possible.** On this page, explain what the problem is in clear language, free of technical jargon users won't understand. Also provide practical steps users can take to solve the problem. For more guidelines, see Error Messages.
    - **Exception:** If the wizard completes with a minor problem from which recovery is possible, present the problem as an additional task instead of an error. Use positive, success-oriented, encouraging language, not terms like error, failure, or problem. Don't use an error icon.

Navigation

- **Use Next only when advancing to the next page without commitment.** Advancing to the next page is considered a commitment when its effect can't be undone by clicking Back or Cancel.
- **Use Back only to correct mistakes.** Aside from correcting mistakes, users shouldn't have to click Back to make progress in a task.
- **Preserve user selections through navigation.** For example, if the user makes changes, clicks Back and then Next, those changes should be preserved. Users don't expect to have to re-enter changes unless they explicitly chose to clear them.
- **Don't disable the Back button unless repeating the steps is harmful.**
- **Permit users to browse or revise choices in the following navigation scenarios:**
    - User gives input, clicks commit button, clicks Back to review previous changes, doesn't change anything, and then clicks commit button again. Normally, this should be possible and the second commit should just advance to the next page (because the task has already been done).
    - User gives input, clicks commit button, clicks Back to review previous changes, changes something, and then clicks commit button again. Normally, this should

be possible and the second commit should redo the task with the changed input (replacing or undoing the effect of the first).

Help

- **Design wizard pages to provide enough information so that referring to documentation in program Help is unnecessary.** A wizard is already taking users away from their desired, direct interaction with the program; requiring users to seek external Help removes them even further from this state. Help should be the exception, not the rule.
- **If you must provide an access point to Help, use a link in the lower-left portion of the content area of the page (above the command area).** This link should be brief, and usually phrased in the form of a question that users are most likely to want answered.
- **Correct:**
- 

- This link to Help is appropriate because basic background information like this would clutter the wizard page too much.

Text

General

- Use you and your to refer to the user and the user's computer, document, settings, and so on. Don't use the first person (I, my) to refer to the computer or the wizard. However, it is acceptable to use the first person in options that the user selects. **Example:My use only** check box.
- Every wizard page must have a main instruction.

Titles

- Put the name of the wizard in the title bar. Use title-style capitalization.
- Titles shouldn't include punctuation, except for those with question marks.
- Don't include the word Wizard in wizard titles. For example, use Connect to a Network instead of Network Setup Wizard.

Buttons

- Don't include text on the Back button. Use the arrow glyph instead, unlabeled.
- Do include text on the Next button. Don't use glyphs (such as > or >>) in addition to the word Next.
- Use specific commit button labels that make sense on their own and are a response to the main instruction. Ideally users shouldn't have to read anything else to understand the label. Users are far more likely to read command button labels than static text.
- If possible, don't use the word Finish for the commit button label, because there is usually a better, more specific commit button:
    - If clicking the button commits to the task (so the task hasn't already been performed), use a specific label that starts with a verb that is a response to the main instruction (examples: Print, Connect, Start).
    - If the task has already been performed within the wizard, use Close instead.

      Exceptions:
        - You can use Finish when the specific label is still generic, such as Save, Select, Choose, or Get.
        - You can use Finish when the task involves changing a setting or collection of settings.
- Start commit button labels with a verb. Exceptions are OK, Yes, and No.
- Use sentence-style capitalization.

- Don't use ending punctuation.

Documentation

- Although most Windows wizards no longer have the word Wizard in the title, it's acceptable to refer to wizards as wizards in documentation. This reference should be lowercase.
- **Correct:**
- If you're setting up a network for the first time, you can get help by using the **Connect to a Network** wizard.
- Some legacy wizards from earlier versions of Windows might include Wizard in the title. When referring to one of those wizards, it's acceptable to use the [X] Wizard to avoid saying the [X] Wizard wizard.
- Refer to an individual screen within a wizard as a page.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Property Windows

Article • 02/09/2021

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

Property window is the collective name for the following types of user interfaces (UIs):

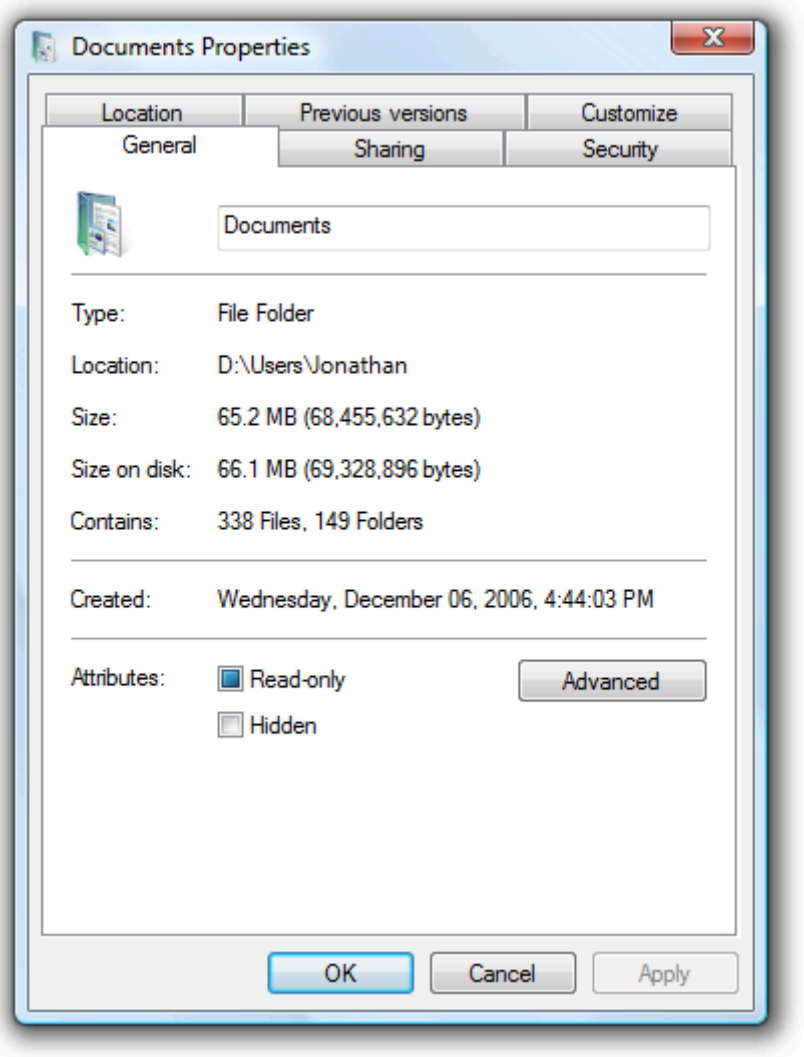
- Property sheet: used to **view and change properties for an object or collection of objects in a dialog box**.
- Property inspector: used to **view and change properties for an object or collection of objects in a pane**.
- Options dialog box: used to **view and change options for an application**.

A property for an object is either of the following:

- A setting that users can change (such as a file's name and read-only attribute).
- An attribute of an object that users cannot directly change (such as a file's size and creation date).

Unlike dialog boxes (other than options dialogs) and wizards, property windows typically support several tasks instead of a single task.

Property windows are usually organized into pages, which are accessed through tabs. Property windows are often associated with tabs (and vice versa), but **tabs are not essential to property windows**.



A typical property sheet.

Note: Guidelines related to layout and tabs are presented in separate articles.

Is this the right user interface?

To decide, consider these questions:

- **Does setting the properties require users to perform a fixed, non-trivial sequence of steps?** If so, use a wizard or task flow instead.
- **Is the content solely an application's options?** If so, use an options dialog box.
- **Is the content solely an application's attributes?** If so, use an About box.
- **Is the content mostly an object's properties (its settings or attributes)?** If not, use a standard dialog boxe or tabbed dialog box.
- Are users **likely to view or change properties frequently or over an extended period of time?** If so, use a property inspector; otherwise, use a property sheet.
- Are users **likely to view or change properties for several different objects at a time?** If so, use a property inspector; otherwise, use a property sheet.

Property sheets and property inspectors aren't exclusive. You can display the most frequently accessed properties in a property inspector, and the complete set in the property sheet.

Design concepts

Property windows often become a dumping ground for an odd assortment of low-level, technology-based settings. Too often, these properties are organized into tabs, but beyond that not designed for any particular tasks or users. As a result, when users are faced with a task in a property window, they often don't know what to do.

To ensure that your property windows are useful and usable, follow these steps:

- Make sure the properties are necessary.
- Present properties in terms of user goals, not technology.
- Present properties at the right level.
- Design pages for specific tasks.
- Design pages for specific users, especially limited users (non-administrators).
- Organize the property pages efficiently.

If you do only one thing...

Present properties in terms of user goals, not technology. Pretend that you are explaining the property and why it is useful to a friend. How would you explain it? What language would you use? That's the language to use in your property pages.

Usage patterns

Property windows have several usage patterns.

- Property sheets. Properties for a single object are displayed in a modeless dialog box.
- Multiple-object property sheets. Properties for multiple objects are displayed in a modeless dialog box.
- Effective settings property sheets. The effective properties for a single object are displayed in a modeless dialog box.
- Options dialog boxes. Properties for an application are displayed in a modal dialog box.
- Property inspectors. Properties for the current selection (a single object or group of objects) are displayed in a modeless window pane or undocked window.

All property window patterns except property inspectors use a delayed commit, meaning that changes take effect only when users click OK or Apply. Property inspectors use an immediate commit (properties are changed as soon as users make changes), so there is no need for OK, Cancel, and Apply buttons.

Guidelines

Property sheets

- **Display a property sheet when users:**
  - Select the Properties command for an object.
  - Set input focus on an object and press Alt+Enter.

Multiple-object property sheets

- **Display the common properties of all the selected objects.** Where the property values differ, display the controls associated with those values using a mixed state. (See the respective control guidelines for using mixed-state values.)
- If the selected object is a collection of multiple discrete objects (such as a file folder), **display the properties of the single grouped object instead of a multiple-object property sheet for the discrete objects.**

Options dialog boxes

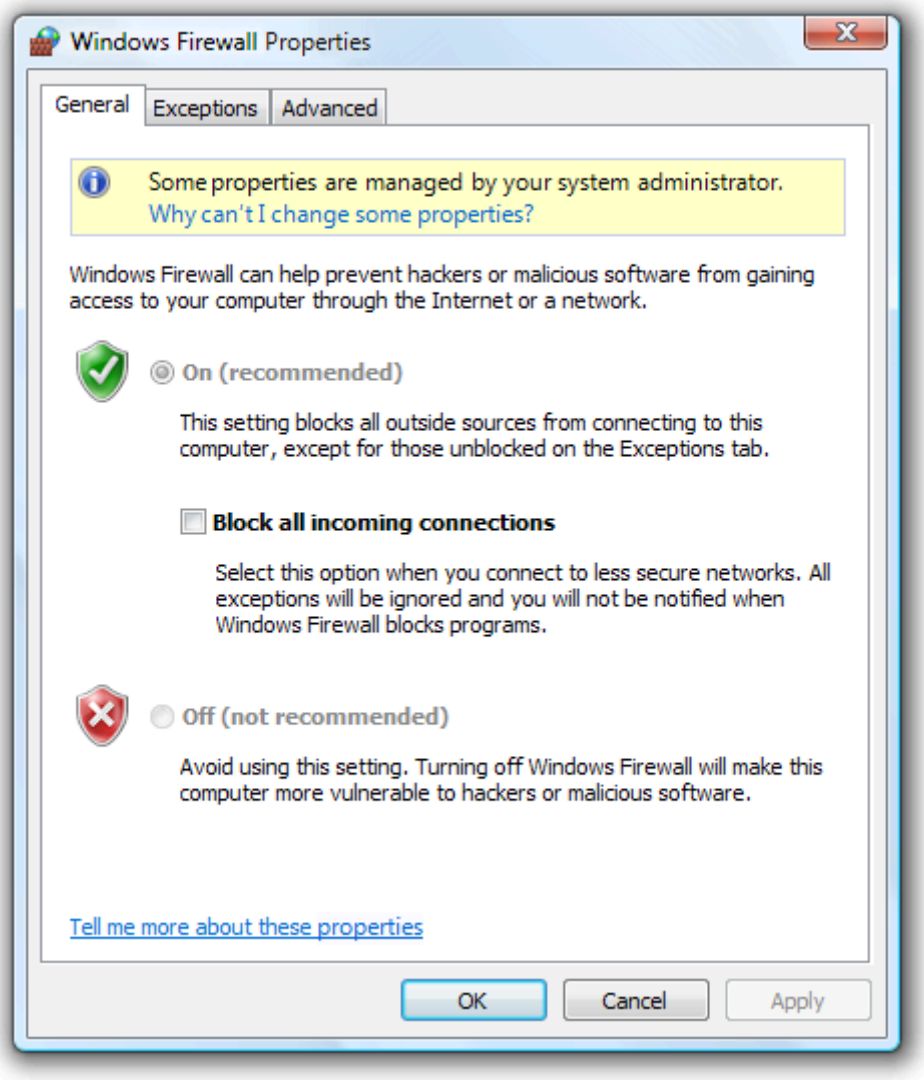
- **Don't separate options from customization.** That is, don't have both an Options command and a Customize command. Users are often confused by this separation. Instead, access customization through options.

Property pages

- **Follow these guidelines for page order:**
  - Make the General page or its equivalent the first page.
  - Make the Advanced page or its equivalent the last page.
  - For the remaining pages:
    - Organize them into groups of related pages.
    - Order the groups by the likelihood of their usage.
    - Within each group, order the pages either by their relationships or by the likelihood of their use.
    - You shouldn't have so many pages that there is a need to display them in alphabetical order.

- **Make pages coherent by relating all properties on each page to a single, specific, task-based purpose.**

- **If space allows, explain the purpose of the property window at the top of the page if it isn't obvious to your target users.** If the page is used to perform only a single task, **phrase the text as a clear instruction about how to perform that task**. Use complete sentences, ending with a period.



  In this example, the purpose of Microsoft Windows Firewall is explained at the top of the General page.

- **Make similar content consistent across pages by using consistent control names and locations.** For example, if several pages have Name boxes, try to place them in the same location on the page and use consistent labels. Similar content shouldn't bounce around from page to page.

- **Place the same property on the same page throughout your application.** For example, don't put an Expiration property on the General tab for one object type, and on the Advanced tab for another type.

- **If users are likely to start with the last page displayed, make the page tab persist, and select it by default.** Make the settings persist on a per-property window, per-user basis. Otherwise, select the first page by default.

- **Don't make the settings on a page dependent upon settings on other pages.** Put the dependent settings on a single page instead. Changing a setting on one page should never automatically change settings on other pages.
  - **Exception:** If the dependent settings are in two different property windows, use static text labels to explain this relationship in both locations.

- **Don't scroll property pages.** Both tabs and scrollbars are used to increase the effective area of a window, but one mechanism should be sufficient. Instead of using scrollbars, make the property pages larger and lay out the pages efficiently.

First pages

- For object properties, **put the object's name on the first page.**
- If you are associating (optional) icons with your objects, **display the appropriate icon in the upper-left corner** of the first page.

General pages

- **Avoid General pages.** You aren't required to have a General page. Use a General page only if:
  - The properties apply to several tasks and are meaningful to most users. Don't put specialized or advanced properties on a General page, but you can make them accessible through a command button on the General page.
  - The properties don't fit a more specific category. If they do, use that name for the page instead.

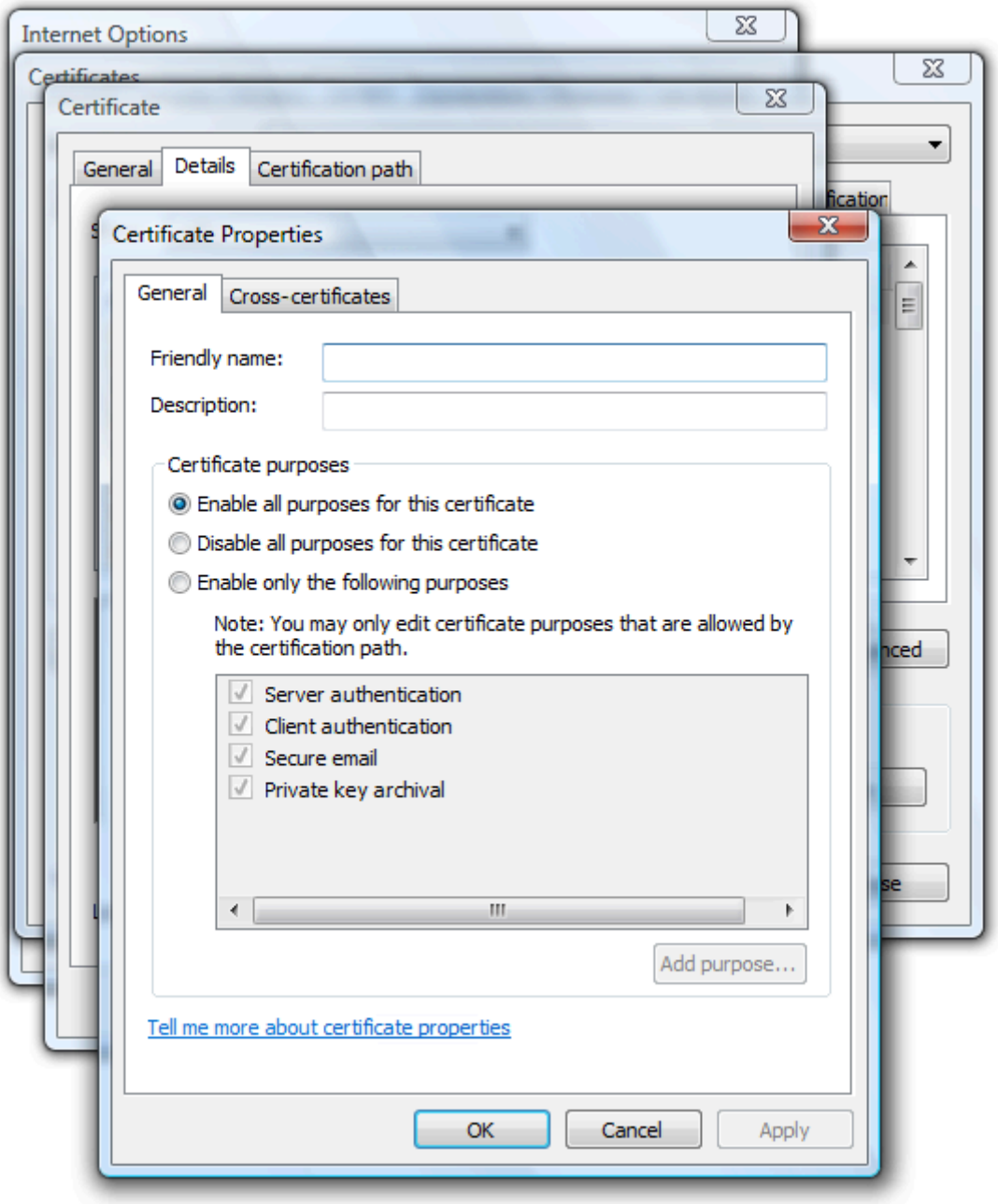
Advanced pages

- **Avoid Advanced pages.** Use an Advanced page only if:
  - The properties apply to uncommon tasks and are meaningful primarily to advanced users.
  - The properties don't fit a more specific category. If they do, use that name for the page instead.
- **Don't call properties advanced based solely on technological measures.** For example, a printer stapling option may be an advanced printer feature, but it is meaningful to all users, so it shouldn't be on an Advanced page.

Owned property windows

- **Don't display more than one owned property window from a property window.** Displaying more than one makes the meaning of the OK and Cancel buttons difficult to understand. You can display other types of auxiliary dialog boxes (such as object pickers) as needed.

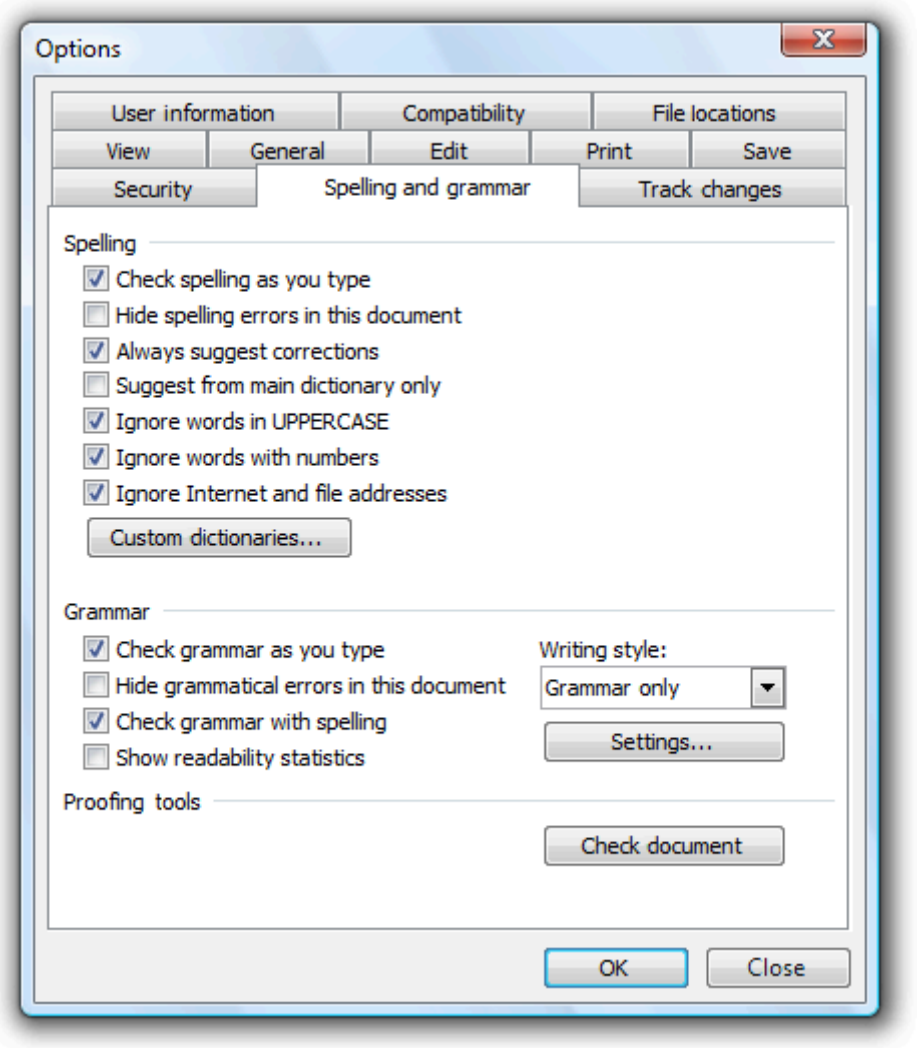
  Incorrect:



  In this example, the owner options dialog box has three levels of owned property windows. As a result, the meanings of OK and Cancel are confusing.

- For property windows that use a delayed commit model, **make sure users can cancel changes made in an owned property window by clicking Cancel on the owner window.**

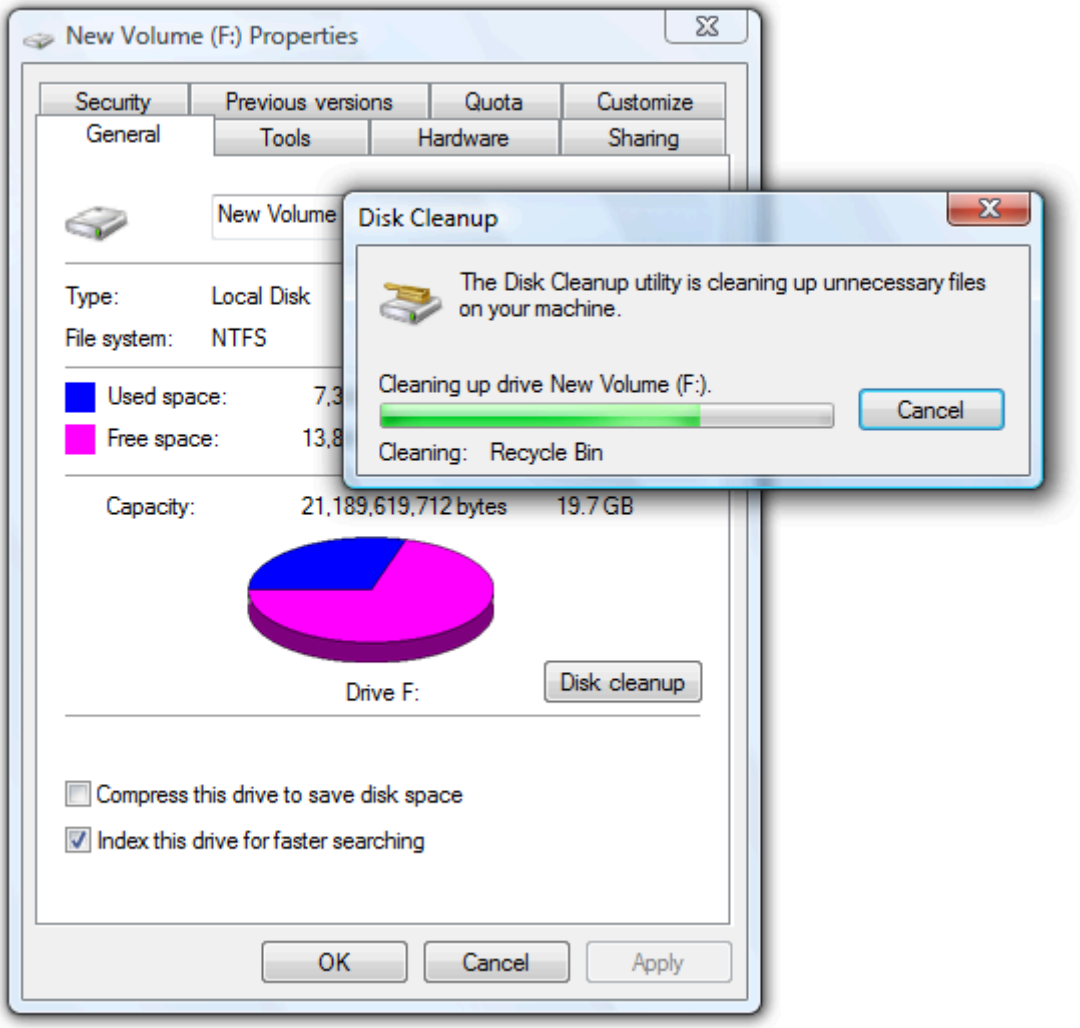
- If an owned property window requires an immediate commit, **indicate that changes were committed by renaming the Cancel button on the owner window to Close.** Revert the button back to Cancel if the user clicks Apply.



  In this example, changes to custom dictionaries and grammar settings can't be cancelled. You can give users this feedback by changing Cancel to Close.

Other owned windows

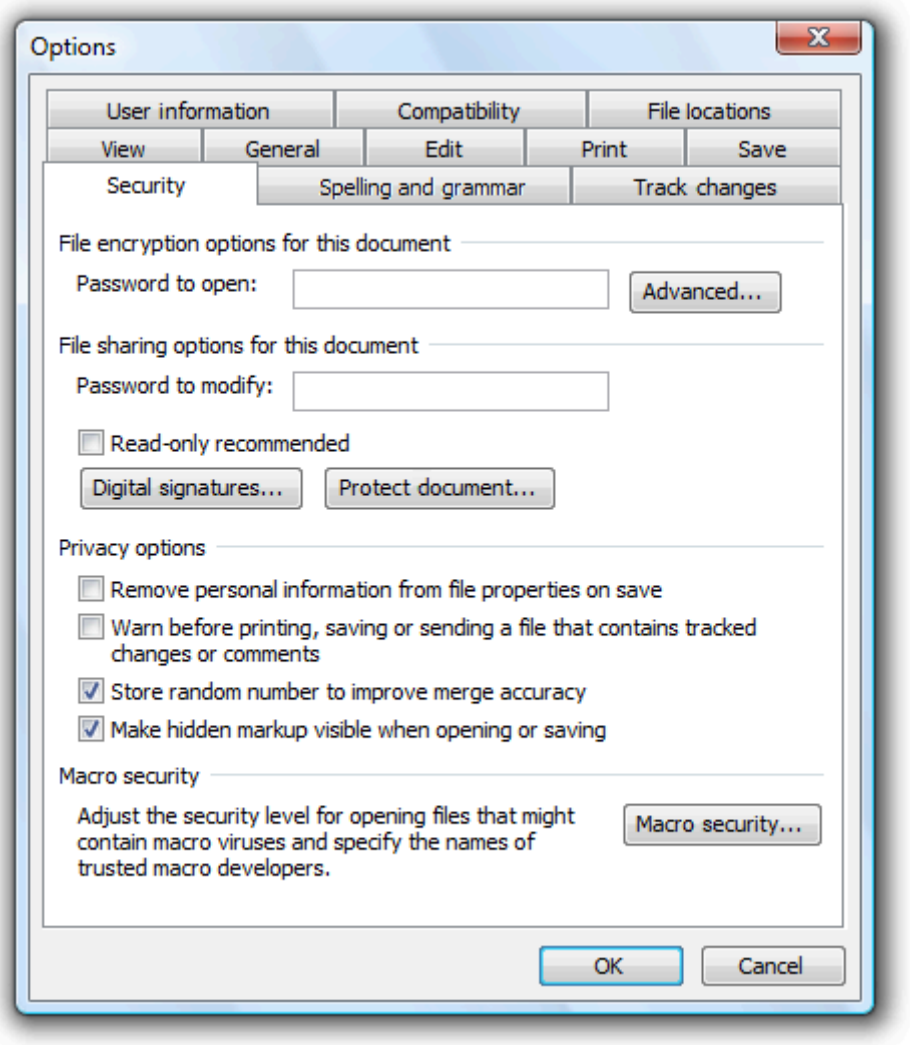
- If an owned window is used to perform an auxiliary task, **don't rename the Cancel button.** The preceding guidelines apply only to owned property windows, not dialog boxes used to perform auxiliary tasks.



In this example, Disk Cleanup is an auxiliary task, so the previous guidelines don't apply. For example, the Cancel button on the owner window shouldn't be changed to Close.

- If the owned window is used to perform an auxiliary task, **don't close the owner property window when the command button is clicked.** Doing so is disorienting and assumes that the only reason the user displayed the property window was to perform that command.

  Incorrect:

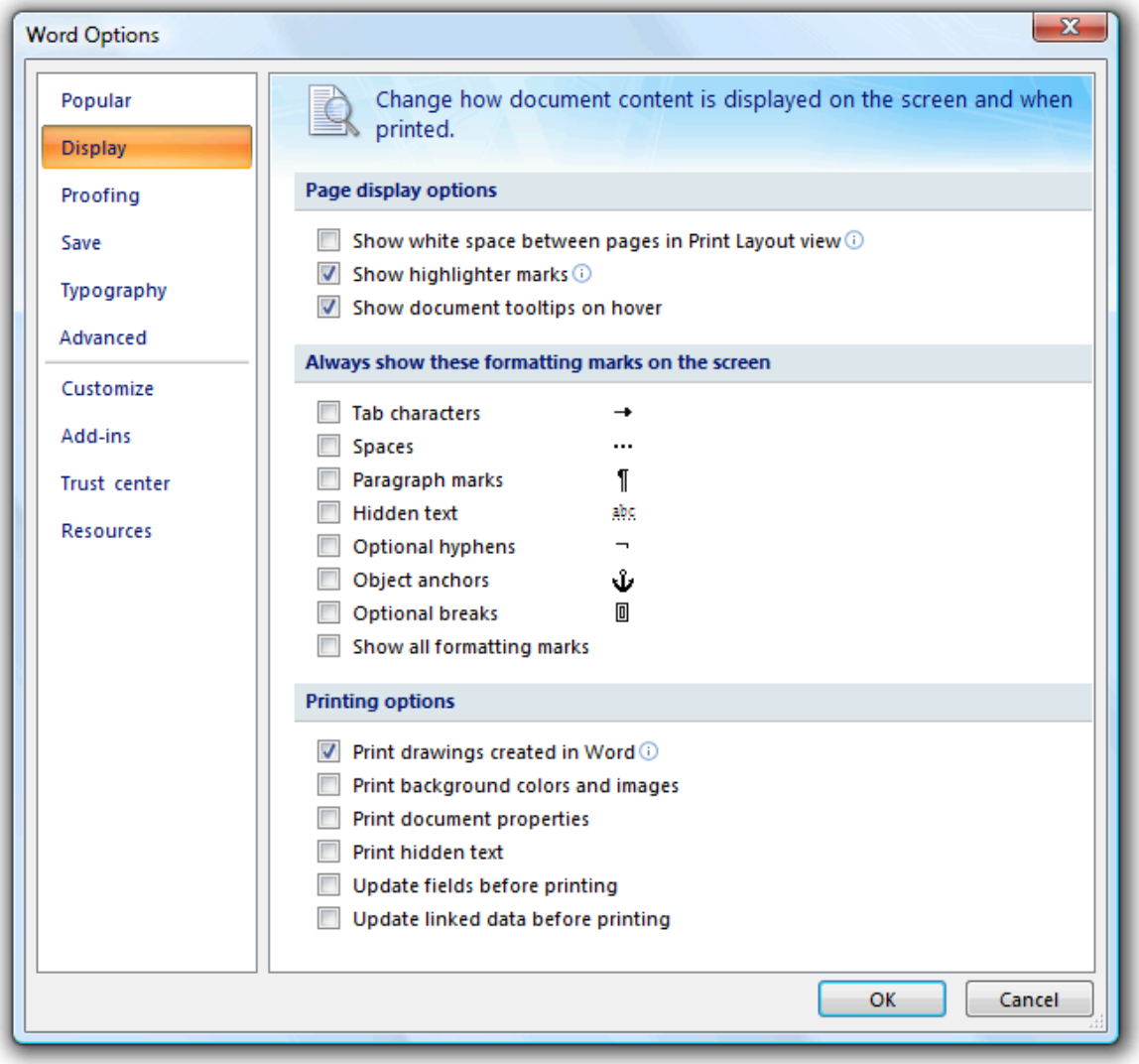


In this example, clicking **Protect document** incorrectly closes the Options dialog box.

Tabs

- **Use concise tab labels.** Use one or two words that clearly describe the content of the page. Longer labels result in an inefficient use of screen space, especially when the labels are localized.
- **Use specific, meaningful tab labels.** Avoid generic tab labels that could apply to any tab, such as General, Advanced, or Settings.
- **Use horizontal tabs if:**
  - The property window has seven or fewer tabs (including any third-party extensions).
  - **All the tabs fit on one row, even when the UI is localized.**
  - You use horizontal tabs on the other property windows in your application.
- **Use vertical tabs if:**

- The property window has eight or more tabs (including any third-party extensions).
- **Using horizontal tabs would require more than one row.**
- You use vertical tabs on the other property windows in your application.

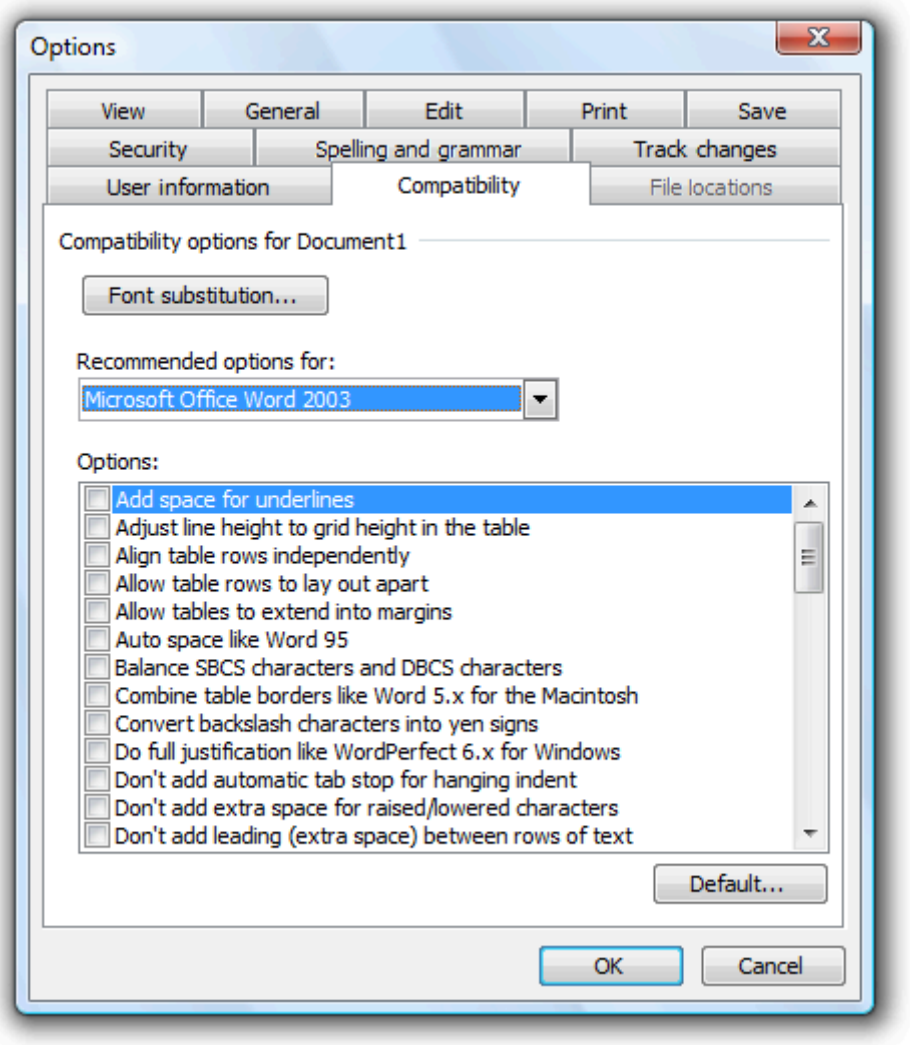


In this example, vertical tabs are used to accommodate eight or more tabs.

- **For property inspectors, to conserve space, consider using a drop-down list instead of tabs**, especially if the current tab is rarely changed by the user.

- **If a tab doesn't apply to the current context and users don't expect it to, remove the tab.** Doing so simplifies the UI, and users won't miss it.

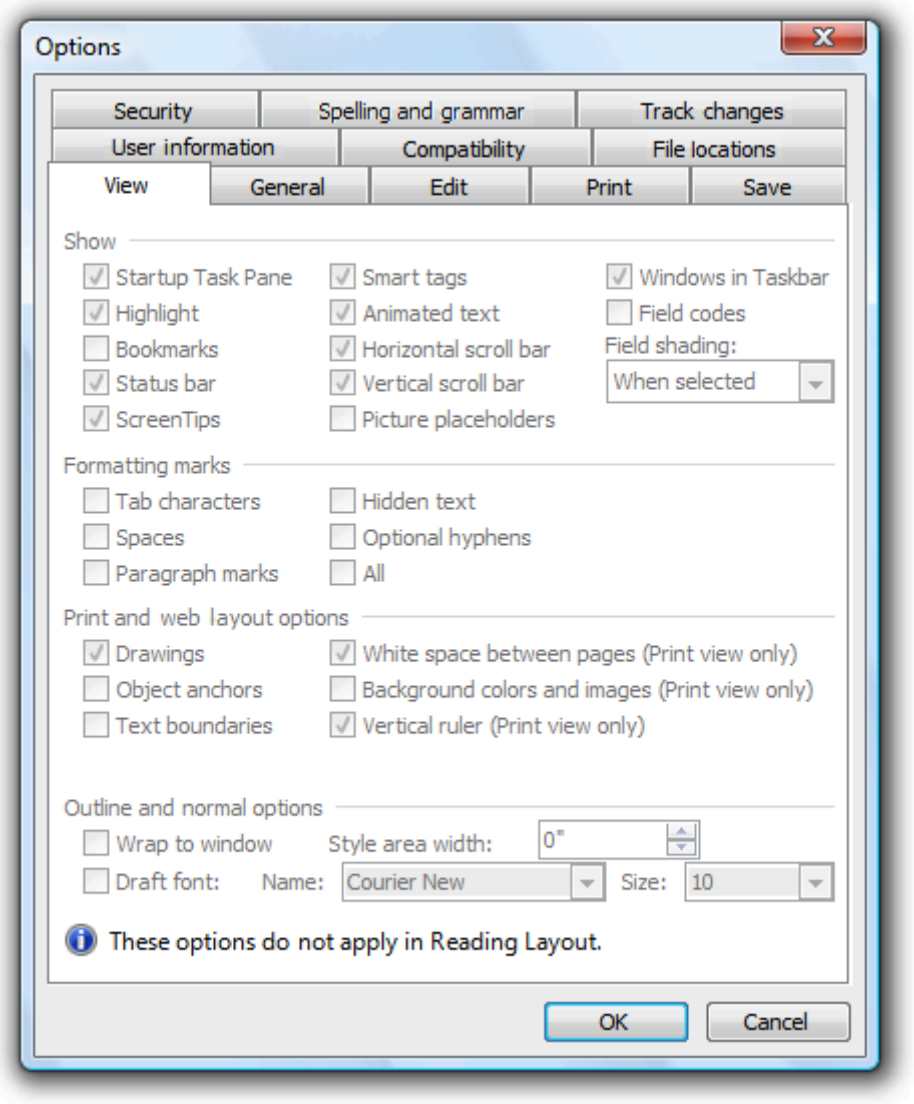
  Incorrect:



In this example, the File Locations tab is incorrectly disabled when Microsoft Word 2003 is used as an e-mail editor. The page should be removed because users wouldn't expect to view or change file locations in this context.

- **If a tab doesn't apply to the current context and users might expect it to:**
  - **Display the tab.**
  - **Disable the controls on the page.**
  - **Include text explaining why the controls are disabled.**

  Don't disable the tab because doing so isn't self-explanatory and prohibits exploration. Furthermore, users looking for a specific property would be forced to look on all other tabs.



In this example from Word 2003, none of the View options apply in Reading Layout. However, users might expect them to apply based on the tab label, so the page is displayed but the options are disabled.

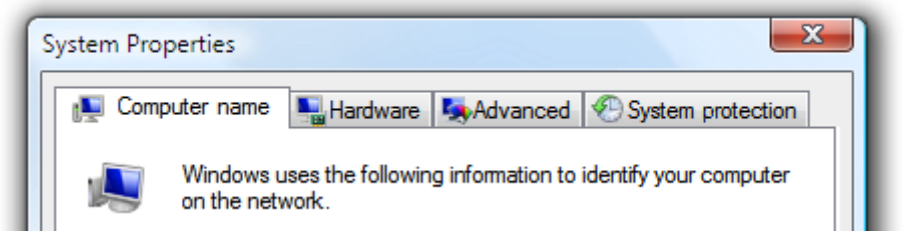
- **Don't assign effects to changing tabs.** Changing the current tab should never have side effects, apply settings, or result in an error message.

- **Don't nest tabs or combine horizontal tabs with vertical tabs.** Instead, reduce the number of tabs, use only vertical tabs, or use another control such as a drop-down list.

- **Don't use tabs if a property window has only a single tab and isn't extensible.** Use a regular dialog box with OK, Cancel, and an optional Apply button instead. Extensible property windows (which can be extended by third parties) always need to use tabs.

- **Don't put icons on tabs.** Icons usually add unnecessary visual clutter, consume screen space, and often don't improve user comprehension. Only add icons that aid in comprehension, such as standard symbols.

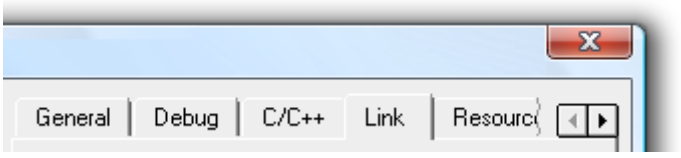
Incorrect:



In this example, the graphics add unnecessary visual clutter and do little to improve user comprehension.

- **Don't use product logos for tab graphics.** Tabs aren't for branding.
- **Don't scroll horizontal tabs.** Horizontal scrolling isn't readily discoverable. You may scroll vertical tabs, however.

  Incorrect:



  In this example, the horizontal tabs are scrolled.

Command buttons

- **Place command buttons that apply to all property pages at the bottom of the property window.** Right-align the buttons and use this order (from left to right): OK, Cancel, and Apply.
- **Place command buttons that apply only to individual property pages directly on the property page.**

Commit buttons

OK buttons

- **For owner property windows, the OK button means apply the pending changes (made since the window was opened or the last Apply), and close the window.**
- **For owned property windows, the OK button means keep the changes, close the window, and apply the changes when the owner window's changes are applied.**
- **Don't rename the OK button.** Unlike other dialog boxes, property windows aren't used to perform any one specific task. If it makes sense to rename the OK button (to Print, for example), the window isn't a property window.

- **Don't assign an access key.**

Cancel buttons

- **The Cancel button means discard all pending changes (made since the window was opened or the last Apply), and close the window.**
- **If all pending changes can't be abandoned, rename the Cancel button to Close.** Clicking Cancel must abandon all pending changes.
- **If the owned property window requires an immediate commit, rename the Cancel button on the owner window to Close to show that changes were committed.**
- **Don't assign an access key.**

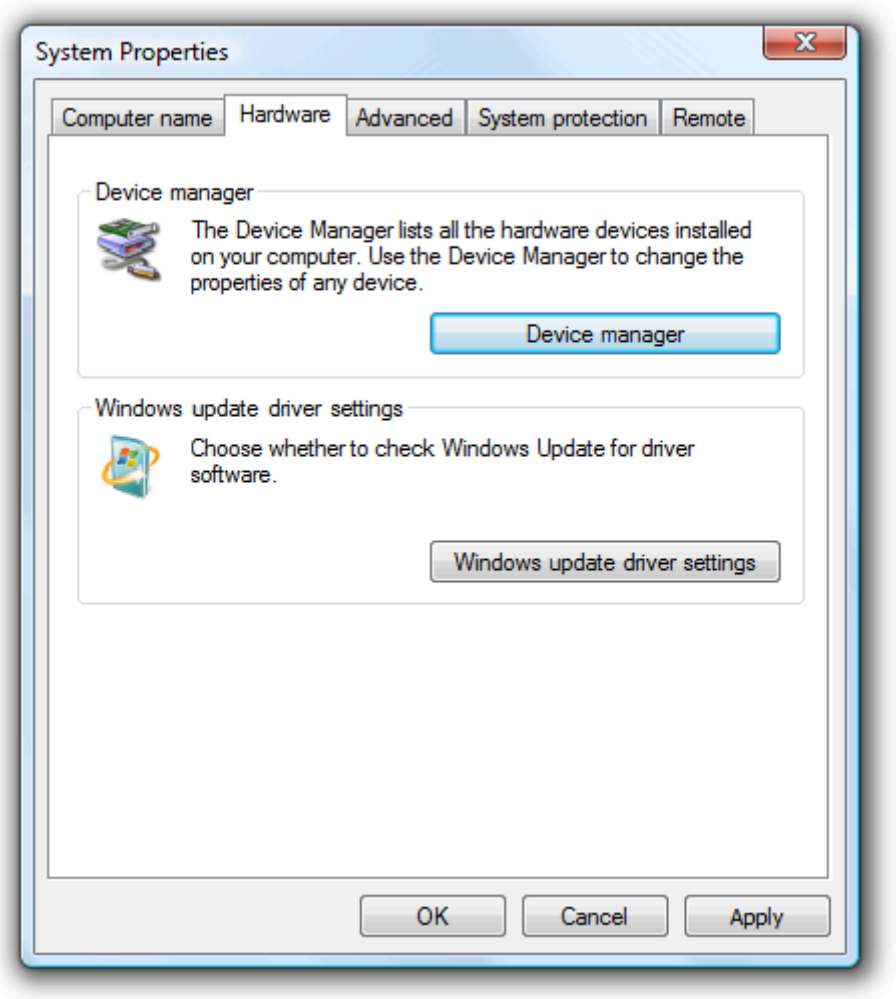
Apply buttons

- **For owner property sheets, the Apply button means apply the pending changes (made since the window was opened or the last Apply), but leave the window open.** Doing so allows users to evaluate the changes before closing the property sheet.

- **For owned property sheets, don't use.** Using an Apply button on an owned property sheet makes the meaning of the commit buttons on the owner property sheet difficult to understand.

- **Provide an Apply button only if the property sheet has settings (at least one) with effects that users can evaluate in a meaningful way.** Typically, Apply buttons are used when settings make visible changes. Users should be able to apply a change, evaluate the change, and make further changes based on that evaluation. If not, remove the Apply button instead of disabling it.

  Incorrect:



In this example, none of the system properties have a visual effect, so the Apply button has no value and should be removed.

- **Place all settings that users may want to apply on owner pages.** Don't use Apply buttons on owned property sheets, because doing so is confusing.
- **Use Apply buttons only with property sheets, not with options dialog boxes.**
- **Enable the Apply button only when there are pending changes**; otherwise, disable it.
- **Assign "A" as the access key.**

Close buttons

- **If all pending changes can't be abandoned, rename the Cancel button to Close.** Clicking Cancel must abandon all pending changes.
- **Don't confirm if users discard their changes.**
  - **Exception:** If the property window has settings that require significant effort to set and the user has made changes, you may display a confirmation if the user clicks the Close button on the title bar. The reason is that some users mistakenly

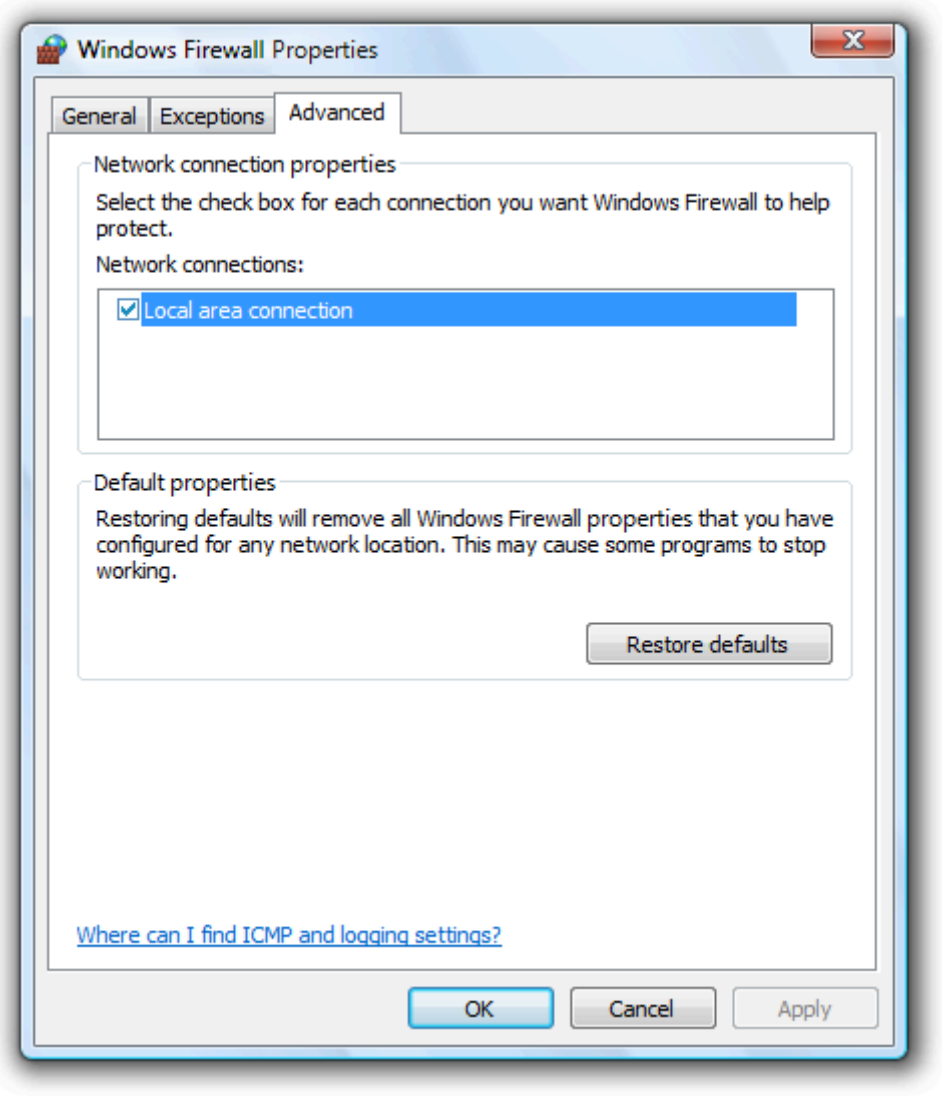
assume that the Close button on the title bar has the same effect as the OK button.
- **With the exception of the confirmation message, make sure the Close button on the title bar has the same effect as Cancel or Close.**

Page contents

- **Make sure the properties are necessary.** Don't clutter your pages with unnecessary properties just to avoid making hard design decisions.
- **Present properties in terms of user goals, not technology.** Just because a property configures a specific technology doesn't mean that you must present the property in terms of that technology.
  - If you must present settings in terms of technology (perhaps because your users recognize the technology's name), include a brief description of how the user benefits from that setting.
- **Present properties at the right level.** You don't need to present individual, low-level settings on a property page, so present the properties at a level that makes sense to your users.
- **Design property pages for specific tasks.** Determine the tasks that users will perform, and make sure there is a clear path to perform those tasks.
- **Organize property pages efficiently** by reducing the number of tabs, deciding what goes on a page based on logical grouping and coherence, and simplifying the page's presentation.

- **If an option is strongly recommended, consider adding "(recommended)" to the label.**

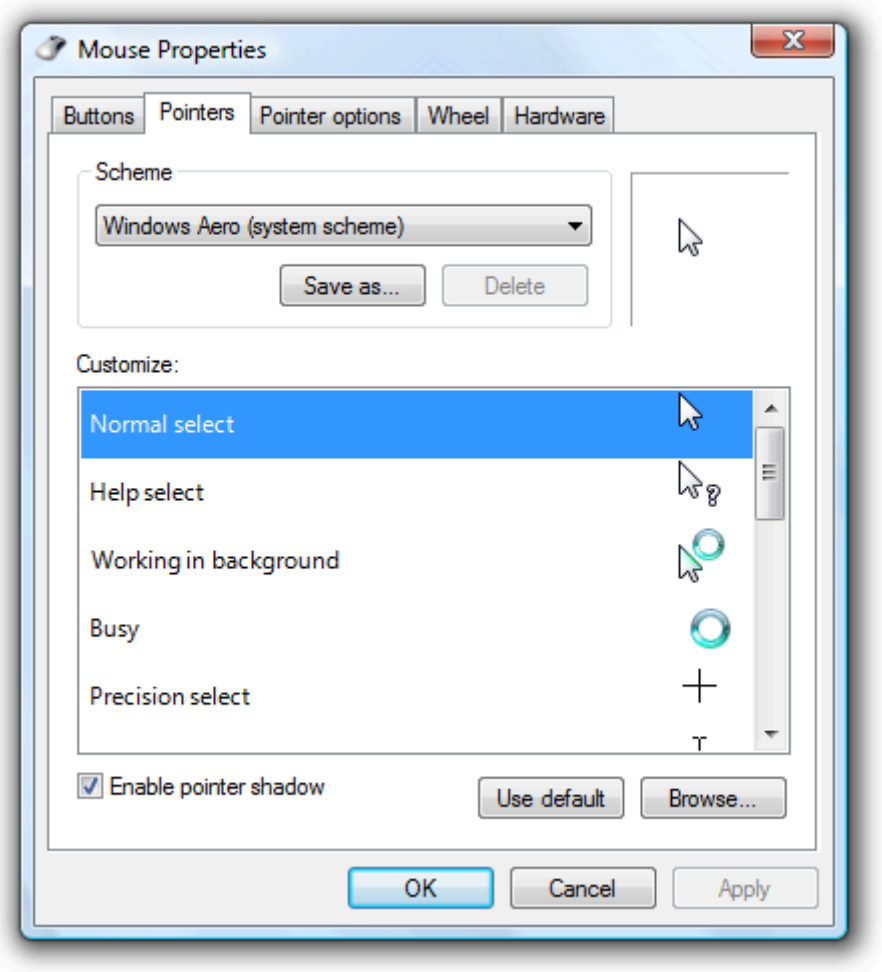
- **Provide a Restore Defaults command button for a property page or the entire property window when:**
  - Your users are likely to consider the settings complex and difficult to understand.
  - Having incorrect settings may result in breaking functionality, but the defaults might restore functionality.
  - It's easier for users to start over when the object is misconfigured.



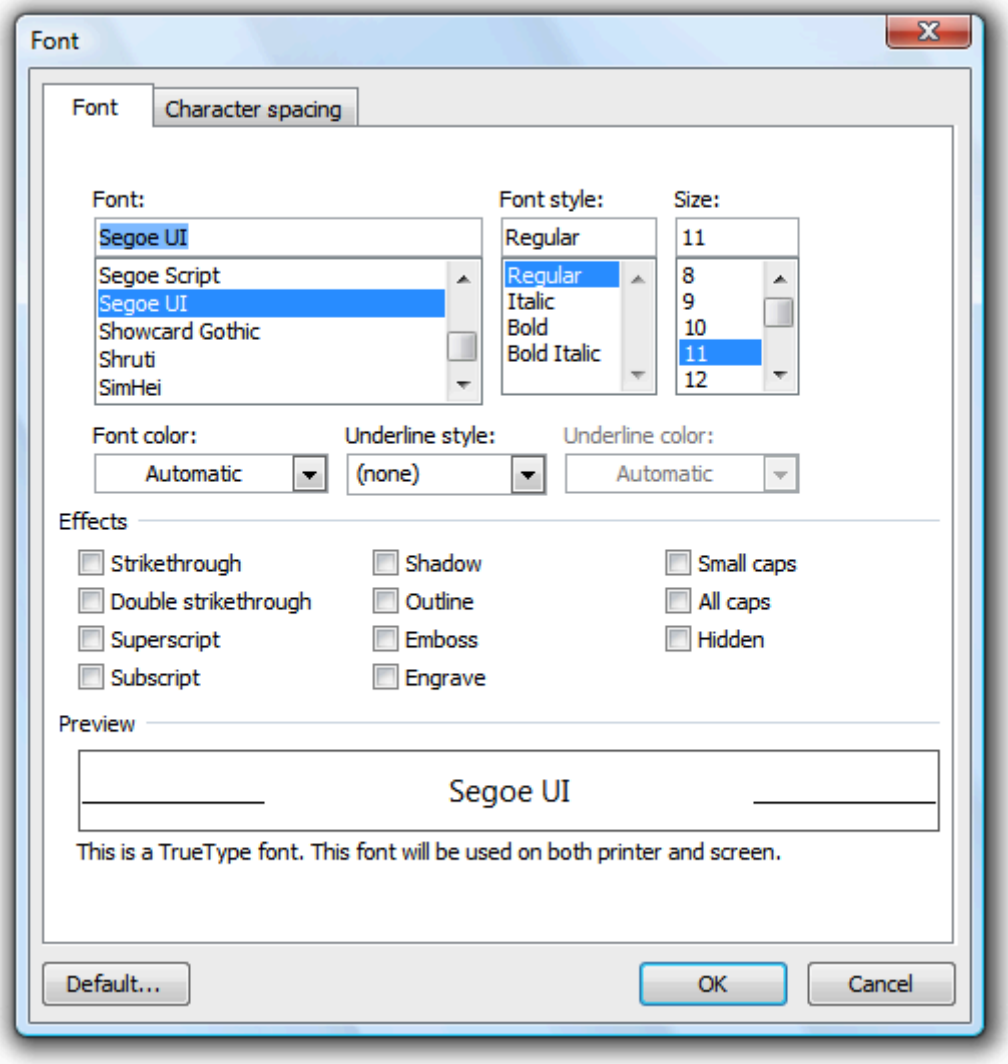
In this example, the Windows Firewall settings are complex and may result in broken functionality. If there's a problem, it is often easier for users to start over by clicking Restore Defaults.

- Confirm the Restore Defaults command if its effect isn't obvious or the settings are complex. Indicate the confirmation by using ellipses.

- **When appropriate, display a preview of the results of a setting.**



In this example, the page displays a preview of the pointer schemes. While clicking Apply also shows a preview, having a preview on the page is more efficient for users.



In this example, the Preview box shows the results of the font settings. This example shows that you can preview settings that aren't graphical.

Help

- When providing user assistance, **consider using the following options** (listed in their order of preference):
  - Give interactive controls self-explanatory labels. Users are more likely to read the labels on interactive controls than any other text.
  - Provide in-context explanations using static text labels.
  - Provide a specific link to a relevant Help topic.
- **Locate Help links at the bottom of each page.** If a page has several distinct groups of settings that have a Help topic (perhaps within group boxes), locate the Help link at the bottom of the group.
- **Don't use general or vague Help topic links or generic Help buttons.** Users often ignore generic Help.

For more information and examples, see Help.

Standard users and Protected administrators

Many settings require administrator privileges to change. If a process requires administrator privileges, Windows and later requires Standard users and Protected administrators to elevate their privileges explicitly. Doing so helps prevent malicious code from running with administrator privileges.

For more information and examples, see User Account Control.

Default values

- **The settings within a property window must reflect the current state of the application, object, or collection of objects.** Doing otherwise would be misleading and possibly lead to undesired results. For example, if the settings reflect the recommendations but not the current state, users might click Cancel instead of making changes, thinking that no changes are needed.
- **Choose the safest (to prevent loss of data or system access) and most secure initial state.** Assume that most users won't change the settings.
- **If safety and security aren't factors, choose the initial state that is most likely or convenient.**

Text

Commands

- To display program options, use "Options."
- To display an object's property window, use "Properties."
- To display a summary of the commonly used program customization settings, use "Personalize."
- Don't use "Settings" or "Preferences."
- Don't use ellipses for these commands.

Property sheet titles

- For a single object, use "[object name] Properties."
  - If the object has no name, use the object's type name. (For example, User Account Properties.)
- For multiple objects, use "[first object name], ... Properties."
  - If the objects have no names, use the objects' type name. (For example, User Accounts Properties.)

    - If the objects have different types, use "Selection Properties."
- Use title-style capitalization.
- Don't use ending punctuation.
- Don't use hyphens, such as "[object name] - Properties."

Property inspector titles

- Use "Properties."
- Use title-style capitalization.
- Don't use ending punctuation.

Options dialog box titles

- Use "Options."
- Use title-style capitalization.
- Don't use ending punctuation.

Property page tab names

- **Use concise tab labels.** Use one or two words that clearly describe the content of the page. Using longer tab names results in an inefficient use of screen space, especially when the tab names are localized.
- **Use specific, meaningful tab labels.** Avoid generic tab labels that could apply to any tab, such as General, Advanced, or Settings.
- Write the label as a one- or two-word phrase and use no ending punctuation.
- Use sentence-style capitalization.
- Don't assign a unique access key.

Property page text

- Avoid large blocks of text.
- Provide enough room for the text to expand 30 percent if it will be localized.
- Don't use text phrased as a command on property windows. Because users might want to simply view settings, you don't want to prompt them to change settings.
- Use sentence-style capitalization and ending punctuation.

Documentation

When referring to property windows:

- In programming and other technical documentation, refer to property sheets and options dialog boxes as property sheets. Everywhere else, use dialog box, especially in user documentation.
- Use the exact title text, including its capitalization.
- To describe user interaction, use open and close.
- When possible, format the title using bold text. Otherwise, put the title in quotation marks only if required to prevent confusion.

When referring to property pages:

- In programming and other technical documentation, refer to property pages as property pages. Everywhere else, use tab, especially in user documentation.
- Use the exact title text, including its capitalization.
- To describe user interaction, use click to refer to clicking a tab.
- When possible, format the name using bold text. Otherwise, put the name in quotation marks only if required to prevent confusion.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Visuals

Article • 02/09/2021

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

Visuals include the visual elements other than the controls. These guidelines help you make decisions about layout, fonts, color, icons, and so on in your app.

In this section

Expand table

Topic	Description
Layout	Layout is the sizing, spacing, and placement of content within a window or page. Effective layout is crucial in helping users find what they are looking for quickly, as well as making the appearance visually appealing. Effective layout can make the difference between designs that users immediately understand and those that leave users feeling puzzled and overwhelmed.
Fonts	Users interact with text more than with any other element in Microsoft Windows. Segoe UI (pronounced "SEE-go") is the Windows system font. The standard font size has been increased to 9 point.
Color	Color is an important visual element of most user interfaces. Beyond pure aesthetics, color has associated meanings and elicits emotional responses. To prevent confusion in meaning, color must be used consistently. To obtain the desired emotional responses, color must be used appropriately.
Icons	Icons are pictorial representations of objects, important not only for aesthetic reasons as part of the visual identity of a program, but also for utilitarian reasons as shorthand for conveying meaning that users perceive almost instantaneously. Windows Vista introduces a new style of iconography that brings a higher level of detail and sophistication to Windows.
Standard Icons	Standard icons are the error, warning, information, and question mark icons that are part of Windows.

Topic	Description
Animations and Transitions	Strategic use of animations and transitions can make your program easier to understand, feel smoother, more natural, and of higher quality, and be more engaging. But the gratuitous use of animations and transitions can make your program distracting and even annoying.
Graphic Elements	*Graphic elements* show relationships, hierarchy, and emphasis visually. They include backgrounds, banners, glass, aggregators, separators, shadows, and handles.
Sound	*Sound* is the audio element of the user experience. When used appropriately, sound can be an effective form of communication that establishes a non-verbal and even emotional relationship with your users. Sounds can be used alone or as a supplement to visual UI. For example, adding a sound effect to a notification increases the likelihood that it will be noticed, especially if the user isn't looking at the screen when an event occurs.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Layout

Article • 06/03/2021

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

Layout is the sizing, spacing, and placement of content within a window or page. Effective layout is crucial in helping users find what they are looking for quickly, as well as making the appearance visually appealing. Effective layout can make the difference between designs that users immediately understand and those that leave users feeling puzzled and overwhelmed.

Note: Guidelines related to window management are presented in a separate article. Recommended specific control sizing and spacing are presented in their respective guideline articles.

Design concepts

Visual hierarchy

A window or page has a clear visual hierarchy when its appearance indicates the relationship and priority of its elements. Without a visual hierarchy, users would have to figure out these relationships and priorities themselves.

Visual hierarchy is achieved by skillfully combining the following attributes:

- **Focus.** The layout indicates where users need to look first.
- **Flow.** The eye flows smoothly and naturally by a clear path through the surface, finding user interface (UI) elements in the order appropriate for their use.
- **Grouping.** Logically related UI elements have a clear visual relationship. Related items are grouped together; unrelated items are separate.
- **Emphasis.** UI elements are emphasized based on their relative importance.
- **Alignment.** The UI elements have coordinated placement, so they are easy to scan and appear orderly.

Additionally, effective layout has these attributes:

- **Device independence.** The layout appears as intended regardless of the font typeface or size, dots per inch (dpi), display, or graphic adaptor.
- **Easy to scan.** Users can find the content they are looking for at a glance.
- **Efficiency.** UI elements that are large need to be large, and elements that are small work well small.
- **Resizability.** If helpful, a window is resizable and its content layout is effective no matter how large or small the surface is.
- **Balance.** The content appears evenly distributed across the surface.
- **Visual simplicity.** The perception that a layout is not more complicated than it needs to be. Users don't feel overwhelmed by the layout's appearance.
- **Consistency.** Similar windows or pages use a similar layout, so users always feel oriented.

While sizing, spacing, and placement are simple concepts, the challenge with layout is to achieve the right mixture of these attributes.

In Windows, layout is communicated using device independent metrics such as dialog units (DLUs) and relative pixels.

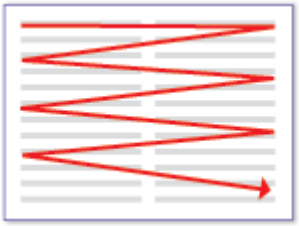
A design model for reading

Users choose what they read by the content's appearance and organization. To create an effective layout, you'll need to understand what users tend to read and why.

You can make layout decisions using this design model for reading:

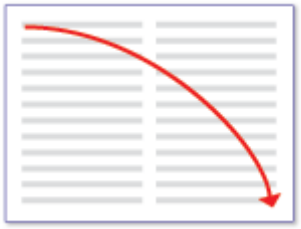
- People read in a left-to-right, top-to-bottom order (in Western cultures).

- There are two modes of reading: immersive reading and scanning. The goal of immersive reading is comprehension.

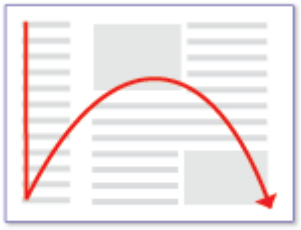
  

  This diagram models immersive reading.

  By contrast, the goal of scanning is to locate things. The overall scan path looks like this:

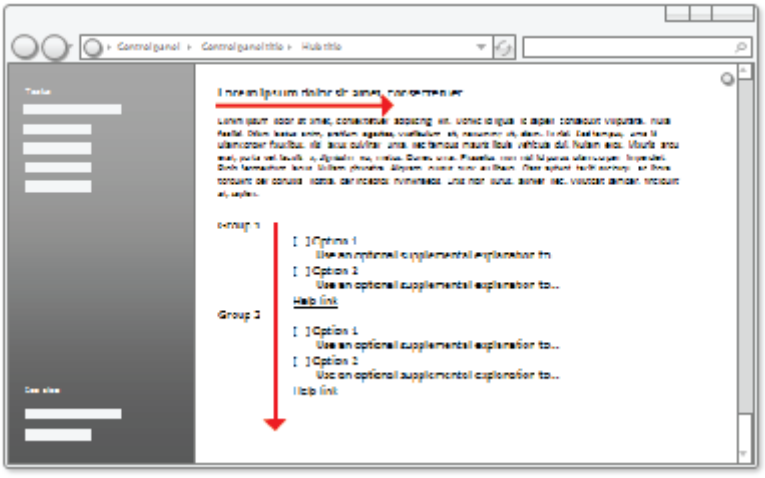


This diagram models scanning.



If there is text running along the left edge of a page, users scan the left edge first.

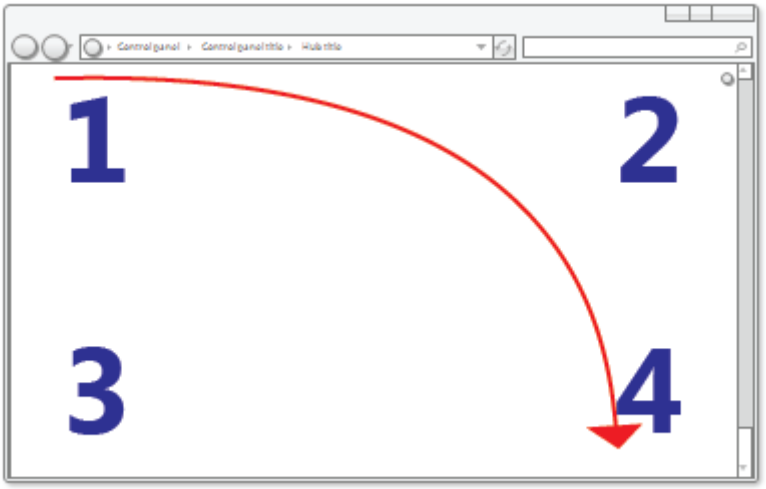
- When using software, users aren't immersed in the UI itself but in their work. Consequently, users usually don't read UI text they scan it. They then read bits of text comprehensively only when they believe they need to.
- Users tend to skip over navigation panes on the left or right side of a page. Users recognize that they are there, but look at navigation panes only when they want to navigate.
- Users tend to skip over large blocks of unformatted text without reading them at all.



Users tend to skip over large blocks of text and navigation panes when they scan.

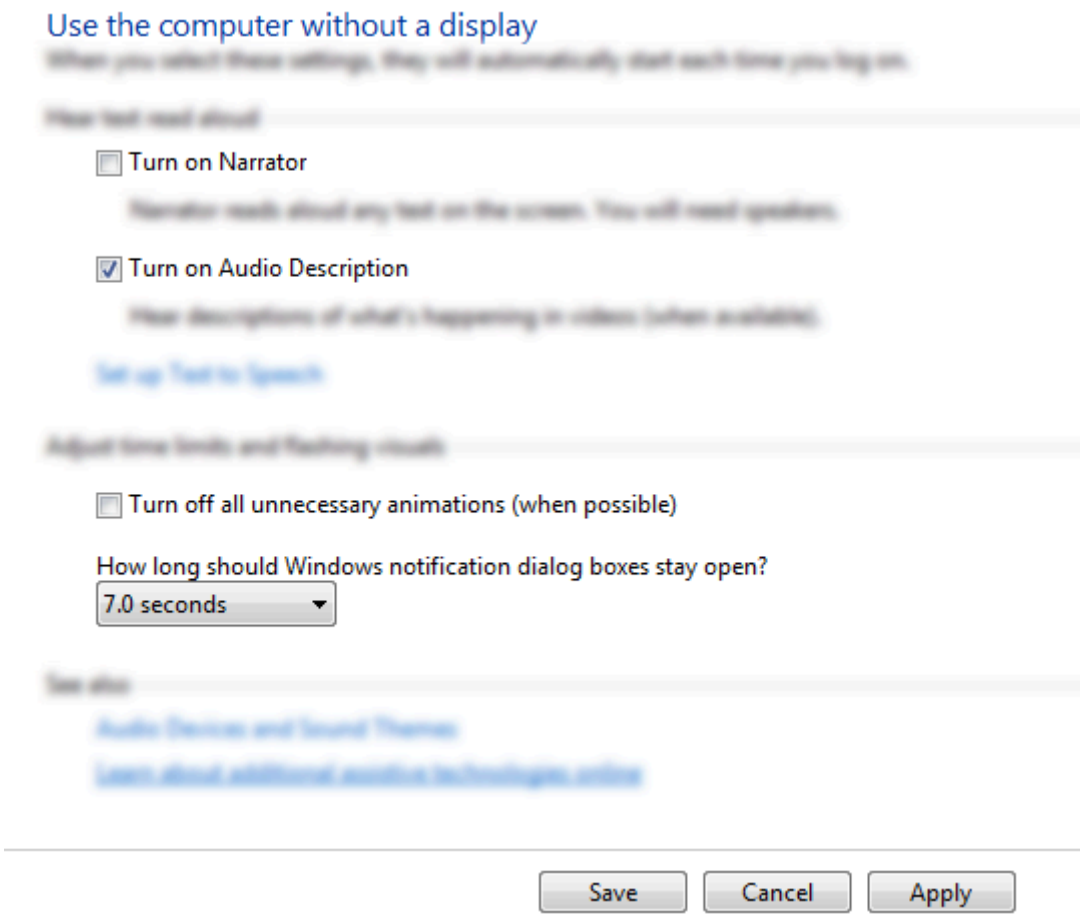
- All things being equal, users first look in the upper left corner of a window, scan across the page, and end their scan in the lower right corner. They tend to ignore

the lower left corner.



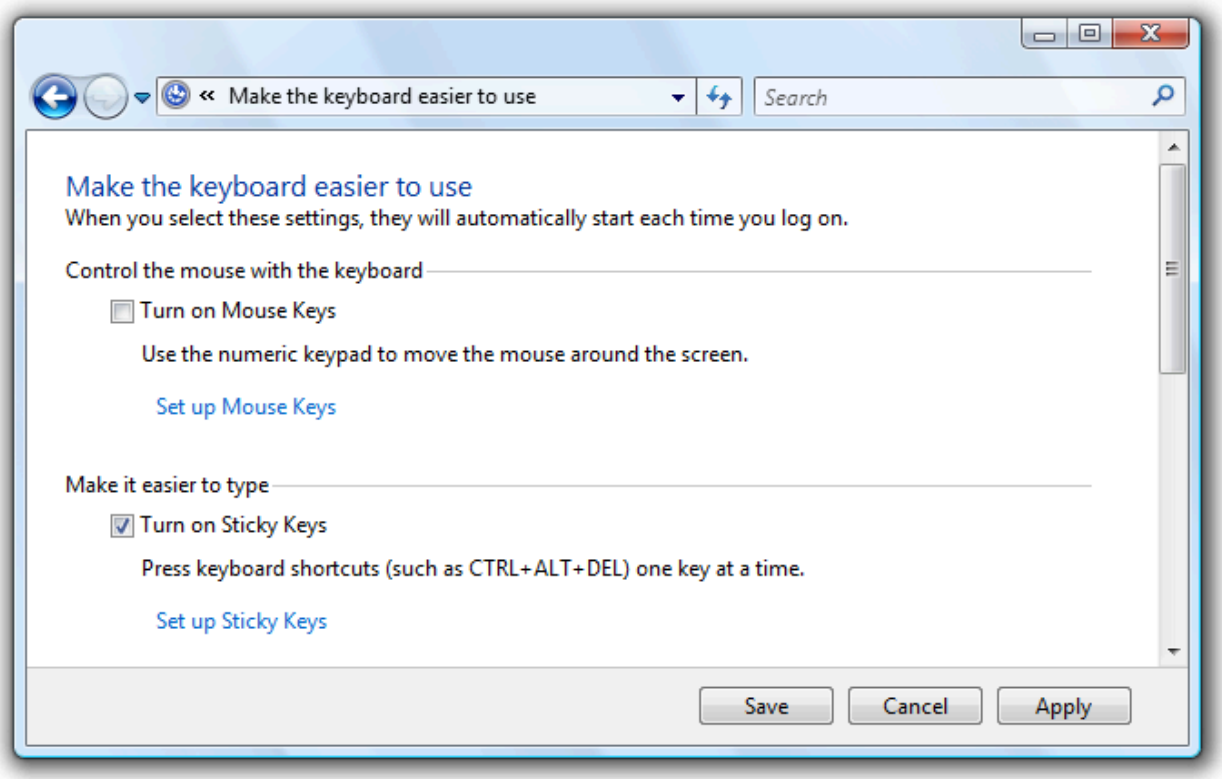
All things being equal, users will read these numbers in the following order: 1, 2, 4, and 3.

- But in interactive UI, not all things are equal so different UI elements receive different levels of attention. Users tend to look at interactive controls especially controls in the upper left and center of the window and prominent text first.



Users focus on the main interactive controls and the prominent main instruction, and look at other things only when they need to.

- Users tend to read interactive control labels, especially those that appear relevant to completing the task at hand. By contrast, users tend to read static text only when they think they need to.
- Items that appear different attract attention. Bold text and large text stands out from normal text. UI items with color or on a colored background stand out. Items with icons stand out from items without icons.
- Users don't scroll unless they have a reason to. If the content above the fold doesn't provide a reason to scroll, they won't.
- Once users have decided what to do, they immediately stop scanning and do it.
- Because users stop scanning when they think they are done, they tend to ignore anything beyond what appears to be the point of completion.



Users stop scanning when they think they're done.

Of course, there will be exceptions to this general model. Eye tracking devices indicate that real users' behavior is quite erratic. The goal of this model is to help you make good decisions and tradeoffs, not to model user behavior accurately. But as you've read this list, hopefully you've recognized many of your own reading patterns, too.

Designing for scanning

Users don't read, they scan so you should design UI surfaces for scanning. Don't assume that users will read the text as written in a left-to-right, top-to-bottom order but rather that they look at the UI elements that attract their attention.

To design for scanning:

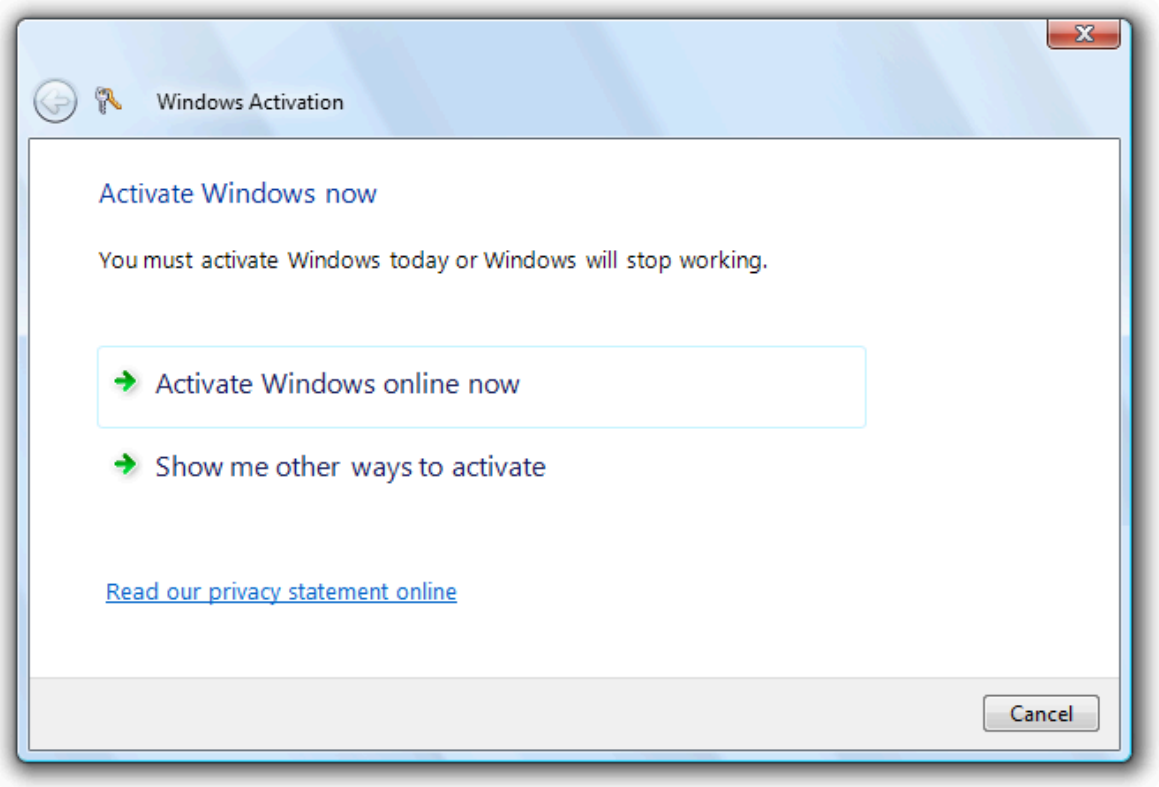
- Assume that users start by quickly scanning the whole window, then reading the UI elements in roughly the following order:
  - Interactive controls in the center
  - The commit buttons
  - Interactive controls found elsewhere
  - Main instruction
  - Supplemental explanations
  - Text presented with a warning icon
  - Window title
  - Other static text in main body
  - Footnotes
- Place UI elements that initiate a task in the upper-left corner or upper-center.
- Place UI elements that complete a task in the lower-right corner.
- Whenever possible, put crucial text on interactive controls instead of static text.
- Avoid putting crucial information in the lower-left corner or at the bottom of a long scrollable control or page.
- Don't present big blocks of text. Eliminate unnecessary text. Use the inverted pyramid presentation style.
- If you do something to attract users' attention, make sure that attention is warranted.

Whenever possible, work with this model rather than fighting it; but there will be times when you need to either emphasize or de-emphasize particular UI elements.

To emphasize primary UI elements:

- Put primary UI elements in the scan path.
- Put any UI to initiate a task in the upper-left corner or upper-center.
- Put commit buttons in the lower-right corner.
- Put the remaining primary UI in the center.
- Use controls that attract attention, such as command buttons, command links, and icons.
- Use prominent text, including large text and bold text.

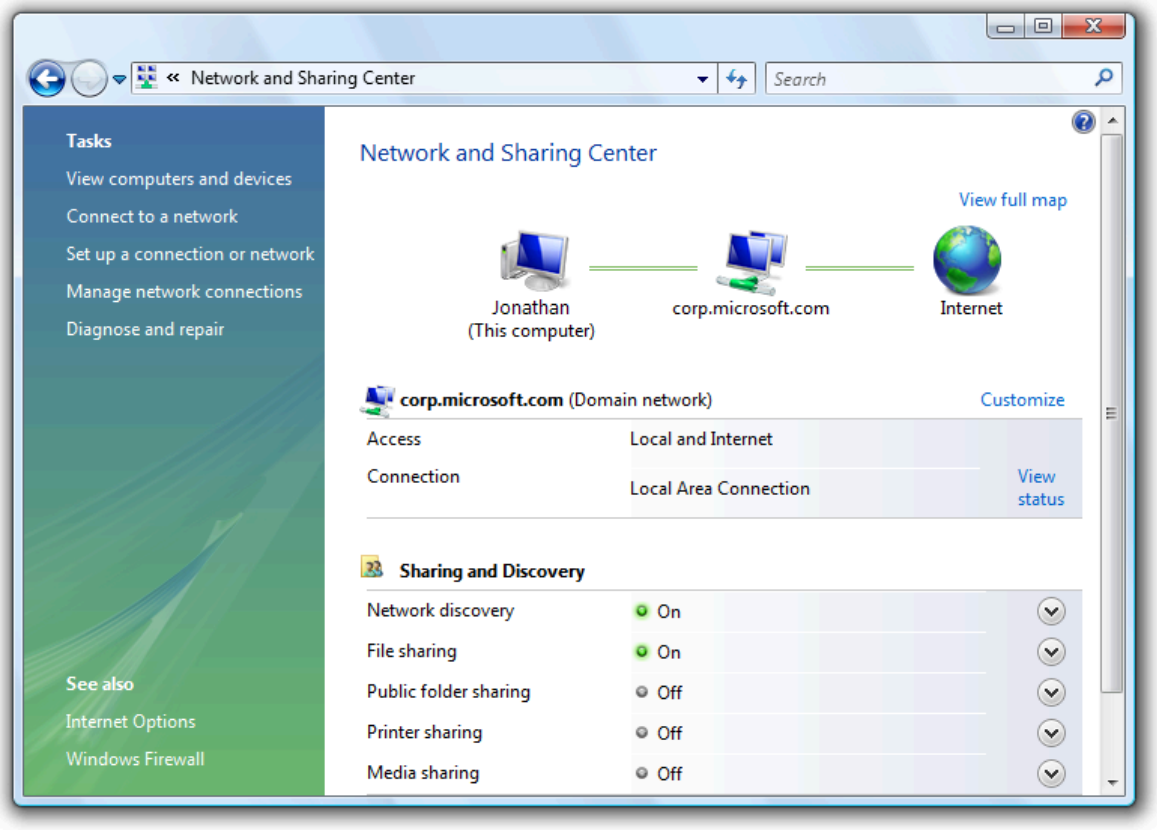
- Put text users must read in interactive controls, or with icons, or on banners.
- Use dark text on a light background.
- Surround the elements with generous space.
- Don't require any interaction, such as pointing or hovering, to see the element you are emphasizing.



This example shows many ways to emphasize primary UI elements.

To de-emphasize secondary UI elements:

- Put secondary UI elements outside the scan path.
- Put anything users usually don't need to see in the lower-left corner or bottom of the window.
- Use controls that don't attract attention, such as task links instead of command buttons.
- Use normal or gray text.
- Use light text on a dark background. White text on a dark gray or blue background works well.
- Surround the elements with minimal space.
- Consider using progressive disclosure to hide secondary UI elements.

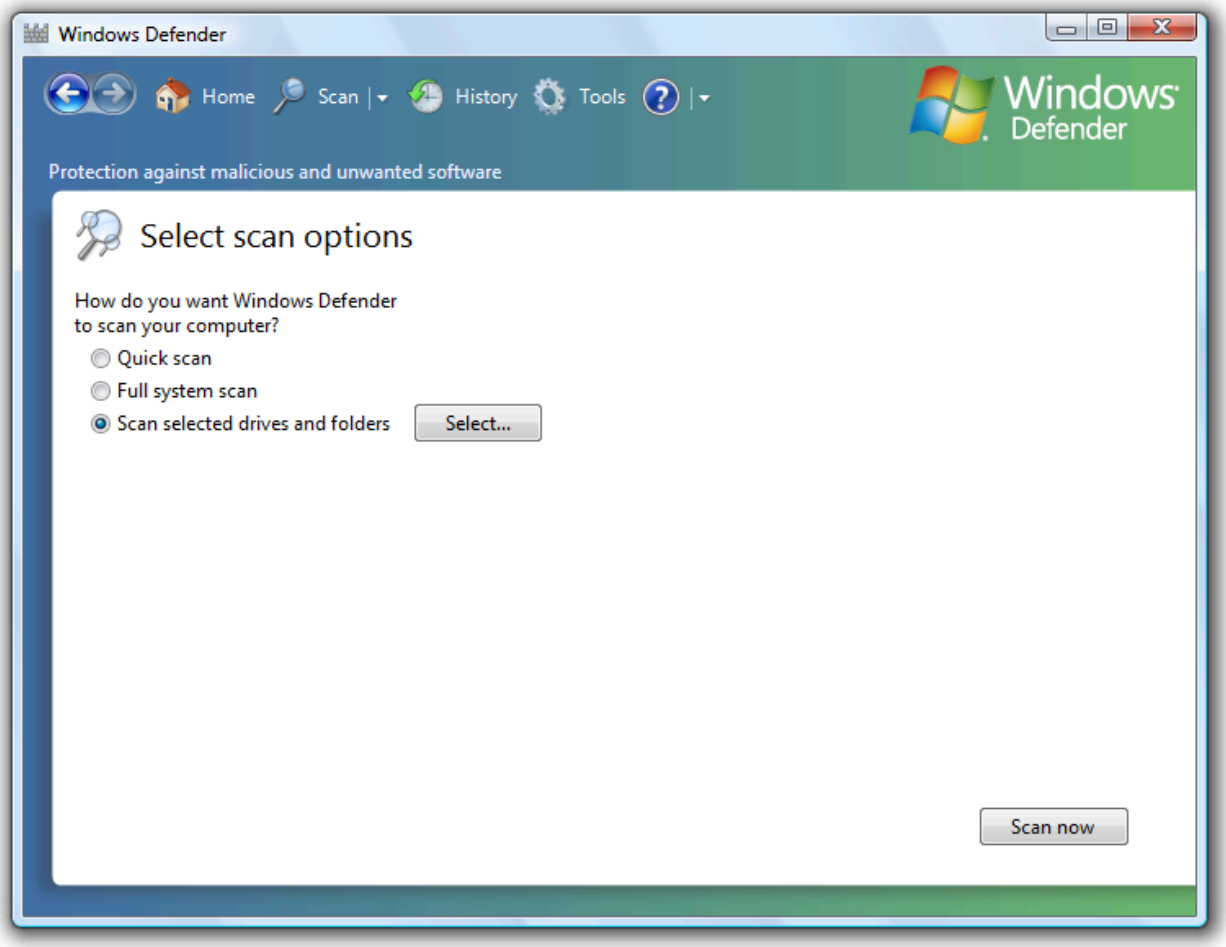


This example shows many ways to de-emphasize secondary UI elements.

Using screen space effectively

Using screen space effectively requires you to balance several factors: use too much space and a window feels heavy and wasteful, and even difficult to use based on Fitts' Law.

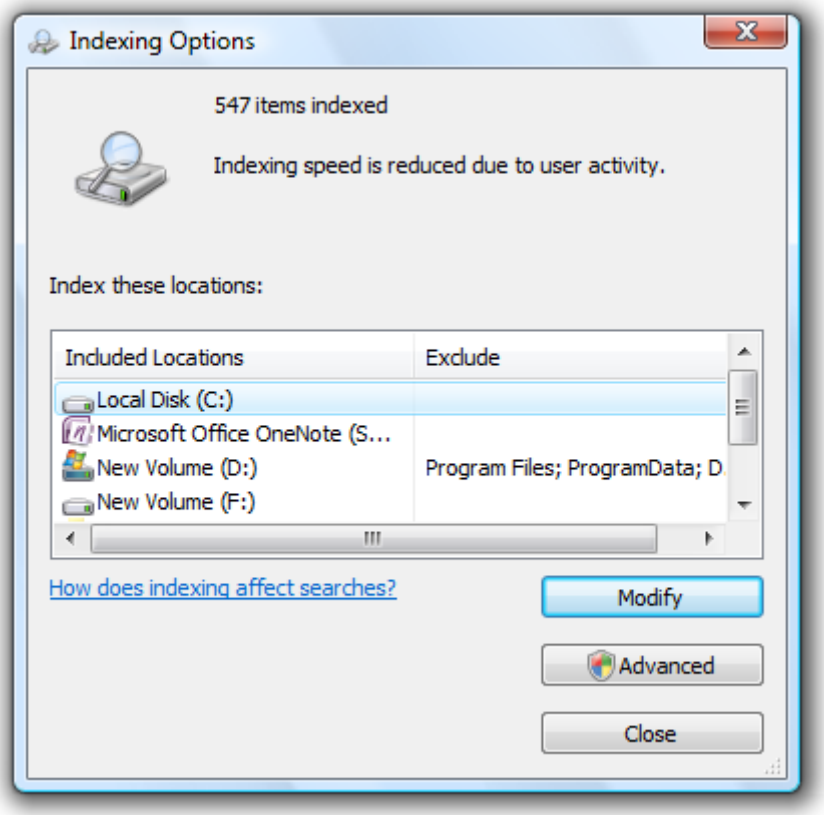
Incorrect:



In this example, the window is too large for its content.

On the other hand, use too little space and a window feels cramped, uncomfortable, and intimidating, and difficult to use if it requires scrolling and other manipulation to use.

Incorrect:



In this example, the window is too small for its content.

While critical UI must fit in the minimum supported effective resolution, don't assume that using screen space effectively means that windows should be as small as possible they shouldn't be. **Effective layout has respect for open space and doesn't attempt to cram everything into the smallest space possible.** Modern displays have significant screen space and it makes sense to use this space effectively when you can. Consequently, err on the side of using too much screen space rather than too little. Doing so makes your windows feel lighter and more approachable.

You know a layout is using screen space effectively when:

- Windows, window panes, and controls don't have to be resized to be usable. If the first thing users do is resize a window, pane, or control, its size is wrong.
- Data isn't truncated. Most data in list views and tree views doesn't have ellipses, and data in other controls isn't clipped unless the data length is unusually large. Data that must be read to perform a task shouldn't be truncated.
- The windows and controls are sized appropriately to eliminate unnecessary scrolling. There are few horizontal scrollbars and no unnecessary vertical scrollbars.
- Controls mostly use their standard sizes. Strive to reduce the number of control sizes, by, for example, using only one or two command button widths on a surface.
- The UI surface is balanced. There aren't large unused screen areas.

Choose window sizes that are just large enough to fulfill their purpose well. (And if the window is resizable, this goal applies to its default size.) **A combination of truncated**

data or scrollbars and plenty of available screen space is a clear sign of ineffective layout.

Control sizing

Usually the first step in using screen space effectively is to determine the right size for the various UI elements. Refer to the Control sizing table as well as the recommended sizing in the specific control guideline articles.

Fitts' Law states that the smaller a target is, the longer it takes to acquire it with the mouse. Furthermore, for computers using Windows Tablet and Touch Technology, the "mouse" might actually be a pen or the user's finger, so you should consider alternative input devices when determining sizes for small controls. **A control size of 16x16 relative pixels is a good minimum size for any input device.** By contrast, the standard 15x9 relative pixel spin control buttons are too small to be used effectively by pens.

Spacing

Providing generous (but not excessive) space makes the layout feel more comfortable and easier to parse. Effective space isn't unused space it plays an important role in improving the ability for users to scan, and also adds to visual appeal of your design. For guidelines, refer to the Spacing table.

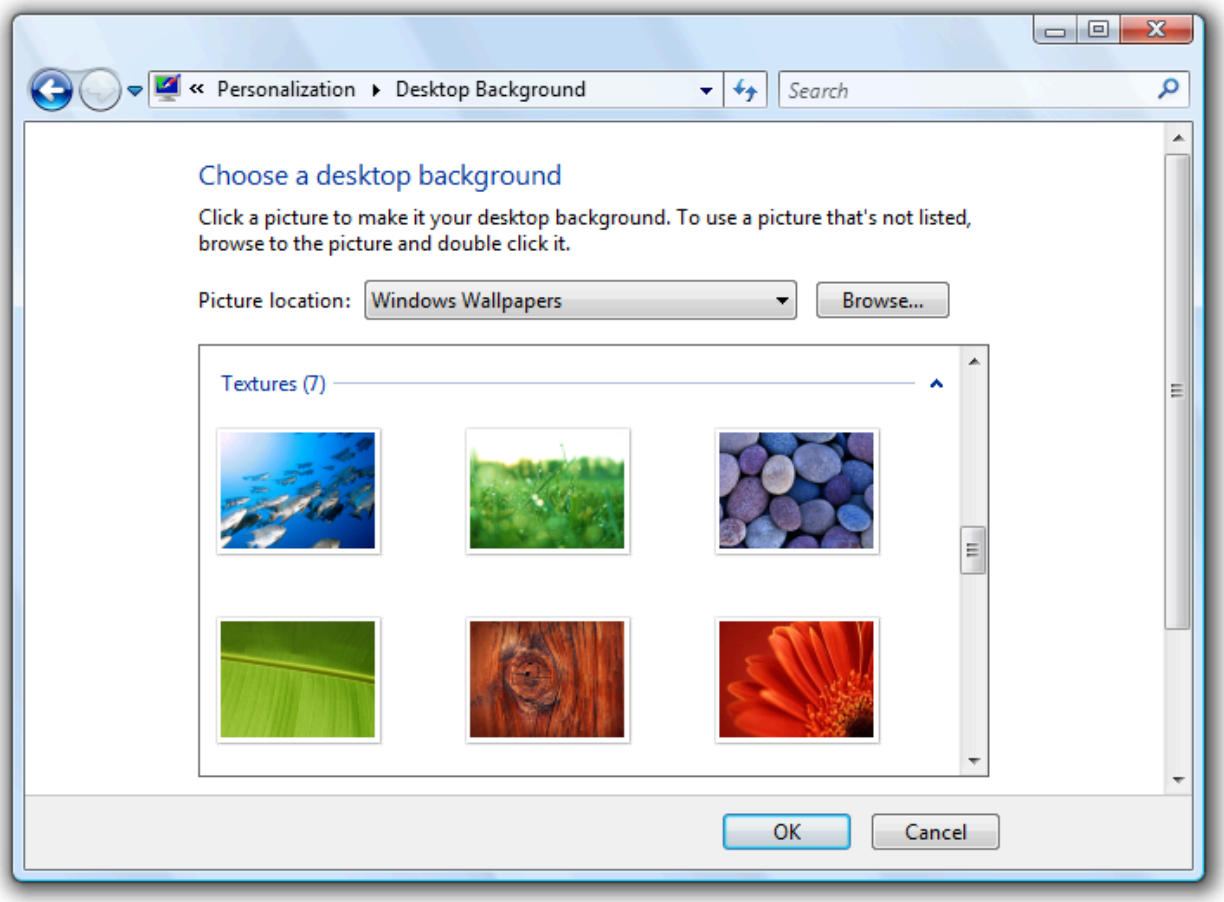
For computers using Windows Tablet and Touch Technology, again the "mouse" might actually be a pen or the user's finger. Targeting is more difficult when using a pen or finger as the pointing device, resulting in users tapping outside the intended target. When interactive controls are placed very close together but are not actually touching, users may click on inactive space between the controls. Since clicking inactive space has no result or visual feedback, users are often uncertain what went wrong. If small controls are too closely spaced, the user needs to tap with precision to avoid tapping the wrong object. **To address these issues, the target regions of interactive controls should either be touching or have at least 3 DLUs (5 relative pixels) of space between them.**

You know a layout has good spacing when:

- Overall, the UI surface feels comfortable and doesn't feel cramped.
- The space appears uniform and balanced.
- Related elements are close together and unrelated elements are relatively far apart.
- There is no dead space between controls that are meant to be together, such as toolbar buttons.

Resizable windows

Resizable windows are also a factor in using screen space effectively. Some windows consist of fixed content and don't benefit from being resizable, but windows with resizable content should be resizable. Of course, the reason users resize a window is to take advanced of the additional screen space, so the content should expand accordingly by giving more space to the UI elements that need it. Windows with dynamic content, documents, images, lists, and trees benefit the most from resizable windows.

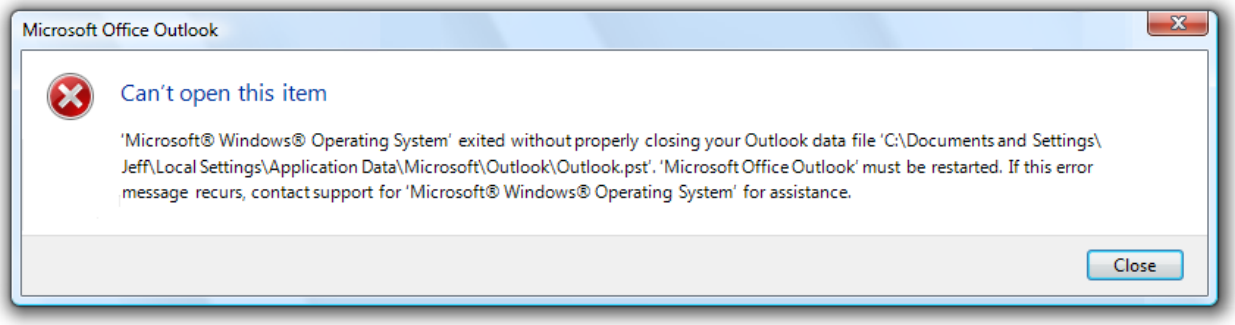


In this example, resizing the window resizes the list view control.

That said, windows can be stretched too wide. For example, many control panel pages become unwieldy when the content is wider than 600 relative pixels. In this case, it's better not to resize the content area beyond this maximum width or change the content's origin as the window is resized larger. Instead, keep a maximum width and a fixed upper-left origin.

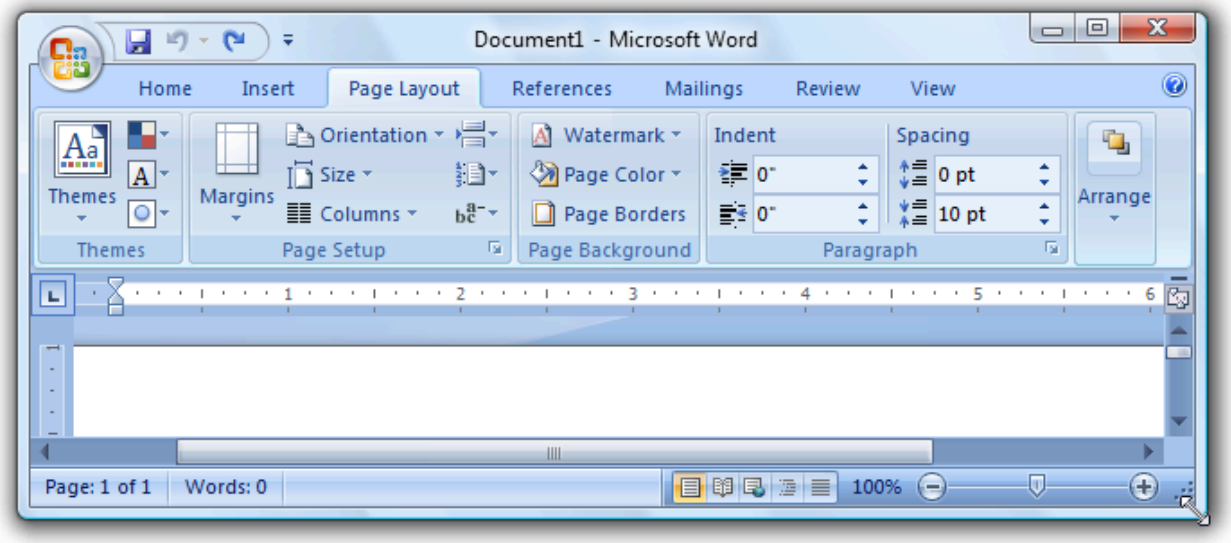
Text becomes difficult to read as the line length increases. For text documents, consider a maximum line length of 80 characters to make the text easy to read. (Characters include letters, punctuation, and spaces.)

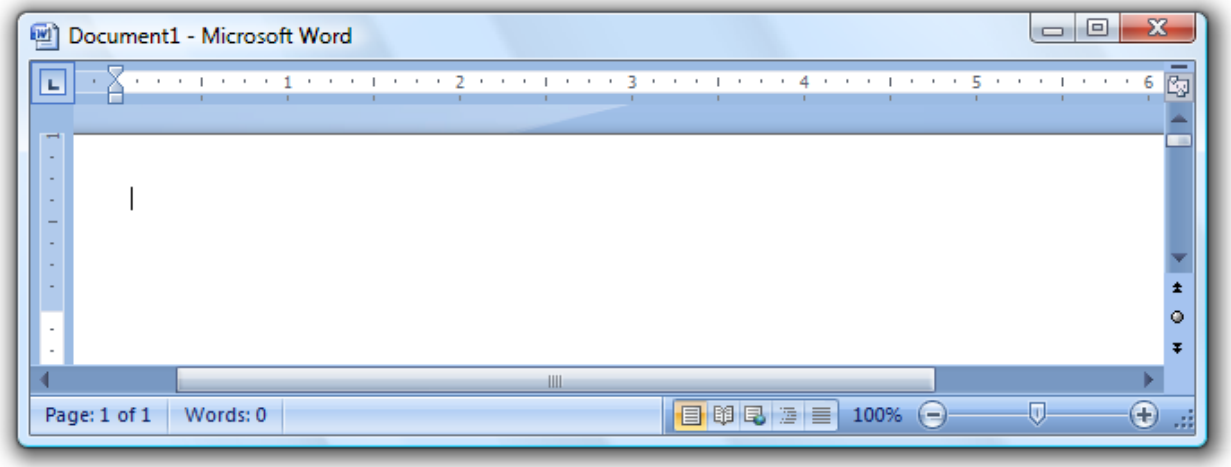
Incorrect:



In this example, the long text length makes for difficult reading.

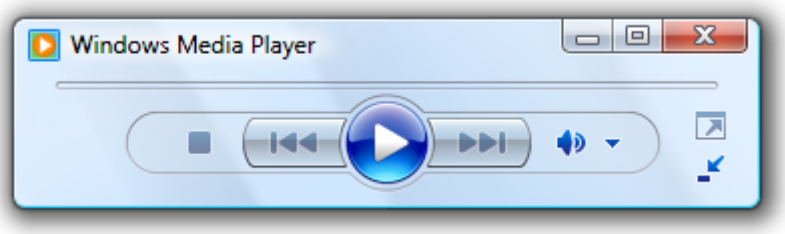
Finally, resizable windows also need to use screen space effectively when made smaller, by making resizable content smaller and by removing space from UI elements that can work effectively without it. At some point, the window or its UI elements become too small to be usable, so they should be assigned a minimum size or some elements should be removed completely.





In this example, the pane has a minimum size.

Some programs benefit from using a completely different presentation to make the content usable at smaller sizes.



In this example, Windows Media Player changes its format when the window becomes too small for the standard format.

Focus

A layout has focus when there is one obvious place to look first. Focus is important to show users where to start scanning your window or page. Without clear focus, the user's eye will wander aimlessly. The focal point should be something important that users need to find and understand quickly, and should have the greatest visual emphasis. The upper-left corner is the natural focal point for most windows.

There should be only one focal point. Just as in real life, the eye can focus on only one thing at a time, users can't focus on multiple places simultaneously.

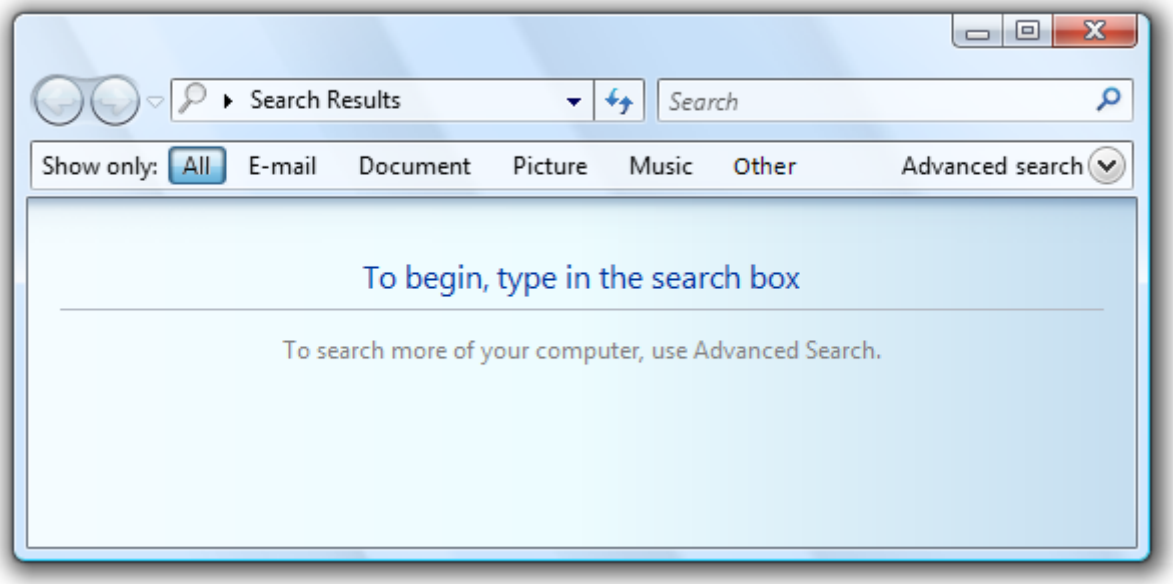
To make a UI element the focal point, you can give it visual emphasis by:

- Placing it in the upper-left or upper-center portion of the surface.
- Using interactive controls that are important and readily comprehensible.
- Using prominent text, such as a main instruction.
- Giving the controls default selection and initial input focus.
- Placing the controls in a different colored background.

Consider Windows Search. The focal point for Windows Search should be the Search box because it is the starting point for the task. However, it is located in the upper-right corner to be consistent with the standard Search box placement. The Search box has input focus, but given its location in the scan path, that clue alone isn't sufficient.

To address this problem, there is prominent instruction in the upper-center portion of the window to direct users to the right location.

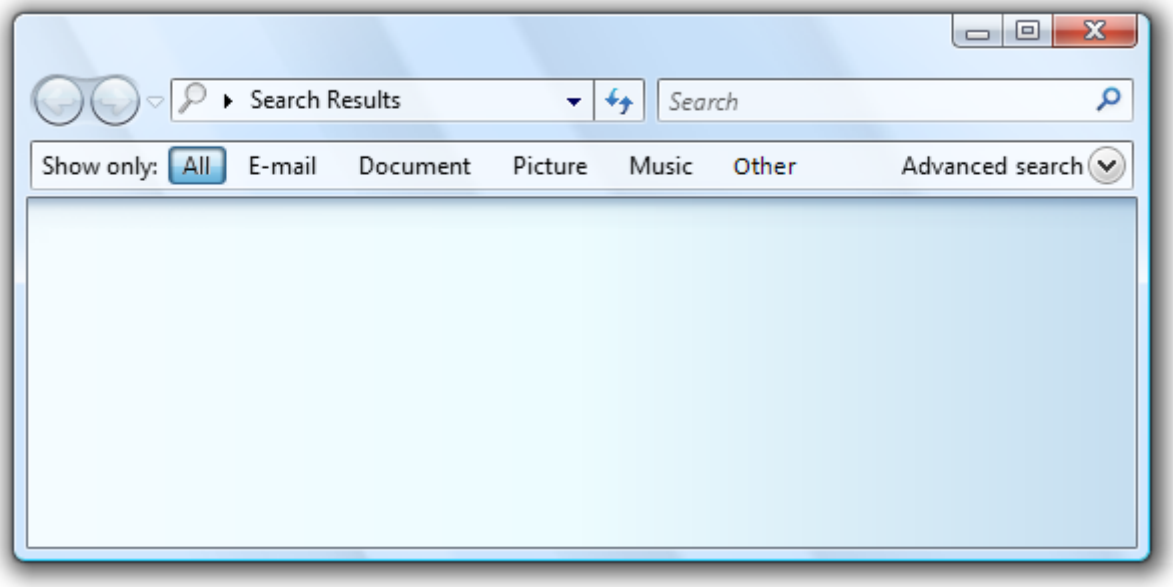
Acceptable:



In this example, a prominent instruction in the upper-center portion of the window directs users to the Search box.

Without the instructions, the window wouldn't have an obvious focal point.

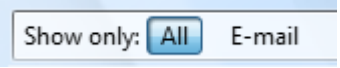
Incorrect:



This example has no obvious focal point. Users don't know where to look.

If you give a UI element visual emphasis, make sure that attention is warranted. In the previous incorrect Windows Search example, the highlighted All button is in the upper-left corner and has the most visual emphasis, but it's not the intended focal point. Users may get stuck looking at this button trying to figure out what to do with it.

Incorrect:



Without the prominent instruction as the focal point, the highlighted All button is an unintentional focal point.

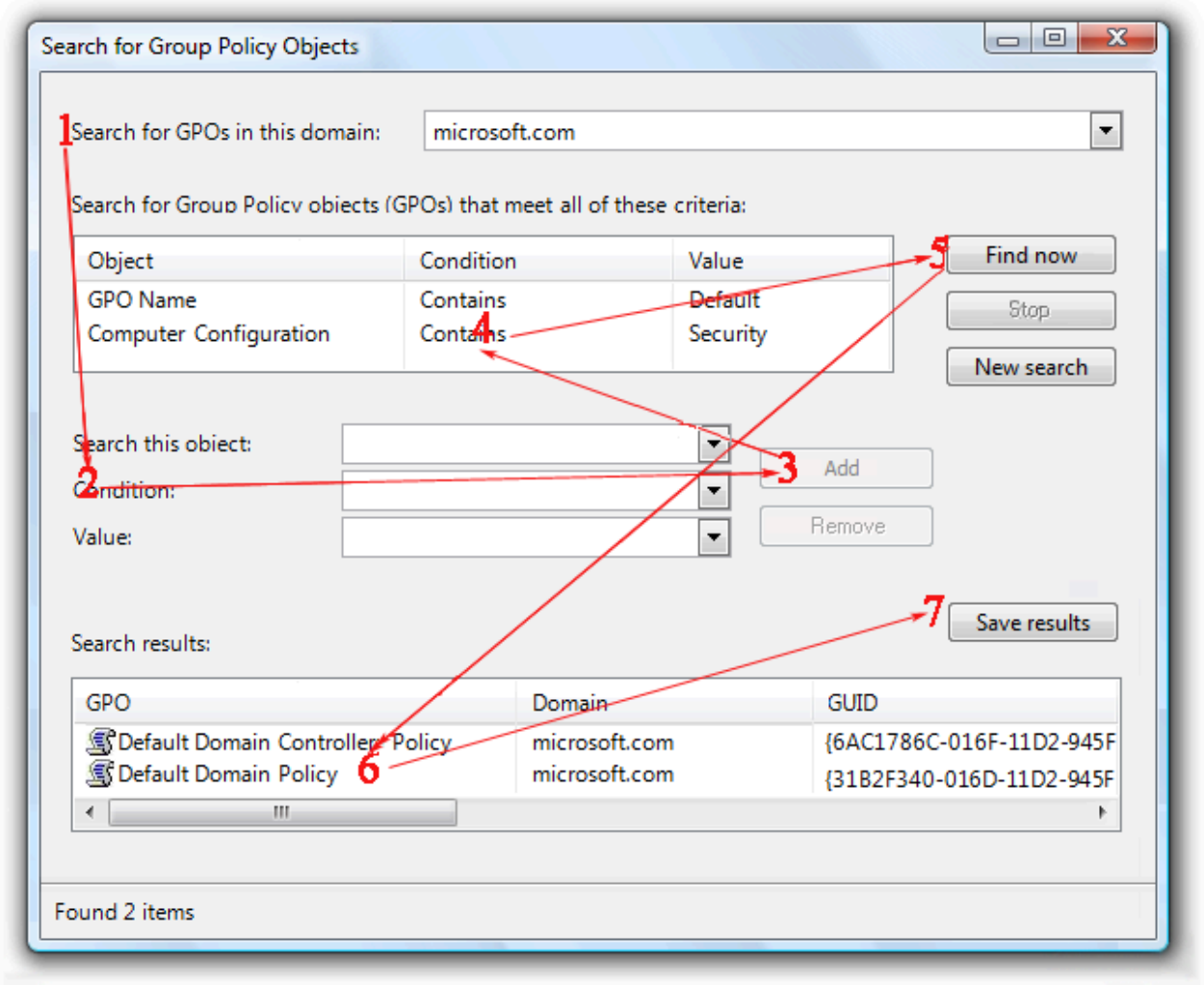
Flow

A layout has flow when users are guided smoothly and naturally by a clear path through its surface, finding UI elements in the order appropriate for their use. Once users identify the focal point, they need to determine how to complete the task. The placement of the UI elements conveys their relationship and should mirror the steps to perform the task. Typically, this means the task's steps should flow naturally in a left-to-right, top-to-bottom order (in Western cultures).

You know a layout has good flow when:

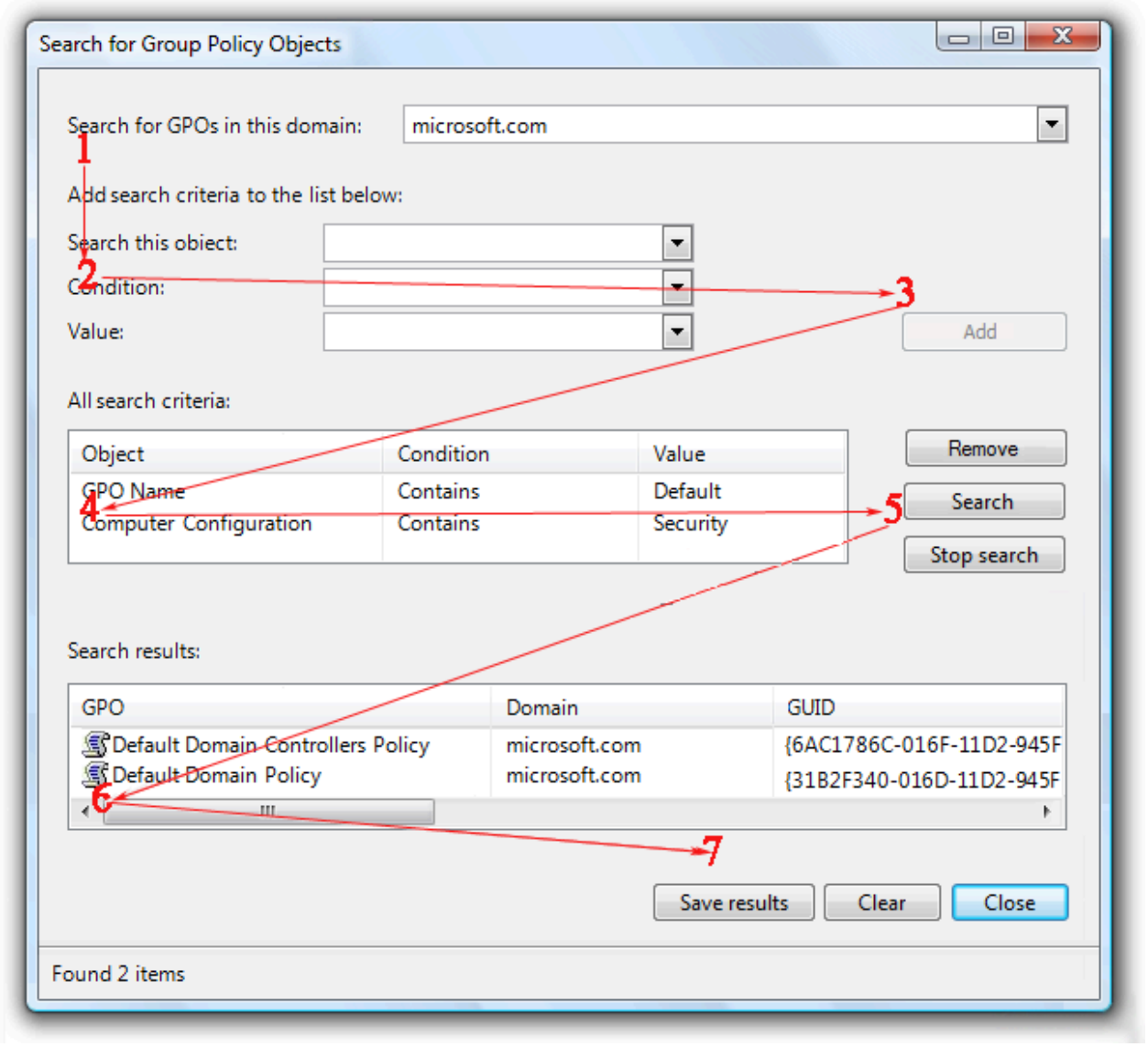
- The placement of the UI elements mirrors the steps users need to perform the task.
- UI elements that initiate a task are in the upper-left corner or upper-center.
- UI elements that complete a task are in the lower-right corner.
- Related UI elements are together; unrelated elements are separate.
- Required steps are in the main flow.
- Optional steps are outside the main flow, possibly de-emphasized by using a suitable background or progressive disclosure.
- Frequently used elements appear before infrequently used elements in the scan path.
- Users always know what to do next. There are no unexpected jumps or breaks in the task flow.

Incorrect:



In this example, users don't know what to do next. There are unexpected jumps and breaks in the task flow.

Correct:



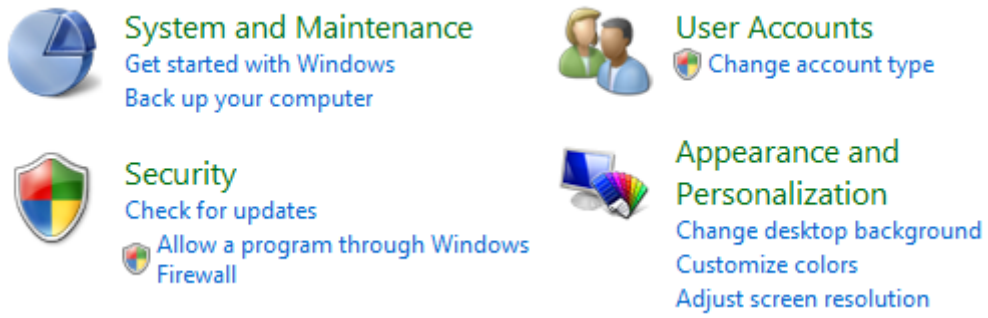
In this example, the presentation of the UI elements mirrors the steps to perform the task.

Grouping

A layout has grouping when logically related UI elements have a clear visual relationship. Groups are important because it is easier for users to understand and focus on a group of related items than the items individually. Groups make a layout appear simpler and easier to parse.

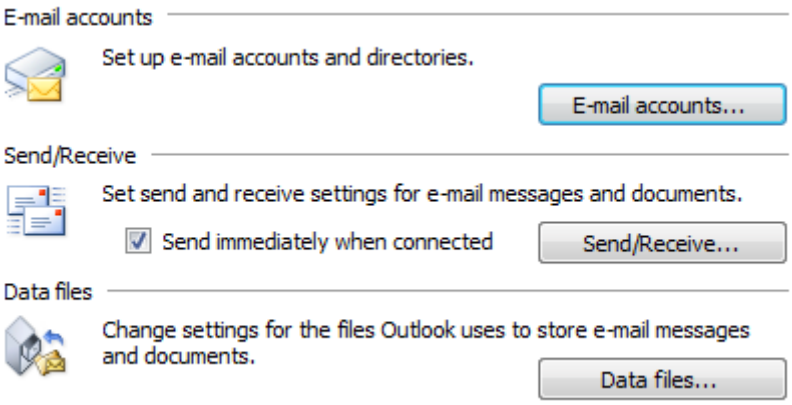
You can show grouping in the following ways (in increasing heaviness):

- **Layout.** You can group related controls next to each other and put extra spacing between unrelated controls.



In this example, layout alone is used to show control relationships.

- **Separators.** A separator is a horizontal or vertical line that unifies a group of controls. Separators provide a simpler, cleaner look. However, unlike group boxes, they work best when they span the full surface.

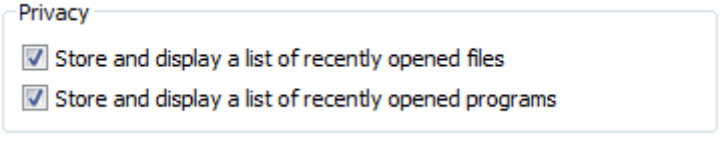


In this example, labeled separators are used to show control relationships.

- **Aggregators.** An aggregator is a graphic that creates a visual relationship between strongly related controls.

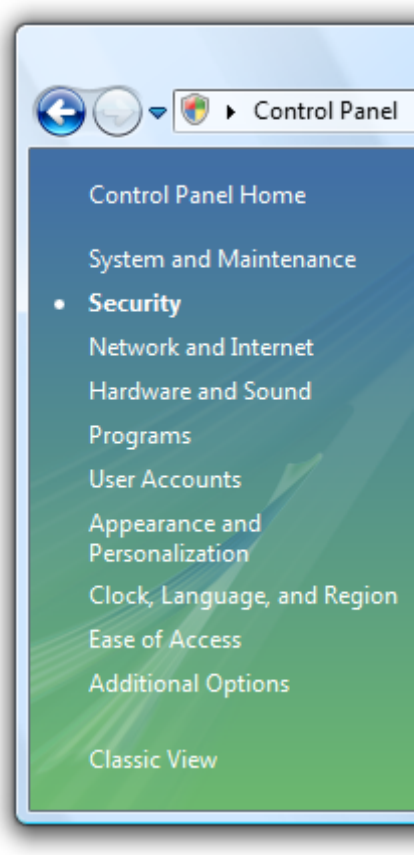
In this example, a boundary aggregator is used to emphasize the relationship between the controls and make them feel like a single control instead of eight.

- **Group boxes.** A group box is a labeled rectangular frame that surrounds a set of related controls.



In this example, a group box surrounds and labels a set of related controls.

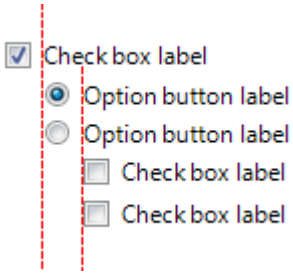
- **Backgrounds.** You can use backgrounds to emphasize or de-emphasize different types of content.



  In this example, the control panel task pane is used to group related tasks and control panel items.

  To avoid visual clutter, the lightest weight grouping that does the job well is the best choice. For more information, see Group Boxes, Tabs, Separators and Backgrounds.

Regardless of the grouping style, you can use indenting to show the relationship of the controls within a group. Controls that are peers to each other should be left-aligned, and dependent controls are indented 12 DLUs or 18 relative pixels.



Dependent controls are indented 12 DLUS or 18 relative pixels, which by design is the distance between check boxes and radio buttons from their labels.

You know a layout has good grouping when:

- The window or pages has at most 7 groups.
- The purpose of each group is obvious.
- The relationship of controls within each group is obvious, especially control dependency.
- The grouping simplifies the content rather than making it more complex.

Alignment

Alignment is the coordinated placement of UI elements. Alignment is important because it makes content easier to scan and affects users' perception of visual complexity.

There are several goals to consider when determining alignment:

- **Ease in horizontal scanning.** Users can read horizontally and find related items next to each other, without any awkward gaps.
- **Ease in vertical scanning.** Users can scan columns of related items and immediately find what they are looking for, with minimal horizontal eye movement.
- **Minimal visual complexity.** Users perceive a layout to be visually complex if it has unnecessary vertical alignment grid lines.

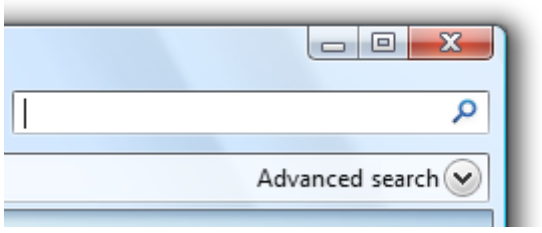
Horizontal alignment

Left alignment

Because of the left-to-right reading order, left alignment works well for most content. Left alignment makes columnar data easy to scan vertically.

Right alignment

Right alignment is the best choice for numeric data, especially columns of numeric data. Right alignment also works well for commit buttons as well as controls aligned with right window edge.



In this example, the Advanced search progressive disclosure control is right aligned because it is placed against the right window edge.

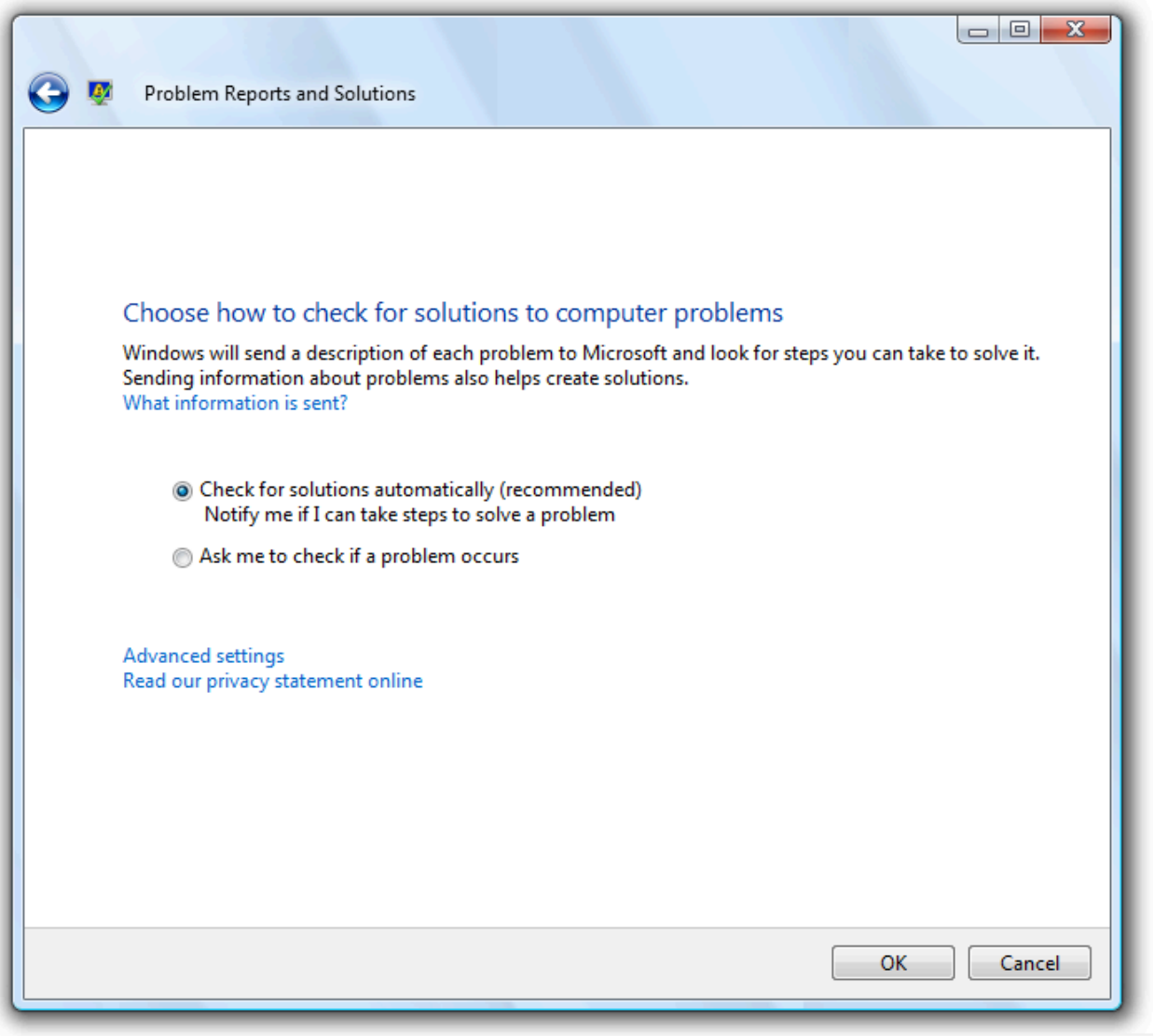
Center alignment

Center alignment is best reserved for situations where either left or right alignment is inappropriate or appears unbalanced.

In this example, the media player control is centered to give a balanced appearance.

Don't center window content just to fill space.

Incorrect:



In this example, content is incorrectly centered in a resizable window to fill space.

Vertical alignment

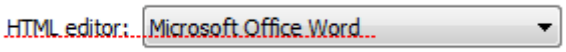
Element tops

Because of the top-to-bottom reading order, top alignment works well for most content. Top alignment makes UI elements easy to scan horizontally.

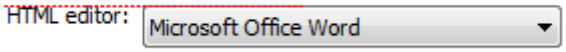
Text baselines

When vertically aligning controls with text, align the text baselines to give a smooth horizontal reading flow.

Correct:



Incorrect:



In the correct example, the control and its label are vertically aligned by their text baselines.

You know a layout has good alignment when:

- It is easy to scan both horizontally and vertically.
- It has a simple visual appearance.

Label alignment

The general alignment rules apply to control labels, but it is a common problem worthy of specific attention. Label alignment has these goals:

- Ease in vertical scanning to find the right control.
- Ease in horizontal scanning to associate labels with their controls.
- Ease in localization, handling labels that differ in length across languages.
- Works well with a mixture of different label lengths.
- Makes efficient use of available space while avoiding truncated text.

The overall goal is to reduce the amount of eye movement required to find what users are likely looking for, but the nature of the controls and what users are looking for depends on the context.

There are four common label placement and alignment styles, each with its benefits:

- Left-justified labels above controls
- Left-justified labels on left of controls
- Left-justified labels on left of controls, controls ragged on left

- Right-justified labels on left of controls

Left-justified labels above controls

This style is the easiest to localize because the layout doesn't depend upon the length of the labels, but it takes the most vertical space.

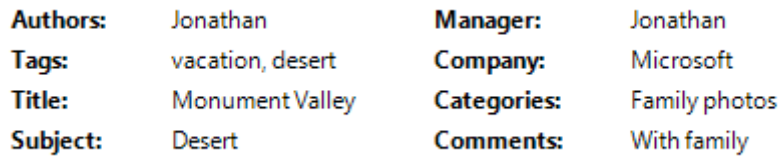
This style takes the most vertical space but is easiest to localize. It's a better choice for labeling mostly interactive controls.

Best used when:

- The controls being labeled are interactive (not just text).
- The UI will be localized. This style often affords room to double or even triple the length of the label.
- The UI is using a fixed layout technology (such as Win32).
- There are ten or fewer controls. With more controls, the labels are hard to scan.
- There is sufficient vertical space to accommodate the labels.
- The layout needs to be free form, not just columns.

Left-justified labels on left of controls

This style is the easiest to scan vertically and it also works well when labels differ greatly in length, but it is harder to associate the label with its control. This style can use multi-line labels if necessary.



This style works well. However, there are two columns but visually it looks like there are four making the data appear more complex.

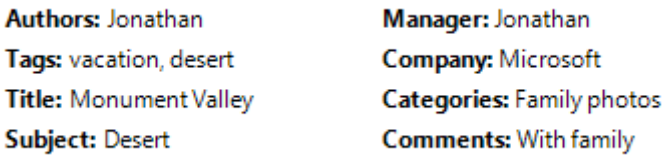
Best used when:

- Users are likely to scan vertically to find specific labels.

- Users aren't likely to read the labels and controls in a left-to-right, top-to-bottom manner.
- There is sufficient horizontal space to accommodate the labels.
- The labels vary significantly in length.
- There are many controls, such as with forms.
- There are few columns. Visually the labels and controls appear as two individual columns.

Left-justified labels on left of controls, controls ragged on left

This style makes it easy to scan the labels vertically and the labels and controls horizontally, and is very space efficient; but it is harder to scan the controls vertically. The controls are right-justified to take full advantage of the available space.



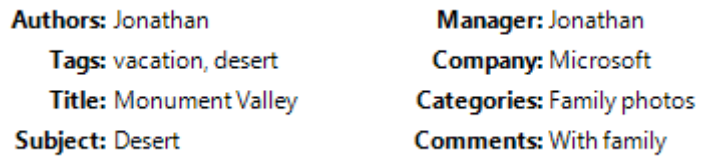
This style is compact and easy to read, but it's hard to scan controls vertically.

Best used when:

- The UI is using a variable layout technology (such as Windows Presentation Foundation).
- Users are likely to scan vertically to find specific labels.
- Users are likely to read the labels and controls in a left-to-right, top-to-bottom manner.
- Users aren't likely to scan the controls vertically.
- The control text varies in length and would likely be truncated if another style were used.
- The controls are read-only, such as read only text boxes. For other controls, this alignment will look sloppy. However, the controls may become editable on click.
- There are many columns, but few controls in a column.

Right-justified labels on left of controls

This style is the easiest to read horizontally to associate the labels with their controls, but it is hard to scan the labels vertically and doesn't work well when labelsList with indented labels and controls differ greatly in length.



This style permits easy vertical scanning of the controls, but makes it hard to scan the labels vertically.

Best used when:

- Users are likely to read the labels and controls in a left-to-right, top-to-bottom manner.
- Users aren't likely to scan vertically to find specific labels, possibly because:
  - There are few controls.
  - The labels are well known.
  - The controls are mostly self explanatory and are rarely blank (possibly having default values to prevent blank controls).
- There is sufficient horizontal space to accommodate the labels.
- The labels don't vary significantly in length.
- There are many columns. Visually the labels and controls appear as a single column.

Before adopting any of these styles, however, consider two more factors:

- Prefer a style that you can use consistently across your program.
- Left-justified labels either above controls on left of controls are the most common styles, so they should be given preference.

Balance

A window or page has balance when its content appears evenly distributed across its surface. If the surface physically had the same weighting as it has visually, a balanced layout wouldn't tip over to one side.

The most common balance problem is having too much content on the left side of a window or page. You can create balance in the following ways:

- Using larger margins on the left side than the right.
- Placing UI elements used to complete a task on the right.
- Placing UI elements used throughout the task in the center.
- Lengthening resizable or multi-line controls.
- Using center-alignment strategically.

This well balanced wizard page layout shows a larger left margin than right to improve balance.

If these techniques don't achieve balance, consider reducing the width of the window or page to better match its content.

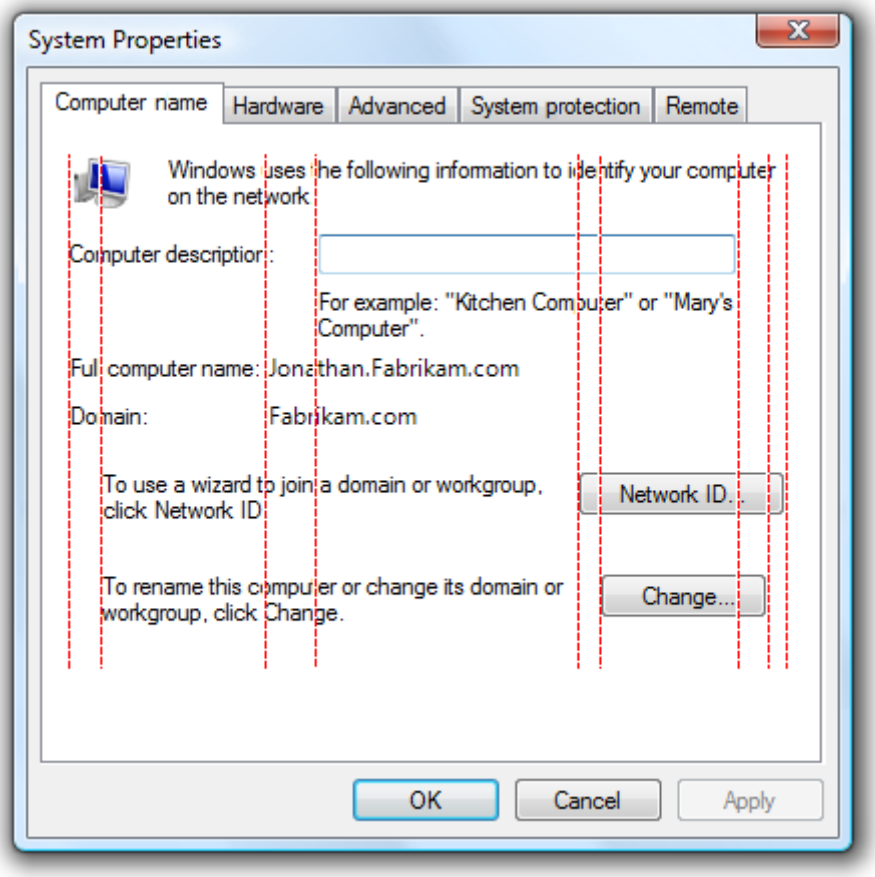
For resizable surfaces, don't center content just to achieve balance. Rather, maintain a fixed upper left origin, define a maximum surface area, and balance the content within the space used.

Grids

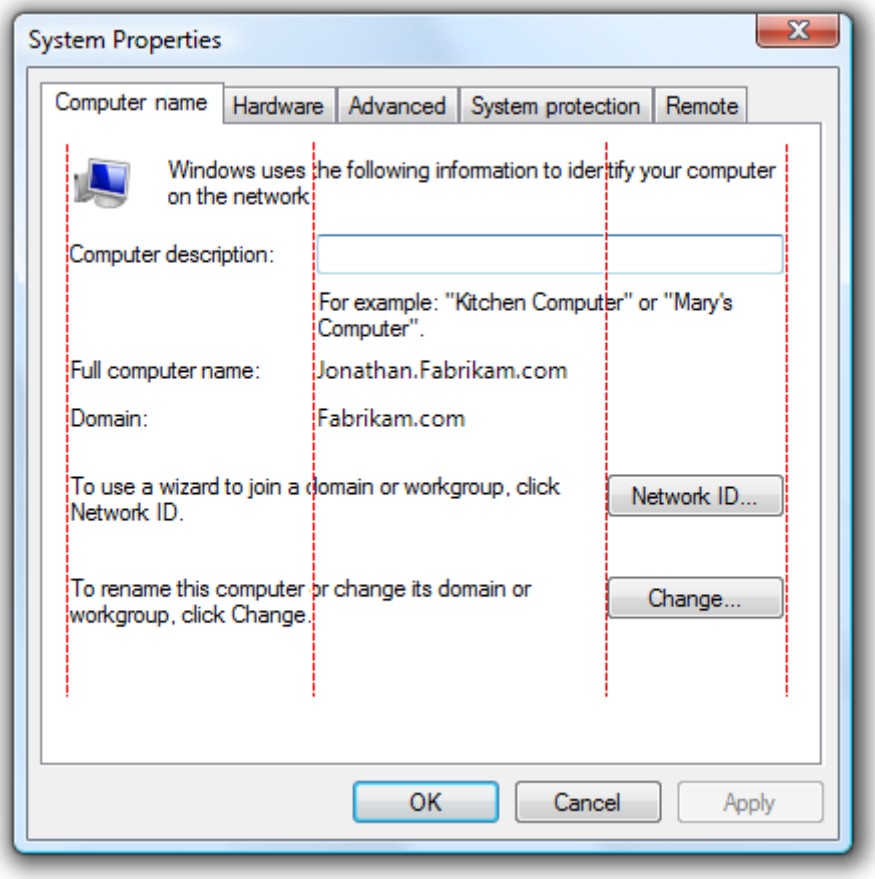
A grid is an invisible underlying alignment system. Grids can be symmetrical, but asymmetric grids work just as well. When used by a single window or page, grids help organize content within a surface. When reused, grids create consistent layout across surfaces.

The number of grid lines affect the perception of visual complexity. A layout with fewer grid lines appears simpler than a layout with more grid lines.

Visually complex:



Visually simple:



Unnecessary grid lines create visual complexity.

You know a layout is using grids effectively when:

- Windows or pages with similar content or function have similar layout.
- Repeated design elements appear in similar locations across windows and pages.
- There are no unnecessary vertical and horizontal alignment grid lines.

Visual simplicity

Visual simplicity is the perception that a layout is not more complicated than it needs to be.

You know a layout has visual simplicity when it:

- Eliminates unnecessary layers of window chrome.
- Presents the content using at most seven easily identifiable groups.
- Uses lightweight grouping, such as layout and separators instead of group boxes.
- Uses lightweight controls, such as links instead of command buttons for secondary commands, and drop-down lists instead of lists for choices.
- Reduces the number of vertical and horizontal alignment grid lines.
- Reduces the number of control sizes, by, for example, using only one or two command button widths on a surface.
- Uses progressive disclosure to hide UI elements until they are needed.
- Uses sufficient space so that the window or page doesn't feel cramped.
- Sizes windows and controls appropriately to eliminate unnecessary scrolling.
- Uses a single font with a small number of sizes and text colors.

As a general rule, if a layout element can be eliminated without harming the effectiveness of the UI, it probably should be.

Guidelines

Screen resolution and dpi

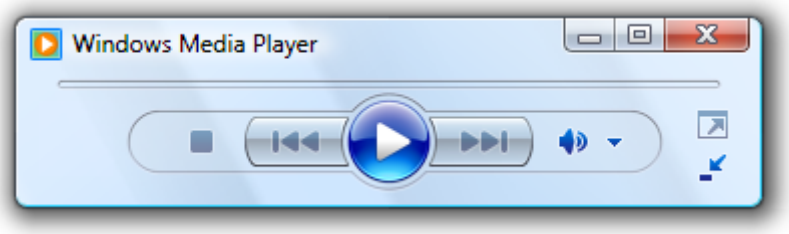
- **Support the minimum Windows effective resolution of 800x600 pixels.** For critical UIs that must work in safe mode, support an effective resolution of 640x480 pixels. Be sure to account for the space used by the taskbar by reserving 48 vertical relative pixels for windows displayed with the taskbar.
- **Optimize resizable window layouts for an effective resolution of 1024x768 pixels.** Automatically resize these windows for lower screen resolutions in a way that is still functional.

- **Be sure to test your windows in 96 dots per inch (dpi) (at 800x600 pixels), 120 dpi (at 1024x768 pixels), and 144 dpi (at 1200x900 pixels) modes.** Check for layout problems, such as clipping of controls, text, and windows, and stretching of icons and bitmaps.
- **For programs with touch and mobile use scenarios, optimize for 120 dpi.** High-dpi screens are currently prevalent on touch and mobile PCs.

Window size

- **Choose a default window size appropriate for its contents.** Don't be afraid to use larger initial window sizes if you can use the space effectively.
- **Use a balanced height to width aspect ratio.** An aspect ratio between 3:5 and 5:3 is preferred, although an aspect ratio of 1:3 can be used for message dialog boxes (such as errors and warnings).
- **Use resizable windows whenever practical to avoid scroll bars and truncated data.** Windows with dynamic content, documents, images, lists, and trees benefit the most from resizable windows.
- **For text documents, consider a maximum line length of 80 characters** to make the text easy to read. (Characters include letters, punctuation, and spaces.)
- Fixed-sized windows:
  - **Fixed-sized windows must be entirely visible and sized to fit within the work area.**
- Resizable windows:
  - **Resizable windows may be optimized for higher resolutions, but sized down as needed at display time to the actual screen resolution.**
  - **Progressively larger window sizes must show progressively more information.** Make sure that at least one window portion or control has resizable content.
  - **Keep the upper-left origin of the content fixed as the window is resized.** Don't move the origin to balance the content as the window size changes.
  - **Set a maximum content size if the content can be too stretched too wide.** If the content becomes unwieldy, don't resize the content area beyond its maximum width or change the content's origin as the window is resized larger. Instead, keep a maximum width and a fixed upper-left origin.
  - **Set a minimum window size if there is a size below which the content is no longer usable.** For resizable controls, set minimum resizable element sizes to their smallest functional sizes, such as minimum functional column widths in list views. Optional UI elements should be removed completely.

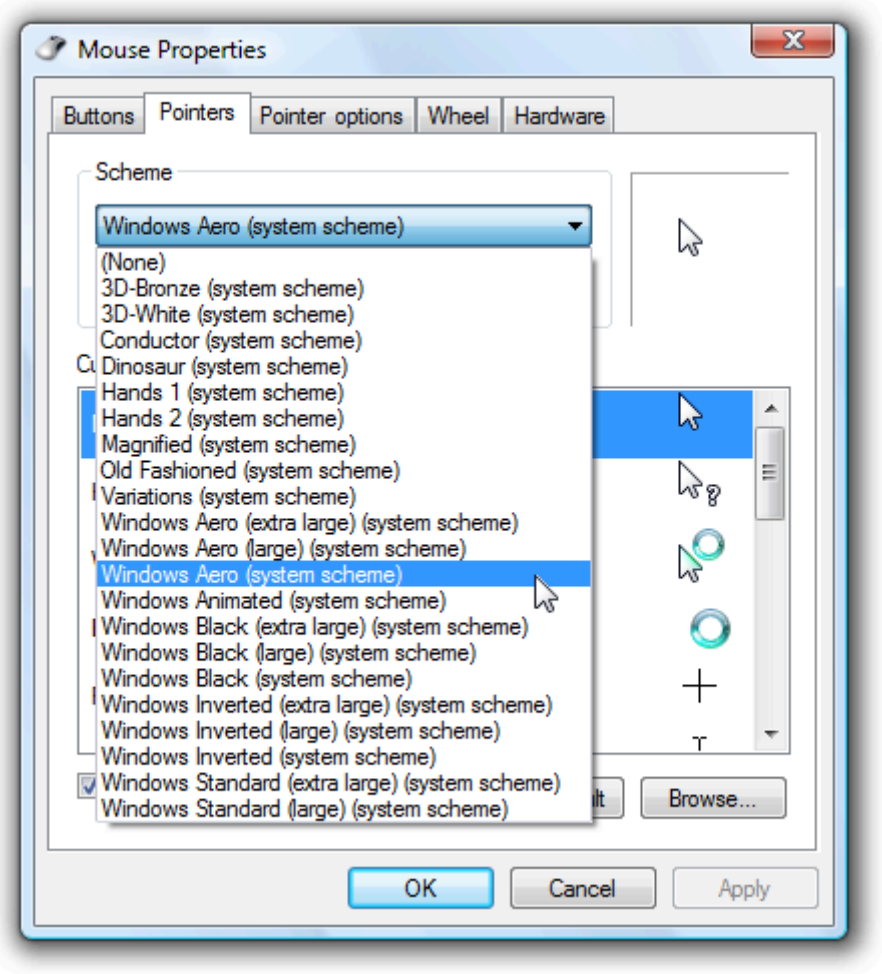
- **Consider altering the presentation to make the content usable at smaller sizes.**



In this example, Windows Media Player changes its format when the window becomes too small for the standard format.

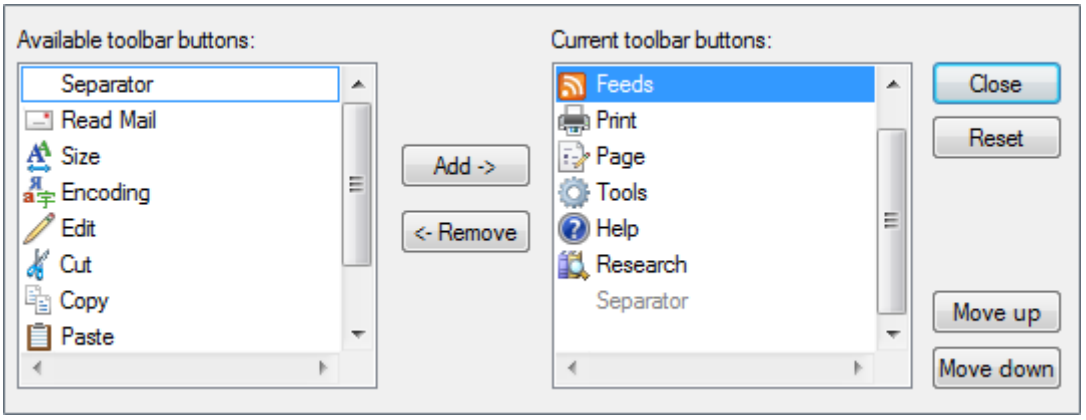
Control size

- **Make all interactive controls at least relative 16x16 pixels.** Doing so works well for all input devices, including Windows Tablet and Touch Technology.
- **Size controls to avoid truncated data.** Don't truncate data that must be read to perform a task. Size list view columns to avoid truncated data.
- **Size controls to eliminate unnecessary scrolling.** Make controls slightly larger if doing so eliminates a scrollbar. There should be few vertical scrollbars and no unnecessary horizontal scrollbars.



In this example, the drop-down list is sized to eliminate the scrollbar.

- **Reduce the number of control sizes on a surface.** Prefer using the standard recommended control sizes and when necessary, use a few consistently sized larger or smaller controls. Try to use a single width for list boxes and tree views, and no more than three widths for command buttons and drop-down lists. However, text box and combo box widths should suggest the length of their longest or expected input.



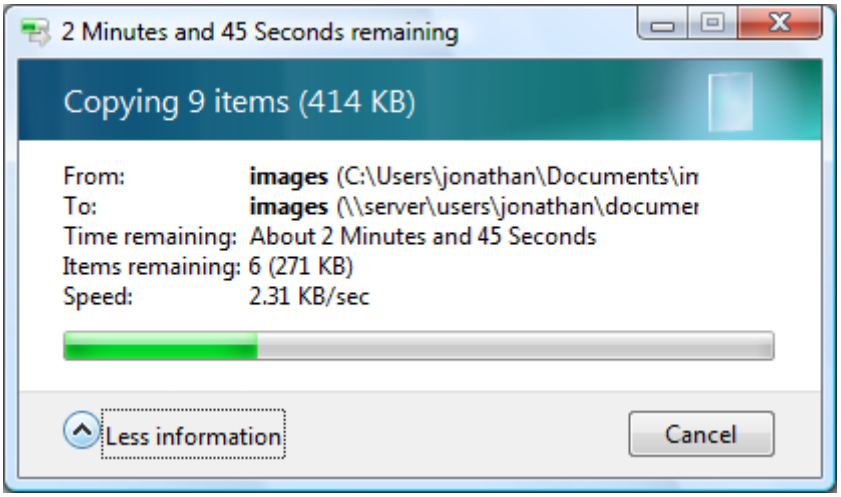
In this example, one list box and command button size is used consistently.

- **For controls that are sized based on their text, include an additional 30 percent (up to 200 percent for shorter text) for any text that will be localized.** This

guideline assumes the layout is designed using English text. Note also that this guideline refers to localized text, not numbers.

- **Extend static text controls, check boxes, and radio buttons to the maximum width that will fit in the layout.** Doing so avoids truncation from variable length text and localization.

  Incorrect:

  In this example, control text is unnecessarily truncated.

Control spacing

- **If controls aren't touching, have at least 3 DLUs (5 relative pixels) of space between them.** Otherwise, users may click on inactive space between the controls. Since clicking inactive space has no result or visual feedback, users are often uncertain what went wrong.

Placement

- **Arrange the UI elements within a surface to flow naturally in a left-to-right, top-to-bottom order (in Western cultures).** The placement of the UI elements conveys their relationship and should mirror the steps to perform the task.
- **Place UI elements that initiate a task in the upper-left corner or upper-center.** Give the UI element that users should look at first the greatest visual emphasis.
- **Place UI elements that complete a task in the lower-right corner.**
- **Place related UI elements together, and separate unrelated elements.**
- **Place required steps in the main flow.**
- **Place optional steps outside the main flow,** possibly de-emphasized by using a suitable background or progressive disclosure.

- **Place frequently used elements before infrequently used elements** in the scan path.

Focus

- **Choose a single UI element that users need to look at first to be the focal point.** The focal point should be something important that users need to find and understand quickly.
- **Place the focal point in the upper-left corner or upper-center.**
- **Give the focal point the greatest visual emphasis,** such as prominent text, default selection, or initial input focus.

Alignment

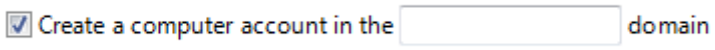
- Normally, use left alignment.
- Use right alignment for numeric data, especially columns of numeric data.
- Use right alignment for commit buttons, as well as controls aligned with right window edge.
- Use center alignment when either left or right alignment is inappropriate or appears unbalanced.
- When vertically aligning controls with text, align the text baselines to give a smooth horizontal reading flow.
- For label alignment, refer to the Label alignment section in Design concepts.

Accessibility

- **Don't use layout as the only means to convey important information about a UI.** Users who have visual impairments may not be able to interpret this presentation. For example, make sure that controls labels communicate their relationship to other items.

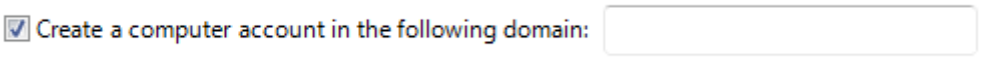
- **Don't embed subordinate controls within control labels to create a sentence or phrase.** Such associations are based purely on layout and aren't handled well by keyboard navigation or accessibility assistive technologies. Furthermore, this technique isn't localizable because sentence structure varies with language.

  Incorrect:

  In this example, the text box is incorrectly placed within the check box label.

Correct:



Here, the text box is placed after the check box label.

- **Make grouping accessible.** Groups defined by window panes, group boxes, separators, text labels, and aggregators are automatically handled by accessibility aids. However, groups defined only by placement and backgrounds are not, and must be defined programmatically for accessibility.

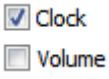
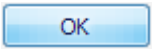
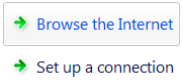
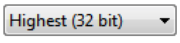
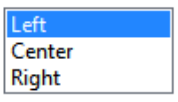
For more guidelines, see Accessibility.

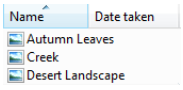
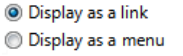
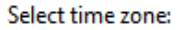
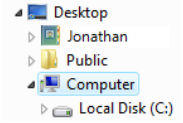
Recommended sizing and spacing

Control sizing

The following table lists the recommended sizes (width x height, or height if a single number) for common UI elements (for 9 pt. Segoe UI at 96 dpi). The widths based on the longest item in English add 30 percent for localization (up to 200 percent for shorter text) for any text (but not numbers) that will be localized.

Expand table

Example	Control	Dialog units	Relative pixels
	Check boxes	10	17
	Combo boxes	width of longest item + 30% x 14	width of longest item + 30% x 23
	Command buttons	50 x 14	75 x 23
	Command links	25 (one line) or 35 (two lines)	41 (one line) or 58 (two lines)
	Drop-down lists	width of longest valid data + 30% x 14	width of longest item + 30% x 23
	List boxes	width of longest item + 30% x an integral number of items (3 items minimum)	

Example	Control	Dialog units	Relative pixels
	List views	columns widths that avoid truncated data x an integral number of items	
	Progress bars	107 or 237 x 8	160 or 355 x 15
	Radio buttons	10	17
	Sliders	15	24
	Text (static)	8	13
	Text boxes	width of longest or expected input + 30% x 14 (one line) + 10 for each additional line	width of longest valid data + 30% x 23 relative pixels (one line) + 16 for each additional line
	Tree views	width of longest item + 30% x an integral number of items (5 items minimum)	

Spacing

The following table lists the recommended spacing between common UI elements (for 9 pt. Segoe UI at 96 dpi).

Expand table

	Element	Dialog units	Relative pixels
	Dialog box margins	7 on all sides	11 on all sides
	Between text labels and their associated controls (for example, text boxes and list boxes)	3	5
	Between related controls	4	7
	Between unrelated controls	7	11
	First control in a group box	11 down from the top of the group box; align vertically to the group box title	16 down from the top of the group box; align vertically to the group box title

	Element	Dialog units	Relative pixels
	Between controls in a group box	4	7
	Between horizontally or vertically arranged buttons	4	7
	Last control in a group box	7 above the bottom of the group box	11 above the bottom of the group box
	From the left edge of a group box	6	9
	Text label beside a control	3 down from the top of the control	5 down from the top of the control
	Between paragraphs of text	7	11
	Smallest space between interactive controls	3 or no space	5 or no space
	Smallest space between a non-interactive control and any other control	2	3

Feedback

Was this page helpful? Yes No

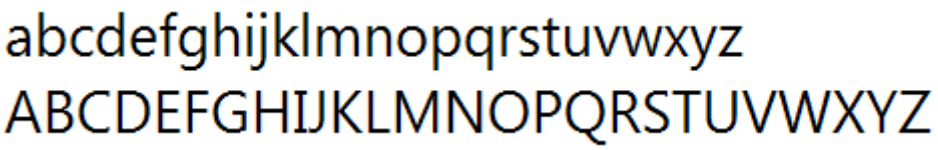
Provide product feedback | Get help at Microsoft Q&A

Fonts

Article • 06/03/2021

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

Users interact with text more than with any other element in Microsoft Windows. Segoe UI (pronounced "SEE-go") is the Windows system font. The standard font size has been increased to 9 point.



The Segoe UI font.

Segoe UI and Segoe are not the same font. Segoe UI is the Windows font intended for user interface text strings. Segoe is a branding font used by Microsoft and partners to produce material for print and advertising.

Segoe UI is an approachable, open, and friendly typeface, and as a result has better readability than Tahoma, Microsoft Sans Serif, and Arial. It has the characteristics of a humanist sans serif: the varying widths of its capitals (narrow E and S, for instance, compared with Helvetica, where the widths are more alike, fairly wide); the stress and letterforms of its lowercase; and its true italic (rather than an "oblique" or slanted roman, like many industrial-looking sans serifs). The typeface is meant to give the same visual effect on screen and in print. It was designed to be a humanist sans serif with no strong character or distracting quirkiness.

Segoe UI is optimized for ClearType, which is on by default in Windows. With ClearType enabled, Segoe UI is an elegant, readable font. Without ClearType enabled, Segoe UI is only marginally acceptable. This factor determines when you should use Segoe UI.

Segoe UI includes Latin, Greek, Cyrillic, and Arabic characters. There are new fonts, also optimized for ClearType, created for other character sets and uses. These include Meiryo for Japanese, Malgun Gothic for Korean, Microsoft JhengHei for Chinese (Traditional), Microsoft YaHei for Chinese (Simplified), Gisha for Hebrew, and Leelawadee for Thai, and the ClearType Collection fonts designed for document use.

Meiryo includes Latin characters based on Verdana. Malgun Gothic, Microsoft JhengHei, and Microsoft YaHei use a customized Segoe UI. Use of italic versions of these fonts is not recommended. Malgun Gothic, Microsoft JhengHei, and Microsoft YaHei are supplied in regular and bold styles only, meaning italic characters are synthesized by slanting the upright styles. Although Meiryo includes true italic and bold italics, these styles only apply to the Latin characters the Japanese characters remain upright when italic styling is applied.

A variation of Meiryo, called Meiryo UI, is preferred in the ribbons command user interface.

To support locales using these character sets, Segoe UI is replaced with the correct fonts depending on each locale during the localization process.

To license Segoe UI and other Microsoft fonts for distribution with a Windows-based program, contact Monotype.

Note: Guidelines related to style and tone and user interface text are presented in separate articles.

Design concepts

Fonts, typefaces, point sizes, and attributes

In traditional typography, a font describes a combination of a typeface, a point size, and attributes. A typeface is the look of the font. Segoe UI, Tahoma, Verdana, and Arial are all typefaces. Point size refers to the size of the font, measured from the top of the ascenders to the bottom of the descenders, minus the internal spacing (called leading). A point is roughly 1/72 inch. Finally, a font can have attributes of bold or italic.

Informally, people often use font in place of typeface as done in this article but technically, Segoe UI is a typeface, not a font. Each combination of attributes is a unique font (for example, 9 point Segoe UI regular, 10 point Segoe UI bold, and so on).

Serif and sans serif

Typefaces are either serif or sans serif. Serif refers to small turns that often finish the strokes of letters in a font. A sans serif typeface doesn't have serifs.

Readers generally prefer serif fonts used as body text within a document. The serifs provide a feeling of formality and elegance to a document. For UI text, the need for a

clean appearance and the lower resolution of computer monitors makes sans serif typefaces the better choice.

Contrast

Text is easiest to read when there is a large difference between the luminance of the text and the background. Black text on a white background gives the highest contrast dark text on a very light background can provide high contrast as well. This combination is best for primary UI surfaces.

Light text on a dark background offers good contrast, but not as good as dark text on a light background. This combination works well for secondary UI surfaces, such as Explorer task panes, that you want to de-emphasize relative to the primary UI surfaces.

If you want to make sure users read your text, use dark text on a light background.

Affordances

Text can use the following affordances to indicate how it is used:

- **Pointer.** The I-bar ("text select") pointer indicates that the text is selectable, whereas the left-pointing arrow ("normal select") pointer indicates that text isn't.
- **Caret.** When text has input focus, the caret is the flashing vertical bar that indicates the insertion/selection point in selectable or editable text.
- **Box.** A box around text that indicates that it's editable. To reduce the weight of the presentation, the box may be displayed dynamically only when the editable text is selected.
- **Foreground color.** Light gray indicates that text is disabled. Non-gray colors, especially blue and purple, indicate that text is a link.
- **Background color.** A light gray background weakly suggests that text is read-only, but in practice read-only text can have any color background.

These affordances are combined for the following meanings:

- **Editable.** Text displayed in a box, with a text select pointer, a caret (on input focus), and usually on a white background.
- **Read-only, selectable.** Text with a select pointer and a caret (on input focus).
- **Read-only, non-selectable.** Text with an arrow pointer.
- **Disabled.** Light gray text with an arrow pointer, sometimes on a gray background.

Read-only text traditionally has a gray background, but a gray background isn't necessary. In fact, a gray background can be undesirable, especially for large blocks of text, because it suggests that the text is disabled and discourages reading.

Accessibility and the system font, sizes, and colors

The guidelines for making text accessible to users with disabilities or impairments can be boiled down to one simple rule: Respect the user's settings by always using the system font, sizes, and colors.

If you do only one thing...

Respect the user's settings by always using the system font, sizes, and colors.

Developers: From code, you can determine the system font properties (including its size) using the GetThemeFont API function. You can determine the system colors using the GetThemeSysColor API function.

Because you can't make any assumptions about users' system theme settings, you should:

- Always base your font colors and backgrounds off system theme colors. Never make your own colors based on fixed RGB (red, green, blue) values.
- Always match system text colors with their corresponding background colors. For example, if you choose COLOR_STATICTEXT for the text color, you must also choose COLOR_STATIC for the background color.
- Always create new fonts based on proportional-sized variations of the system font. Given the system font metrics, you can create bold, italic, larger, and smaller variations.

A simple way to ensure that your program respects users' settings is to test using a different font size and a high contrast color scheme. All text should resize and display correctly in the chosen color scheme.

Usage patterns

Text has several usage patterns:

Expand table

Usage	Description
Title bar text Text on the title bar that identifies the window.	Title bar text
Main instructions Text that explains what to do on a page, window, or dialog box.	Main instructions

Usage	Description
Secondary instructions Supplemental text that explains what to do on a page, window, or dialog box.	Secondary instructions
Normal text Ordinary (read-only) text displayed in a user interface.	Normal text
Emphasized text Bold text is used to make the text easier to parse and to draw attention to text users must read. italic text is used to refer to text literally (instead of quotation marks) and to emphasize specific words.	**Emphasized text**
Editable text Text that users can edit is shown in a box. to reduce the weight of the presentation, the box may be displayed only when the editable text is selected.	Editable text
Disabled text Text that doesn't apply to the current context, such as labels for disabled controls. disabled text indicates that users (normally) shouldn't bother reading the text.	Disabled text
Links Text used to navigate to another page, window, or help topic, or initiate a command.	Link Links (Hover)
Group header Text used to group items in a list view.	Group header
File name File name text (in content view only).	File name (in content view)
Document text Text used in documents (as opposed to ui text).	Document text
Document headings Text used as a heading within a document.	Document headings

Guidelines

Fonts and colors

- **The following fonts and colors are defaults for Windows Vista and Windows 7.**

Expand table

Pattern	Theme symbol	Font, Color
Title bar text	CaptionFont	9 pt. black (#000000) Segoe UI
Main instructions	MainInstruction	12 pt. blue (#003399) Segoe UI
Secondary instructions	Instruction	9 pt. black (#000000) Segoe UI
Normal text	BodyText	9 pt. black (#000000) Segoe UI
Emphasized text	BodyText	9 pt. black (#000000) Segoe UI, bold or italic
Editable text	BodyText	9 pt. black (#000000) Segoe UI, in a box
Disabled text	Disabled	9 pt. dark gray (#323232) Segoe UI
Link	HyperLinkText	9 pt. blue (#0066CC) Segoe UI
Links (Hover)	Hot	9 pt. light blue (#3399FF) Segoe UI
Group header		11 pt. blue (#003399) Segoe UI
File name (in content view)		11 pt. black (#000000) Segoe UI
Document text	(none)	9 pt. black (#000000) Calibri
Document headings	(none)	17 pt. black (#000000) Calibri

- **Choose fonts and optimize window layouts based on the UI technology and the target version of Windows:**

Expand table

UI technology	Target Windows version	Fonts to use and optimize for
Windows Presentation Foundation	All	Use WPF theme parts.
Win32 or	Windows Vista or	Use the appropriate Segoe UI font.

UI technology	Target Windows version	Fonts to use and optimize for
WinForms	later	
	Extensible components or pre-Windows Vista	To target Windows XP and Windows 2000, use the 8 point MS Shell Dlg 2 pseudo font, which maps to Tahoma. To target earlier versions of Windows, use 8 point MS Shell Dlg pseudo font, which maps to Tahoma on Windows 2000 and Windows XP, and to MS Sans Serif on Windows 95, Windows 98, Windows Millennium Edition, and Windows NT 4.0.

- **Developers:**
  - For elements that use fixed layout (such as Windows dialog templates and WinForms), hard code the appropriate font from the preceding table.
  - For elements that use dynamic layout (such as Windows Presentation Foundation), use the theme fonts. Use theme APIs like DrawThemeText to draw text based on the theme symbol. Be sure to have an alternative based on system metrics in case the theme service isn't running.
- **For Segoe UI, use a 9 point font size or larger.** The Segoe UI font is optimized for these sizes, so avoid using smaller sizes.
- **Always match system text colors with their corresponding background colors.** For example, if you choose COLOR_STATICTEXT for the text color, you must also choose COLOR_STATIC for the background color.
- **Always create new fonts based on proportional-sized variations of the system font.** Given the system font metrics, you can create bold, italic, larger, and smaller variations.
- Display large blocks of read-only text (such as license terms) against a light background instead of a gray background. Gray backgrounds suggest that the text is disabled and discourages reading.
- **Consider a maximum line length of 65 characters** to make the text easy to read. (Characters include letters, punctuation, and spaces.)

Attributes

- Most UI text should be plain without any attributes. Attributes may be used as follows:
  - **Bold.** Use in control labels to make the text easier to parse. Use sparingly to draw attention to text users must read. Using too much bold lessens its impact.
  - **Italic.** Use to refer to text literally instead of quotation marks. Use sparingly to emphasize specific words. Use for prompts in text boxes and editable drop-

down lists.
    - **Bold italic.** Don't use.
    - **Underline.** Don't use except for links. Use italic instead for emphasis.
- Not all fonts support bold and italic, so they should never be crucial to understanding the text.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

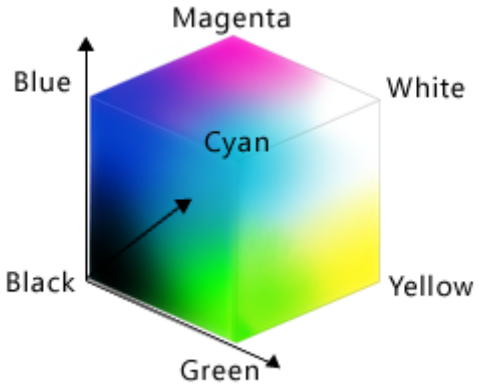
Color

Article • 04/26/2022

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

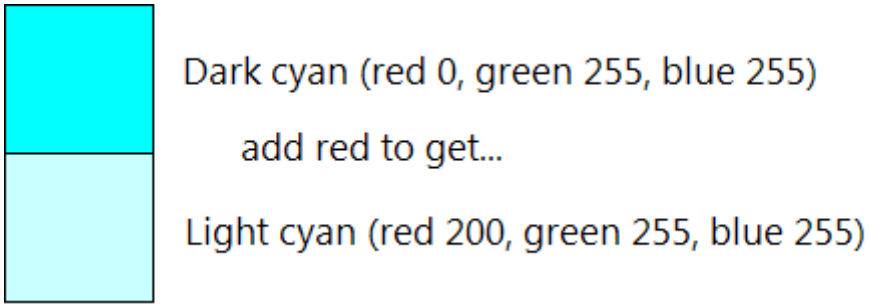
Color is an important visual element of most user interfaces. Beyond pure aesthetics, color has associated meanings and elicits emotional responses. To prevent confusion in meaning, color must be used consistently. To obtain the desired emotional responses, color must be used appropriately.

Color is often thought of in terms of a color space, where RGB (red, green, blue), HSL (hue, saturation, luminosity), and HSV (hue, saturation, value) are the most commonly used color spaces.



The RGB color space can be visualized as a cube.

While display technology uses RGB values and consequently developers often think of colors in terms of RGB, the RGB color space doesn't correspond to how people perceive color. For example, if you add red to dark cyan, the result isn't perceived as more red but as lighter cyan.



In this example, adding red to dark cyan makes it lighter, not more red. The RGB color space doesn't correspond to how people perceive color.

The HSL/HSV color spaces consist of three components: hue, saturation, and luminosity or value. These color spaces are often used instead of RGB because they better match how people perceive color.

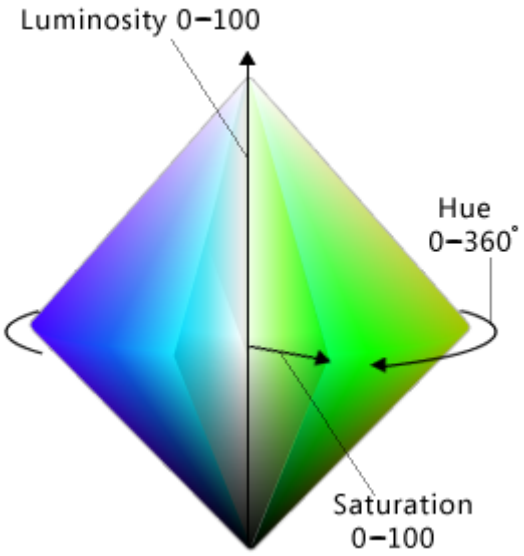
The HSL color space forms a double cone that is white on the top, black on the bottom, and neutral in the middle:

- **Hue:** The basic color in the color wheel, ranging from 0 to 360 degrees where both 0 and 360 degrees are red.

  The color wheel, where red is 0 degrees, yellow is 60 degrees, green is 120 degrees, cyan is 180 degrees, blue is 240 degrees, and magenta is 300 degrees.

- **Saturation:** How pure (vs. dull) the color is, ranging from 0 to 100, where 100 is fully saturated and 0 is gray.

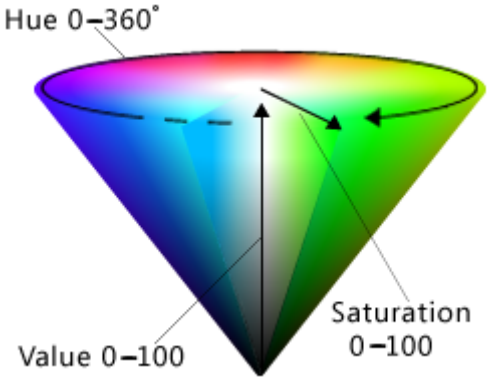
- **Luminosity:** How light the color is, ranging from 0 to 100, where 100 is as light as possible (white, regardless of the hue and saturation) and 0 is as dark as possible (black).



The HSL color space can be visualized as a double cone.

The HSV color space is similar, except that its space forms a single cone:

- **Hue:** The basic color in the color wheel, ranging from 0 to 360 degrees where both 0 and 360 degrees are red.
- **Saturation:** How pure (vs. dull) the color is, ranging from 0 to 100, where 100 is fully saturated and 0 is gray.
- **Value:** How bright the color is, ranging from 0 to 100, where 100 is as bright as possible (which is half luminosity in the HSL space) and 0 is as dark as possible (black).



The HSV color space can be visualized as a single cone.

In both HSL and HSV spaces, if saturation is 0 then luminosity specifies a shade of gray. In Windows, the HSL and HSV spaces are usually remapped to a scale between 0 to 240 so that colors can be represented with a 32-bit value.

Note: Guidelines related to fonts and accessibility are presented in separate articles.

Design concepts

Effective use of color can make your program's user interface (UI) more effective. Color can help users understand certain meanings at a glance. Color can also make your product appear more aesthetically pleasing and refined.

Unfortunately, it's all too easy to use color ineffectively, especially if you are not trained in visual design. Poor use of color results in designs that look unprofessional, dated, confusing, or just plain ugly. A poor use of color can be worse than not using color at all.

This section explains what you need to know to use color effectively.

How color is used

Color is typically used in UI to communicate:

- **Meaning.** The meaning of a message can be summarized through color. For example, color is often used to communicate status where red is a problem or error, yellow is caution or warning, and green is good.
- **State.** An object's state can be indicated through color. For example, Windows uses color to indicate selection and hover states. Links within Web pages use blue for unvisited and purple for visited.
- **Differentiation.** People assume that there is a relationship between items of the same color, so color coding is an effective way to differentiate between objects. For example, in a control panel item, task panes use a green background to visually separate them from the main content. Also, Microsoft Outlook allows users to assign different colored flags to messages.
- **Emphasis.** Color can be used to draw users' attention. For example, Windows uses blue main instructions to help them stand out from the other text.

Of course, color is often used in graphics for purely aesthetic reasons. While aesthetics are important, you should choose the colors of UI elements primarily based on what they mean, not how they look.

Color interpretation

Users' interpretation of color is often culturally dependent. For example, in the United States, wedding attire for the bride is largely associated with the color white, while black is associated with funerals. However, long ago in Japan the color symbolism was just the opposite: white was the predominant color at funerals, and black was considered a color that brings good luck for weddings.

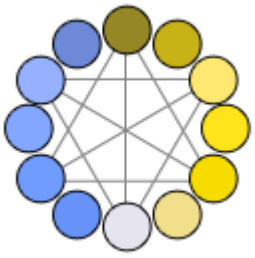
That said, **the interpretation of red, yellow, and green for status is consistent globally.** This is due to the UNESCO Vienna Convention on Road Signs and Signals, which defines the worldwide convention for traffic lights (where red means stop, green means proceed, and yellow means proceed with caution). You can use these status colors without concern for culturally dependent interpretations.

Beyond the status colors, Windows assigns meanings to colors based on convention, as presented in the Guidelines section of this article. Be sure that your program's color usage is compatible with these color conventions.

Color accessibility

Use of color affects the accessibility of your software to the widest possible audience. Users with blindness or low vision may not be able to see the colors well, if at all. Approximately 8 percent of adult males have some form of color confusion (often incorrectly referred to as "color blindness"), of which red-green color confusion is the most common.

The primary colors as seen with normal color vision.



The primary colors as seen with Protanopia (1% of male population).

The primary colors as seen with Deuteranopia (6% of male population).

The primary colors as seen with Tritanopia (1% of male population).

For more information, see Can Color-Blind Users See Your Site?

Use color to reinforce visually

The best solution to the color interpretation and accessibility problems is to use color to visually reinforce the meaning of one of these primary methods of communication:

- **Text.** Concise text is usually the most effective primary communication either directly on the UI or through a tooltip.

In this example, tooltip text is used to communicate an icon's meaning.

- **Design.** Icons are easily distinguished by the designs, especially their outline shape.

In this example, the standard icons are readily distinguishable based on their designs.

- **Location.** Relative location can also be used, but this approach is weaker than the alternatives. To be effective, the location should be standard and well known, as with traffic lights.

While color is the most obvious attribute of many designs, it must always be redundant.

Designing with color

Ironically, the best way to design for color is to start by designing without color, using either wireframes or monochrome, and then add color later. Doing so helps ensure that information isn't being communicated using color alone. It also helps ensure that your printouts look great on monochrome printers.

Use theme or system colors

While there are many complex factors in using color effectively, in Windows UI choosing color often boils down to simply choosing the appropriate theme color or system color according to a few simple rules. Users can then select and customize these color schemes as they choose.

By doing so, not only do you accommodate the color preferences of all your users, but you eliminate the burden of choosing the one perfect color scheme that works for all tastes, styles, and cultures (which, of course, is otherwise impossible).

If you do only one thing...

Choose colors by selecting the appropriate theme color or system color. Never use color as a primary method of communication, but as a secondary method to reinforce meaning visually. Design using wireframes or monochrome to help ensure that color is secondary.

Use theme or system colors correctly

Assume that users choose theme or system colors based on their personal needs and that the theme or system colors are constructed appropriately. Based on this assumption, if you always choose theme or system colors based on their intended purpose and pair foregrounds with their associated backgrounds, the colors are guaranteed to be legible and respect users' wishes in all video modes, including high-contrast mode. For example, the window text system color is guaranteed to be legible against the window background system color.

Specifically, always:

- **Choose colors based on their purpose.** Don't choose colors based on their current appearance because that appearance can be changed by the user or future versions of Windows.
- **Match foreground colors with their associated background colors.** Foreground colors are guaranteed to be legible only against their associated background colors. Don't mix and match foreground colors with other background colors, or worse yet, other foreground colors.
- **Don't mix color types.** That is, always match theme colors with their associated theme colors, system colors with their associated system colors, and hardwired colors with other hardwired colors. For example, a theme text color isn't guaranteed to be legible against a hardwired background.
- **If you must hardwire colors, handle high-contrast mode as a special case.**

If you do only one thing...

Always choose theme or system colors based on their intended purpose, and pair foregrounds with their associated backgrounds.

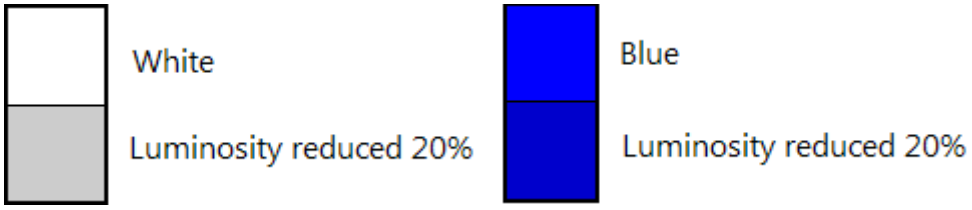
Using other colors

While the Windows theme defines a comprehensive set of theme parts, you may find that your program needs colors that aren't defined in the theme file. While you could hardwire such colors, a better approach is to derive colors from the theme or system colors. Strategically using this approach gives you all the benefits of using theme and system colors, but with much more flexibility.

For example, suppose you need a window background that is darker than the theme window background color. In the HSL color space, having a darker color means a color with a lower luminosity. Thus, you can derive a darker window background color using the following steps:

- Obtain the window background theme color RGB.
- Convert the RGB to its HSL value.
- Reduce the luminosity value (by, say, 20 percent).
- Convert back to RGB values.

Using this approach, the derived color is guaranteed to be perceived as a darker shade of the original color (unless the original color was very dark to begin with.)



In this example, a darker window background color is derived from the theme color.

Testing colors

To determine if your program's use of color is accessible and not used as a primary method of communication, we recommend using the Fujitsu ColorDoctor or the Vischeck utilities to check for:

- Overall dependency on color using the Gray scale filter.
- Specific color confusion problems using the Protanopia, Deuteranopia, and Tritanopia filters.

To determine if your program's use of color is programmed correctly, test your program in the following modes:

- Theming enabled using the default Windows theme.
- Theming enabled using a non-default theme.
- Theming disabled ("Windows Classic style" in the Theme Settings in the Personalization Control Panel item).
- High Contrast Black theme (white text on a black background, known as Aquatic in Windows 11).
- High Contrast White theme (black text on a white background, known as Desert on Windows 11).

All the screen elements should be legible and appear as expected, even immediately after mode changes.

Guidelines

General

- **Never use color as a primary method of communication,** but as a secondary method to reinforce meaning visually.

Using theme and system colors

- Whenever possible, **choose colors by selecting the appropriate theme color or system color.** By doing so, you can always respect users' color preference.
- **Choose theme and system colors based on their purpose.** Don't choose colors based on their current appearance, as that appearance can be changed by the user or future versions of Windows.
- **Match foreground colors with their associated background colors.** Foreground colors are guaranteed to be legible only against their associated background colors. Don't mix and match foreground colors with other background colors, or worse yet, other foreground colors.
- **Don't mix color types.** That is, always match theme colors with their associated theme colors, system colors with their associated system colors, and hardwired colors with other hardwired colors. For example, a theme text color isn't guaranteed to be legible against a hardwired background.
- **If you must use a color that isn't a theme or system color:**
  - **Prefer to derive the color from a theme or system color over hardwiring its value.** Use the process described earlier in this article, in Using other colors.

    - **Handle high-contrast mode as a special case.**
- **Handle theme changes.** Theme changes are handled automatically by windows with standard window frames and common controls. Windows with custom window frames, custom or owner-draw controls, and other use of color must handle theme changes explicitly.
    - **Developers:** You can respond to theme change events by handling the WM_THEMECHANGED message.

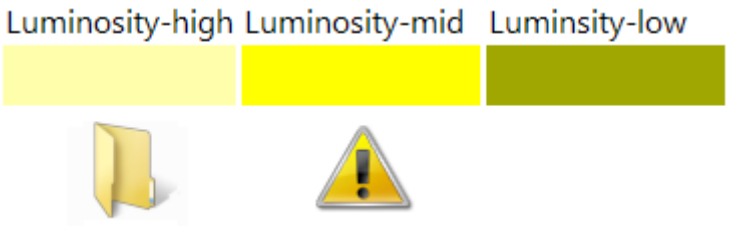
Color meaning

- While you should use theme and system colors (or derived colors) whenever you can, make sure any other use of color is compatible with the following use of color within Windows.

Expand table

Hue	Meaning	Use in Windows
blue/green	Windows brand	Background: Windows branding.
glass, black, gray, white	neutral	Background: standard window frames, Start menu, taskbar, Sidebar. Foreground: normal text.
blue	start, commit	Background: default command buttons, search, log on. Icons: information, Help. Foreground: main instructions, links.
red	error, stop, vulnerable, critical, immediate attention, restricted	Background: status, stopped progress (progress bars). Icons: error, stop, close window, delete, required input, missing, unavailable.
yellow	warning, caution, questionable	Background: status, paused progress (progress bars). Icons: warning
green	go, proceed, progress, safe	Background: status, normal progress (progress bars). Icons: go, done, refresh. Foreground: Paths and URLs (in search results).
purple	visited	Foreground: visited links (for links within Windows Internet Explorer and documents).

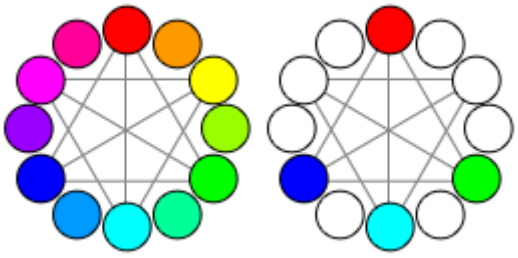
- **To avoid communicating the previous meanings, choose colors have high mid to low saturation and high or low luminosity.** Users associate the previous meanings to colors that have full or high saturation and mid-level luminosity, so you can avoid these associations by choosing different shades.



In this example, there are three different shades of yellow, but only the highly saturated, mid-level luminosity shade communicates warning. The yellow folder icon doesn't feel like a warning.

Using color with data

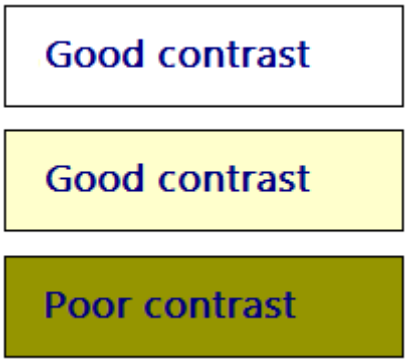
- When helpful, **assign color to data to help users differentiate it.** Note that users will assume that data with similar colors have similar meanings.
- **Assign colors by default that are easy to distinguish.** Generally, colors are easy to distinguish if they are far apart from each other in the HSL/HSV color spaces, while maintaining high contrast with their background:
  - When choosing colors, prefer triad harmonies or complementary hues, but not adjacent hues.

    In this example, if the first color assignment is red, the next color should be blue, green, or cyan, but not magenta, purple, orange, or yellow.
  - Colors have high contrast if there is a large difference in their hue, saturation, or luminosity.

In this example, the light blue base color contrasts with backgrounds with large differences in hue, saturation, or luminosity.

- Using a white or very light background makes contrasting foreground colors easier to distinguish.



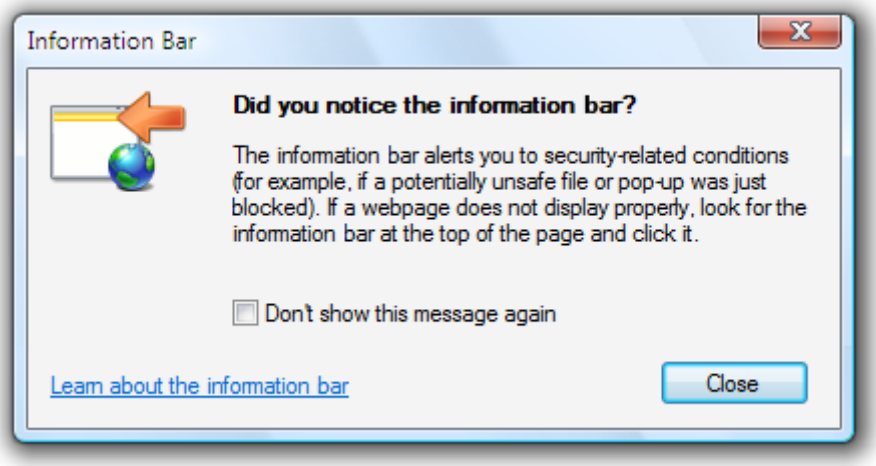
In this example, white and light background colors make the foreground color easier to distinguish.

- **Allow users to customize these color assignments** because color choice is subjective and a personal preference. If there are many coordinated colors, allow users to change them as a group using color schemes.
- **Allow users to label these color assignments.** Doing so helps make them easier to identify and find.
- **Unlike UI colors, data should not change when the system colors change.**

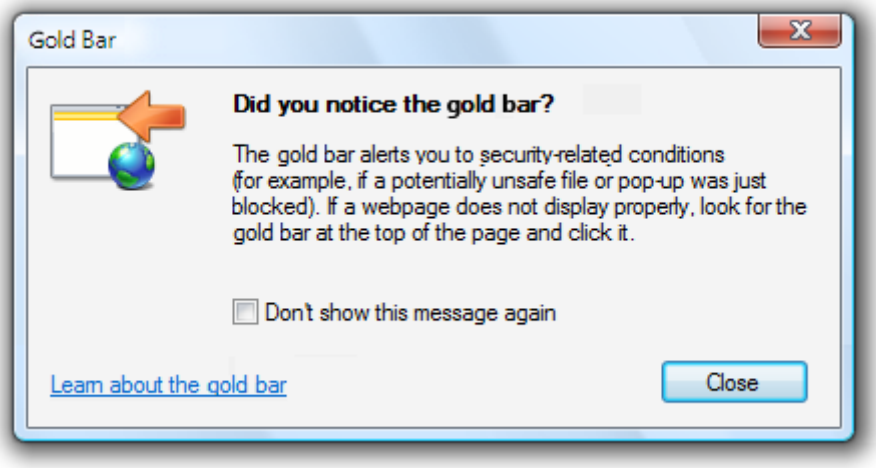
Documentation

- **Refer to UI elements by their names, not by their colors.** Such references aren't accessible and the system colors may change. If a UI element's name isn't well known or not descriptive enough, show a screenshot to clarify.

Correct:



Incorrect:



In the incorrect example, the message refers to the Windows Internet Explorer information bar by its color instead of its name.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Icons (Design basics)

Article • 02/09/2021

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

Icons are pictorial representations of objects, important not only for aesthetic reasons as part of the visual identity of a program, but also for utilitarian reasons as shorthand for conveying meaning that users perceive almost instantaneously. Windows Vista introduces a new style of iconography that brings a higher level of detail and sophistication to Windows.

Note: Guidelines related to standard icons are presented in a separate article.

Design concepts

Aero is the name for the user experience of Windows Vista, representing both the values embodied in the design of the aesthetics, as well as the vision behind the user interface (UI). Aero stands for: authentic, energetic, reflective, and open. Aero aims to establish a design that is both professional and beautiful. The Aero aesthetic creates a high quality and elegant experience that facilitates user productivity and even drives an emotional response.

Windows Vista icons differ from Windows XP-style icons in the following ways:

- The style is more realistic than illustrative, but not quite photorealistic. Icons are symbolic images they should look better than photorealistic!
- Icons have a maximum size of 256x256 pixels, making them suitable for high-dpi (dots per inch) displays. These high-resolution icons allow for high visual quality in list views with large icons.
- Wherever practical, fixed document icons are replaced by thumbnails of the content, making documents easier to identify and find.
- Toolbar icons have less detail and no perspective, to optimize for smaller sizes and visual distinctiveness.

Well-designed icons:

- Improve the visual communication of your program.

- Strongly impact users' overall impression of your program's visual design, and appreciation for its fit-and-finish.
- Improve usability by making programs, objects, and actions easier to identify, learn, and find.

The following images depict what makes the Aero style of iconography in Windows Vista different from that used in Windows XP.

The Windows Vista icons (the lock and key on the left) are authentic, crisp, and detailed. They are rendered rather than drawn, but are not completely photorealistic.

The Windows Vista icons (the two on the left) are professional and beautiful, with attention to details that improve icon production quality.

These Windows Vista icons show optical balance and perceived accuracy in perspective and details. This allows them to look great big or small, up-close or from a distance. Moreover, this style of iconography works for high-resolution screens.

These examples show different types of icons, including a three-dimensional object in perspective, a front-facing (flat) icon, and a toolbar icon.

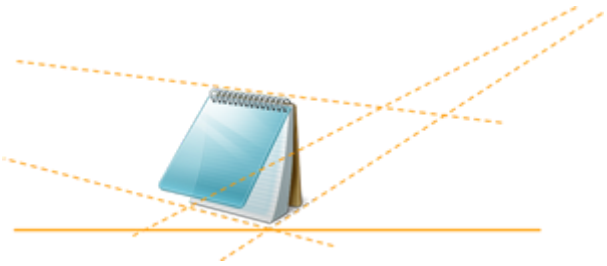
Guidelines

Perspective

- **Icons in Windows Vista are either three-dimensional and shown in perspective as solid objects, or two-dimensional objects shown straight-on.** Use flat icons for files and for objects that are actually flat, like documents or pieces of paper.

  Typical 3D and flat icons.

- **Three-dimensional objects are represented in perspective as solid objects, seen from a low birds-eye view with two vanishing points.**

  This example shows perspective and vanishing points typical of 3D icons.

- **In the smaller sizes, the same icon may change from perspective to straight-on.** At the size of 16x16 pixels and smaller, render icons straight-on (front-facing). For larger icons, use perspective.

- **Exception:** Toolbar icons are always front-facing, even in larger sizes.

This example shows how the same icon is treated differently, depending on size.

Light source

- **The light source for objects within the perspective grid is above, slightly in front of, and slightly to the left of the object.**
- **The light source casts shadows that are slightly to the rear and right of the object's base.**
- **All light rays are parallel, and strike the object along the same angle (like the sun).** The goal is to have a uniform lighting appearance across all icons and spotlight effects. Parallel light rays produce shadows that all have the same length and density, providing further unity across multiple icons.

Shadows

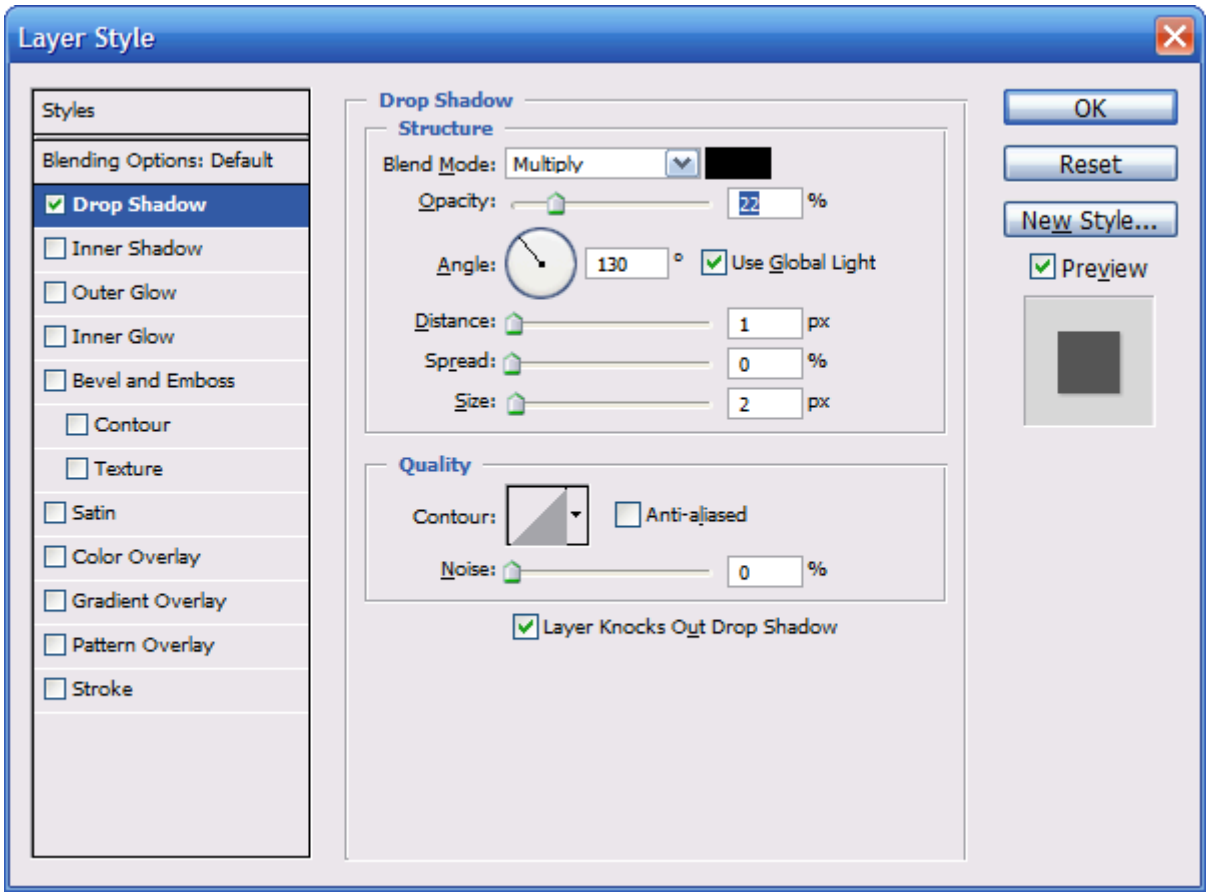
General

- **Use shadows to lift objects visually from the background, and to make 3D objects appear grounded, rather than awkwardly floating in space.**
- **Use an opacity range of 30-50 percent for shadows.** Sometimes a different level of shadow should be used, depending on the shape or color of an icon.
- **Feather or shorten the shadow if necessary, to keep it from being cropped by the icon box size.**
- **Don't use shadows in icons at 24x24 or smaller sizes.**

Typical icon shadows.

Flat icons

- **Flat icons are generally used for file icons and flat real-world objects,** such as a document or a piece of paper.
- **Flat icon lighting comes from the upper-left at 130 degrees.**
- **Smaller icons (for example, 16x16 and 32x32) are simplified for readability.** However, if they contain a reflection within the icon (often simplified), they may have a tight drop shadow. The drop shadow ranges in opacity from 30-50 percent.
- **Layer effects can be used for flat icons, but should be compared with other flat icons.** The shadows for objects will vary somewhat, according to what looks best and is most consistent within the size set and with the other icons in Windows Vista. On some occasions, it may even be necessary to modify the shadows. This will especially be true when objects are laid over others.
- **A subtle range of colors may be used to achieve desired outcome.** Shadows help objects sit in space. Color impacts the perceived weight of the shadow, and may distort the image if it is too heavy.



The Drop Shadow option in the Layer Style dialog box, and a typical shadow for a flat icon.

Basic flat icon shadow ranges

Expand table

Characteristic	Range
Color	Black

Characteristic	Range
Blend mode	Multiply
Opacity	22-50 percent, depending on color of the item
Angle	120-130 (use global light)
Distance	3 for 256x256, ranging down to 1 for 32x32
Spread	0
Size	7 for 256x256, ranging down to 2 for 32x32

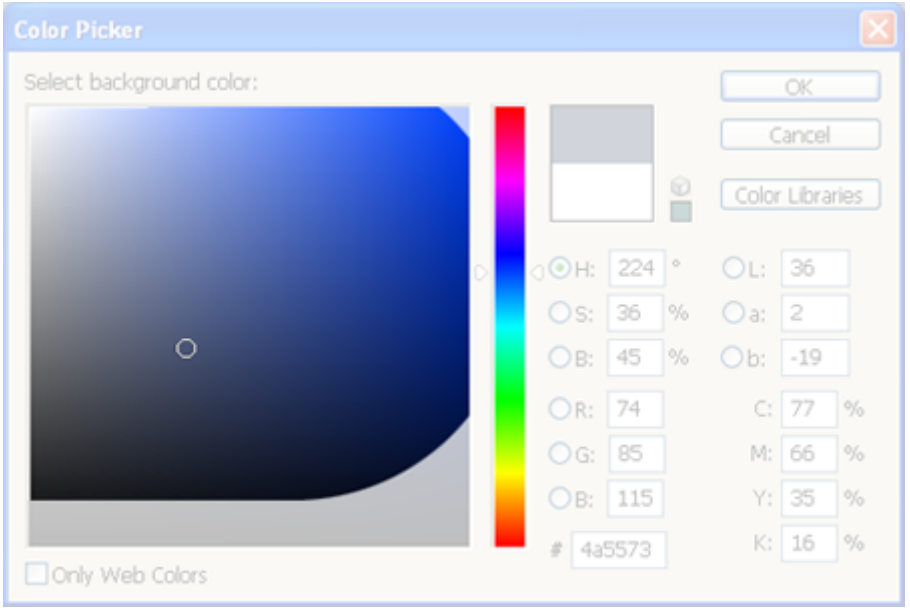
Three-dimensional icons

- Create shadows for 3D icons on a case-by-case basis, with an effort to fit within a range of cast distance and feathering to fully transparent. Create the images in a size a bit smaller than the overall icon size demands to allow space for a drop shadow (for those sizes that will require one). Make sure the shadow doesn't end abruptly at the edge of the icon.

These examples help demonstrate variations created based on the shape and position of the object itself. The shadow sometimes needs to be feathered or shortened to keep it from being cropped by the icon box size.

Color and saturation

- **Colors are generally less saturated than they were Windows XP.**
- **Use gradients to create a more realistic looking image.**
- **Although there is no specific color palette for standard icons, remember that they need to work well together in many contexts and themes.** Prefer the standard set of colors; don't re-color standard icons, such as warning icons, because this disrupts users' ability to interpret meaning. For more guidelines, see Color.
- **Icon files require 8-bit and 4-bit palette versions as well,** to support the default setting in a remote desktop. These files can be created through a batch process, but they should be reviewed, as some will require retouching for better readability.



There is no strict color palette restriction. Only full-saturation (top right) is avoided.

- Bit levels: ICO design for 32-bit (alpha included) + 8-bit + 4-bit (dithered down automatically pixel poke only most critical). Only a 32-bit copy of the 256x256 pixel image should be included, and only the 256x256 pixel image should be compressed to keep the file size down. Several icon tools offer compression for Windows Vista.

- Bit levels: Toolbars 24-bit + alpha (1 bit mask), 8-bit and 4-bit.

- Toolbars or AVI files: Use magenta (R255 G0 B255) as the background transparency color.

Size requirements

General

- **Pay special attention to high visibility icons,** such as main application icons, file icons that can appear in Windows Explorer, and icons appearing in the Start Menu or on the desktop.
  - **Application icons and Control Panel items:** The full set includes 16x16, 32x32, 48x48, and 256x256 (code scales between 32 and 256). The .ico file format is required. For Classic Mode, the full set is 16x16, 24x24, 32x32, 48x48 and 64x64.
  - **List item icon options:** Use live thumbnails or file icons of the file type (for example, .doc); full set.
  - **Toolbar icons:** 16x16, 24x24, 32x32. Note that toolbar icons are always flat, not 3D, even at the 32x32 size.
  - **Dialog and wizard icons:** 32x32 and 48x48.
  - **Overlays:** Core shell code (for example, a shortcut) 10x10 (for 16x16), 16x16 (for 32x32), 24x24 (for 48x48), 128x128 (for 256x256). Note that some of these are slightly smaller but are close to this size, depending on shape and optical balance.
  - **Quick Launch area:** Icons will scale down from 48x48 in Alt+Tab dynamic overlays, but for a more crisp version, add a 40x40 to .ico file.
  - **Balloon icons:** 32x32 and 40x40.
  - **Additional sizes:** These are useful to have on hand as resources to make other files (for example, annotations, toolbar strips, overlays, high dpi, and special cases): 128x128, 96x96, 64x64, 40x40, 24x24, 22x22, 14x14, 10x10, and 8x8. You can use .ico, .png, .bmp, or other file formats, depending on code in that area.

For high dpi

- Windows Vista targets 96 dpi and 120 dpi.

The following tables show examples of scaling ratios applied to two common icon sizes. Note that not all of these sizes must be included in the .ico file. The code will scale larger ones down.

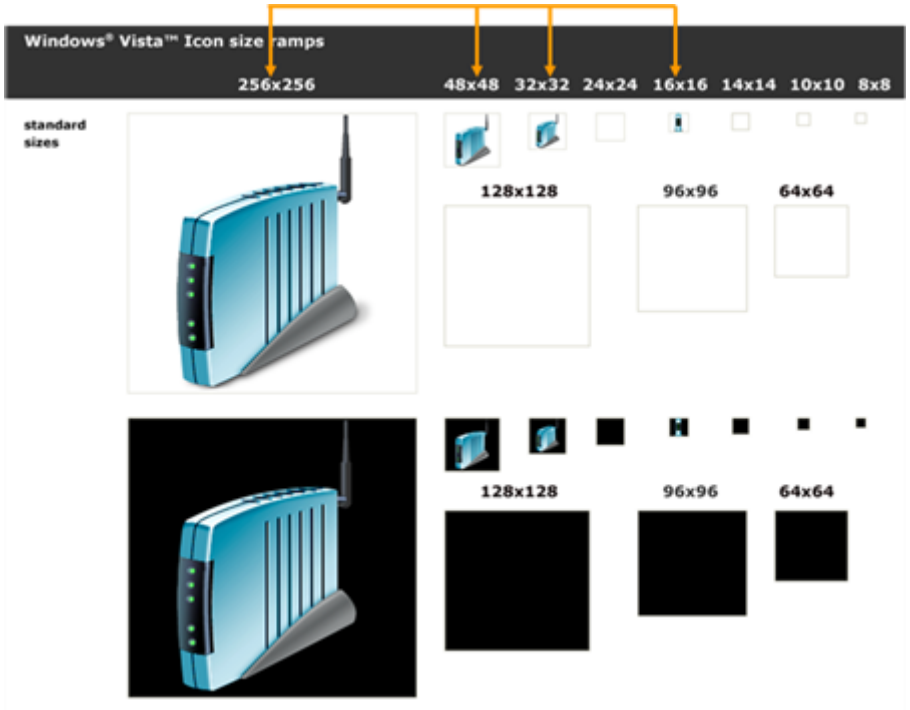
Expand table

dpi	Icon size	Scale factor
96	16x16	1.0 (100%)
120	20x20	1.25 (125%)
144	24x24	1.5 (150%)
192	32x32	2.0 (200%)

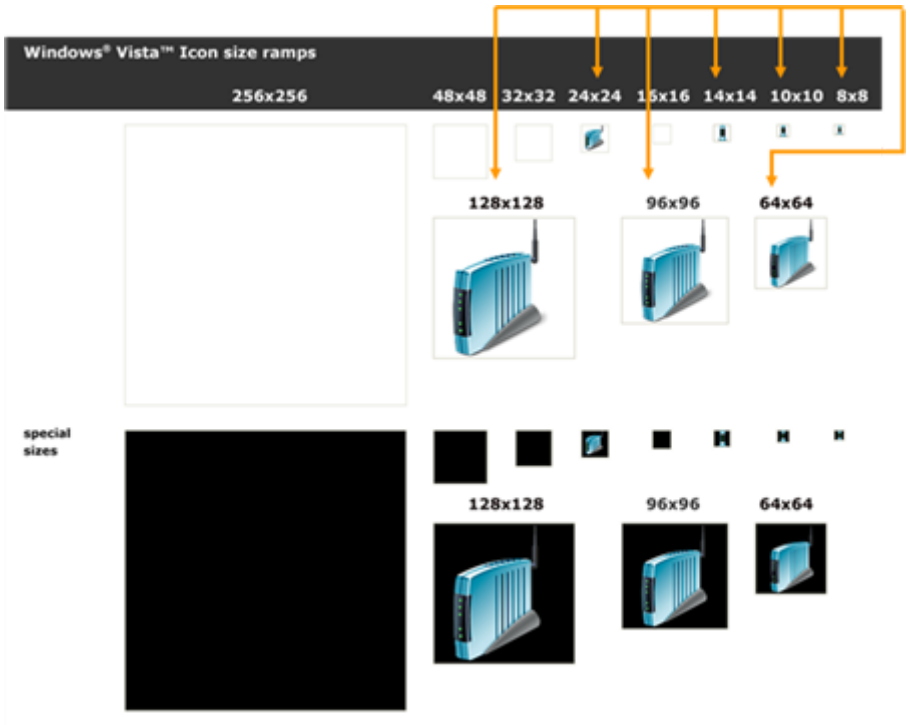
Expand table

dpi	Icon size	Scale factor
96	32x32	1.0 (100%)
120	40x40	1.25 (125%)
144	48x48	1.5 (150%)
192	64x64	2.0 (200%)

.ico file sizes (standard)



.ico file sizes (special cases)

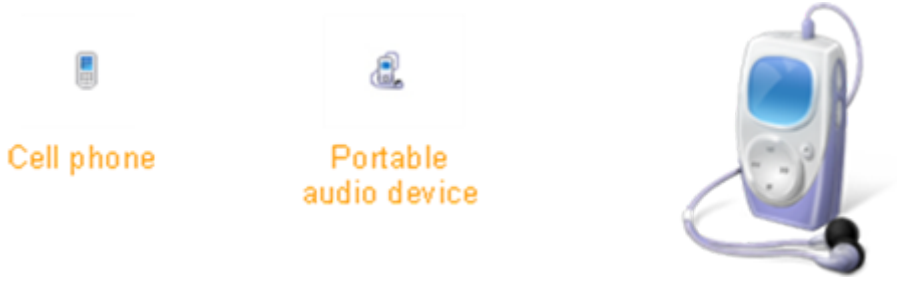


Annotations and overlays

- Annotations go in bottom-right corner of icon, and should fill 25 percent of icon area.
  - **Exception:** 16x16 icons take 10x10 annotations.
- Don't use more than one annotation over an icon.
- Overlays go in bottom-left corner of icon, and should fill 25 percent of icon area.
  - **Exception:** 16x16 icons take 10x10 overlays.

Level of detail

- 16x16 size of many of these icons is still widely used and therefore important.
- The details in an icon of this size must clearly show the key point of the icon.
- As an icon gets smaller, transparency and some special details found in larger sizes should be sacrificed in order to simplify and get the point across.
- Attributes and colors should be exaggerated and used to emphasize the key forms.



At 16x16, the icon for the portable audio device could easily be mistaken for a cell phone so the ear piece is a key visual detail to show.

- Simply scaling down from the 256x256 size does not work.
- All sizes need relevant level of detail; the smaller the icon the more you need to exaggerate the defining details.

Icon development

Designing and producing icons

- **Hire an experienced graphic designer.** For great graphics, images, and icons work with experts. Experience in illustrations using vector art or 3D programs is recommended.
- **Plan to do series of iterations,** from initial concept sketches, to in-context mock-ups, to final production review and fit-and-finish of icons in the working product.
- **Think ahead icon creation can be expensive.** Gather all existing details and requirements, such as: the complete set of icons needed; the main function and meaning for each; families or clusters in the set you want to be apparent; brand requirements; the exact file names; image formats used in your code; and size requirements. Ensure up front that you can make the most of your time with the designer.
- **Remember that the designer may not be familiar with your product, so provide functional information, screen shots, and spec sections, as appropriate.**
- **Plan for geopolitical and legal reviews as appropriate.**
- **Map out a timeframe and have regular communication.**

From concept sketch to end-product

- Create concept sketches.
- Try out the concept in different sizes.
- Render in 3D if necessary.
- Test sizes on different background colors.
- Evaluate icons in the context of the real UI.
- Produce final .ico file or other graphic resource formats.

Tools

- **Pencil and paper:** Initial concept ideas, listed and sketched.
- **3D Studio Max:** Render 3D objects in perspective.
- **Adobe Photoshop:** Sketch and iterate, mock-up in context, and finalize details.
- **Adobe Illustrator/ Macromedia Freehand:** Sketch and iterate, finalize details.
- **Gamani Gif Movie Gear:** Produce .ico file (with compression if needed).
- **Axialis Icon Workshop:** Produce .ico file (with compression if needed).
- Microsoft Visual Studio doesn't support Windows Vista icons (there is no support for alpha channel or more than 256 colors).

Production

> **Tip**
>
> Follow these steps to create a single .ico file that contains multiple image sizes and color depths.

Step 1: Conceptualize

- Use established concepts where possible, to ensure consistency of meanings for the icon and its relevance to other uses.
- Consider how the icon will appear in the context of the UI, and how it might work as part of a set of icons.
- If revising an existing icon, consider whether complexity can be reduced.
- Consider the cultural impact of your graphics. Avoid using letters, words, hands, or faces in icons. Depict representations of people or users as generically as possible, if needed.

- If combining multiple objects into a single image in an icon, consider how the image will scale to smaller sizes. Use no more than three objects in an icon (two is preferred). For the 16x16 size, consider removing objects or simplifying the image to improve recognition.
- Do not use the Windows flag in icons.

Step 2: Illustrate

- To illustrate Windows Aero style icons, use a vector tool such as Macromedia Freehand or Adobe Illustrator. Use the palette and style characteristics as outlined earlier in this article.
- Illustrate image using Freehand or Illustrator. Copy and paste the vector images into Adobe Photoshop.
- Make and use a template layer in Photoshop to make sure that work is done within square regions of the regulated sizes.
- Create the images in a size a bit smaller than the overall icon size demands to allow space for a drop shadow (for those sizes that require one).
- Place images at the bottom of the squares, so that all icons in a directory are positioned consistently. Avoid cutting off shadows.
- If you are adding another object to an image or a series, keep the main object in a fixed position, and place flat smaller sized images in a fixed position, such as the lower-left or upper-right depending on the case.

Step 3: Create the 24-bit images

- Once you've pasted sizes in Photoshop, check the readability of images, especially at 16x16 and smaller sizes. Pixel-poking using percentages of colors may be required. Reduction of transparency may also be needed. It is common to exaggerate aspects at smaller sizes and to eliminate aspects as well, in order to focus on the key point.
- The 8-bit icons will be displayed in any color mode lower than 32-bit and will not have the 8-bit alpha channel, so they may need to have their edges or more cleaned up because there's no anti-aliasing (edges may be jagged and image may be hard to read).
- In Photoshop, duplicate the 24-bit image layer and rename the layer to 4-bit images. Index 4-bit images to the Windows 16 color palette.
- Clean up images using only the colors from the 16 color palette. Outlines made from darker or lighter versions of the object's colors are usually preferable to grey or black.
- If working on a bitmap, be sure that the background color isn't used in the image itself, because that color that will be the transparent color. Magenta (R255 G0 B255) is often used as the background transparency color.

Step 4: Create the 8-bit and 4-bit images

- Now that the 24-bit images are ready to be made into 32-bit icons, 8-bit versions need to be created.
- This is a great time to test contextual screen shots. It's amazing what can be discovered by viewing other icons or a family of icons in context. This step can save time and money. It is much better to catch issues before files go through production and are handed off.
- Add the drop shadow to your images in sizes that require them.
- Merge the drop shadow and the 24-bit images together.
- Create a new Photoshop file for each size. Copy and paste the appropriate image. Save each file as a .psd file.
- Do not merge the image layer with the background layer. It's helpful to include the size and color depth in the file name while working, but the file may ultimately need to be renamed.

Step 5: Create the .ico file

- Choose the application that best meets needs and skills of artists. Remember that icons to be used in a shipping product must be created in a tool that has been purchased or licensed. This means that trial versions cannot be used.
- Both of the products listed below have been used by designers who have produced icons for Windows Vista, and each offers the ability to export to Adobe Photoshop CS.
  - Gamani Gif Movie Gear: Produce .ico file
  - Axialis Icon Workshop: Produce .ico file
- Visual Studio doesn't support Windows Vista icons (there is no support for alpha channel or more than 256 colors), so its use is not recommended.
- Icon (.ico format) files must contain the 4- and 8-bit versions, as well as the 24-bit + alpha.
- Save files as a "Windows icon (.ico)" no matter which icon creation program you choose to use.
- Some iconographic assets may actually be bitmap strips, which also require an alpha channel (for example, for toolbars), or .png files saved with transparency. Not all are necessarily .ico format; check for what format is supported in code.

Step 6: Evaluate

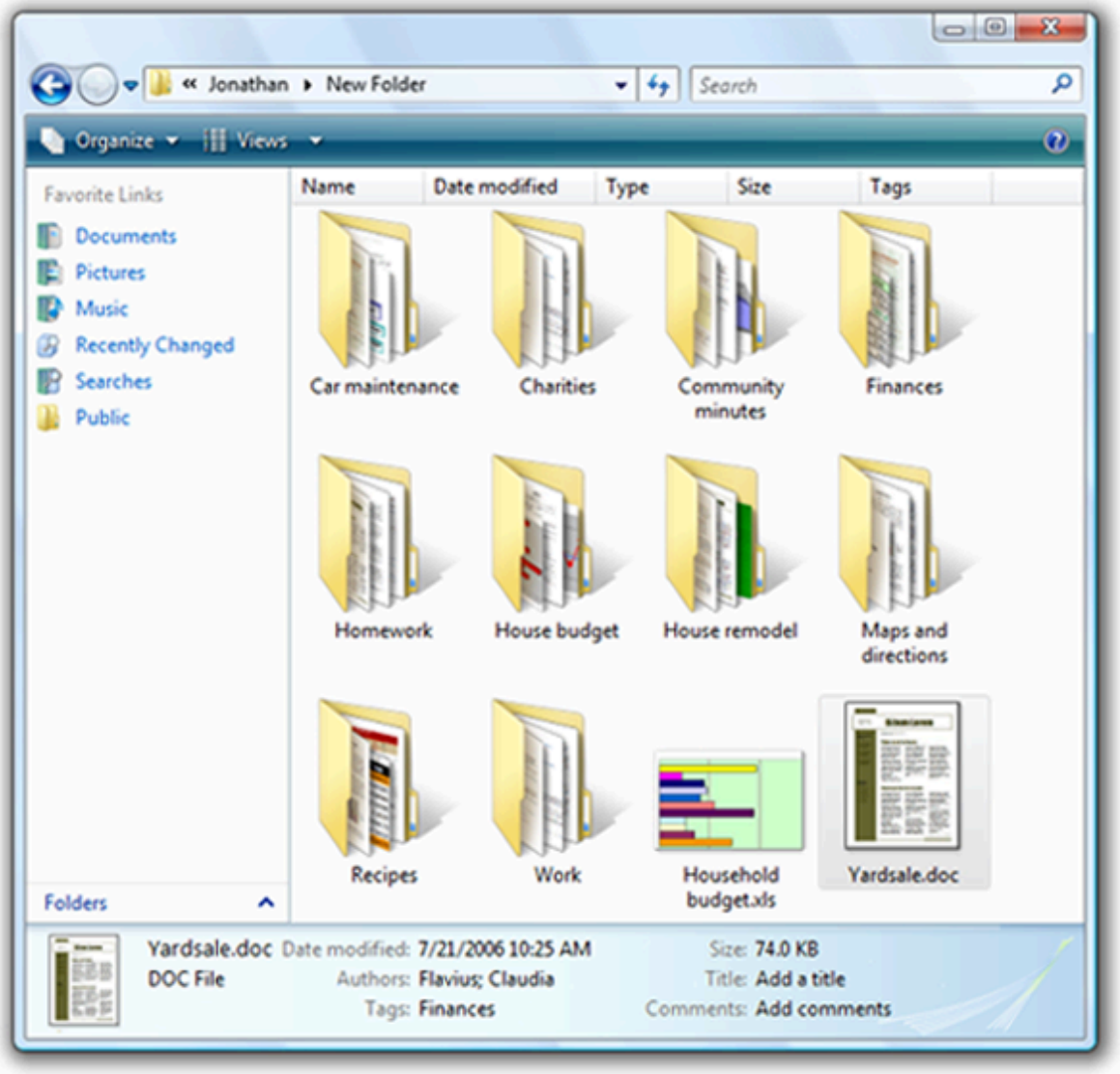
- Look at all sizes.
- Look at the family together to evaluate family resemblance, optical balance, and distinction.

- Look at in context to evaluate relative weights and visibility (make sure that one doesn't dominate).
- Consider cases that may not be used now, but could be in the near future. Could this icon ever be annotated or have an overlay?
- Look at in code.

Icons in the context of list views, toolbars, and tree views

List views

- For Windows Vista, use thumbnails for files holding content that is visually distinct at small scale, such that users can directly recognize the file they are looking for. (Use the Windows Thumbnailing application programming interface for this.)



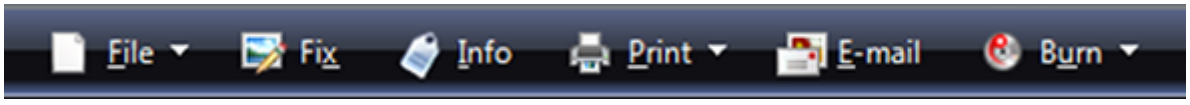
- Application icon overlays (not shown here) on thumbnails help association with the application for the file type, in addition to showing the file's preview.

Note: For files without visually distinct content, don't use thumbnails. Instead, use traditional symbolic file icons showing object representation and the associated

application or type.

Toolbars

- Icons that appear in a toolbar must have an optical balance in size, color, and complexity.
- Test potential icons in a contextual screen shot to avoid any undesired dominance or imbalances.
- Testing in screen shots easily helps avoid expensive iterations in code.
- Review the icons in code as well. Motion and other factors can impact the success of an icon; in some cases further iterations may be needed.

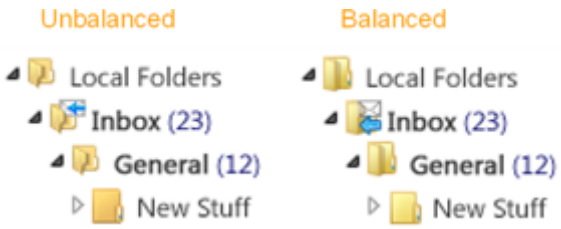


In the above example, the optical balance has not yet been achieved.

Try iterations in context.

Tree views

- Optical balance is needed to preserve the hierarchy in a tree view control.
- Therefore, icons that are typically used in this context should be evaluated there. Sometimes a particular 16x16 icon should be made smaller because its shape has an optical dominance over others.
- Compensation for optical imbalances is an important part of producing top quality icons.



Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Standard Icons

Article • 02/09/2021

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

Standard icons are the error, warning, information, and question mark icons that are part of Windows.

The standard error, warning, information, and question mark icons.

The standard icons have these meanings:

- **Error icon.** The user interface (UI) is presenting an error or problem that has occurred.
- **Warning icon.** The UI is presenting a condition that might cause a problem in the future.
- **Information icon.** The UI is presenting useful information.
- **Question mark icon.** The UI indicates a Help entry point.

The standard icons are notable because they are built into many Windows application programming interfaces (APIs), such as task dialogs, message boxes, balloons, and notifications. They are also commonly used on in-place messages and status bars.

Note: Guidelines related to icons are presented in a separate article.

Design concepts

There are several factors in choosing the appropriate standard icon which in part explains why they are so often used incorrectly. The most common mistakes are:

- Using a warning icon for minor errors. Warnings are not "softened" errors.
- Using a standard icon when it is better to use no icon at all. Not every message needs an icon.

- Alarming users by giving warnings for minor issues or presenting routine questions as warnings. Doing so makes programs appear prone to hazard, and detracts from truly significant issues.

The remainder of this section explains how to think about standard icons in order to avoid these common mistakes.

Message type vs. severity

Choose standard icons based the message type, not the severity of the underlying issue. The message types are:

- **Error.** An error or problem that has occurred.
- **Warning.** A condition that might cause a problem in the future.
- **Information.** Useful information.

Consequently, an error message might take an error icon but never a warning icon. Don't use warning icons as a way to "soften" minor errors. So despite their difference in severity, "Incorrect font size" is an error, whereas "Continuing with this operation will set your house on fire" is a warning.

Determining the appropriate message type

Some issues can be presented as an error, warning, or information, depending on the emphasis and phrasing. For example, suppose a Web page cannot load an unsigned ActiveX control based on the current Windows Internet Explorer configuration:

- **Error.** "This page cannot load an unsigned ActiveX control." (Phrased as an existing problem.)
- **Warning.** "This page might not behave as expected because Windows Internet Explorer isn't configured to load unsigned ActiveX controls." or "Allow this page to install an unsigned ActiveX Control? Doing so from untrusted sources may harm your computer." (Both phrased as conditions that may cause future problems.)
- **Information.** "You have configured Windows Internet Explorer to block unsigned ActiveX controls." (Phrased as a statement of fact.)

To determine the appropriate message type, focus on the most important aspect of the issue that users need to know or act upon. Typically, if an issue blocks the user from proceeding, it is presented as an error; if the user can proceed, it's a warning. Craft the main instruction or other corresponding text based on that focus, and then choose an icon (standard or otherwise) that matches the text. The main instruction text and icons should always match.

Severity

While severity isn't a consideration when choosing among the error, warning, and information icons, **severity is a factor in determining if a standard icon should be used at all.**

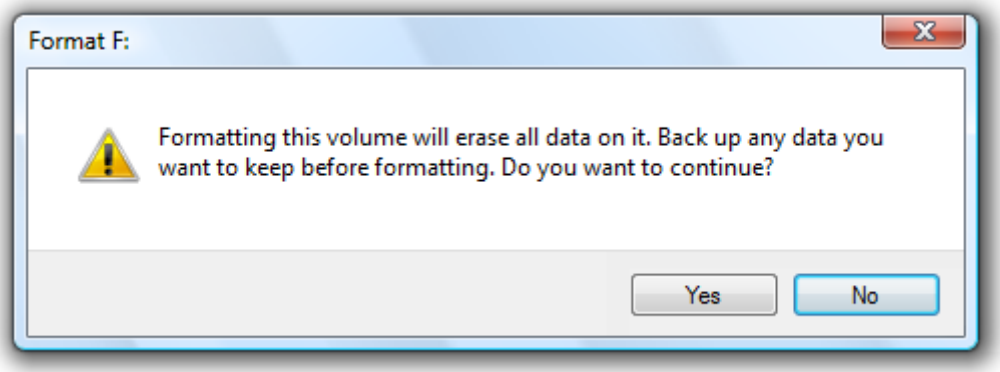
Icons work best when used to communicate visually. (Note that for accessibility reasons, this visual communication must always be redundant with another form, such as text or sound.) Users should be able to tell at a glance the nature of the information and the consequences of their response, so we must differentiate critical errors and warnings from their ordinary counterparts. Critical errors and warnings have these characteristics:

- They involve potential loss of one or more of the following:
  - A valuable asset, such as data loss or financial loss.
  - System access or integrity.
  - Privacy or control over confidential information.
  - User's time (a significant amount, such as 30 seconds or more).
- They have unexpected or unintended consequences.
- They require correct handling now, because mistakes can't be easily fixed and may even be irreversible.

To distinguish non-critical errors and warnings from critical ones, non-critical messages are usually displayed without an icon. Doing so draws attention to critical messages, makes critical and non-critical messages visually distinct, and is consistent with the Windows tone.

Not every message needs an icon. Icons are not a way to decorate messages.

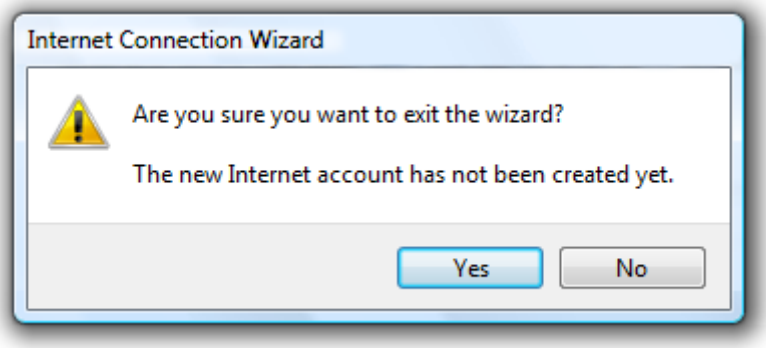
The following is a good example of a critical warning because it meets the previously defined characteristics.



In this example, a critical warning alerts users of potential irreversible data loss.

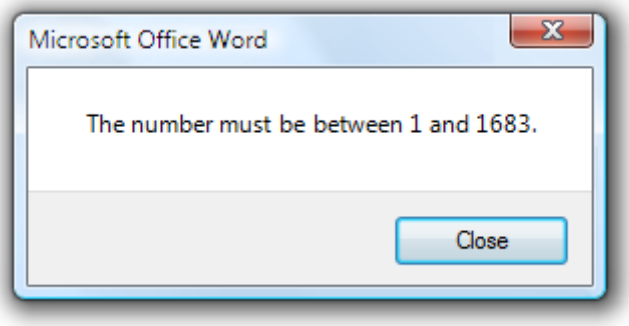
However, the next example isn't critical because it is likely to be intentional and its results are easily undone.

Incorrect:



In this example, this confirmation isn't critical because it's likely to be intentional and easily undone.

In a typical UI, most errors relate to user input errors. Most user input errors aren't critical because they are easily corrected, and users must correct them before continuing. Also, drawing too much attention to minor user mistakes is contrary to the Windows tone. Consequently, minor user input errors are usually displayed without an error icon. To reinforce their non-critical nature, we refer to these as user input problems.



In this example, this minor user input problem isn't critical, so it doesn't need an icon when presented in a dialog box.

Avoid overwarning

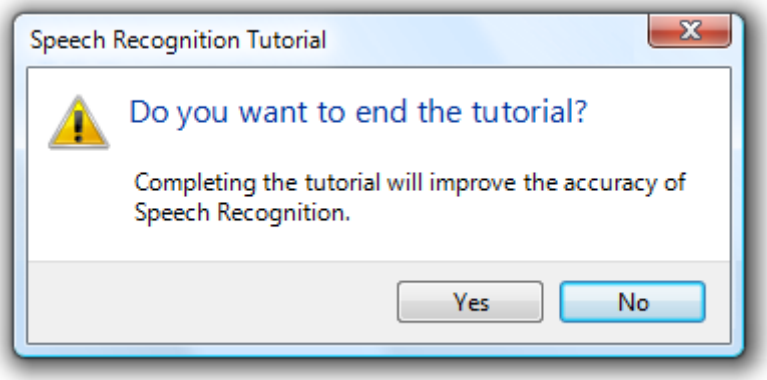
We overwarn in Windows programs. The typical Windows program has warning icons seemingly everywhere, warning about things that have little significance. In some programs, nearly every question is presented as a warning. Overwarning makes using a program feel like a hazardous activity, and it detracts from truly significant issues.

The mere potential for data loss alone is insufficient to call for a warning icon. Additionally, any undesirable results should be unexpected or unintended and not easily corrected. Otherwise, just about any incorrectly answered question could be construed to result in data loss of some kind and merit a warning icon.

To focus warning icons on truly critical issues:

- Make sure that the issue warrants heightened user attention. Routine confirmations and questions shouldn't have warning icons.
- Are users likely to behave differently as a result of the warning icon? Are users likely to consider the decision more carefully?

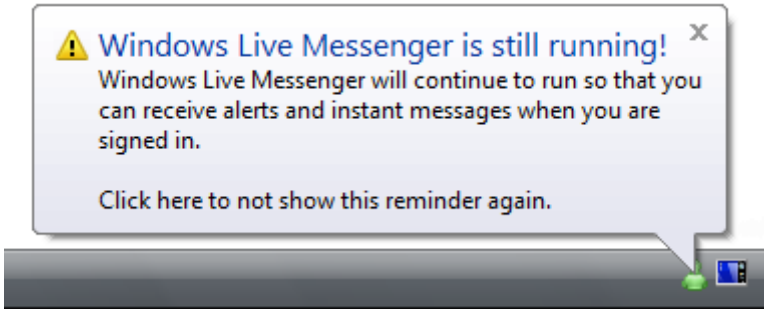
Incorrect:



In this example, are users likely to answer this question differently because of the warning icon?

- Is there some significant action to do or decision to make? Warnings without actions just make users feel paranoid.

Incorrect:



Why is this notification a warning? What are users supposed to do (beside worry)?

Context

Context is also a consideration in using standard icons because the context itself communicates information. Specifically:

- While dialog boxes (including task dialogs and message boxes) and notifications don't need icons for non-critical errors, in-place errors always need error icons. Otherwise, such non-modal feedback would be too easy to overlook.

- In-place warnings always need warning icons to distinguish them from regular text.
- Dialog boxes, notifications, and balloons don't need information icons because they are clearly presenting information. By contrast, banners need 16x16 pixel information or other icons because such non-modal feedback would be too easy to overlook.

Because context is a significant factor in icon usage, the standard icon guidelines in this article are given in terms of their context.

Evaluating standard icon appropriateness

When evaluating your UI text, read any standard icons as well. Read error icons as "error!", warning icons as "warning, be very careful here!", and information icons as "attention!". Then continue to read the remaining context, such as the main instruction, content area, and commit buttons. Make sure the meaning and the tone of each standard icon matches the meaning and the tone of its context. If they don't, you've found a problem.

If you do only one thing...

Make sure the meaning and the tone of each standard icon matches the meaning and the tone of its context. If they don't match, change or remove the icon.

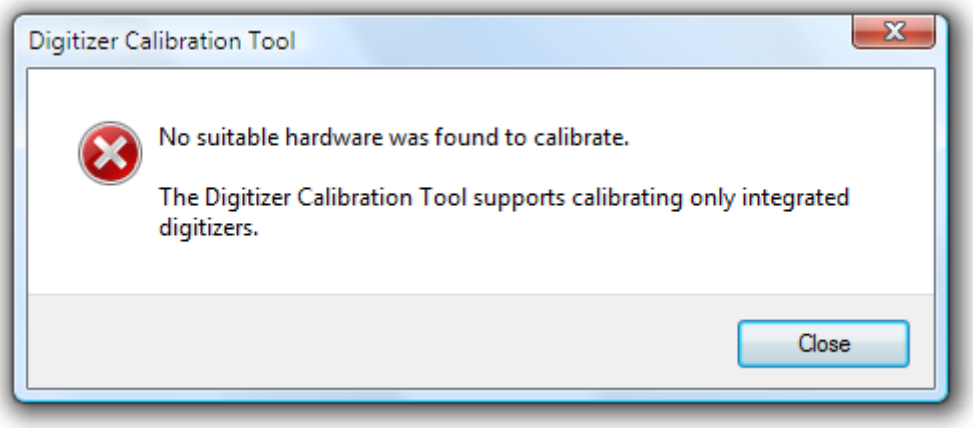
Guidelines

Note: For the following guidelines, "in-place" means on any normal window surface, such as within the content area of a wizard, property sheet, or control panel item page.

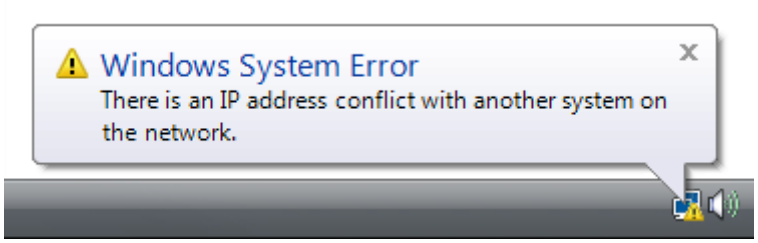
General

- Choose standard icons based their message type, not the severity of the underlying issue:
  - **Error.** An error or problem that has occurred.
  - **Warning.** A condition that might cause a problem in the future.
  - **Information.** Useful information.
- If an issue straddles different message types, focus on the most important aspect that users need to act on.
- Icons must always match the main instruction or other corresponding text.

Correct:



Incorrect:



In the incorrect example, the standard warning icon doesn't match the main instruction (which gives an error).

Icon size

- **Choose the standard icon size based on the context:**

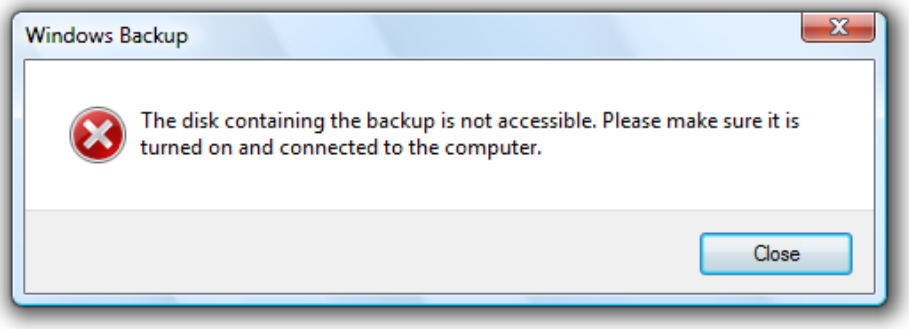
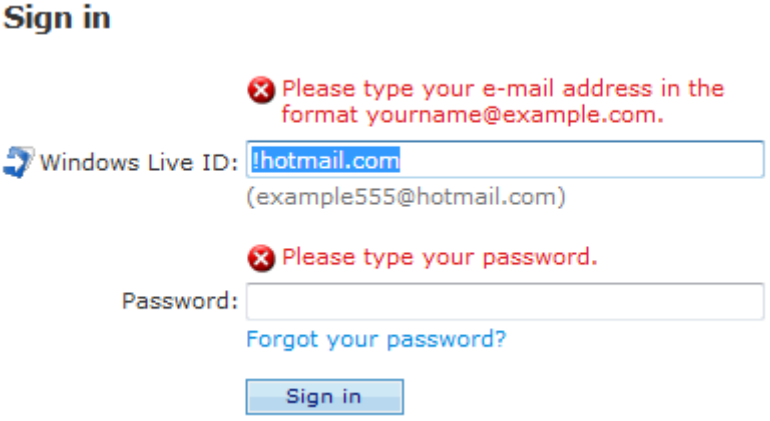
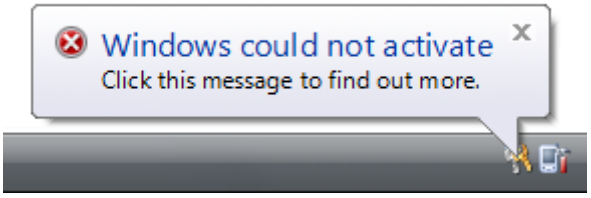
Expand table

Context	When to use
Dialog boxes	Use 32x32 pixel for content area icons; 16x16 pixel for footnote area icons.
In-place	Use 32x32 pixel for error pages; 16x16 pixel icons for all others.
Notifications	Use 16x16 pixel icons.
Balloons	Use 16x16 pixel icons.
Banners	Use 16x16 pixel icons.

Error icons

- **Use error icons only when an error or a problem has occurred:**

Expand table

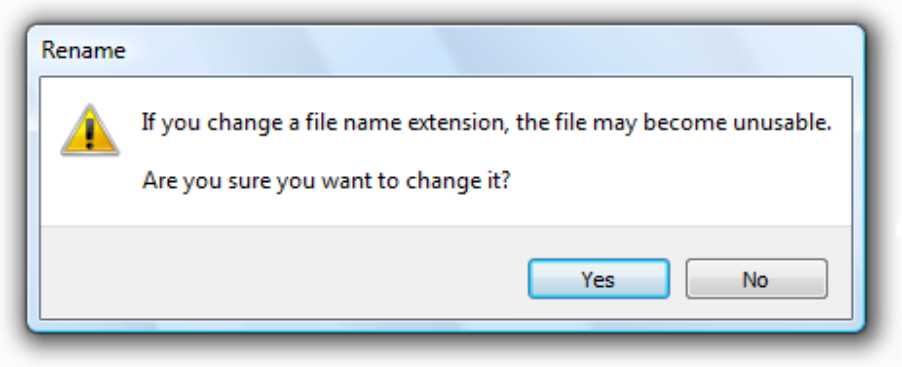
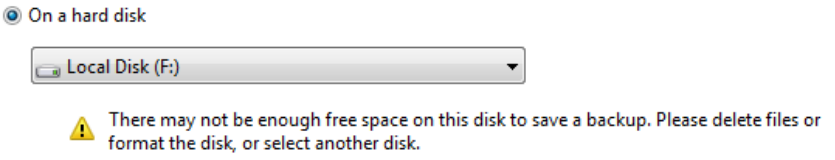
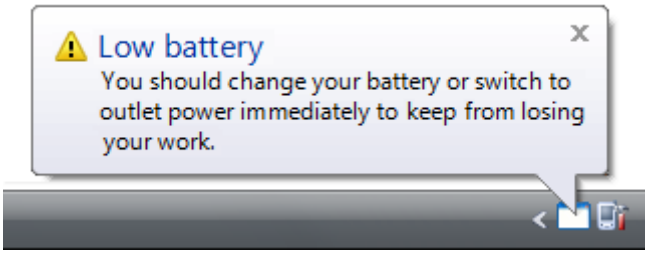
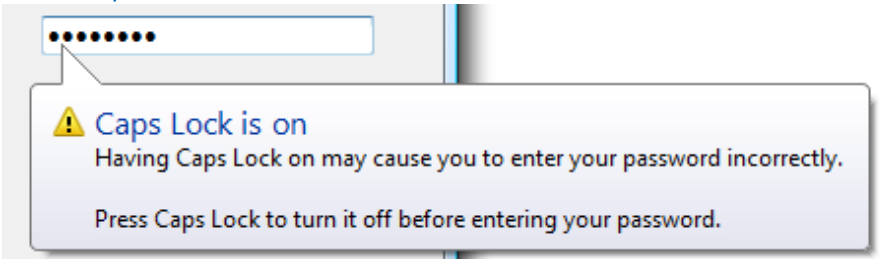
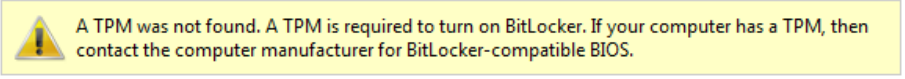
Context	When to use
Dialog boxes	Use for critical errors only. (don't use standard icons for non-critical errors.) 
In-place errors	Use for all errors. 
Notifications	Use for critical errors only. (for action failures .) 
Balloons	Don't use. Balloons shouldn't be used for critical errors, and they don't need error icons for non-critical errors.
Banners	Don't use. Banners shouldn't be used for errors.

- Generally, error icons aren't needed for non-critical user input problems. However, icons are needed for in-place errors, because otherwise such contextual feedback would be too easy to overlook.
- **For task dialogs, don't use error footnote icons.** Error icons must be presented in the content area only.

Warning icons

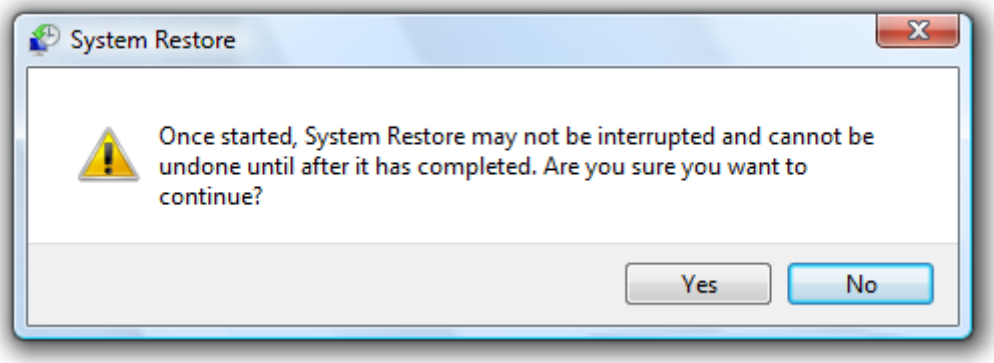
- **Use warning icons only when a condition might cause a problem in the future:**

Expand table

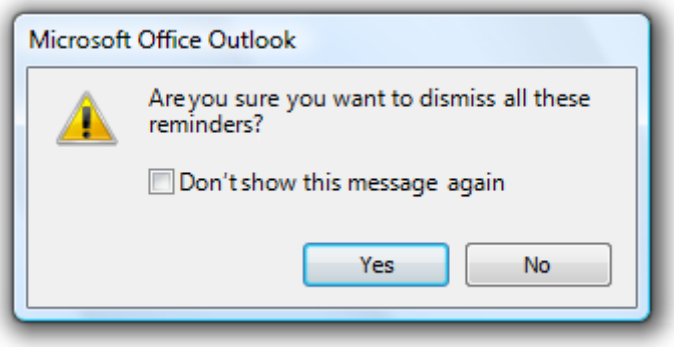
Context	When to use
Dialog boxes	Use for all warnings. 
In-place warnings	Use to identify the text as a warning. 
Notifications	Use for all warnings. (for non-critical system events.) 
Balloons	Use for special conditions. 
Banners	Use to draw attention to the banner. 

- **Don't use warning icons to "soften" non-critical errors.** Errors aren't warnings apply the error icon guidelines instead.
- **For question dialogs, use warning icons only for questions with significant consequences.** Don't use warning icons for routine questions.

Correct:

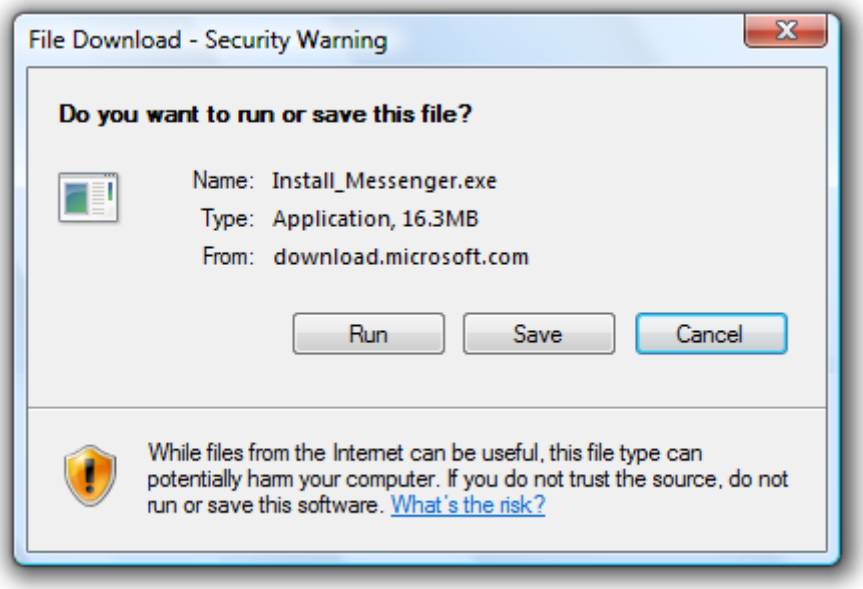


Incorrect:



In the incorrect example, a warning icon is incorrectly used for a routine question.

- **For task dialogs, you can use a warning footnote icon to alert users of risky consequences.** However, use a warning icon either in the content area or the footnote area, but not both.

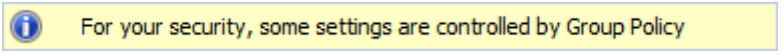


In this example, a yellow security shield is used in a footnote.

Information icons

- **Use information icons only when the context isn't obviously presenting information:**

Expand table

Context	When to use
Dialog boxes	Don't use.
In-place	Don't use. Use either plain static text or a banner instead.
Notifications	Don't use.
Balloons	Don't use.
Banners	use to draw attention to the banner. 

- Information icons aren't needed in dialog boxes, notifications, and balloons because their context sufficiently communicates that they are providing users with information.

- **For task dialogs, don't use information footnote icons.** Footnotes are sufficiently visible and it goes without saying that they are information.

Question mark icons

- **Use the question mark icon only for Help entry points.** For more information, see the Help entry point guidelines.
- **Don't use the question mark icon to ask questions.** Again, use the question mark icon only for Help entry points. There is no need to ask questions using the question mark icon anyway it's sufficient to present a main instruction as a question.
- **Don't routinely replace question mark icons with warning icons.** Replace a question mark icon with a warning icon only if the question has significant consequences. Otherwise, use no icon.

Feedback

Was this page helpful? Yes No

Animations and Transitions

Article • 06/03/2021

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

Strategic use of animations and transitions can make your program easier to understand, feel smoother, more natural, and of higher quality, and be more engaging. But the gratuitous use of animations and transitions can make your program distracting and even annoying.

Animations give the appearance of motion or change over time. Use animation to give feedback, preview the effect of an action, show the relationship between objects, draw attention to change, or explain a task visually.

Microsoft Windows uses a background flash animation to give feedback that the object was clicked.

Transitions are animations used to keep users oriented during user interface (UI) state changes and object manipulations, and make those changes feel smooth instead of jarring. Good transitions feel natural, often giving the illusion that users are interacting with real-world objects.

Windows Desktop Gadgets use smooth transitions between their concise and details states.

Generally, the best animations and transitions are used to communicate to users non-verbally, and to make state changes more natural and less noticeable. By contrast, the least effective are gratuitous in that they don't communicate anything or draw unnecessary attention. Animations are best used as a secondary form of communication. They should communicate information that is useful but not critical, and to be accessible, users should be able to determine equivalent information through other means.

Note: Guidelines related to software branding, sound, and accessibility are presented in separate articles.

Is this the right user interface?

To decide, consider the following questions.

Animations

Do the following conditions apply?

- The animation visually communicates something useful, such as giving feedback, showing relationships, causes and effects, or drawing attention to important change.
- Seeing the animation isn't essential. Equivalent information can be obtained in another way. Users might not benefit from the animation if:
  - They have turned animations off.
  - Their attention is elsewhere.
  - They are visually impaired.
  - The animation is obscured by another window.
  - The animation isn't played due to insufficient system performance.
- The animation doesn't affect the user's productivity. Either:
  - It happens quickly (200 milliseconds or less).
  - It doesn't interfere with interaction or it can be interrupted.
  - The user has to wait anyway.
- The animation doesn't affect the user's flow.
  - It is either at the user's center of attention, or it draws attention to something outside the center of attention that is important or useful in completing a task.
  - It is easily ignorable, not distracting or annoying.

- It doesn't become tiresome. Users still find it appropriate and enjoyable even after repeated viewing.

If so, consider using an animation.

Transitions

Is an object or scene changing state, and do all of the above conditions for using animations as well as any of the following conditions apply?

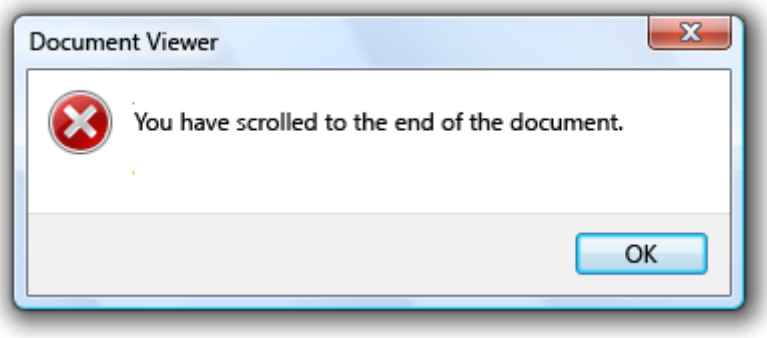
- The state change is conceptually disorienting, confusing, or otherwise hard to understand.
- The state change is visually jarring, lacks continuity or smoothness, or flashes; or appears unnatural, unpolished, or of poor quality, especially if it involves a large screen area.
- Using a transition would make the state change appear faster.
- The state change is worthy of special user attention.

If so, consider using a transition.

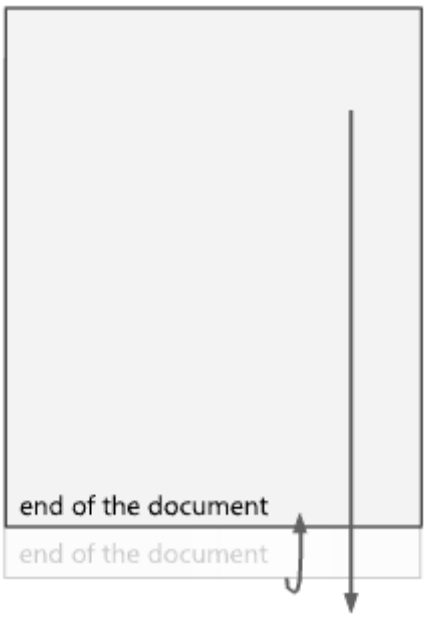
Design concepts

Animations and transitions are an effective way to communicate information visually that would otherwise require text to explain or might be missed by users.

Incorrect:

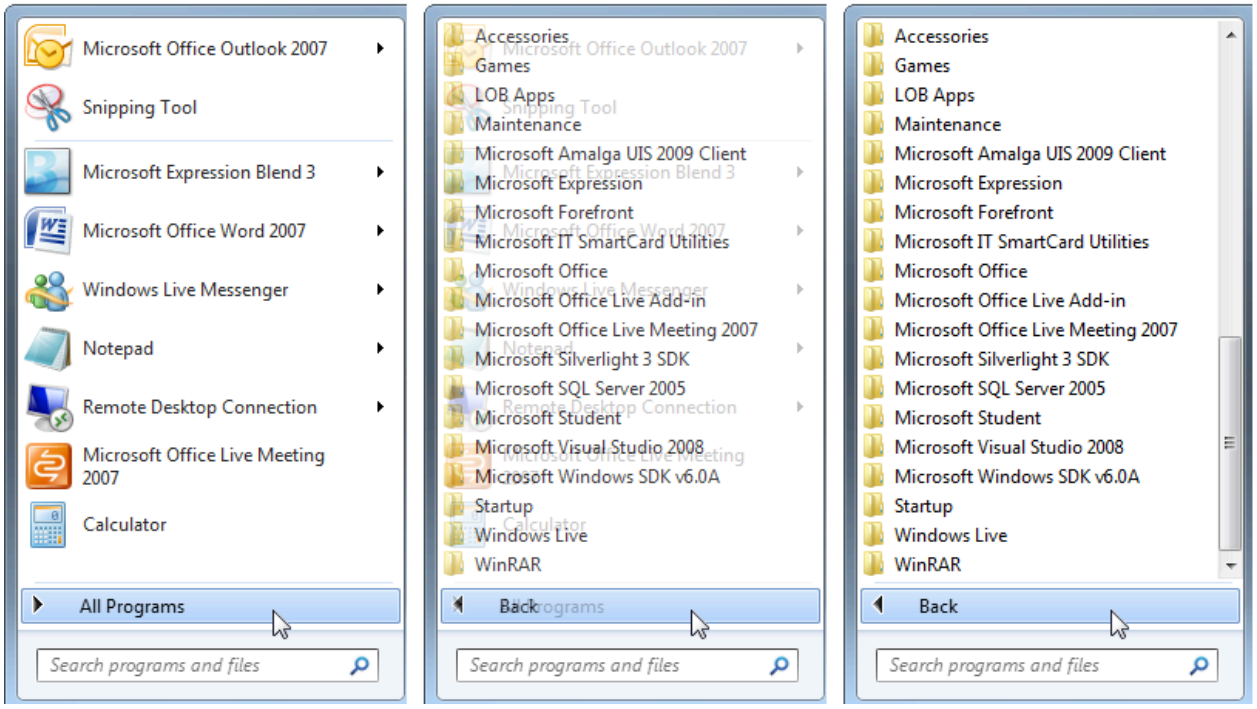


Correct:



Using an animation communicates the same information, but in a natural, unobtrusive way. Which would you rather see a thousand times?

Animations and transitions don't have to demand attention to be successful. In fact, they are often used to avoid drawing attention to program mechanics that users don't need to be aware of. Many successful animations are so natural that users aren't even aware of them; rather users would only notice their absence. Frequency of occurrence increases the need for subtlety, so save effects that demand attention for infrequent events that really deserve the attention.



A Start menu transition that avoids drawing attention.

Beyond making your program easier to understand and feel smoother, **well-designed animations and transitions are a great way to add personality, character, and style to your program.** They can make the user experience more immersive and engaging by giving it a natural, real-world feel.

Windows 7 highlights taskbar buttons on hover based on the current mouse position and program icon color. This approach is visually attractive, yet subtle, conveying a humble personality.

However, animations and transitions are most effective and welcome when they serve a clear purpose. They should be used to improved usability, smoothness and flow, and the perception of quality, without harming performance significantly.

While some types of animations are used to draw the user's attention, make sure that attention is well deserved and worthy of interrupting the user's train of thought. The human eye is sensitive to motion, especially peripheral motion. It can be difficult for users to concentrate when there is a flashing taskbar button or a spinning notification area icon. **Avoid using animations to interrupt or distract users, or draw attention to things that don't warrant the user's attention.**

Incorrect:

Programs shouldn't flash their taskbar button unless users must do something important immediately. In this case, the only thing the user needs to do is activate the program.

Use animations and transitions because your program needs them, not merely because you can. And for accessibility, don't use animation as the only way to convey essential information. Make sure that users can obtain equivalent information in a different way.

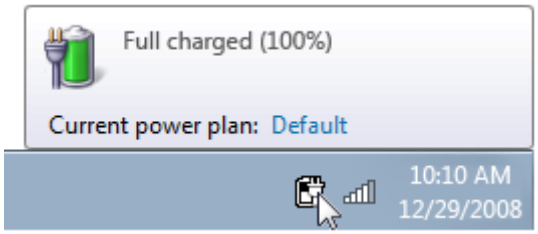
Attributes of good animations and transitions

Good animations and transitions strike the right balance among these attributes:

- **Are clearly purposeful.** Good animations are there because they need to be, whether to communicate information, make an interaction feel real, or draw attention to something noteworthy. And purposeful animations are accurate; if an

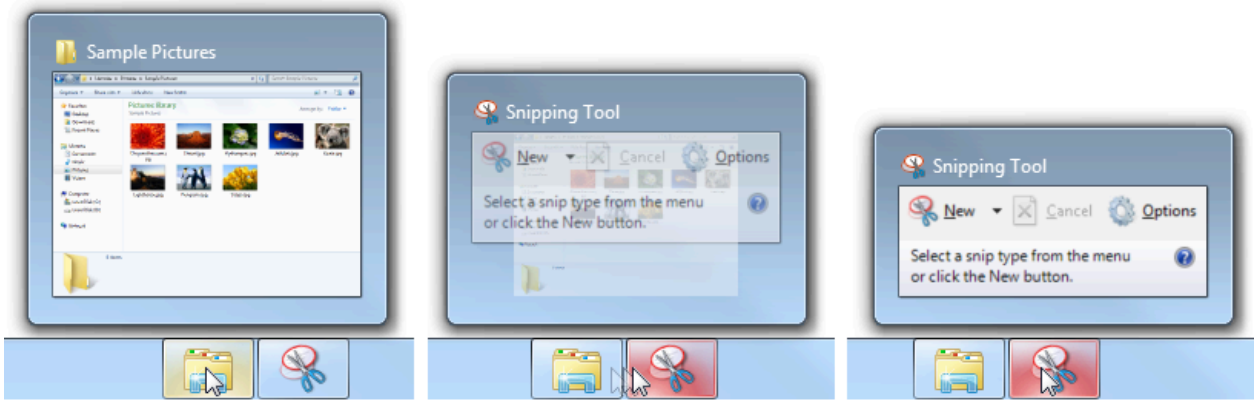
animation shows that a task is being done, it's because the task is in fact being done.

Incorrect:



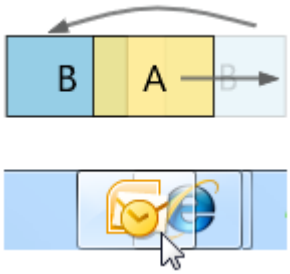
In this example, the animation shows that a fully charged battery is being charged.

- **Look smooth and continuous.** Good animations smoothly remove seams between scene or element state changes by showing relationships and providing a sense of place and context. Continuity helps users understand how they got where they are, and how to get back to where they came from.



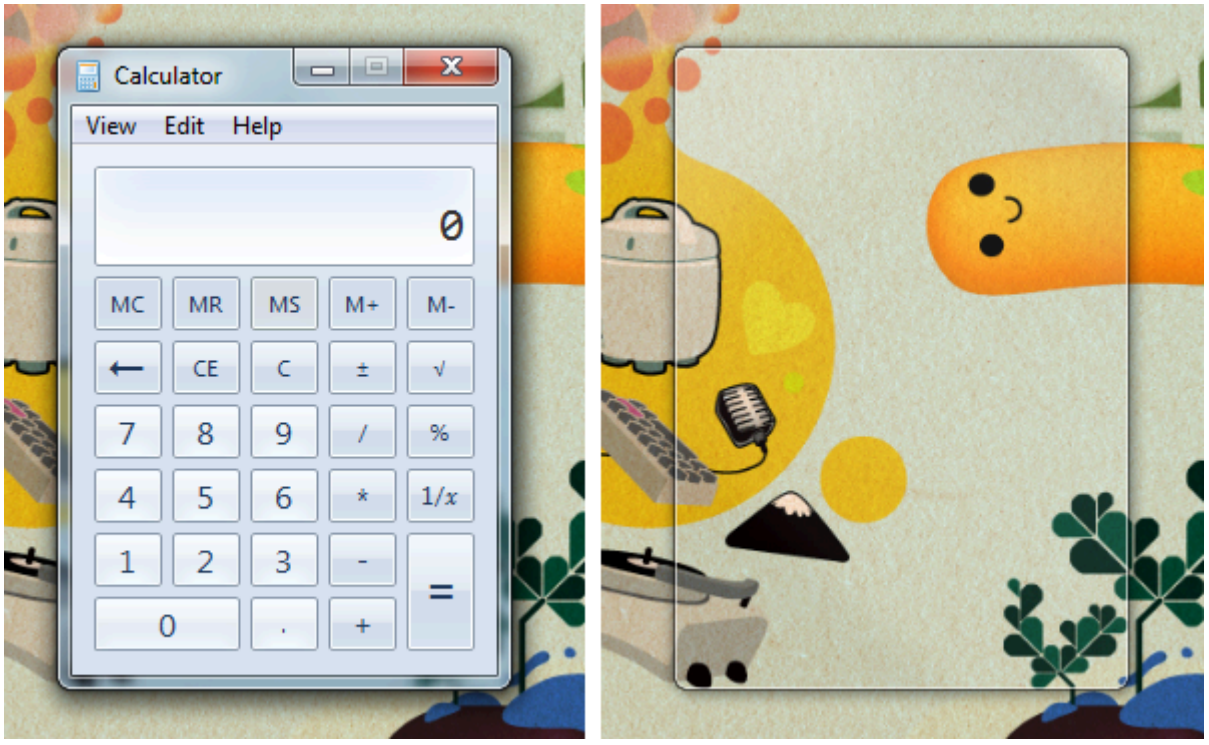
The Windows 7 taskbar window preview morphs for continuity as the user moves from one program to another.

- **Are realistic.** Good animations simulate an object's real-world physical properties and behavior. This helps users predict and understand the results of their interactions. You don't have to model the real world exactly, but if you use realistic animations, you must keep them consistent with the real world. Users should never be surprised or confused by the results, especially with animations used for direct manipulation.



In this example, the "move out of the way" animation used by the Windows 7 taskbar feels more realistic than a static insertion point.

- **Are authentic.** Even objects that aren't found in the real world can appear natural by being based on the real-world behavior of a different, but related, object. This metaphor only works if the relationship clearly communicates the intended purpose and behavior.



In this example, the window "squeegee" animation used by Windows 7 feels authentic because it is consistent with how glass windows might behave in the real world.

- **Use natural mapping.** Natural mappings are either physical or cultural. A culturally-based natural mapping, for example, might start from the fact that in Western cultures, people read from left to right. Consequently, to express a time sequence of objects, the middle object is current, objects to the left are from the past, and objects to the right are in the future. Going forward in time is indicated by left-to-right movement.

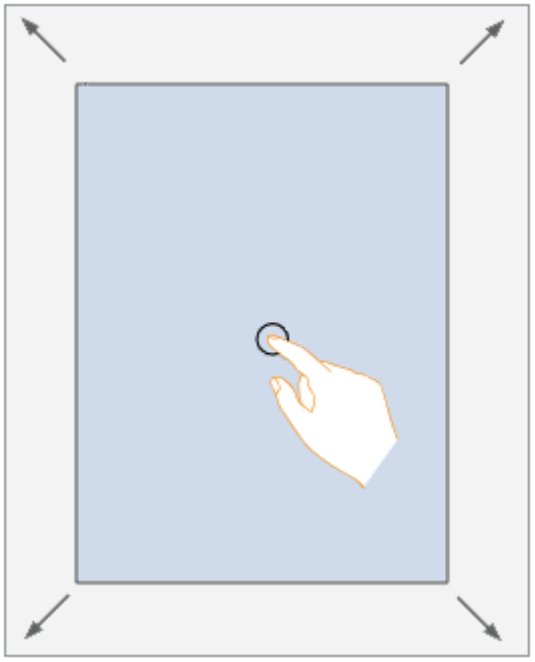
In this example, the Windows Media Player control has a natural mapping because playing moves the position from left to right.

- **Have personality.** Well-chosen animations are great ways to add personality, character, and style to your program. They can make the user experience more immersive and engaging. While the type of animation determines what it communicates, the specific way in which the animation is performed shows the program's personality. Good animations project the right personality for your program, whether serious or whimsical, or somewhere in between.

In this example, Zune's use of animated text and dynamic perspective help shape its personality.

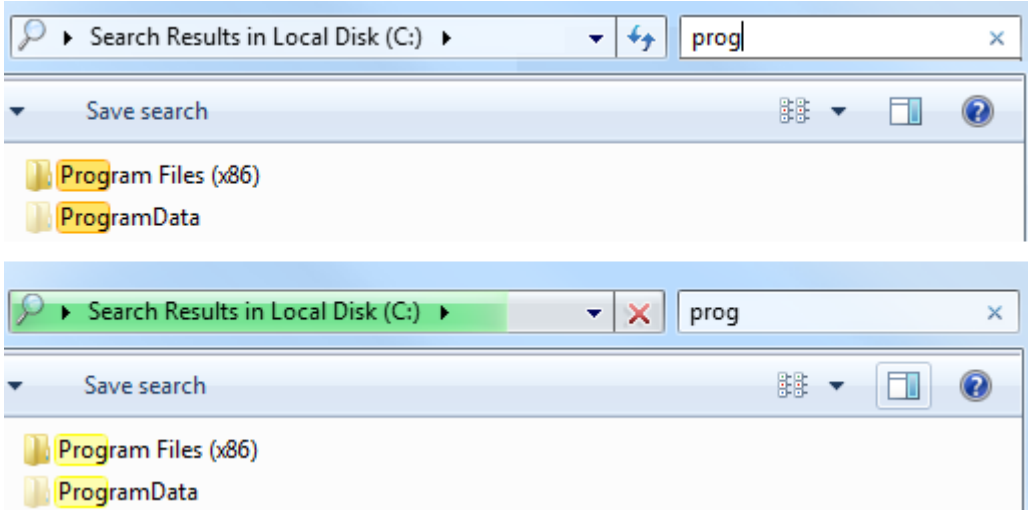
- **Look and feel responsive.** Good animations don't harm the user's productivity by blocking users from other interactions, or forcing users to watch. No matter how natural and engaging your program's animations are, nobody wants to wait for them exclusively. Good animations also look responsive without being jarring by having a fast start with a soft landing. Responsive animations also benefit from

communicating their purpose quickly. Users shouldn't have to watch an animation for a long time just to figure out what it is doing or when it is done. For direct manipulation, responsive animations are essential to maintain a direct and engaging real-world feel. To feel direct, an object's contact points must stay under the pointer smoothly throughout the manipulation. Any lag, choppy response, or loss of contact destroys the perception of direct manipulation.



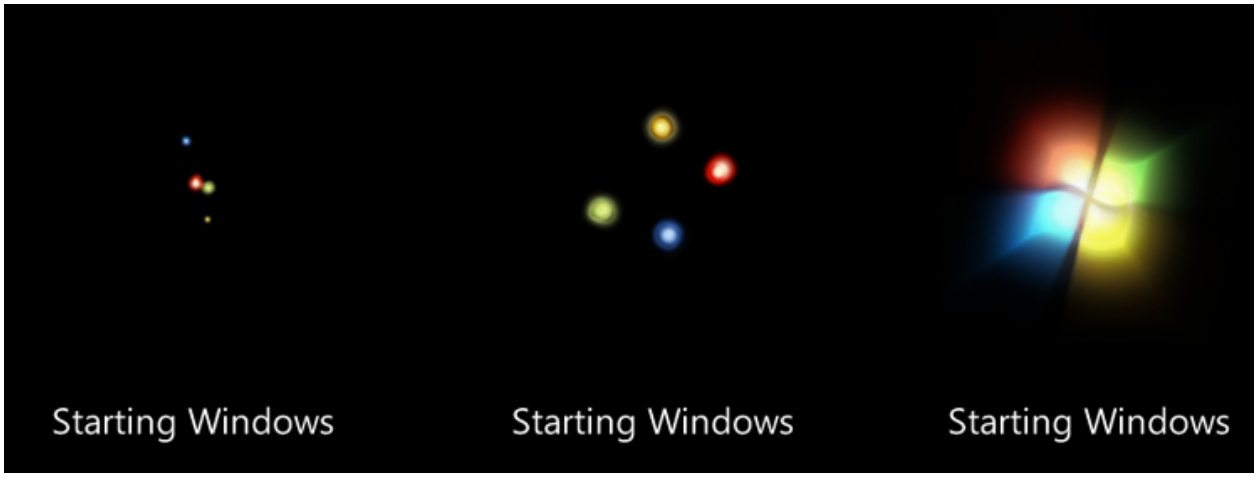
In this example, the touch panning transition feels responsive by keeping the contact point under the user's finger throughout the manipulation.

- **Attract the right level of attention.** Good animations are usually subtle and draw only the attention required to fulfill their purpose. As a result, they aren't distracting, annoying, overly complex, overly long, or repetitive. They don't become tiresome after repeated viewings.



In this example, Windows search temporarily draws attention to matching search words, then fades down.

- **Look special only if genuinely special.** Frequency increases the necessity for subtlety, so common interactions need simple animations that communicate a simple idea in a simple way. Reserve special, complex animations for special, infrequent experiences.



In this example, Windows uses an attention-getting animation at startup to make the experience feel special, but such an animation would be inappropriate elsewhere.

You'll know that you have achieved the right balance when the overall experience would be harmed if any of these attributes were removed.

Creating an animation vocabulary

Good animations are about effective visual communication, and consistency is crucial to their effectiveness. If you use a specific transition, such as pushing a scene in from the right to advance to the next scene, that should be the only transition used for that purpose and that transition shouldn't be used for any other purpose. Assigning different meanings to the same animation harms its ability to communicate. By assigning specific animations and transitions to specific meanings, you are creating an animation vocabulary.

This issue applies to animations and transitions that have meaning, not to generic ones that users aren't likely to assign meaning to or those whose purpose is to be unnoticeable. For example, animations like fades and special effects like dissolves have no particular meaning, so they can be used freely.

A good vocabulary assigns animations that model an object's real world, physical behavior. If you need to assign an animation to an object or action that doesn't have a

real world counterpart, choose an animation that shows how the object might behave were it real.

While the Start menu isn't a real world object, its hover effect lights up like a real-world object might when activated.

Each animation in a vocabulary needs to be clearly distinct. The animations should have similar behaviors only if their associated actions are similarly related. For example, movement transitions suggest navigation, so you can use movement transitions from different directions to indicate different types of navigation.

You'll know that your animations and transitions aren't communicating well when users find the results confusing, surprising, or unexpected. Generally, it's better to achieve a single purpose well than multiple purposes not so well.

Ideally, your animation vocabulary should be comprehensive across all areas of your program that need them. If only a few interactions have natural animations, that will draw attention to those that don't.

To learn more about the Windows animation vocabulary, see the Usage patterns section of this article.

Designing the right personality

While the type of animation determines what it communicates, the specific way in which the animation is performed speaks to the program's personality and reinforces its brand.

Your program's personality should reflect the nature of its tasks and the personality of its users, so it's not an arbitrary choice. Rather, a well-designed personality should feel authentic; never try to force it. The personality should make an emotional connection to the user. Some factors to consider:

- **Tasks:** Serious or fun; optional or required.
- **Consequences:** Serious or minor.
- **Cost:** Free or purchased; if purchased, moderately priced or expensive.
- **User focus:** Relatively narrow group of target users, or wide, general audience.
- **User environment:** Professional, casual, or home.
- **User age:** Younger or older.
- **Usage frequency:** Frequent or infrequent.

The combination of these factors helps determine an appropriate personality for your program. Here are some suitable combinations for common types of programs:

Productivity applications

Naturally, productivity applications must focus on productivity. While a few special experiences can stand out, most other animations should have these characteristics:

- Small
- Natural, realistic
- Subtle, subdued
- Fast, efficient
- Relaxed

Utilities

Utilities are typically used briefly, so their use of animation can be more aggressive:

- Realistic, illustrative, self-explanatory
- Safe
- Engaging

Entertainment, games

Because the goal of these programs is to engage and delight users, the animations and transitions can be much more aggressive by having these characteristics:

- Large (possibly becoming an integral part of the experience)
- Artificial, surreal
- Impactful, vibrant
- Emotional, playful, whimsical
- Energetic

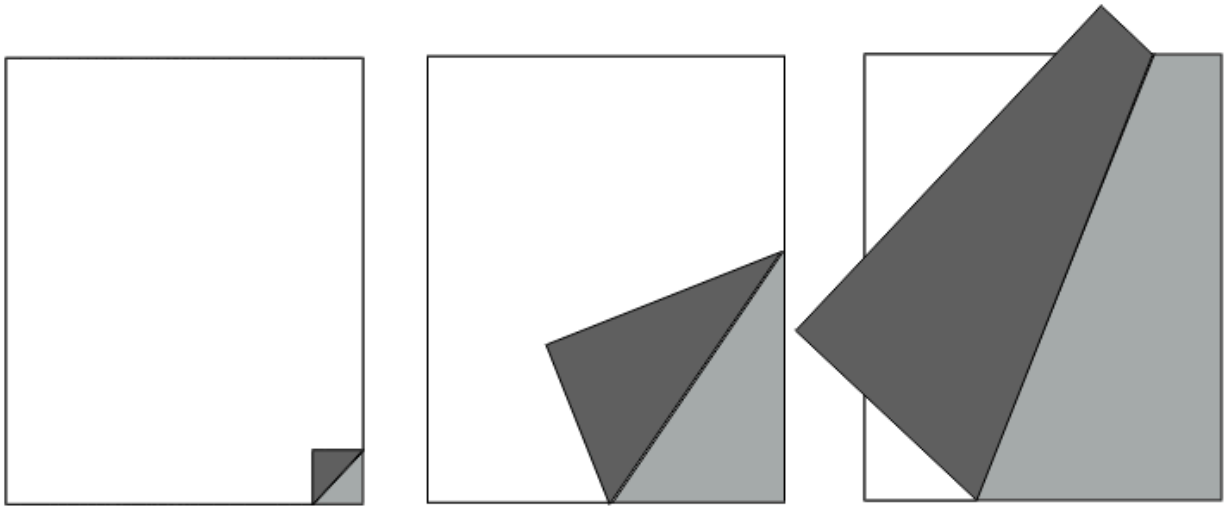
Making an emotional connection is so important for entertainment programs that it's acceptable to bend some rules if doing so helps make users fall in love with the program. For example, it's acceptable if an animation or transition becomes tiresome after the hundredth time if most users are unlikely to use the program that often.

Generally, animations and transitions that are small, natural, subdued, efficient, yet relaxed are the safest bet. Transitions with these characteristics typically take the shortest path from beginning to end, start quickly, end softly, and don't overshoot. Also, well designed transitions are designed to work well across the entire range of distances in which they will be used.

Animation performance

When designing animations, make sure that they don't affect users' ability to use your program efficiently. Generally, make your animations slow enough to fulfill their purpose, but fast enough that they don't interfere with responsiveness, demand too much attention, or become tiresome.

Incorrect:



While this page turning animation has an engaging, real-world feel, it lessens users' productivity by taking longer to turn pages.

Brief transitions (200 milliseconds or less) are a special case (especially when they often work off of a delay) because users will be aware that they have to wait a split second for them. Users are willing to wait for such animations if:

- The perceived wait is extremely brief (200 milliseconds or less).
- The transition makes the interaction feel more smooth and natural.
- The transition makes the interaction feel more responsive.
- Any delay helps keep the user in control of the interaction.

Users will accept a brief delay for the taskbar button reordering animation because it is very brief and it makes the interaction feel more natural.

There are three ways in which animations can adversely affect performance: speed, responsiveness, and perception.

For speed, some animations are visual veneers over CPU-intensive tasks, so the last thing you should do is make these tasks slower with CPU-intensive animations. The most CPU-intensive animations ("heavy" animations) tend to:

- Involve many elements moving independently.
- Play for a long duration or distance.
- Involve a large amount of screen space.
- Are mathematically intensive.

Animations with less impact on performance:

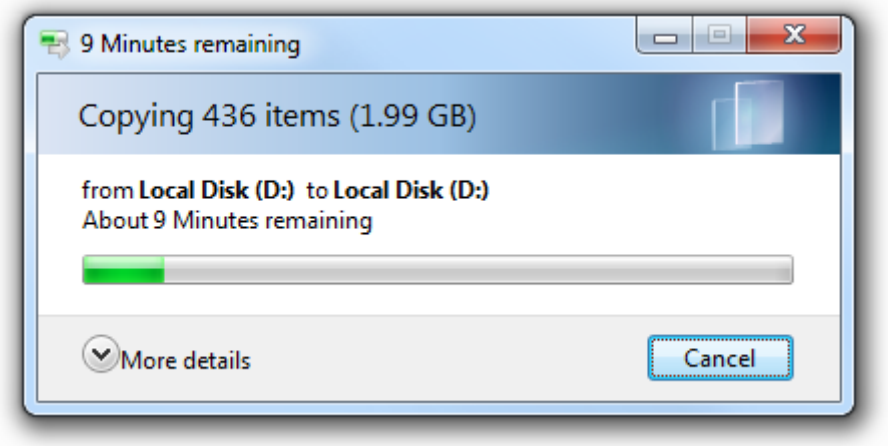
- Involve a single object.
- Play for a short duration or distance.
- Involve a small amount of screen space.
- Aren't mathematically intensive.

To ensure good performance, heavy animations should be used only for tasks that aren't CPU intensive, whereas light animations can be used anywhere.

For responsiveness, most animations and transitions should be designed so that users can interact while the animation is running. Unless an animation is part of a process, make it independent of the user's primary interaction and allow users to interrupt it.

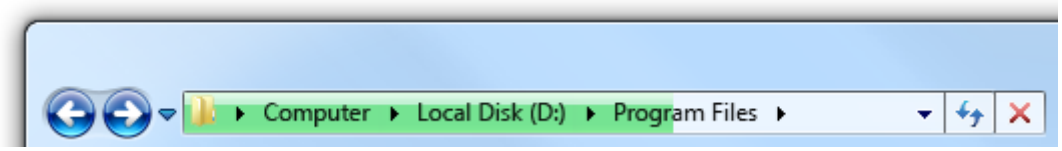
An animation might not adversely affect a task's performance in reality, yet users may have the perception that it does. For example, don't use an animation that appears heavy for a slow, CPU-intensive task even if it doesn't harm performance, because users might conclude that the animation is the reason the task is slow. **If something looks slow, it will feel slow, so it's better to use animations that feel simple, lightweight, and fast.** Using animations with snappy beginnings for CPU-intensive tasks helps.

Risky:



While the animation in the Windows file copy dialog doesn't harm file copy performance, it runs the risk of having users think that it does.

Also risky:



In this example, the sluggish looking progress animation in the Windows Explorer address bar makes some tasks look painfully slow.

Animations and transitions have no value if their quality is so poor that they make the experience less smooth and less engaging. To maintain their quality, animations should be designed to degrade gracefully whenever sufficient system resources aren't available. Animations can degrade by having variations that require fewer resources (such as shorter lengths or lower frame rates), or even not running at all. Regardless of the resources available, make sure the animations have high quality and look like animations instead of software bugs.

Finally, if users believe that your program's animations and transitions detract from their productivity, there's a good chance that some users will want to turn them off. To support this ability, respect the option to **Turn off all unnecessary animations** found in the Windows Ease of Access Center.

Attracting the right level of attention

While only some types of animations and transitions are specifically designed to attract the user's attention, they should be designed to attract the right level of attention to fulfill their purpose well. What are the different ways to attract attention, and how do you pick the right one?

Animation effects

Different animation effects attract different levels of attention. The following list summarizes the most common methods, starting with the most attention-getting:

- **Rapid flashing.** Demands immediate attention. Can break users' concentration no matter where the flashing is occurring.
- **Moderate flashing.** Same, but demands less attention with lower frequency.
- **Bouncing.** Noticeable in peripheral vision, and relatively demanding in nature. Users are likely to notice, but can continue to concentrate elsewhere only if duration is short.
- **Motion.** Noticeable in peripheral vision, but not demanding. However, complex or 3-D motions attract more attention than simple or 2-D motions. Users are likely to notice, but can continue to concentrate elsewhere.

- **Moderate pulsing.** Noticeable but not distracting in peripheral vision. Users can continue to concentrate elsewhere. Can pulse brightness, colors, and sizes.
- **Slow pulsing/glowing.** Noticeable but subtle. Attracts more attention than a static effect, but users might not notice the animation unless they are already looking.
- **Fade.** Even less noticeable. Attracts more attention than a static effect, but users might not notice the animation unless they are already looking.
- **Static highlight/gleam.** Noticeable if users choose to look, but doesn't demand attention if it is elsewhere.
- **Ambient/natural.** Purposefully not noticeable by having a natural, real-world appearance.

To determine the right approach for your program or feature, consider how these factors relate to your feature's scenarios.

For example, suppose you are designing an instant message program and someone just sent the user a message. This scenario requires the user's attention, it should be noticeable anywhere, and usually the user will want to respond quickly. This scenario suggests that a moderate flashing animation would be a good choice. By contrast, suppose you want to inform users that a print job has completed. Users should be able to continue to concentrate and work productively elsewhere, and it's acceptable if users don't notice. This scenario suggests that moderate to slow pulsing or glowing would be a good choice.

Duration

The appropriate duration for an attention getting animation depends upon the scenario and the specific type of animation used. The more attention an animation effect demands, the shorter the duration should be. While very subtle effects that demand little attention (like slow pulsing) can be played indefinitely, attention demanding effects should only be played between 1 and 3 seconds. Anything longer risks making the animation overwhelming and annoying.

In Windows 7, the taskbar flashes for attention for only a second. Any longer would be annoying.

Effect decay

You should design attention-getting animations based on the assumption that if users don't respond right away, it is because they are busy doing something else and don't want to be interrupted. Thus, your goal should be to attract attention without demanding it.

To get the right balance of attracting attention without demanding it, decay the intensity of an effect over time. For example, to attract attention you can make the effect initially strong, but then slow down the effect quickly. By doing so, the attractive power is mostly determined by the initial effect, but the user's overall impression is determined mostly by its finish.

In Windows 7, the taskbar flash effect slows down at the end.

What about PowerPoint?

Microsoft PowerPoint transitions often deliberately violate these guidelines because they are designed to draw attention to slide transitions and require users to wait for them. Furthermore, they don't have any particular meaning so they communicate nothing beyond the fact that a slide is changing.

PowerPoint-style transitions, when used properly, have these purposes:

- They break long presentations into smaller chunks by forcing the presenter to pause.
- They draw the audience's attention toward changes in the presentation, helping people refocus if their minds have wondered.
- They give the presentation a rhythm so that it doesn't feel monotonous or overwhelming.
- Their style reflects the personality of the presenter or of the material.

While these are important goals for a presentation, such transitions would draw unnecessary attention in the UI of most types of programs, and would become tiresome quickly.

Bottom line: Don't use PowerPoint-style transitions as a model for your program.

If you do only six things...

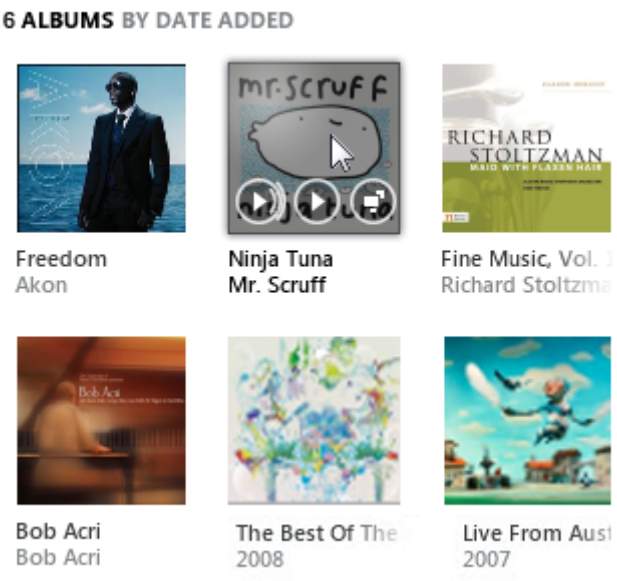
1. Use animations and transitions to make your program easier to understand, and feel smoother and more engaging. They should have a clear purpose. Don't use animations just because you can, or to draw unnecessary attention to your program.
2. Define an animation vocabulary and use it consistently throughout your program. Use the Windows 7 animation vocabulary when appropriate.
3. Use the characteristics of your animations to give your program personality and reinforce its brand.

4. Make most animations simple, brief, and subtle. Remember that animations don't have to demand attention to be successful. If an animation is appropriate and natural, users will only notice its absence.
5. Make your animations fast and responsive, and give them a lightweight feel. No matter how engaging your animations are, nobody is going to want to feel like they are waiting for them. Design heavier animations to have graceful degradation.
6. Design for the long run. If an animation is annoying, distracting, or tiresome, redesign it or remove it.

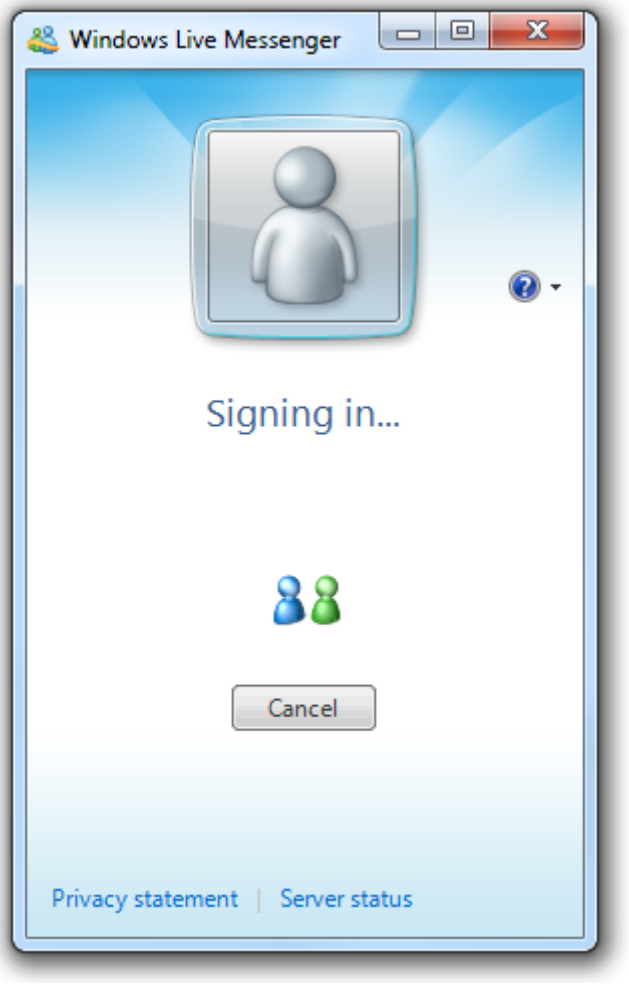
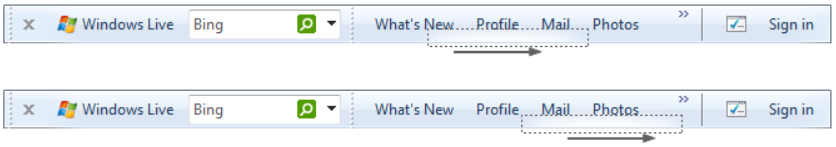
Usage patterns

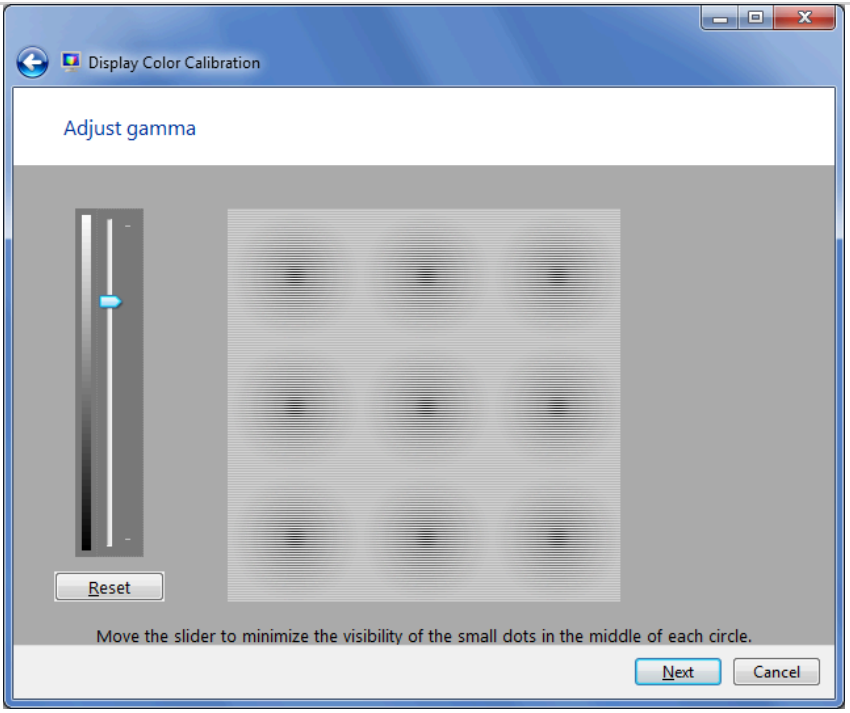
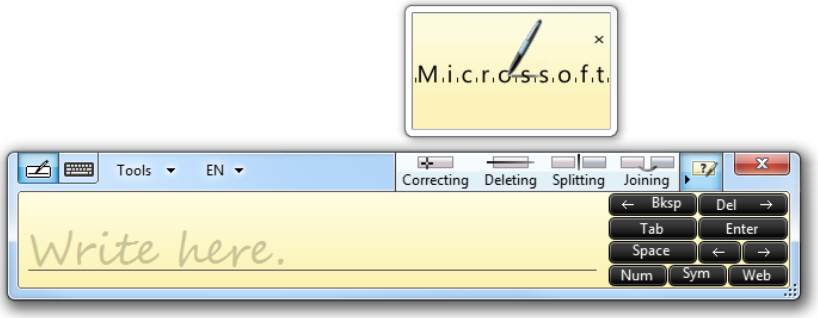
Animations have several usage patterns:

Expand table

Usage	Description
Hover feedback to show where the interaction point is.	Indicates that the interaction point is active. hover can be shown through a static effect as well. windows vocabulary: display hover effect (bounding rectangle, highlight, enlargement) with a fade in/fade out effect for smoothness.  In the Zune digital media player, album covers highlight and add playback controls on hover.
Click feedback to show that a clickable object is responsive and received a click.	Indicates that an object has been clicked. windows vocabulary: flash object background on click down event. to show touch contact, use a ripple effect.

Usage	Description
	 Touch displays a ripple animation so that the user knows the interaction was recognized.
Selection feedback to show that an object is selected.	Indicates that an object is selected. selection can be shown through a static effect as well. windows vocabulary: draw selection rectangle with a fade in/fade out effect for smoothness. Ninja Tuna Mr. Scruff, 2008 Ninja Tuna Mr. Scruff, 2008 Ninja Tuna Mr. Scruff, 2008 In Zune, album covers blink on click, and then get a selection rectangle on selection.
Progress feedback to show that a task is being performed.	Progress feedback indicates that a task is making progress, typically with activity indicators, progress bars, or animations that illustrate the task. determinate progress feedback shows roughly how much of the task has been done and how much remains, whereas indeterminate progress only indicates that the task is being done. windows vocabulary: spinning activity indicators, progress bars, progress backgrounds, illustration animations.

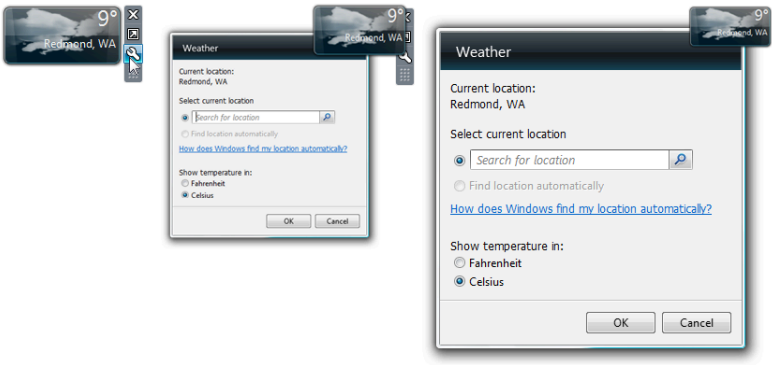
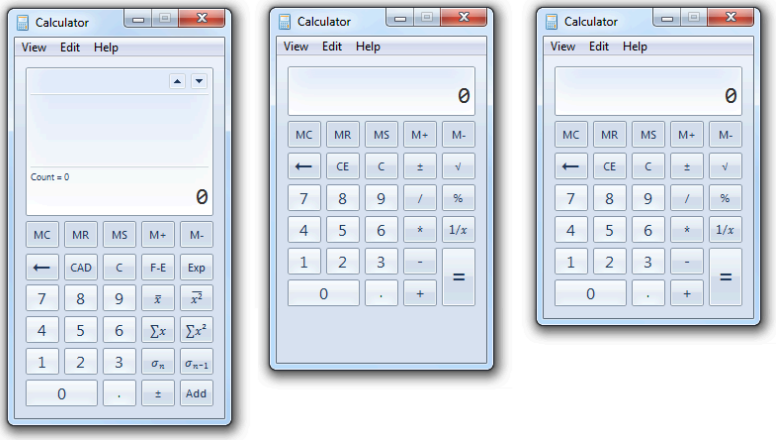
Usage	Description
	 In this example, Windows Live Messenger displays indeterminate progress feedback during sign-in.
Attractor to show that something needs the user's attention.	Attract the right level of attention when significant objects are created or need attention (often due to change), or important or urgent events happen. see attracting the right level of attention for design techniques. windows vocabulary: flashing, moving, pulsing, glowing, gleam.  The Windows Live toolbar animates on first appearance to make it obvious where it is.
Relationship to show the relationship between objects or causality in effects.	Show relationships, especially when the relationship might not be understood or expected, in a way that's not distracting or confusing. windows vocabulary: morphing, transportation, physical change such as flipping over, growing from a point source, shrinking to a point destination.

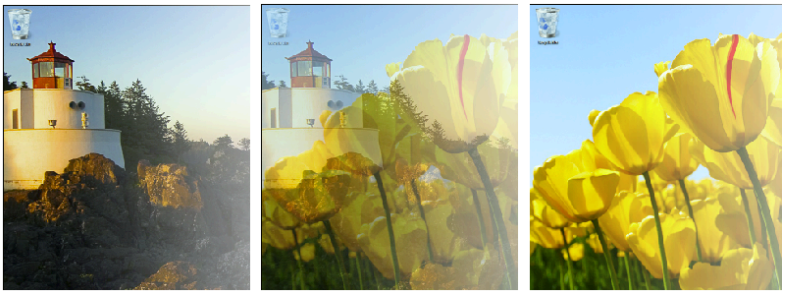
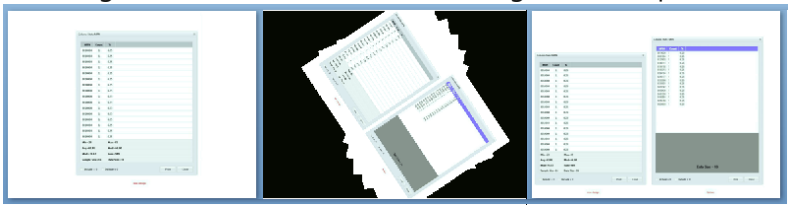
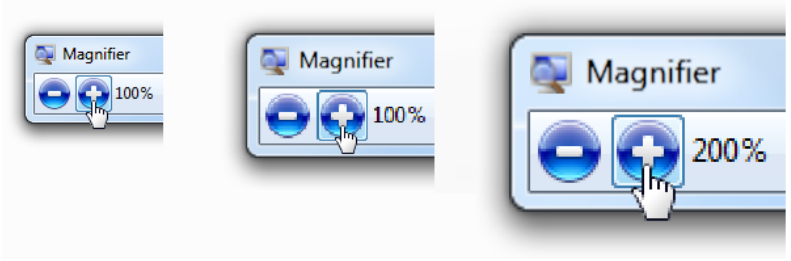
Usage	Description
	 In this example, the animation shows the relationship between the gamma setting and its affect on the display.
Illustration/Preview to explain visually a concept, task, or the effect of a command.	An animation or video that explains a concept or how something works visually, either to supplement or replace a textual explanation. doing so allows users to perform tasks or choose commands efficiently and confidently.  In this example, the Tablet PC Input Panel "show me" commands use illustrations to show how to correct, delete, split, and join.

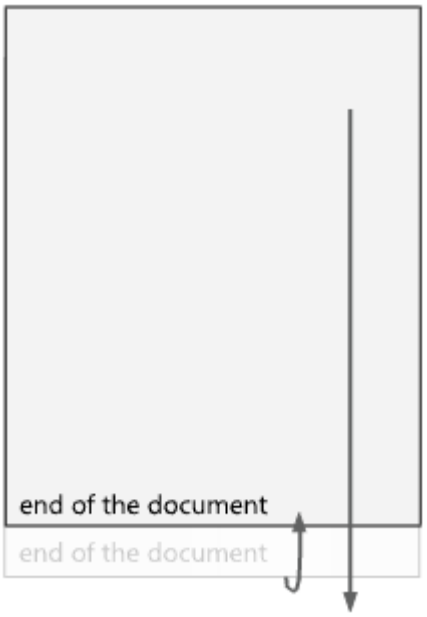
Transitions have several usage patterns:

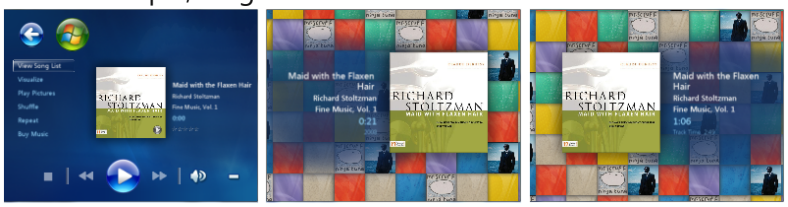
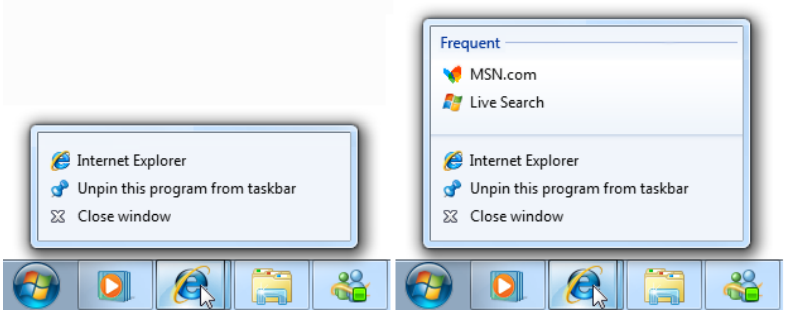
Expand table

Usage	Description
Object grow/shrink/appear to change an object's size or state smoothly.	Object changes between states, possibly while moving. transition keeps users oriented during changes. windows vocabulary: morph, change size, object slides in or out.

Usage	Description
	 In this example, the Weather Gadget morphs from its concise state to display its Options dialog box.
Content show/hide/change to show, hide, or change content smoothly, typically for progressive disclosure.	Window interior reshapes to display more, less, or different content. transition keeps users oriented during changes. windows vocabulary: pane slides in or out. flyout windows fade in and out. different content fades or rolls in.  The Windows Calculator has a smooth transition between view modes.
Control or affordance show/hide to smoothly show or hide controls or their affordances on hover or mouse move in order to simplify the normal visual appearance.	Display controls when users are hovering the pointer over a command area, or display affordances when users are hovering over a control. hovering over these areas indicates that the user intends to interact. affordances may hide if the pointer becomes stationary.

Usage	Description
	In this example, the Windows Media Player controls fade in on hover when in full screen mode.
Scene transitions to make a scene transition smooth and seamless in order to avoid attention.	Abrupt scene changes can be jarring, especially for large screen areas, so use scene transitions to create smoothness and continuity, and to provide context. scene transitions are designed to be natural and low key, to avoid calling attention to the process of transition itself. windows vocabulary: fade in/out; cross fade; sliding in/left, out/right, up, down; pushes and covers.  In this example, the Windows desktop wallpaper gently cross-fades between images to make the transition feel smooth and controlled.
Special scene transitions to draw attention to a scene change to make it special or refocus the user's attention.	While most scene transitions should not call attention to the process of transition, some are designed to break the flow and draw attention in order to emphasize that something different is about to happen. to draw attention, special scene transitions are designed to be unnatural and have high visual impact.  In this example, PowerPoint uses attention-getting transitions to draw the audience into the change.
Direct manipulations to show the effect of direct manipulations (such as move, scroll/pan, rotate, and zoom).	The transition shows the effect of the manipulation in real time. the effect should feel smooth, continuous, and consistent with the real world. moving and rotating may not be continuous in some places to indicate restrictions or likely preferred choices. zooming makes content larger or smaller, possibly changing the level of detail accordingly.  In this example, Magnifier zooms smoothly between levels.

Usage	Description
Incorrect direct manipulations to indicate that a direct manipulation (such as move, scroll/pan) was attempted but could not be done.	The transition shows the manipulation being attempted, but reverts back to the original state. often the effect looks like the manipulation can't be performed due to some real-world physical restriction. these animations are used instead of text-based error messages, which would disrupt the real-world feel of the manipulation. windows vocabulary: bounce  In this example, the document bounces to show that the user has reached the end.
Sort, filter, reorder transitions to indicate that the presentation or content of a collection of items has changed.	The transition shows (or for complex changes, suggests) the effect of the change.

Usage	Description
	 in this example, bing visual search uses a filter transition.  In this example, Windows Media Center uses a reorder transition as a special experience while a song is playing.
Performance transitions to make an action appear to happen faster.	While any transition has the potential for making an action appear to happen faster, the primary purpose of these transitions is to improve the perception of performance and responsiveness. a good technique is to show the task being performed in deliberate steps. by contrast, delaying the action, rendering the results in a haphazard manner, or using an activity indicator will feel slow. windows vocabulary: perform action in stages, with smooth transitions between the stages.  In this example, a taskbar Jump List immediately displays the standard items, and then slides out to display the destinations once the list is ready. Doing so disguises the time required to build the list. By contrast, delaying the initial display would feel unresponsive, and displaying an incomplete list or progress feedback would feel much slower.
Special experiences to engage and delight users during infrequent, special experiences that are important to your program and have the user's full attention.	While any transition has the potential for being a special experience, these transitions are best reserved for infrequent experiences that are truly special to your program. custom transitions are used to give a special feel. branding and personality are often important design elements. unlike other patterns, special experiences can demand attention, be heavy, and require users to wait a moment. consequently, these transitions wear out quickly if overused because the experience

Usage	Description
	is no longer special. In this example, Windows Media Center displays an animation while loading to immediately engage users.

Guidelines

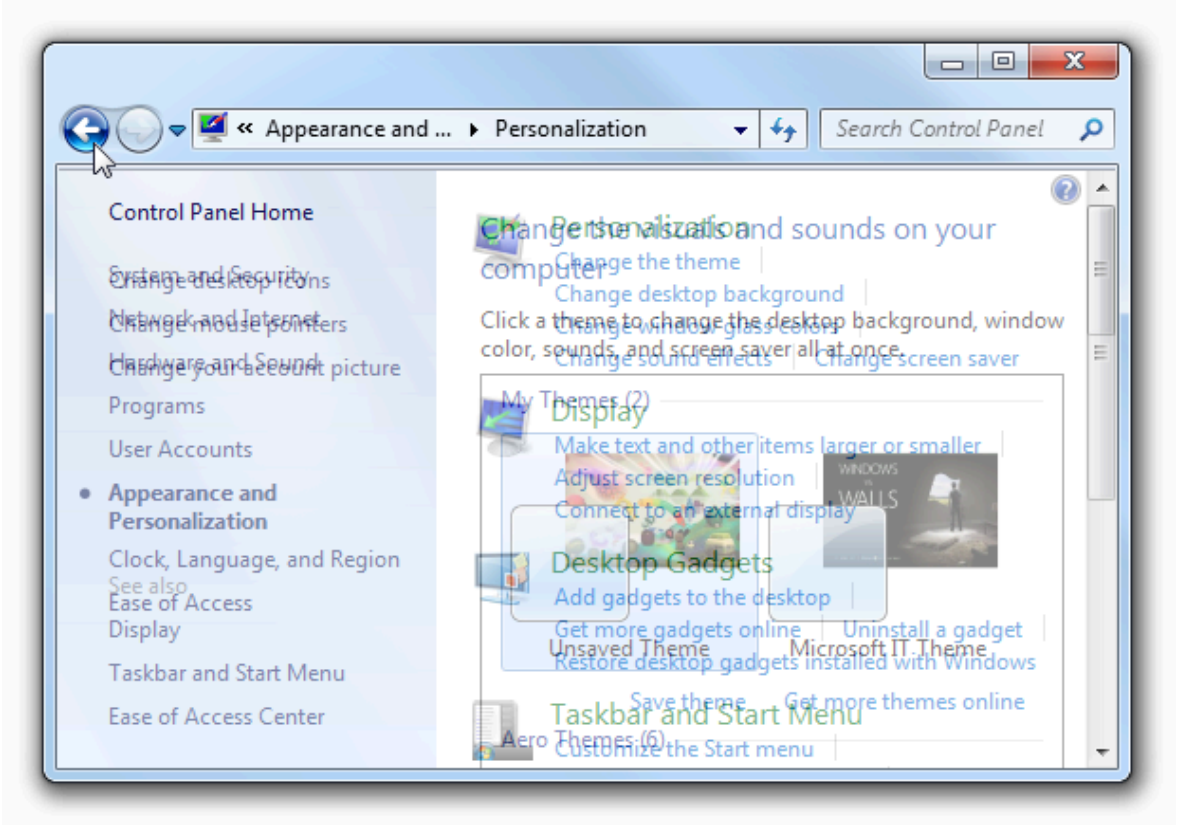
Effective communication

- **Define and use an animation vocabulary** to ensure that your animations and transitions have a consistent meaning, and use it consistently throughout your program. Most vocabularies should include entries for scene and object appearance and disappearance, navigation, basic interaction (hovering, selecting, clicking), object manipulation and interaction (moving, dropping, resizing, scrolling, panning, zooming, rotating, filtering), and attracting attention. Consistent meaning is crucial to effective communication.

- **Whenever practical, use the Windows animation vocabulary.** While your program may have a different audience and different needs, often the benefits of consistency and familiarity outweigh the benefits of being different. If your program's vocabulary must be different, use the same basic animation types as Windows, but give them the right personality for your program.

- **Don't assign specific meanings to generic animations and transitions in an animation vocabulary.** Generic transitions like fades and special effects like dissolves have no particular meaning (beyond appear or disappear), so they can be used freely.

  Incorrect:



In this example, a cross-fade is incorrectly used to navigate to the next item. Because cross-fades have no particular meaning, this transition doesn't provide context.

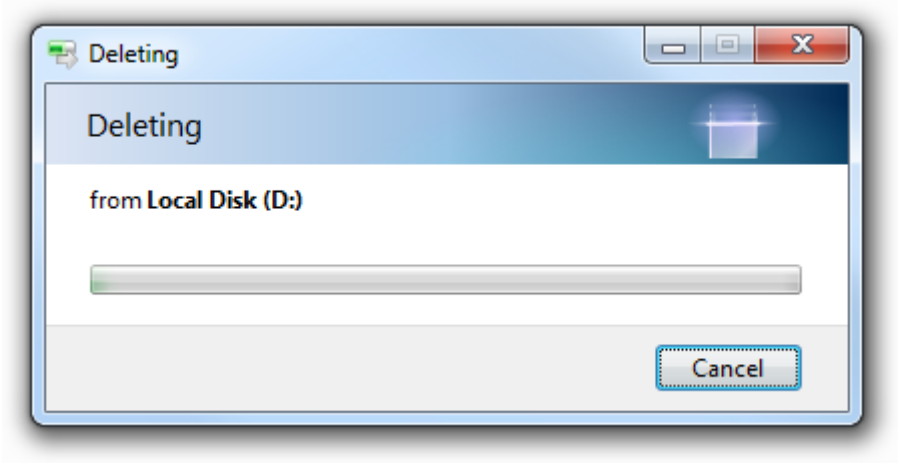
- **Make vocabulary entries clearly distinct.** Related actions may have similar effects (for example, zooming in and zooming out should have inverse transitions), but unrelated actions should have clearly distinct effects (for example, zooming should never be confused with rotating).

- **Keep real-world effects realistic and consistent.** If you use realistic animations and transitions, keep the experience consistent with the real world. Users should never be surprised, confused, or mislead by the results. And for consistency, don't mix metaphors.

- **Give inverse actions inverse animations.** Doing so meets user expectations and simplifies the vocabulary. For example, if a pane appears by sliding in, remove it by sliding out not with some other effect.

- **Make animations comprehensible.** Users should be able to understand quickly the purpose of an animation. It's possible to make an animation too small, too brief (less than 50 milliseconds), or so subtle that users aren't able to comprehend their purpose. In such cases, either redesign to make the meaning clear, or remove.

  Incorrect:



In this example, the effect is so small and subtle that few users can comprehend its purpose. Better to redesign or remove.

Patterns

Hover feedback

- **To appear responsive, strive to play animation within 50 milliseconds of entering or leaving the hover state.**
- **To appear fast, make the duration of hover animations less than 50 milliseconds.**
- **Use a fade in/fade out of hover effect.** Doing so makes hover effects clearly distinct from click and selection feedback.

Click feedback

- **To appear responsive, strive to play animation within 50 milliseconds of click down event.** Click up events don't need click feedback.
- **To appear fast, make the duration of click animations less than 50 milliseconds.**
- **Use a background flash or blink effect.** Doing so makes click effects clearly distinct from hover and selection feedback. Because clicking requires hovering, make click feedback a smooth addition to hover feedback.

Selection feedback

- **To appear responsive, strive to play animation within 50 milliseconds of selection or deselection.**
- **To appear fast, make the duration of selection animations less than 50 milliseconds.**
- **Use a fade in/fade out selection rectangle effect.** Doing so makes selection clearly distinct from hover and click feedback.

Progress feedback

- **Use an activity indicator when an action can't be performed within a second.** Doing so indicates that the command has been received.
- **Use a progress bar when a task will take more than five seconds.** For more guidelines, see Progress Bars.
- **Use progress feedback animations that help users visualize the effect of long-running tasks.** Avoid unnecessary progress feedback animations if an animation doesn't communicate anything helpful, use a progress bar instead.
- **Have clearly identifiable completion and failure states.** Users must be able to determine these final states quickly.
- **Stop showing progress when the underlying task isn't making progress.** Users need to be able to determine if progress isn't being made, and react accordingly.

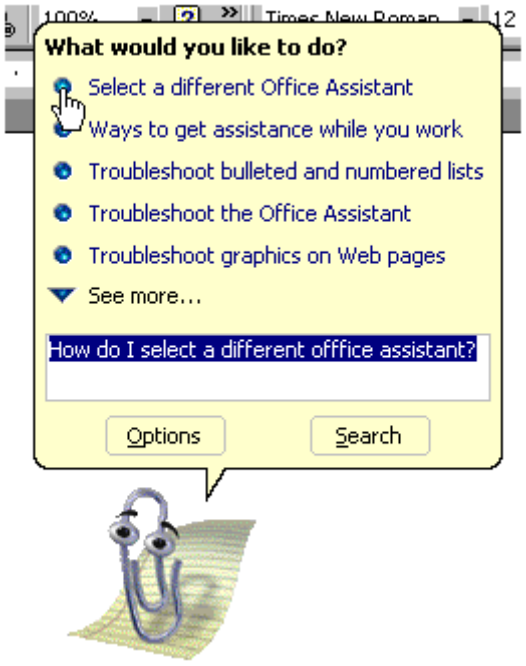
Attractors

- **Use attractors with restraint.** Unless the information is urgent, critical, or otherwise likely to affect the user's immediate behavior, it's usually better to change state inconspicuously and let users discover the change on their own. Solve distractions, not discoverability.

  In this example, the wireless network notification area icon uses an animation for critical problems, but lets users discover weak signals on their own.

- **Choose an animation that draws the right level of attention.** Attractor animations should draw just enough attention to themselves to fulfill their purpose, but no more. If the user must act immediately, choose an effect that demands attention no matter where the user is looking. For other situations, refer to the Attracting the right level of attention section to get the right combination of attention, noticeability, and urgency.

  Incorrect:



The Microsoft Office Assistants attracted unnecessary attention to themselves.

- **If the user doesn't respond, don't repeat the animation or use continuous animations.** Instead, assume that the user chose not to act now, but may act later. Continuous animations make it difficult for users to concentrate on anything else.

Relationship animations

- **Use relationship animations to show where objects came from or where they went.**
- **Relationship animations must start or end with the selected object.** Don't show relationships between objects the user isn't currently interacting with. If users notice at all, what they'll notice is the distraction.

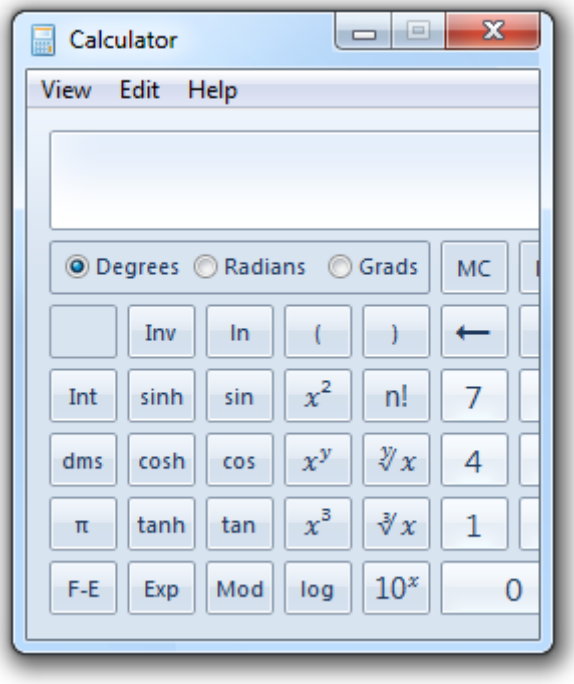
Illustrations/previews

- **Use previews to show the effect of a command without users having to perform it first.** By using helpful previews, you can improve the efficiency and ease of learning of your program, and reduce the need for trial-and-error.
- **Use illustrations and previews that have a clear interpretation.** They have little value if confusing.
- **Play only one illustration at a time** to avoid overwhelming users. If multiple simultaneous illustrations are possible, use mouse hover or a play button to let users indicate their interest.
- **Play an illustration automatically if it is the main purpose of the window or page.** Otherwise, if it's optional, let users play it when they are ready.
- **Play animations at the optimal speed**: not so fast they are difficult to understand, but not so slow they are tedious to watch.

Object grow/shrink

- **Don't clip content during a resize.** Expand containers before adding content. Remove content before reducing containers.

  Incorrect:

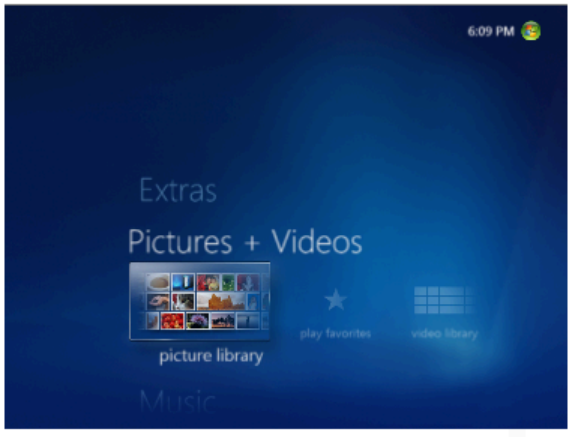
  In this example, content is clipped during a resize.

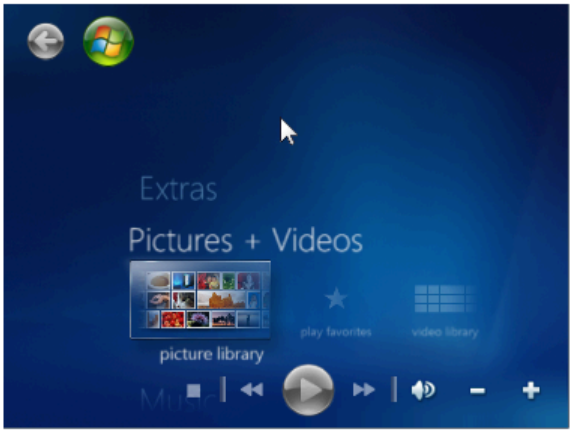
Content show/hide/change

- **Display important information statically.** Users shouldn't have to access important information through progressive disclosure.

Control or affordance show/hide

- **Display important controls when the user positions the pointer anywhere within the window or pane, or, if full screen, on mouse move.** Users shouldn't have to hunt for these controls, so make their discovery certain.

In this example, Windows Media Center displays its controls whenever the pointer is over the window.

- **Display secondary controls or control affordances when the user positions the pointer on or near the commands.** For easy discoverability, make the location obvious and the target large.

In this example, Windows Live Messenger displays a secondary command when the pointer is near the upper right corner.

Scene transitions

- **Make physical scene transitions consistent with natural mapping.** People read from left to right in Western cultures, and hierarchical diagrams flow from top to bottom. Consequently, going forward in time is indicated by left-to-right movement. The following physical scene transitions have natural mapping:

Expand table

Transition	Meaning
From left	Move back in task flow
From right	Move forward in task flow
From top	Move up task hierarchy
From bottom	Move down task hierarchy

- **If your program plays sound, design scene transitions and audio transitions together.** For example, if a scene fades out gradually, any sound should fade gradually as well. Don't ruin seamless visual transitions by having abrupt sound transitions. For more sound guidelines, see Sound.

Direct manipulations

- **When using physical gestures in the interaction (like tossing), design the animation to feel like a natural response to the gesture.** Link the interaction cause with the transition effect. Give the animation real-world physical characteristics

such as acceleration, deceleration, momentum, resistance, weight, bounce, and rotation.

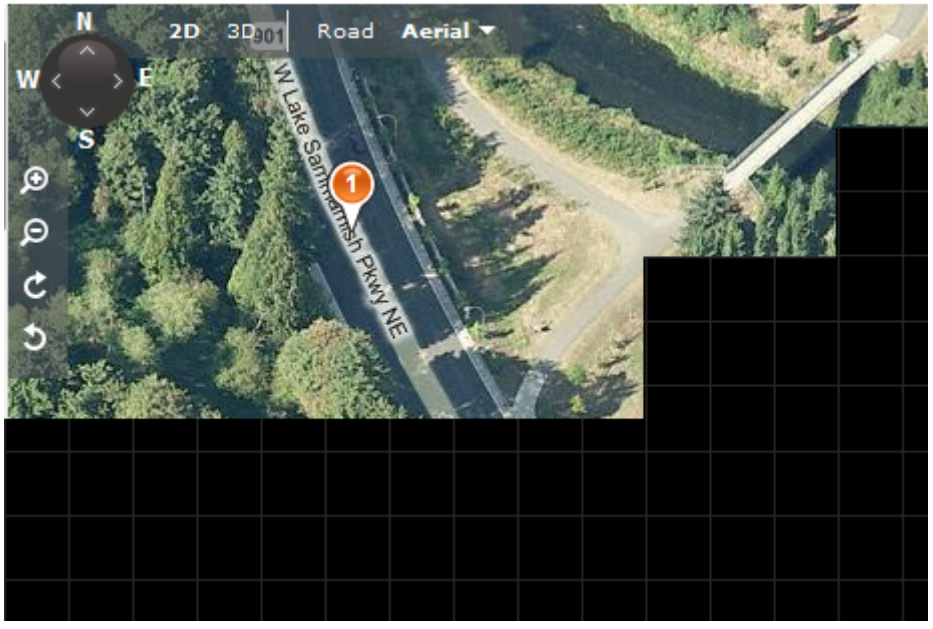
- **To maintain a direct feel, keep an object's contact points under the pointer smoothly throughout the interaction.** Any lag, choppy response, or loss of contact destroys the perception of direct manipulation. Objects should never disappear while being manipulated.

Sort, filter, or reorder transitions

- **For simple changes, show the entire transition.** Users will be able to follow the entire transition easily. Simple changes involve four items or fewer.
- **For complex changes, emphasize the end of the motion as it slows down, and let the eye fill in the rest.** Doing so makes the motion feel much more responsive and orderly.

Performance transitions

- **Consider performing slow transitions in two or three stages to make them appear faster and immediately interactive.** Use the following composition order when appropriate:
  - External frame
  - Background
  - Initial content (using a temporary representation if necessary)
  - Primary controls (so that users can interact immediately)
  - Secondary controls and any remaining UI elements
  - Final content (if a temporary representation was used) Use transitions like fades and slides to make the composition appear smooth, orderly, and refined.



When scrolling in "Bird's eye" view, Bing maps display a temporary grid background. Doing so allows users to continue to scroll immediately, well before the final content is rendered.

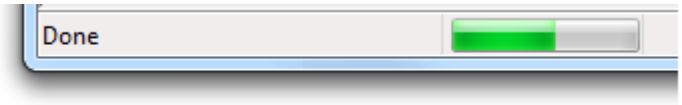
Special experience animations

- **Reconsider animated splash screens (as well as static splash screens).** Often splash screens just draw attention to how long a program takes to load, and they wear out their welcome quickly. While splash screens are acceptable if they are displayed only when user interaction isn't possible, whenever practical a better alternative is to design your program so that users can interact with it immediately, even while it is still loading.
- **Provide a Skip Introduction command if an animated splash screen takes more than three seconds.** Clicking anywhere on the splash screen should also dismiss it. Alternatively, use a short version of the animation after an initial period.

Performance

- **Don't make users wait for your program's animations and transitions.** Use brief animations and transitions (less than 200 milliseconds) whenever practical. Use faster animations (100 milliseconds) for more frequent operations. Design longer animations (more than one second usually the progress feedback, illustration, and special experience patterns) so that users can continue to work while they are running.

- **Design long-running animations to make it clear to users that they can interact while the animation is running.** Users won't attempt to continue to work if the visual clues suggest that they can't.

  In this example from Windows Internet Explorer, the low-key progress bar in the status bar suggests that users don't have to wait for completion before they can interact.

- **Use lightweight animations for CPU-intensive tasks.** Doing so gives full processing power to the task. Furthermore, users won't perceive that the lightweight animation is the reason why the task is CPU-intensive.

- **Don't display an activity indicator during an animation or transition.** Doing so destroys the effect. Design animations and transitions so that they are able to start right away.

- **Design animations to degrade gracefully whenever there are insufficient system resources.** Animations can degrade by having variations that require fewer resources (such as shorter lengths or lower frame rates), or even not running at all. Regardless of the resources available, make sure the animations have high quality and look like animations instead of software bugs.

  Incorrect:

  In this example, the window restore transition is used even though there aren't sufficient system resources to play it well. Consequently, the frozen frame appears to be a bug. If the resources aren't available, it's better to just display the window without a transition.

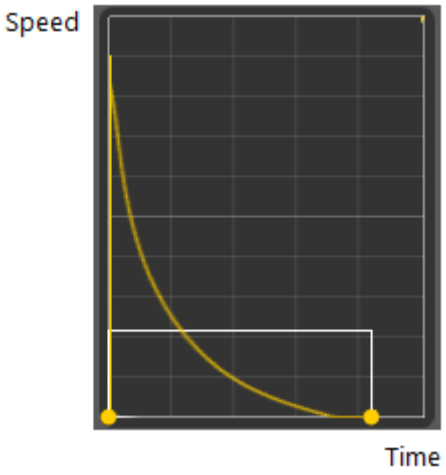
Animation characteristics

Well-designed animations and transitions generally have these characteristics:

- **Brief duration.** Most animations should be between 100 and 300 milliseconds, preferably either 1/6 second (167 milliseconds) or 1/4 second (250 milliseconds). (Special experiences and progress feedback can be longer.) Use faster animation times for more frequent operations. Generally, longer animations take more time to complete, take more time to understand, and feel slow.

- **Responsiveness.** Animations should start within 50 milliseconds of the initiating event or user action. Longer start times feel unresponsive.

- **Acceleration/deceleration.** To look natural, most animation effects need to accelerate when starting and decelerate when stopping. To look responsive, design animations to have fast starts. To appear controlled, design animations to have soft landings at the end. While this applies to motion effects, it also applies to any effect that suggests movement, such as zooms and even fades.



  Most animations should have fast starts and soft endings to have a responsive, yet controlled feel.

- **Motion.** Animations portraying motion in particular need to accelerate and decelerate, so don't use linear motion unless animation duration is very short. Motions should take the shorts path from beginning to end, without overshooting. The full motion path is not always required. When appropriate, emphasize the end of the motion as it slows down, and let the eye fill in the rest. Doing so makes the motion feel much more responsive and orderly. When animating the motion of several objects simultaneously, give them slightly different paths with slightly different timings to feel more natural.

- **Frame rate.** Most animations should use a frame rate of 20 frames per second. If the animation is for a special experience or is related to the main purpose of the program, consider using a higher rate of 24 30 frames per second to improve smoothness and realism.

- **Scale.** Design animations to work well across their entire range of intended usage. For example, page transitions should be designed to work for all page sizes.

- **Personality.** Design animations to feel natural, subdued, and efficient rather than artificial, whimsical, or slow.

Animated text

- While you may display text using a transition, **don't continuously animate text.** Animated text is often distracting and more difficult to read than static text.
  Exceptions:
  - You may animate text in situations where it is traditionally animated, and you provide an accessible alternative.
  - You may animate text if the purpose of the text is primarily decorative.

In this example, Zune animates text but its purpose is primarily decorative. There isn't a problem if users don't carefully read the text.

Reducing power consumption

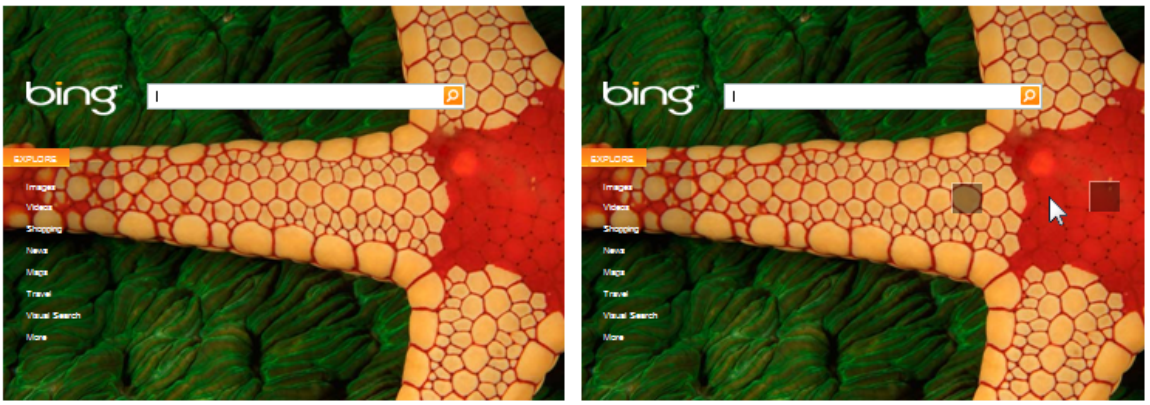
- **Design your animations to reduce power consumption.** When designed properly, animations shouldn't increase power consumption significantly. To reduce power consumption:
  - **Stop animating when the display is off.** The display may be off for the purpose of saving power.
  - **Don't use long-running animations that aren't user initiated.** Animations that use high-resolution periodic timers reduce the efficiency of processor power management. Also, be sure to disable any high-resolution periodic timers when the animations are complete.
  - **Suspend all animations when the system becomes idle.** The period of user inactivity to become idle is determined by Power Options in Control Panel.

Accessibility

- **Don't use animation as the only way to convey essential information.** Animations should communicate information that is useful but not critical, because they aren't

accessible to users with visual impairments.

- **Make sure equivalent information is available through other means,** such as:
  - **By inspection.** Users can determine equivalent information by looking at the screen or objects involved in the animation.
  - **By simple interaction.** Users can determine equivalent information by hovering, clicking, or double clicking.



The Bing home page has an initial animation that reveals several hot spots. Users can also display the hot spots by moving the cursor near them.

Note that "equivalent information" doesn't mean identical information. The information might be in a different format or require simple deduction.

- **When appropriate, set input focus on the object changed during a transition.** Doing so enables assistive technologies to detect where the change happened. But don't change input focus when the user is using the keyboard.

- **Don't use animations or transitions that flash or resize objects quickly.** Flashing and rapid screen changes can cause problems for people with seizure impairments and other neurological disorders.

- **Allow users to turn off your program's animations and transitions.** To support this ability, respect the Turn off all unnecessary animations option in the Ease of Access Center in Windows.

  Developers: You can determine if animations are enabled using the SystemParametersInfo API.

- **Design tasks assuming that users will turn your program's animations off.** Make sure that doing so doesn't disrupt the task flow significantly.

For more accessibility guidelines, see Accessibility.

Documentation

- Avoid referring to animations whenever possible. Instead, refer to the object being animated and, if necessary, the type of animation.
- Don't refer to transitions, except in technical documentation. Instead, refer to the object in its final or initial state.
- If the user explicitly initiates an animation, use the verb play; otherwise use the verb use for technical documentation.

Examples:

- You'll know that an item needs your attention when its icon starts bouncing.
- First, select the photos that you would like to print (note that the photos are enlarged upon selection).
- Use a cross-fade transition to change an object's state seamlessly.

Feedback

Was this page helpful? Yes No

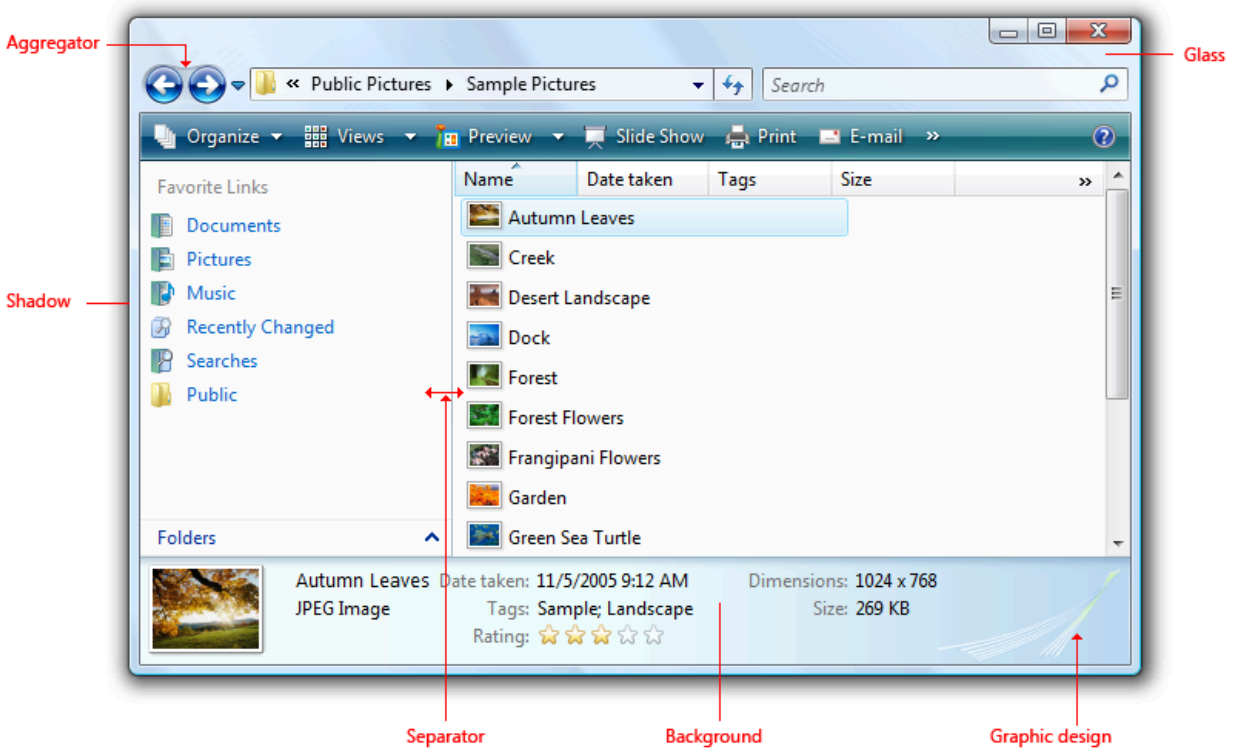
Provide product feedback | Get help at Microsoft Q&A

Graphic Elements

Article • 06/03/2021

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

Graphic elements show relationships, hierarchy, and emphasis visually. They include backgrounds, banners, glass, aggregators, separators, shadows, and handles.



An example with several types of graphic elements.

Graphic elements are usually not interactive. However, separators are interactive for resizable content and handles are graphics that show interactivity.

Note: Guidelines related to group boxes, animations, icons, and branding are presented in separate articles.

Is this the right user interface?

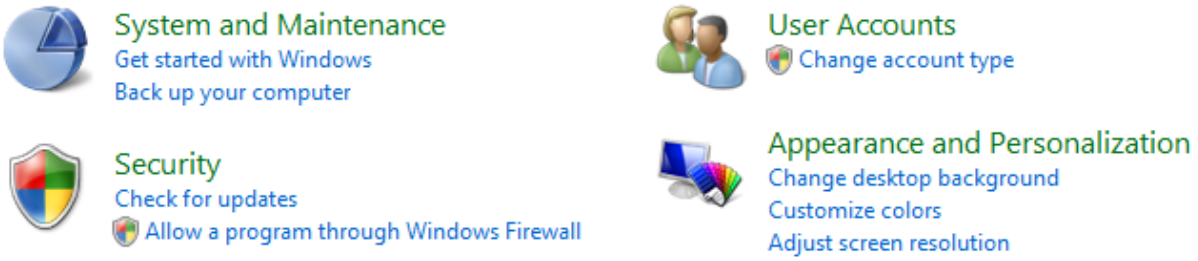
While graphic elements are a strong visual means of indicating relationships, overusing them adds visual clutter and reduces the space available on a surface. They should be

used sparingly.

A design trend in Microsoft Windows is a simpler, cleaner appearance by eliminating unnecessary graphics and lines.

To decide whether a graphic element is necessary, consider these questions:

- **Is the design's presentation and communication just as clear and effective without the element?** If so, remove it.
- **Can you effectively communicate the relationships using layout alone?** If so, use layout instead. You can place related controls next to each other and put extra spacing between unrelated controls. You can also use indenting to show hierarchical relationships.



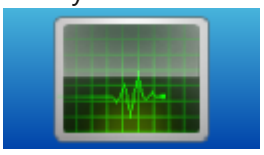
In this example, layout alone is used to show control relationships.

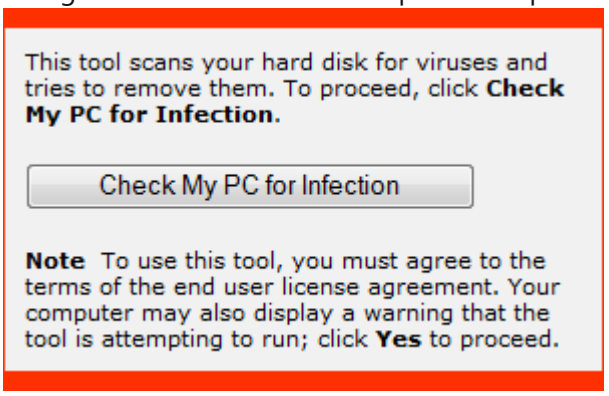
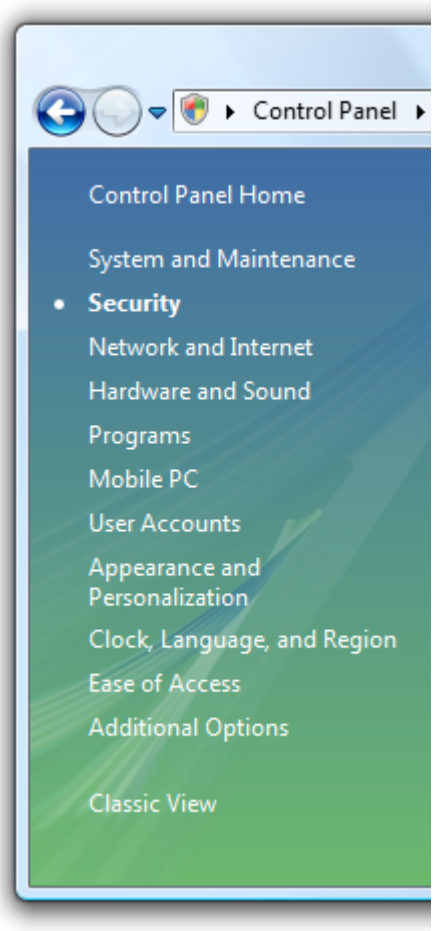
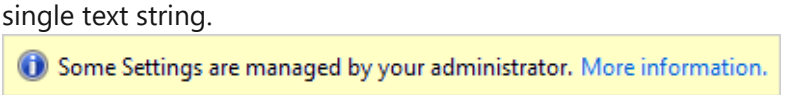
- **Is the communication effective without text?** If not, use a group box, labeled separator, or other label.

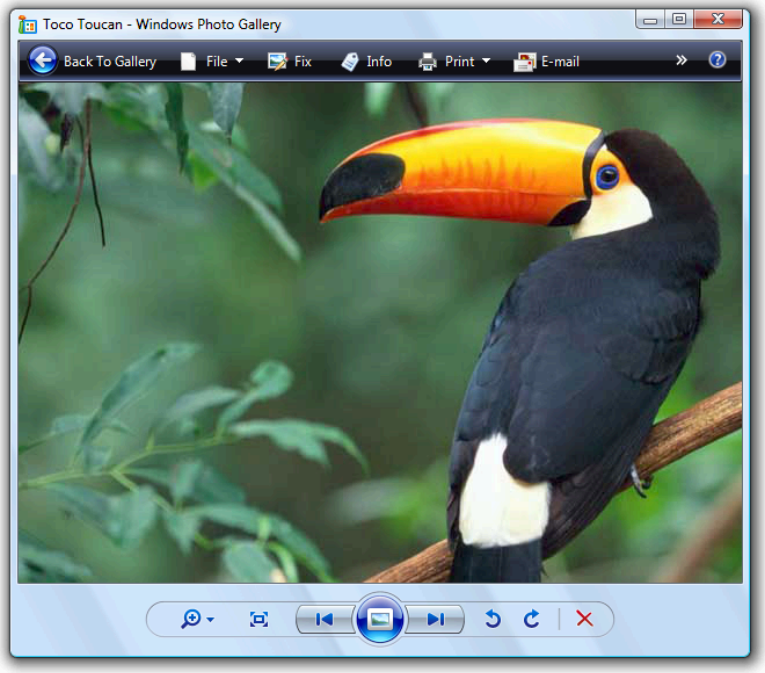
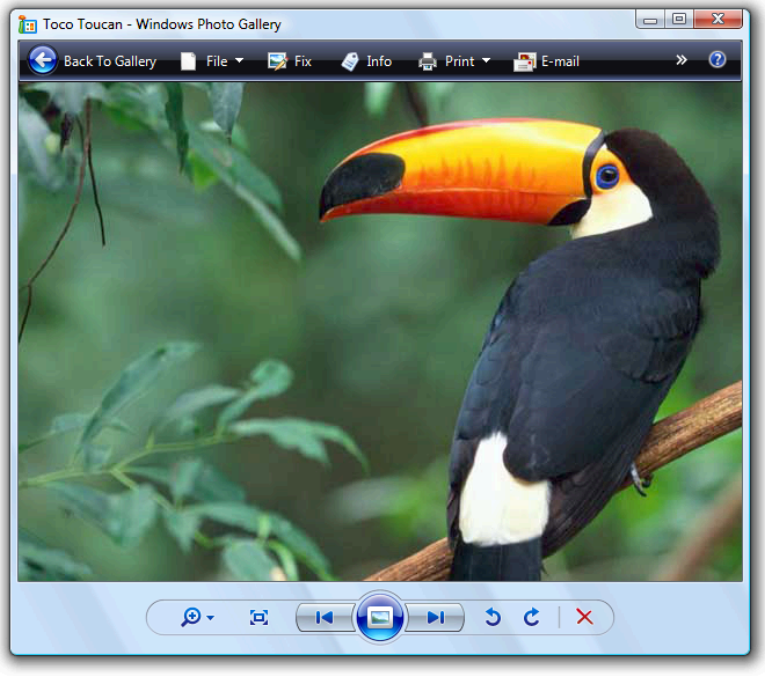
Usage patterns

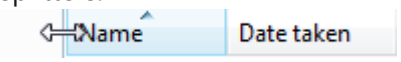
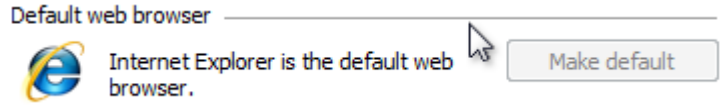
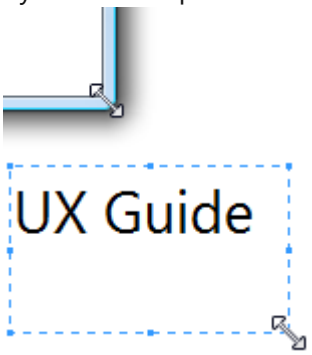
Graphic elements have several usage patterns:

Expand table

Element	Description
Graphic illustrations use to communicate an idea visually.	Graphic illustrations are similar to icons except that they can be any size and usually aren't interactive.  In this example, a graphic illustration is used to suggest the nature of a feature.

Element	Description
Backgrounds use to emphasize or de-emphasize different types of content.	Backgrounds can be used to emphasize important content.  in this example, a background is used to emphasize an important task. backgrounds can also be used to de-emphasize secondary content.  In this example, secondary tasks are de-emphasized by locating them in a task pane.
Banners used to indicate important status.	In contrast to backgrounds, banners emphasize primarily a single text string.  In this example, a banner is used to indicate that the page's settings are controlled by Group Policy.

Element	Description
Glass use strategically to reduce the visual weight of a window.	Glass can reduce the weight of a surface by focusing on the content instead of the window itself.  In this example, glass focuses the user's attention on the content instead of the controls.
Aggregators use to create a visual relationship between strongly related controls.	 in this example, an aggregator background is used to emphasize the relationship between the back and forward buttons in explorer.  In this example, a boundary aggregator is used to emphasize the relationship among the controls, and make them feel like a single control instead of eight.

Element	Description
Separators use to separate weakly related or unrelated controls.	Separators can be either interactive or non-interactive. interactive separators between resizable content are known as splitters.  in this example, an interactive separator is used for resizable content.  In this example, the separator isn't interactive.
Shadows use to make content stand out visually against its background.	 In this example, shadows make the artwork stand out against the background.
Handles use to indicate that an object can be moved or resized.	Handles are always interactive and their behavior is suggested by the mouse pointer on hover.  In these examples, handles indicate that an object can be resized.

Guidelines

General

- **Don't convey essential information through graphic elements alone.** Doing so presents accessibility issues for users with visual disabilities or impairments.

Graphic designs

- **Graphics are most effective when they reinforce a single simple idea.** Complex graphics that require thought to interpret don't work well. Hieroglyphics are best

left for cave drawings.

Incorrect:

In this example, a complex graphic from Windows XP ineffectively attempts to explain a complex trust decision.

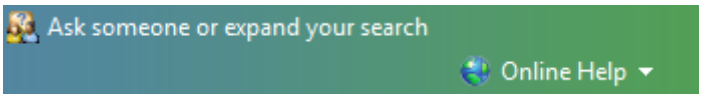
- **Don't use arrows, chevrons, button frames, or other affordances associated with interactive controls.** Doing so invites users to interact with your graphics.
- **Avoid swaths of pure red, yellow, and green in your designs.** To avoid confusion, reserve these colors to communicate status. If you must use these colors for something other than status, use muted tones instead of pure colors.
- **Use culturally neutral designs.** What may have a certain meaning in one country, region, or culture may not have the same meaning in another.
- **Avoid using people, faces, gender, or body parts, as well as religious, political, and national symbols.** Such images may not easily translate or could be offensive.
- **When you must represent people or users, depict them generically;** avoid realistic depictions.

Backgrounds and banners

- **To emphasize content, use dark text on a light background.** Black text on a light gray or yellow background works well.

In this example, the link gets the user's attention because it is on a yellow background.

- **To de-emphasize content, use light text on a dark background.** White text on a dark gray or blue background works well.



In this example, the dark background de-emphasizes the content.

- **If a gradient is used, make sure that the text color has good contrast across the entire gradient.**

- **Always use a 16x16 pixel icon to draw attention to the banner.** Banners are too easy to overlook otherwise. For more guidelines and examples, see Standard Icons.

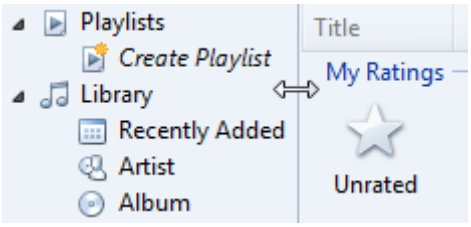
- **Use backgrounds and banners with caution.** While the intent of the background or banner may be to emphasize content, quite often the results are the opposite a phenomenon known as "banner blindness."

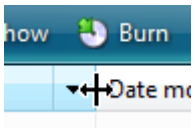
Glass

- **Consider using glass strategically in small regions touching the window frame without text.** Doing so can give a program a simpler, lighter, more cohesive look by making the region appear to be part of the frame.
- **Don't use glass in situations where a plain window background would be more attractive or easier to use.**

Separators

- **Use vertical and horizontal lines for separators.** Be sure to have sufficient space between the separators and the content being separated.
- For separators between sizable content (splitters), display the resize pointer on hover.





In these examples, resize pointers are shown on hover.

Shadows

- **Use only to make your program's most significant content or objects being dragged stand out visually against its background.** Consider shadows to be visual clutter in other circumstances.

High dpi support

- **Support 96 and 120 dots per inch (dpi) video modes.** Detect the dpi mode at startup and handle dpi change events. Windows is optimized for 96 and 120 dpi, and uses 96 dpi by default.
- **Prefer to provide separate bitmaps rendered specifically for 96 and 120 dpi over scaling graphics.** At least provide 96 and 120 dpi versions for the most important, visible bitmaps, and either center or scale the others. Such applications are considered "high-dpi aware" and provide a better overall visual experience than programs that are automatically scaled by Windows.
  - Developers: You can declare a program high-dpi aware (and prevent automatic scaling) setting the dpi aware flag in the program's manifest, or by calling the SetProcessDPIAware() API during program initialization. You can use macros to simplify selecting the right graphics. For Win32 bitmaps, you can use SS_CENTERIMAGE to center or SS_REALSIZECONTROL to scale.
- Check your program in both 96 and 120 dpi for:
  - Graphics that are too small or too large.
  - Graphics being clipped, overlapped, or otherwise not fitting properly.
  - Graphics that are poorly stretched ("pixilated").
  - Text that is clipped or not fitting in graphic backgrounds.

Text

- **For accessibility and localization, don't use any text in graphics.** Make exceptions only to represent branding and text as an abstract concept.

Feedback

Was this page helpful? Yes No

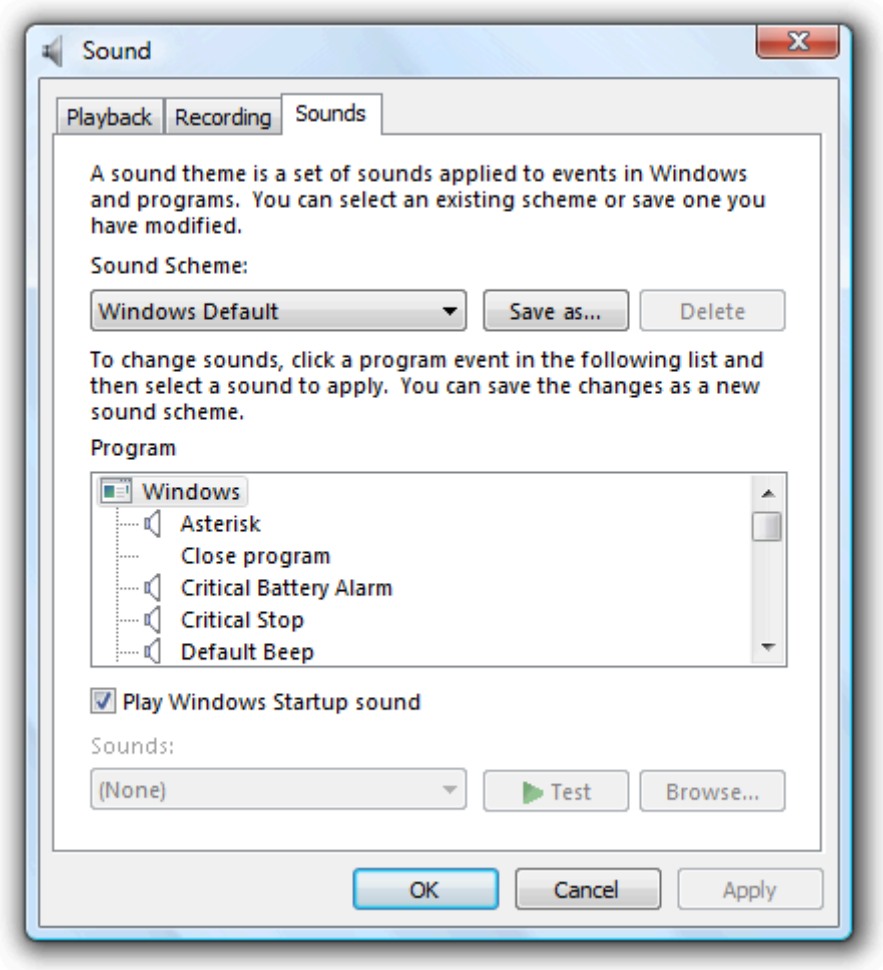
Provide product feedback | Get help at Microsoft Q&A

Sound

Article • 06/03/2021

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

Sound is the audio element of the user experience. When used appropriately, sound can be an effective form of communication that establishes a non-verbal and even emotional relationship with your users. Sounds can be used alone or as a supplement to visual UI. For example, adding a sound effect to a notification increases the likelihood that it will be noticed, especially if the user isn't looking at the screen when an event occurs.



From the Sounds tab of the Sound control panel item, users can make changes to their system sounds.

This article covers the use of sounds within a program as a response to events and user actions, and integrating a program's sound control with Windows. It doesn't cover the use of music or speech.

Note: Guidelines related to notifications and branding are presented in separate articles.

Is this the right user interface?

To decide if you should use sound, consider these questions:

- **Is there a clear user benefit to using sound?** Because the drawbacks of using sound can easily outweigh the benefits, use sound only when there is a clear advantage.
- **Is the use of sound appropriate?** Does the use of sound draw attention to things that are worthy of attention? Would users miss the sound if it were absent? Focus on sounds that keep users informed, are likely to change their behavior, or provide useful feedback.
- **Is the use of sound distracting?** Are there frequent, loud, jarring sounds? Are users likely to reduce the system volume or your program's volume as the result of your use of sound?
- **Are you using sound as a primary form of communication?** In many cases, such as for users who have some level of hearing loss, sound should not be used as the primary means of communication. Sound is more effective as a supplement to other means of communication (such as text or visuals).
- **Are the primary target users IT professionals?** Sound is usually ineffective for tasks targeted at IT professionals because many of their tasks run unattended. Furthermore, sound doesn't scale for them imagine running hundreds of tasks at a time and getting sounds when they complete or fail.

Design concepts

Typically sound achieves any or all of the following purposes:

- **Notification.** Sound can be associated with specific events. For example, a "new mail" sound tells users when mail arrives without disrupting their current task.
- **Feedback.** Sound can provide feedback for specific user actions. For example, a subtle sound that plays when you release the slider on the volume control provides feedback about the level of the current setting.
- **Branding.** Sound can be associated with specific content to brand your product, application, or service. Windows uses sound in this way for the startup of the operating system.

- **Entertainment.** Sound is commonly used to enhance entertainment products and to make any product more engaging. For example, most games, training applications, and consumer products use sound to entertain users and enhance their experience.

Certain sounds can fulfill several of these purposes at once. The Windows Startup sound, for example, indicates that the startup process has completed and the desktop is ready for use. It also provides a powerful form of product branding and even momentarily engages users.

Sounds that fulfill none of these purposes should likely be eliminated.

Inappropriate use of sound

Despite the benefits of sound, appropriate use of sound requires significant restraint to do otherwise can make a program annoying and distracting. Users will turn off their sound completely if they become annoyed by frequent, repetitive, jarring, disrupting, poorly designed sounds; in part this is because by its very nature, sound demands attention and is hard to ignore. For tips on finding a reasonable balance, see the Sound design guidelines.

Because the drawbacks of using sound can easily outweigh the benefits, use sound only when there is a clear advantage. **When in doubt, don't use sound.**

Make sound supplemental

Even if the sound is used appropriately, there are many situations where sound might not be effective for all users:

- Some users may work in a noisy environment where the sounds cannot be heard.
- Some users may work in a quiet environment that requires sound to be turned off or set at a low volume.
- Some users may have hearing impairments or loss.
- The computer may not have speakers.

For these reasons, **sound used for notifications and feedback should never be the only method of communication,** but rather should supplement visual or textual cues.

Desirable characteristics of sound

In general, sounds should be:

- mid to high frequency (600 Hertz [Hz] to 2 kilohertz [kHz]).

- short (less than one second).
- soft or moderate in volume.
- meaningful.
- pleasant, not alarming or jarring.
- non-verbal.
- non-repetitive.

With sound, less is more. **The ideal sound effect is one that users barely notice, but they would miss if it were absent.**

A common misconception is that sounds for critical events need to be loud and jarring to get the user's attention. This isn't true, because sound is really meant to be a supplemental means of communication. In the case of a critical error message, its presentation (perhaps in a modal dialog box), its icon (an error icon), and its text and tone all combine to communicate the nature of the error. An effective error sound can be slightly louder than the typical Windows sound, but need not be significantly louder.

Characteristics of Windows sounds

Beyond this general call for minimalism, the Windows sound aesthetic uses light, pure tones, and glassy and airy sounds, with a soft fade-in and fade-out (soft "edges") to prevent abrupt, jarring, percussive effects. They are designed to feel subtle, gentle, and consonant. Windows sounds use echo, reverb, and equalization to attain a natural, ambient feel.

The default sound scheme for Windows doesn't generally use instrumental or recognizable everyday sounds that are overly specific or musical. Examples of sounds it avoids are musical instruments such as pianos or percussion instruments, animal sounds, environmental noises, speech, voices, movie-like sound effects, or other sounds of humans. Also, Windows sounds are not meant to be perceived as music (that is, as long, multi-note melodies). This makes Windows sounds functionally distinct from other types of sounds.

Because the Windows sounds were professionally designed to have the desirable characteristics and appeal to a broad audience, **consider using these built-in Windows sounds whenever appropriate.**

Designing your own sounds

If you must create your own sounds, design them to have the previously described characteristics. Strive to make them complement their associated tasks or events.

Understand that creating original sounds is difficult to do well especially for sounds intended for a broad audience. Sound can be a polarizing design element. For every user who loves a sound, there will be many who dislike it.

Design the sounds for your program as a group to feel like they are related variations on a theme. Your program's auditory experience should be coordinated with its visual experience. Also, the "tone" of the sounds should be coordinated with the tone of the text. Consider how text with a pleasant, natural tone can be undermined when accompanied by harsh, alarming sounds.

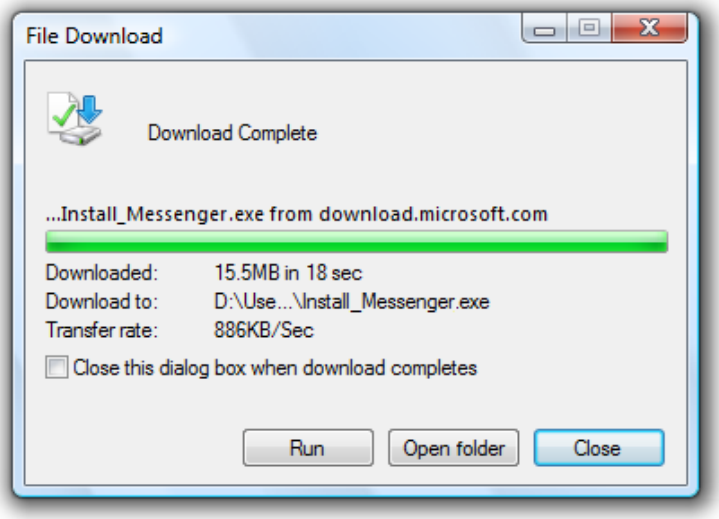
If you do only four things...

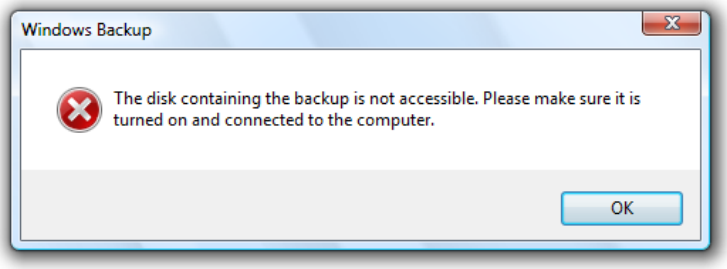
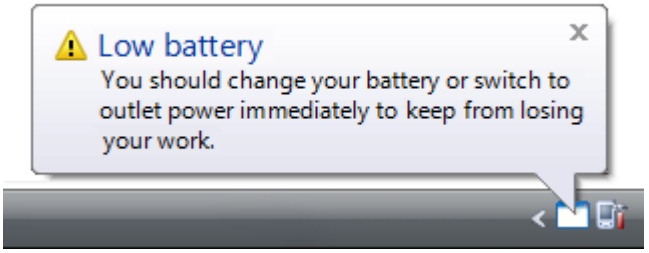
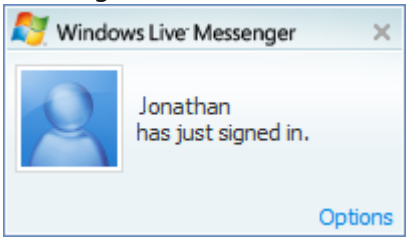
1. Use sound with restraint make sure there is a clear overall user benefit. When in doubt, don't use sound.
2. Use the built-in Windows sounds whenever appropriate.
3. If you design your own sounds, make sure they have the desirable sound characteristics and as a whole feel like variations on a theme.
4. Don't assume that sounds need to be loud and jarring to get the user's attention.

Usage patterns

Sounds have several usage patterns:

Expand table

Sound Use	Example
Action completion sonically notifies users when a long-running, user initiated action completes successfully.	 In this example, the dialog box plays a sound to notify users that the download has completed.

Sound Use	Example
Action failure sonically notifies users when a long-running, user initiated action fails.	 In this example, Windows plays a sound to notify users that the backup operation has failed.
Important system event sonically alerts users of important system events or status that require immediate attention.	 In this example, users are alerted that their low battery requires immediate attention.
FYI sonically notifies users of potentially useful, relevant information.	Because this information usually doesn't require immediate attention, an fyi sound provides subtle feedback without breaking the user's flow.  In this example, a sound plays when a contact signs in to an instant messaging service.
Sound effect sonically provides feedback to user interactions.	Provides real-world or styled sound feedback that is appropriate for the interaction. sound effects often sound as though the user is manipulating a real-world, physical object. sometimes referred to as foley.

Sound Use	Example
	 In this example, the minimize window sound effect sounds like a real-world object is being reduced in size.
Branding sounds a sound provided to enhance the user experience though emotional impact and, as a side effect, promote the product brand.	Branding sounds are best when synchronized to visual events, especially ui transitions such as the display of a program window. true sound brands indicate the source of goods, similar to a trademarked word or logo, and are relatively uncommon. In this example, Windows startup is a branded transitional experience.

Guidelines

Usage

- **Use sound with restraint** make sure there is a clear overall user benefit. Focus on sounds that keep users informed, are likely to change their behavior, or provide

useful feedback. When in doubt, don't use sound.

- **Select the sound and its characteristics based on how it is being used.** For a description of each usage pattern, see the table in the previous section.
- **For notifications and feedback, don't use sound as the only method of communication.** Rather, use sound as a supplemental method to reinforce visual or textual cues. Doing so ensures that users can see the information if they can't hear the sound.
- **Don't play loud or harsh sounds frequently.** Doing so is unnecessary and results in a poor user experience. The more often a sound is played, the less obtrusive it should be. Sounds don't have to be loud or harsh to attract attention.
- **Don't beep.** Beeping isn't appropriate for modern programs. Beeps can't have specific meanings assigned to them, and users can't control them.
  - **Exception:** Critical system functions may beep to alert users of situations that they must attend to immediately, such as critically low battery power.

Playback

- **Don't repeat a sound more than two times consecutively.**
- **For a consecutive sequence of related sound events, play a sound only on the first event.** Avoid using multiple sounds because they may collide with each other or otherwise result in an unpleasant user experience.

Sound selection

- **Choose pleasant sounds.** Don't use unpleasant, alarming sounds, such as buzzing, crashing, and breaking.
- **Use sounds that are short** (less than one second).
- **Use sounds that are roughly the same volume as the typical Windows sound.** Users dislike having to turn the volume down when starting a computer or a program, especially in public environments such as meetings and presentations. The Microsoft Windows sound files are located in the Media folder within the Windows folder.
- **Don't choose sounds that require localization.** You can achieve this by using sounds that don't use speech or have culturally-dependent meanings or connotations.

Windows system sounds

- **Use the built-in Windows system sounds whenever appropriate.**

- **Choose to use system sounds based on their associated meaning, not just on the sound itself.** System sounds must be used consistently.

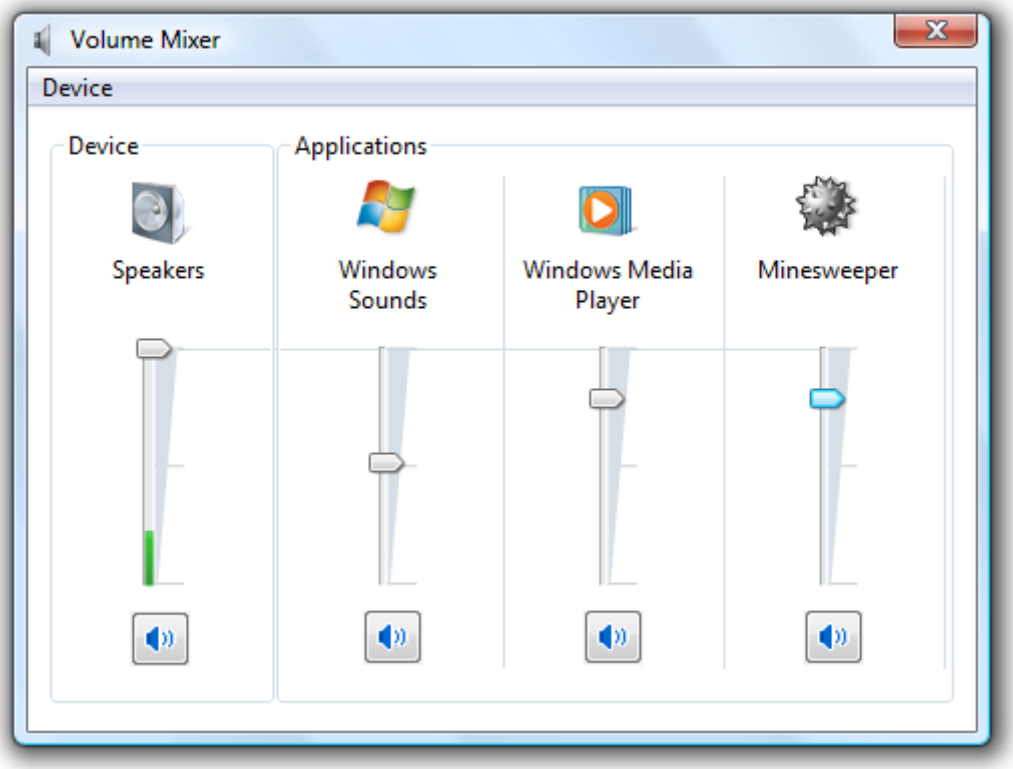
Sound design

When creating your own sounds:

- **Create sounds with the desirable sound characteristics.**
- **Compose sounds with mostly mid-range to high frequencies (600 Hz to 2 kHz).** Don't use low frequencies because they travel farther, are harder to locate, and can be alarming.
- **Set the relative amplitude of normal sounds to the level of the typical Windows sound.** The Windows sounds have been appropriately leveled for home and work environments. Using different levels for your sounds will force users to make volume adjustments.
  - Set important sounds to be slightly louder. Such sounds include action completions, action failures, and important system events.
  - Set frequently occurring sounds to be slightly softer. These include FYIs, branding sounds, and sound effects.
- **Choose sounds consistent with the meaning of the Windows sounds.** To create a custom version of a Windows sound, preserve the same pitch and interval, but change the orchestration or timbre. Don't assign different meanings to sounds with similar pitches and intervals as Windows sounds.
- **Design the sounds for your program to feel like they are related variations on a theme.** Your program's auditory experience should be coordinated with its visual experience.
  - **Design scene transitions and audio transitions together.** For example, if a scene fades out gradually, any sound should fade gradually as well. Don't ruin seamless visual transitions by having abrupt sound transitions.
- **Sounds must be in .wav file format.** The 16-bit, 44.1 kHz stereo uncompressed pulse code modulation (PCM) format is recommended. If file size is important, use compressed or monaural (mono) formats, but be aware that there is an easily discernable quality loss that could reflect badly on your application.

Mixing

- **Don't have volume or mute controls in your program.** Instead, let users control relative volume settings among applications with the Windows volume mixer. If your program has a volume control, there will be multiple places where users adjust their settings, which may lead to confusion.



The Windows volume mixer allows users to control the main volume setting as well as the volume for each program that is currently playing audio.

- **Exception:** If the primary purpose if your program is audio or video playback or creation, it may be useful to have a volume control in the program. Use a slider control for this purpose and provide immediate feedback when the user changes the volume.

Windows integration

- **Register your program's sounds in the Windows Sounds registry.** Doing so allows the Windows volume mixer to add a slider for your program.
- **Register your program's custom sound events.** Doing so allows the Windows Sound control panel item to display them. Create the following key for each custom sound event: HKEY_CURRENT_USER | AppEvents | Event Labels | EventName = Event Name.
- **Don't hardwire the sounds for your program's sound events.** Instead, specify the sounds to be played using registry entries. Doing so allows users to customize the sound events through the Sound control panel item.
  - **Exception:** You can choose to hardwire sounds used for branding.
- **Don't provide a way for users to configure sounds within your program's options.** Rather, use the Windows Sounds control panel item for this purpose.
- **Consider not assigning sounds to frequently occurring events by default.** Don't require users to configure their way out of an annoying initial experience.

DirectSound programming issues

- For DirectSound programs that have their own volume control, **set the program volume to 100 percent by default.** Doing so maximizes the dynamic range of your audio.
- **Don't lock out other sound events by running your program in exclusive mode.** Doing so can prevent other programs from working correctly. For example, using exclusive mode prevents a computer from being used as a telephony device.

Text

- Don't use the phrase sound adapter. Use sound card instead.
- Use device to refer generically to speakers, headphones, and microphones.
- Use controller to refer to audio hardware that controls devices, such as sound cards and chipsets.
- Use the phrase sound scheme to describe a collection of sounds for common program events, such as logging on or receiving new e-mail. Use the phrase desktop theme to describe a collection of visual elements and sounds for your computer desktop.
- Use the term audio to refer broadly to speech, music, and sounds. Use the term sound to refer more narrowly to the program and Windows sounds described in this article.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Experiences

Article • 02/09/2021

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

Experiences are the common experiences and use cases for all apps, like set up, first run, and printing. Learn about the best practices for creating these experiences and communicating your app brand.

In this section

Expand table

Topic	Description
Software Branding	Branding is the emotional positioning of a product as perceived by its customers. Successful branding requires skillful crafting of a product image, and is not achieved just through product logos and color schemes.
Setup	Users don't enjoy installing software, so modern setup experiences need to be simple, efficient, and problem-free.
First Experience	In the ideal first experience, users install your program and use it productively immediately, without answering a bunch of questions or learning a bunch of things.
Printing	Printing is the user experience on paper. It's easy to overlook, but it is an important part of the overall user experience.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Software Branding

Article • 02/09/2021

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

Branding is the emotional positioning of a product as perceived by its customers. Successful branding requires skillful crafting of a product image, and is not achieved just through product logos and color schemes.

Branding is the emotional positioning of a product as perceived by its customers. Product branding is achieved through a combination of factors, including the product name and logo, use of color, text, graphics, and sound, the style of various other design elements, marketing, and most importantly, the attributes of the product experience itself.

Successful branding requires skillful crafting of a product image, and is not achieved simply by plastering a product logo on every surface and using the product's color scheme at every opportunity. Rather, meaningful and high-quality branding that enhances users' experience will be much more successful.

Note: Guidelines related to icons, fonts, color, animation , sound , and window frames are presented in separate articles.

Design concepts

In a competitive marketplace, companies brand their products to help differentiate them from the competition. It would be naive to suggest that product branding in general is wrong or should be avoided, but it is fair to say that software branding is too often executed poorly. The goal of software branding is to associate the brand with the style and quality of the product and its experience. Too often, developers attempt to achieve this by drawing attention to the program itself. The result is to distract users instead of delight them.

Attributes of good software branding

When well done, software branding has these attributes:

- Establishes a clear, distinct style and personality.
- Creates an emotional connection.
- Has high quality.
- Is strategically placed and consistently executed.
- Aligns to the overall brand strategy.
- Is long lasting...as enjoyable the thousandth time as it was the first.

By contrast, poor attempts at software branding have these attributes:

- Has no obvious theme or point.
- Is in the user's face.
- Is annoying.
- Is everywhere.
- Has a custom look and feel without any user benefit.
- Becomes quickly tiresome.

Start with the product itself

Successful software branding starts with the design of product itself. A well-designed program has carefully crafted functionality aimed at an appropriate target audience. Unique functionality and extraordinary attention to detail make powerful branding statements. For more information, see How to Design a Great User Experience.

Carefully choose the product's name

Great product names drive strong brands. A great software product name is memorable and concisely conveys the benefit of the product, providing distinction in a crowded market. Hire a branding professional to help you choose the right product name. In the long term, a well-chosen name is far more important to your branding effort than details like logos, color schemes, and control theming.

What to brand

Software branding elements can be categorized as follows:

Primary

- Product name
- Product logo
- Product color scheme
- Product-specific sounds

Some primary branding elements from Windows Vista.

Primary branding elements tend to draw a lot of attention, so they should be used with restraint. **Limit your use of primary branding elements to a few strategic experiences.** Product-specific sounds aren't recommended for most programs.

Secondary

- Element shapes
- Icon and graphic styles
- Secondary graphic elements
- Accent colors
- Animations
- Transitions
- Shadows
- Backgrounds and transparency

Some secondary branding elements from Windows Vista.

Secondary branding elements tend to be more subtle, and because of that, they can be used more often. While some of these secondary branding elements may not have much impact individually, when taken together they can give your product character and style. Transitions can have more impact than fixed graphics, which users learn to ignore over time. Prefer the secondary level of branding over the primary level.

Tertiary

Finally, there is another category of branding elements to be aware of.

- Custom window frames
- Custom controls

While it's appropriate for certain types of programs (such as games) to create a completely distinct, immersive experience based on custom controls and windows, most programs should use the standard varieties. Having your programs look and act weird doesn't make for a strong brand identity. Rather, your goal should be to create a program with character—a product that stands out while fitting in.

Where to brand

Not everything needs to be branded. A few strategically placed branding elements can make a more powerful impression than slapping uncoordinated branding elements everywhere.

Focus your branding effort on the special experiences in your program. These are the places that have the most emotional impact, such as:

- The first experiences, especially when the program is used for the very first time.
- The main window or home page.
- The start and completion of important tasks.
- Important transitions between tasks or program areas.
- Waiting time during long-running tasks.
- Log in and log off.

Where not to brand

While you can potentially use any element in your program as a branding opportunity, don't use the Windows desktop (including the work area, Quick Launch bar, or notification area) for branding.

The desktop is the user's entry point to Windows. Leave the user in control. Use these entry points appropriately—never view them as ways to promote awareness of your

program or its brand. For more information, see Desktop.

Use branding professionals

Branding is a specialized skill best done by experienced professionals. It is far better to expose your users to minimal branding than to use extensive branding that is annoying and ineffective. Work with your branding and marketing team to create a good end-to-end branding experience.

If you do only five things...

1. Start with the product design. The most powerful branding statement is to satisfy your customers' needs especially well.
2. Choose a good product name that is memorable, distinctive, and concisely conveys the benefit of the product.
3. Think of branding in terms of experiences and making an emotional connection, not product logos and color schemes.
4. Prefer secondary branding elements. Limit your use of primary branding elements to a few strategic experiences.
5. Get help from a branding professional.

Guidelines

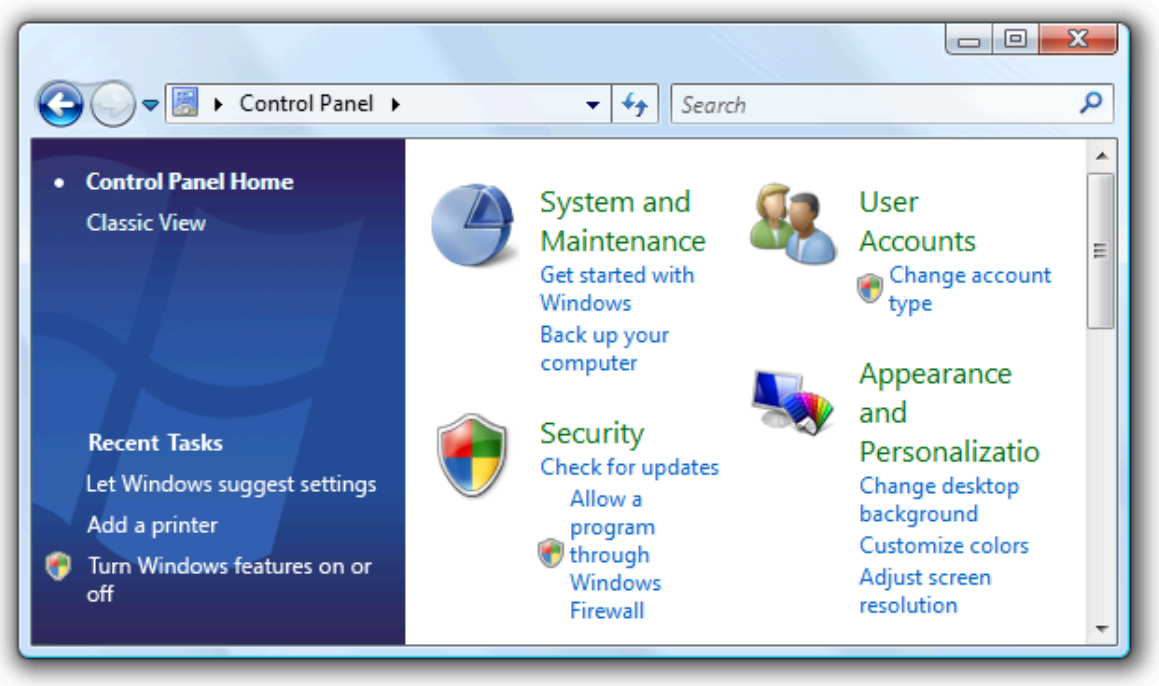
General

- **Choose a good product name that is memorable, distinctive, and concisely conveys the benefit of the product.** This will be the foundation of your brand.

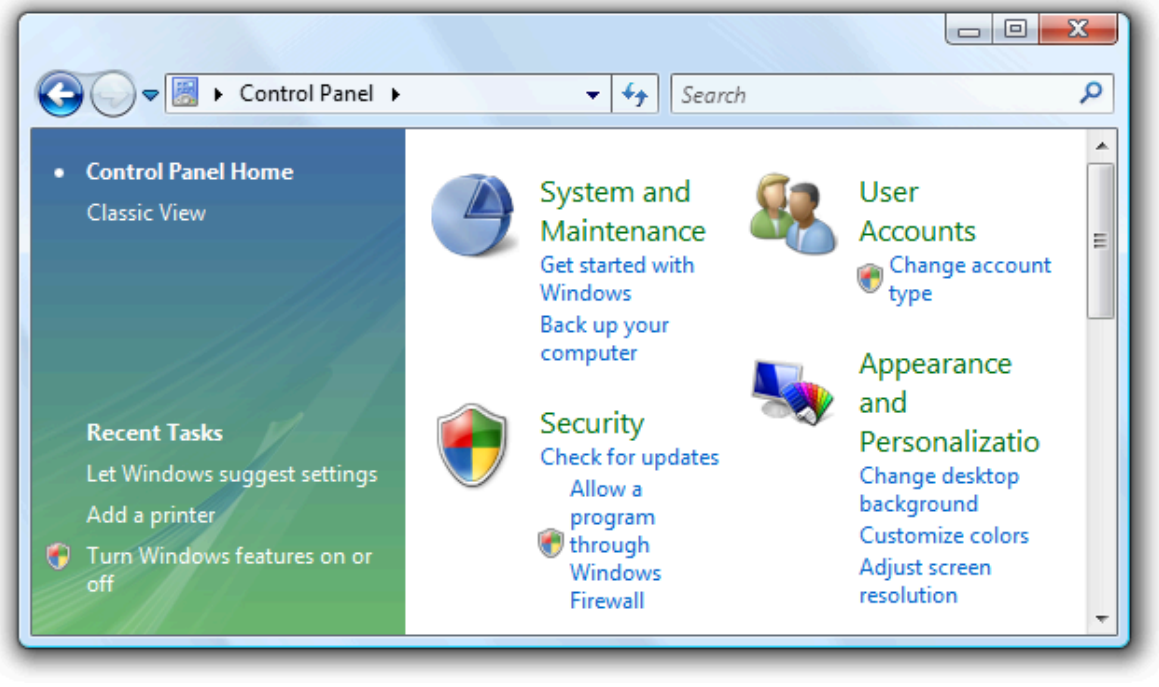
- **Focus your branding effort on the special experiences in your program,** such as:
  - The first experiences, especially during setup and when the program is used for the first time.
  - The main window or home page.
  - The start and completion of important tasks.
  - Important transitions between tasks or program areas.
  - Log in and log off.

- **Prefer secondary branding elements.** Limit your use of primary branding elements to a few strategic experiences. For example, consider using secondary graphics, transitions, and color instead of logos. Also, avoid prominent primary branding elements in places where users spend a lot of time because they may be perceived as clutter.

Acceptable:



Better:



In the better example, a secondary graphic element is used instead of the product logo for Windows control panel items.

- **Don't use branding that is distracting or harms usability or performance.**
- **Don't use the Windows desktop for branding.** For more information, see Desktop.

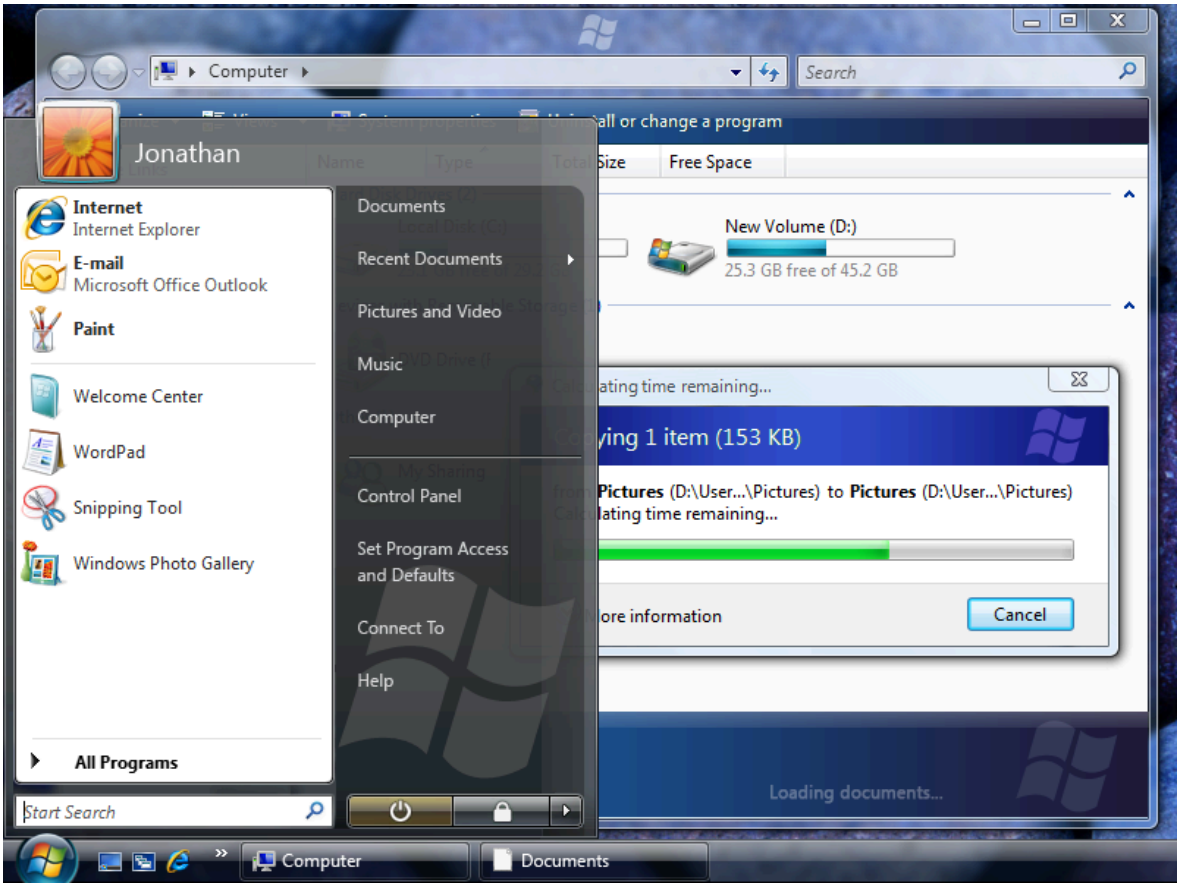
Names and logos

- **Limit the use of product and company logos in the user interface.** Don't plaster company or product logos on every UI surface.
  - Limit product and company logos to at most two different surfaces, such as the main window or home page and the About box.
  - Limit product and company logos to at most twice on any single surface.
  - Limit product and company names to at most three times on any surface.

  Incorrect:

  In this example, the company name is overused.

  Incorrect:

  In this example, while individually the use of the logos is acceptable, the overall effect is overwhelming.

- **Use small product and company logos.** Place the logo out of the user's workflow, and choose a size that is appropriate for its location.

- **Use graphic logos.** Graphic logos are more stable than text logos because they aren't affected by font, text size, language pack, or theme changes.

- **Don't use animated logos.**

Controls

- **Don't use custom controls for branding.** Rather, use custom controls when necessary to create a special immersive experience or when special functionality is needed.

  Incorrect:

  This example shows a custom control incorrectly used for branding.

Splash screens

- **Don't use splash screens for branding.** Avoid using splash screens because they may cause users to associate your program with poor performance. Use them only to give feedback and reduce the perception of time for programs that have unusually long load times.
- **Don't use animated splash screens.** Users often assume that the animated splash screen is the reason for a long load time. Too often, that assumption is correct.

Sound

- **Generally, sound is not recommended just for branding.** If you do use sound for branding:
  - **Play a sound only at program startup,** but only if the program was launched by the user.
  - **Synchronize the sound to a visual event,** such as a UI transition like the display of a program window.

For more information, see Sound.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Setup

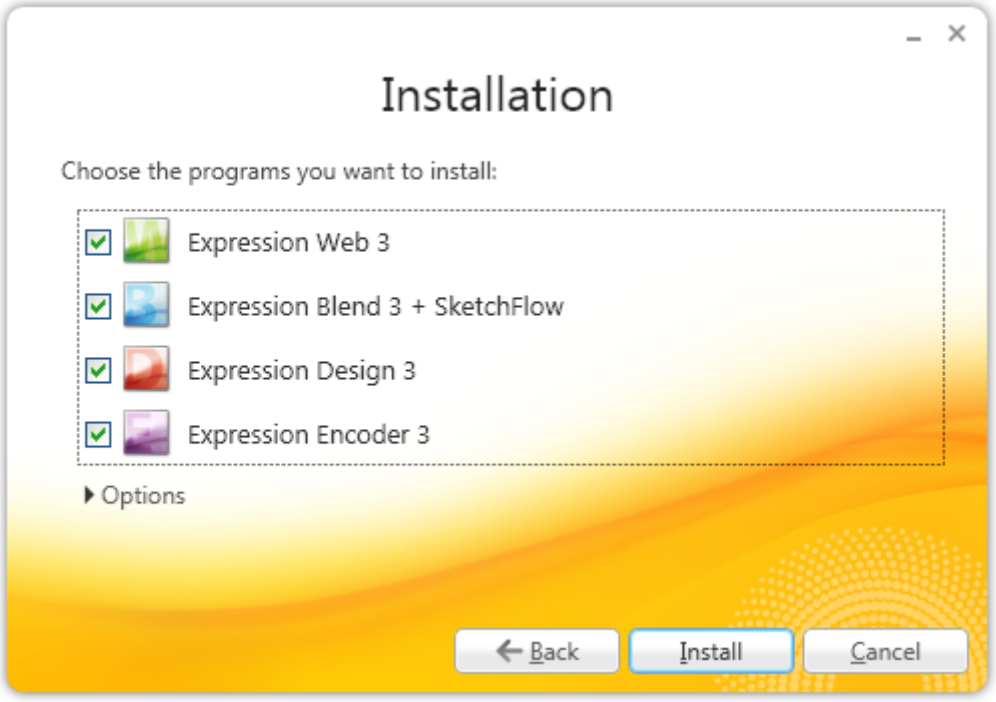
Article • 02/09/2021

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

Users don't enjoy installing software, so modern setup experiences need to be simple, efficient, and problem-free.

Setup usually refers to the experience of installing and initially configuring a program. However, setup can also refer to entire installation life cycle, including initial installation, incremental program updates (such as version upgrades or service packs), repairing, and uninstalling.

Most users regard setup as a necessary evil, to be performed as quickly as possible. The point of installing the program is to use it, not to make innumerable decisions about configuration and usage, or, worse yet, to spend a lot of time answering personal questions used for registration or marketing purposes.



A streamlined setup experience.

The setup experience combined with the first use of the program is known as the first experience. Your program should provide a streamlined first experience for users. Each

question or step that isn't necessary or could be postponed delays them from using your program. Overly complex setup programs are relics from a different age.

Note: Guidelines related to the first experience using a program and wizards are presented in separate articles.

Is this the right user interface?

While all Microsoft Windows programs need some sort of setup program, you have a choice of where to put program settings:

- Setup
- First use of the program
- Centralized program options
- In the context of using the feature

Setup

Present settings in setup if:

- The correct settings are required to use the program, and they apply to all users.
- Using default settings isn't acceptable, either because there is no safe default, users are likely to choose settings that aren't the default, or the default settings require user consent.
- Users should, but aren't likely to, change important settings after setup.

First use of the program

Present settings on the program's first use if:

- The correct settings are required to use the program, and they apply to individual users.
- Using default settings isn't acceptable, either because there is no safe default, users are likely to choose settings that aren't the default, or the default settings require user consent.
- Users should, but aren't likely to, change important settings using the program options.
- The settings customize a core experience, or one that is crucial to a user's personal identification with the program.

For such settings, users are likely to make better choices within the context of the program than within setup.

Centralized program options

Present settings on the program's options dialog box if all of the following conditions apply:

- There are default settings that work well for most users.
- There are many settings and they apply across features and tasks.
- Users are more likely to expect to find the settings in a centralized location.

In the context of using the feature

Present settings in the relevant context if all of the following conditions apply:

- There are default settings that work well for most users.
- There is a small number of self-contained settings for a specific feature.
- Users are more likely to expect to find the settings with the associated feature than a centralized location.
- There is an obvious place in the user interface (UI) to access the settings.

Through careful attention to the placement of configuration settings, you can reduce the burden on users during their first experience with your program.

Design concepts

Design a lightweight setup

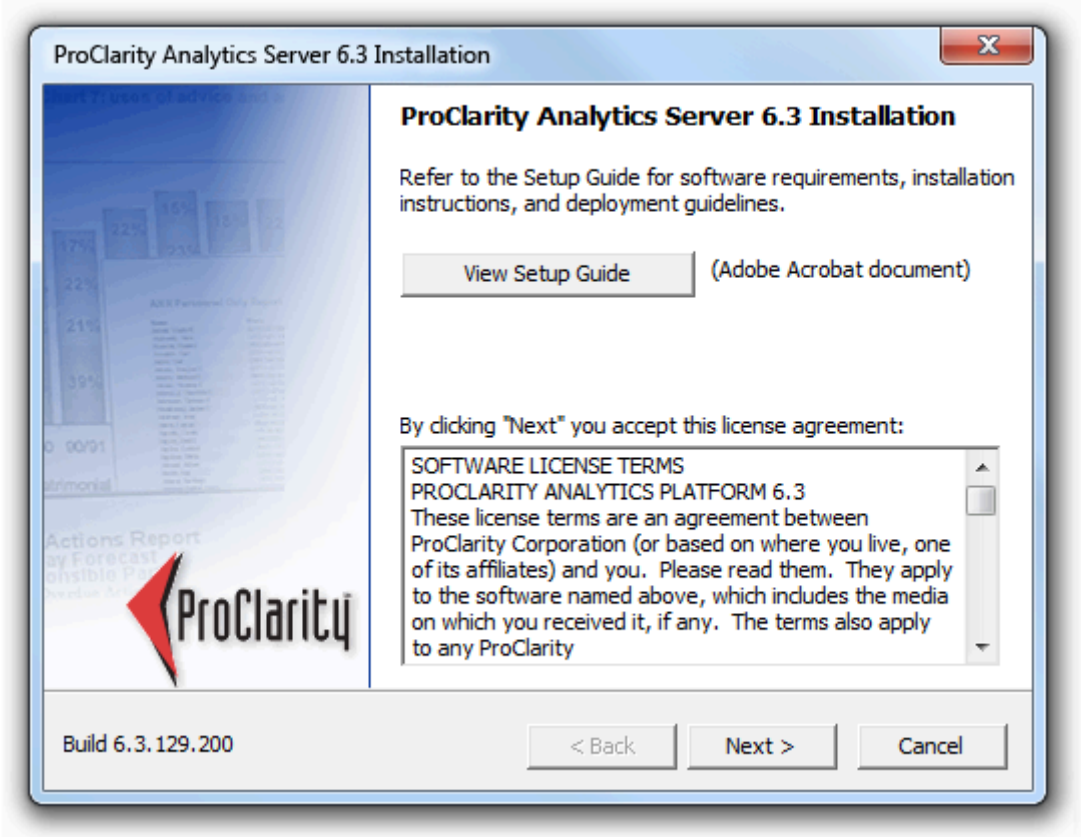
Welcome, next, next, next, next, next, install, finish, congratulations! Does this setup experience sound familiar? Historically, setup programs have adopted this kind of inefficient design: a long sequence of screens, inviting users into a mindless sequence of clicks just to get through it.

If users describe the setup of your program with words like quick and simple, they are surely praising the experience. They would much rather be using your program than setting it up.

Review your setup design for nonessential questions, options, pages, and paths, and be ruthless about eliminating them. Perform user research to find out what options users really need, and make sure they aren't mindlessly clicking the Next button through all the pages. Defer any options or questions that are better addressed in the context of the running program.

Many setup programs offer standard pages not because they are necessary or helpful, but because they are standard. For example, welcome pages, summary pages, and congratulations pages often just add clicks. Instead, your setup program should add

pages only if they are necessary to complete the setup task. For guidelines about types of setup pages and how to evaluate them, see Page types later in this article.



In this example, the setup program eliminates the traditional Welcome page and gets right down to business.

While it might be necessary to offer different branches of setup (a speedy, typical experience and a more controllable, custom experience), make sure that you have enough custom options to warrant the extra complexity. Don't add branches unless you have to. A few unimportant options in a custom branch suggests the need to reorganize the setup design.

Another reason to streamline setup is that inexperienced users sometimes overanalyze options, fearing that a wrong choice could be irreversible or destructive. Forcing users to make decisions about things they don't understand or care about can make them feel anxious, incompetent, and even frustrated. Not a good first impression. It is better just to get them going quickly, feeling comfortable and confident as they explore the features in your program, and making better decisions about feature options at that time. For more guidelines, see Streamlining setup later in this article.

Strive to make your setup experience as simple as possible, but not simpler. Programs targeted at highly technical users may need a complex setup. For example, the Microsoft SQL Server team discovered that database administrators prefer to retain control over many setup options, such as file locations. Moreover, SQL Server is a large business application, with a number of components that differ widely in purpose and

functionality. So while we want to keep things simple, setup needs to reflect the complexity of the product and the expectations and needs of its users.

Still, such complex setup programs should be the exception, not the rule. Most Windows programs should strive to start the setup process with a simple, single step.

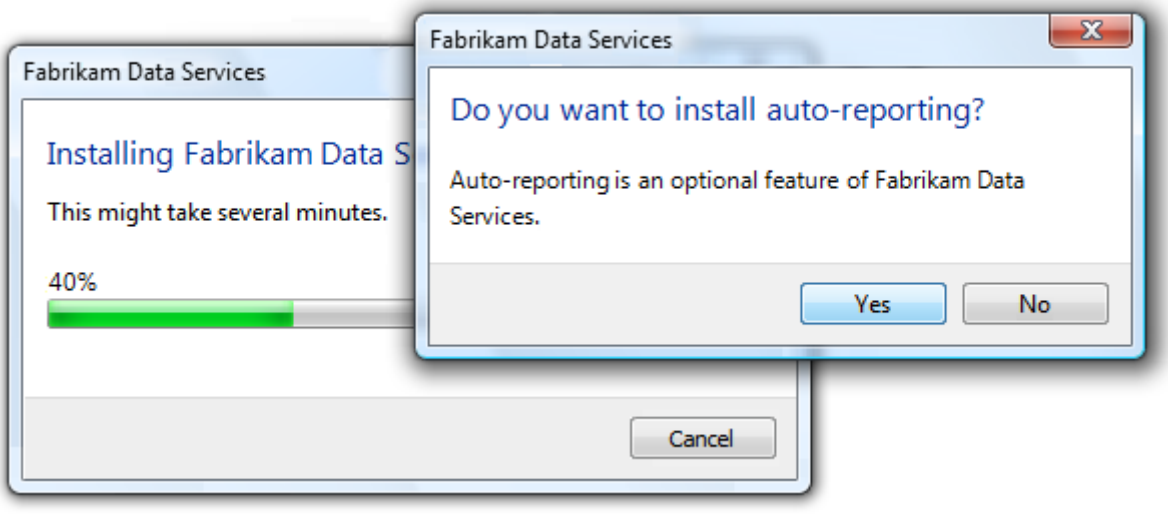
Setup phases

Well-designed setup programs let users perform other activities during the time-consuming task of downloading and copying files. To run unattended, setup programs are designed to have four separate phases:

- **The decision phase.** Users indicate how they want the program installed and configured.
- **The downloading phase.** For programs downloaded from the Internet. If the program has multiple applications or versions, users indicate what to download during the decision phase.
- **The installation phase.** The setup program copies the files and makes the appropriate configuration changes.
- **The completion phase.** Any remaining details, steps, or problems are addressed.

Because the installation phase might take a long time, this phase should be designed to run to completion without any user involvement. This means that all questions should be asked during the decision phase, and any problems that arise should be queued and dealt with in the completion phase. **If the installation phase takes more than one minute to complete, assume that users will be doing something else during the download and installation phases.**

Incorrect:



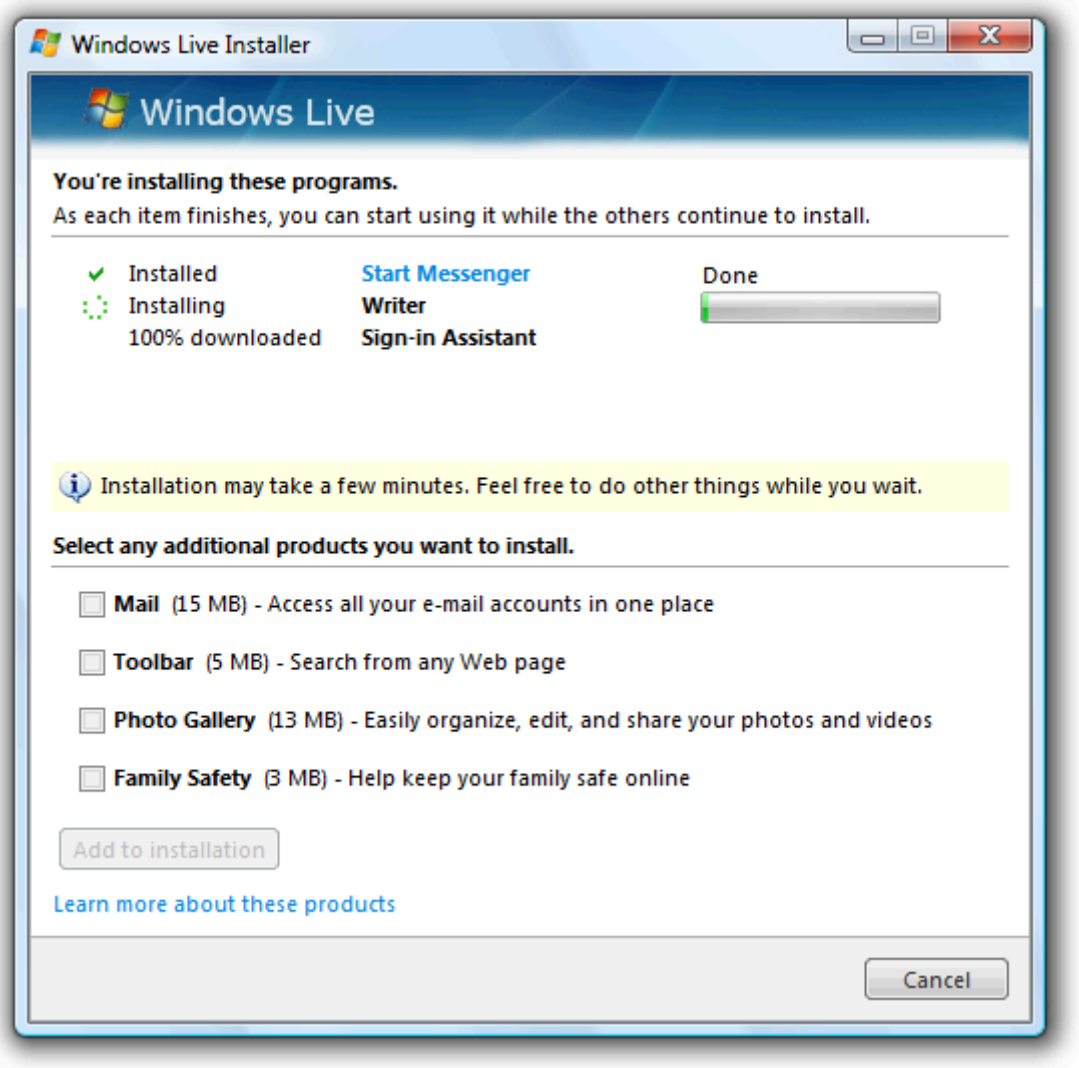
In this example, the setup program interrupts progress to ask a question that should have been asked during the decision phase.

Present helpful progress

If users patiently wait through the installation phase of the setup experience, perhaps watching a progress bar to its apparent completion, only to witness the progress bar reset and start over, there's a real sense of betrayal. The reported progress was misleading and ultimately meaningless.

A variation on this painful scenario is the "brinksmanship" installation: users see progress reach, say, 99 percent complete, yet are forced to wait for a disproportionate amount of time before finally getting to 100 percent complete. So in terms of what is most important to the user, an implied promise about the amount of time to wait, the claim of 99 percent complete is deceptive.

During the downloading and installation phases, users typically have two things they want to know: should they wait or do something else, and is the setup going to be done soon. While there are enough variables in the setup process to prevent you from providing perfectly accurate progress information, the progress feedback needs to be accurate enough to answer these two questions and set appropriate expectations. In addition to a progress bar, you can include a brief statement about the overall time expected for the process.



In this example, the progress page includes a brief, general statement about how much time the installation could take.

Good setup programs use progress bars effectively to provide users with helpful information about the setup program's progress. For more guidelines, see Progress Bars.

Design for all setup scenarios

Modern setup programs must be designed to handle a variety of installation scenarios:

- The user of the program is installing it from a disc or network file share.
- The user of the program is downloading it from the Web.
- An original equipment manufacturer (OEM) is including the program on the computer at the factory.
- An IT professional is installing the program on many computers across an organization.
- Someone other than the user is installing the program (for example, a parent on behalf of a child, or a co-worker who is using the same computer as another co-worker).

Given these scenarios, you shouldn't assume that users are always installing the program for themselves (making options about personal preferences inappropriate), going to be monitoring the process closely (making unattended setup important), or even want a graphical user interface for the task.

Don't forget the uninstall experience

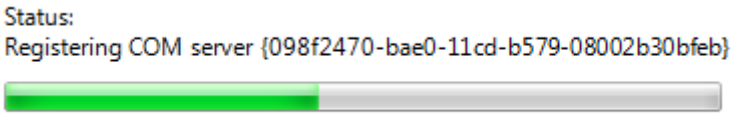
To complete the software setup lifecycle, users need to be able to remove software they don't want or no longer need. This is especially important if they didn't install the program themselves (for example, if it came pre-loaded on the computer).

Handle technical support strategically

Installing your program is the one task that all your users must complete successfully. If users fail to install your program, either you need to provide them with costly technical support or they aren't your users anymore.

Design your setup program to provide your technical support team with the features and information they need to help users install successfully. These details shouldn't normally be exposed to users, but they should be readily accessible when needed.

Incorrect:



In this example, the progress bar is showing details meaningful only to technical support.

Keep the normal user experience simple—don't clutter it with information that has value only to technical support. Rather, record support information in a setup log file instead. And more importantly, help users avoid the need for technical support with clear, concise error messages that explain problems well and provide practical solutions. Provide links to Help articles when necessary. Consider providing a Repair option to your setup program to repair missing or corrupted files or settings.

If you do only three things...

1. a. Make setup as simple and lightweight as possible. Remember that users don't enjoy setup, they endure it. Look carefully at every question, option, page, and path, and trim away everything that isn't essential to completing setup.

2. b. Design for all setup scenarios, including unattended installations, scripted installations, and uninstall. For efficient unattended installations, make sure there is a clean separation between the setup phases.
3. c. Design your setup program so that users can resolve setup problems on their own, but also log the information needed for technical support just in case. Keep in mind that setup is the one task that all users must complete successfully.

Guidelines

General

- **Apply the standard wizard guidelines for wizard-based setup programs.** Use these guidelines to determine good page design, effective navigation, good control labels, use of main instructions, and use of help.
- **Allow users to restart the setup program where they left off if it requires a lot of user input or takes a long time to complete.** If users restart the program after closing it before completion, restore previous user input, and restart where the setup was stopped.
- **Don't display setup windows maximized.** Displaying a setup window maximized presumes that users will give setup their undivided attention, which is unlikely. Instead, choose a size that is appropriate for the content to maintain a simple appearance.

Windows integration

- **Name the setup file "Setup.exe".** "Install.exe" is an acceptable alternative. This enables Windows (and users) to recognize the file as a setup program.
  - **Exception:** For programs downloaded from the Internet, help users manage and organize their Downloads folder by including the name of the program in the name of the setup file. For example, SetupVisualStudioExpress2008.exe.
- **Copy program files to the proper file system locations.** Doing so allows users and Windows to find and organize the files better. For more information, see the Windows File System Namespace Usage Guidelines.

User Account Control

- **Digitally sign the setup executable file.** Signed executables have many advantages, including using a more specific User Account Control Elevation UI. For information about signing files, see Introduction to Code Signing.

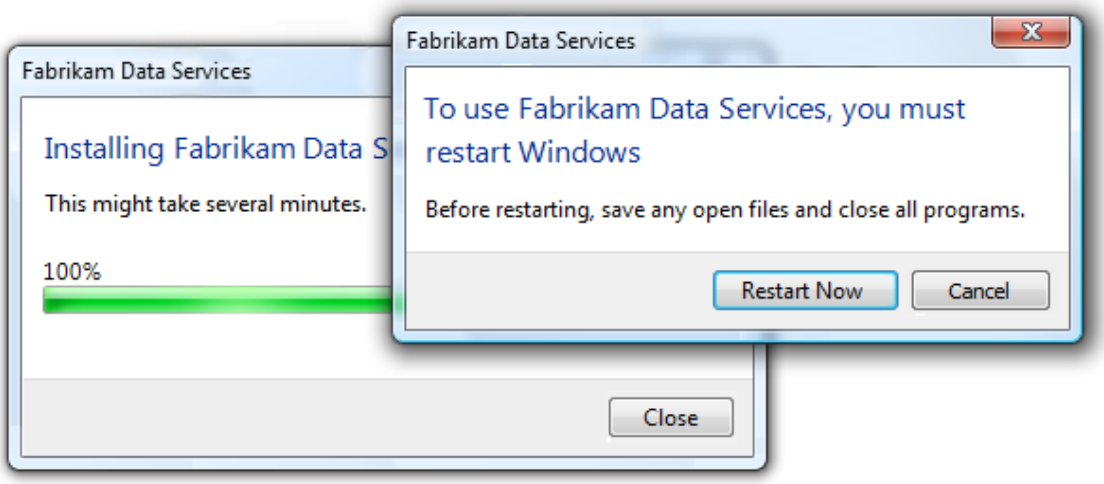
- **If a setup might require elevation, elevate as late as possible.** Display the elevation UI only after the user has committed to an option that requires elevation. Usually, the elevation UI appears during the installation phase, not the decision phase. However, if a setup always requires elevation, elevate at its entry point.
- **Always require elevation for uninstall.** Doing so prevents malware from uninstalling critical software without users knowing about it.
- **Once elevated, stay elevated until elevated privileges are no longer necessary.** Users shouldn't have to elevate multiple times to perform complete a program installation.
- **If special privileges are required for installation, verify the user's credentials and report any problems on the first or second page.** Don't let users perform a lot of work only to find that they don't have the right credentials to complete the installation.
- **Require the least privileges possible.** For example, administrators are reluctant to install software that requires domain administrator credentials.

For more guidelines, see User Account Control.

Restarting Windows

- **Avoid restarting Windows.** Most programs should install without restarting Windows. The primary reason program installations or updates require a system restart is that some of the files involved are currently being used by a running program. In this case, a better alternative is to make users aware of the situation, allow users to close these programs, and retry the action. For more information about avoiding restarts, see Restart Manager.
- **If your setup must restart Windows:**
    - **Use a single restart.** Delay the restart required by any prerequisites until the program and its updates are completely installed.
    - **Let users determine when it happens.** Don't restart Windows automatically, because users may lose work. Make sure that it's clear to users that they have a choice.

      Incorrect:

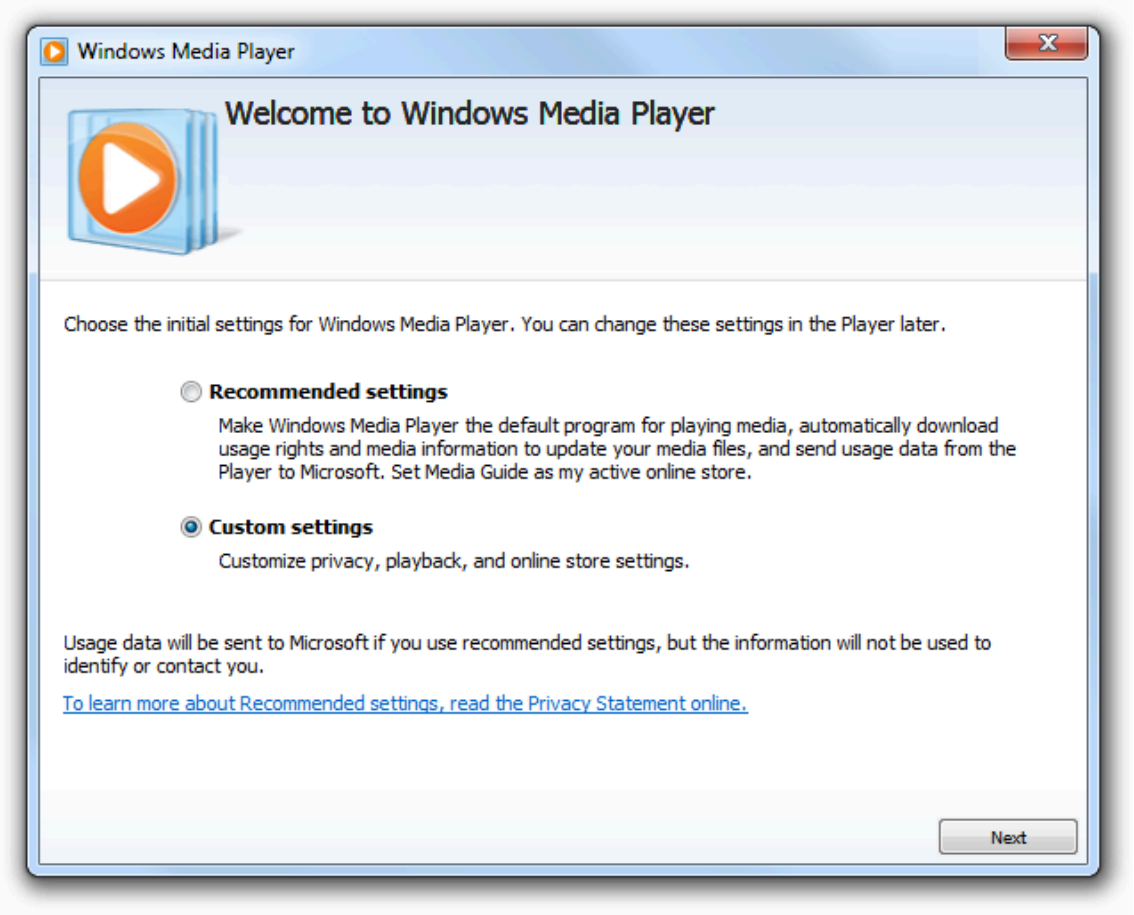


In this example, users don't appear to have a choice about when to restart Windows.

- **If the user chooses to not restart Windows immediately, present any final feedback as a success, not a failure.** While technically the installation isn't complete until restart, it was successful from the user's point of view.

Streamlining setup

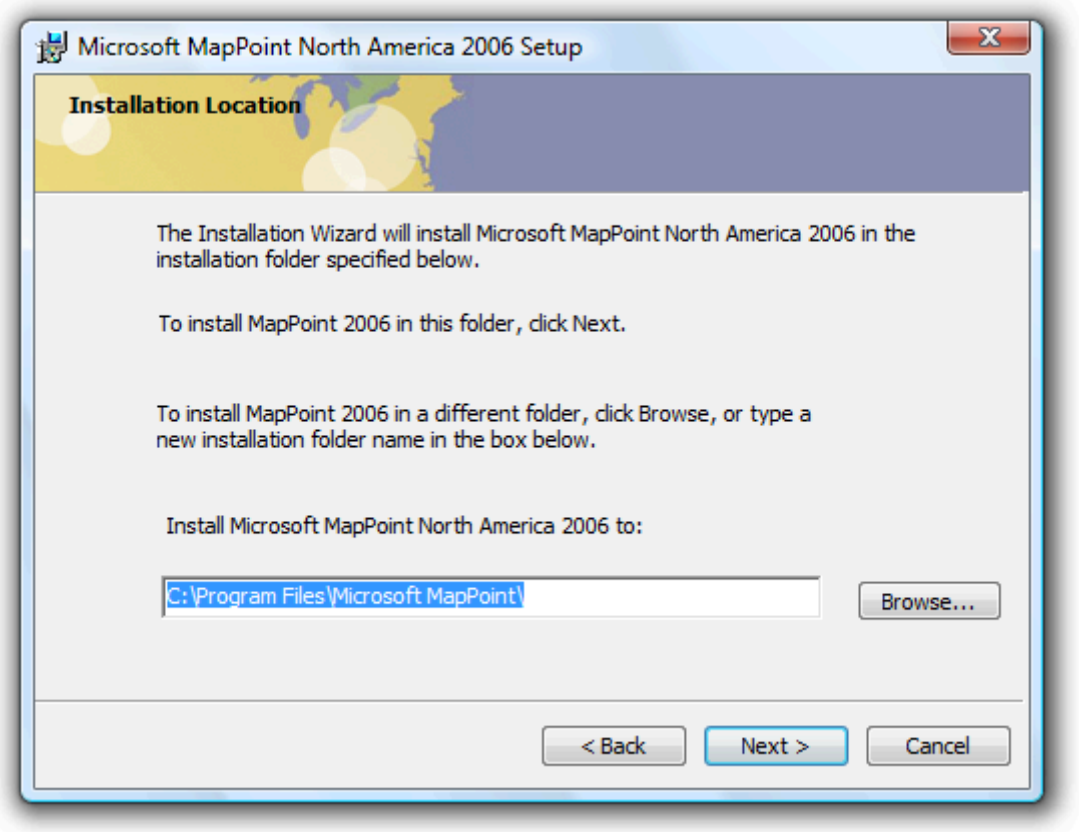
- **Whenever practical, start the installation process with a single step.** For example, instead of adding a separate page in setup for the license terms, you may provide a link to them instead. If you link to the terms:
  - Phrase the commit button as "Agree and install" to require explicit consent to accept the license terms.
  - Ensure that the license agreement link cannot be broken by linking to a file local to the setup instead of a Web page.
  - Provide the ability to print the license agreement from its display window.
- **Eliminate unnecessary options and questions.**
  - Postpone options that are more appropriate for the first use of the program or feature.



In this example, Windows Media Player presents per-user privacy options on first use of the program.

- Don't ask users questions about the system state. Detect this information automatically instead, and ask users to verify only if there is a reason to change.
- Don't ask questions about unimportant details. For example, for typical Windows programs it is safe to assume that you should copy program files to the Program Files folder.

  Incorrect:



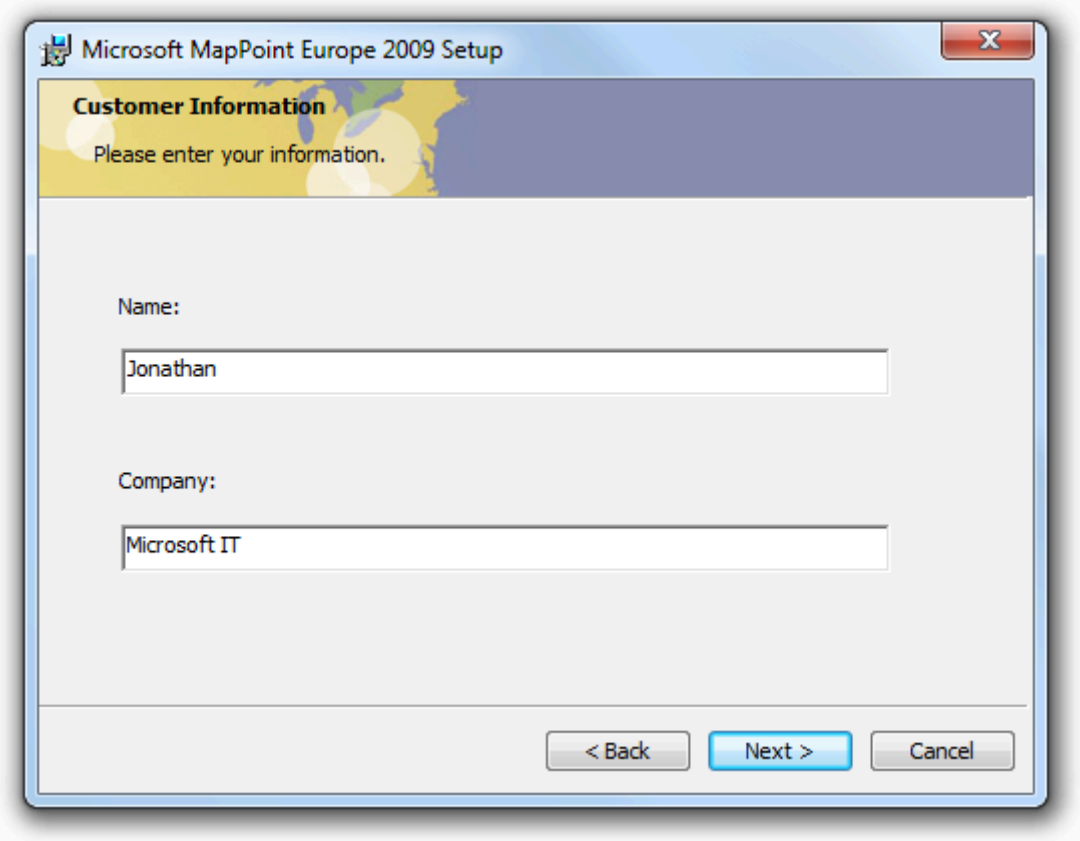
In this example, setup should be streamlined by eliminating the request for file location input. Given the program's size, most users don't care, and simply click Next.

- Don't ask permission to do what you shouldn't do anyway. For example, most programs shouldn't include an option to put the program icon on the desktop.
- Don't confirm setup cancellation. If users click Cancel during setup, assume the cancellation was intentional and close the program without confirmation. If doing so risks losing significant time or effort, allow users to restart your setup program and pick up where they left off.

- **Optimize for unattended installation.**
  - Present all options and questions during the decision phase.
  - For the download and installation phases, delay requiring user input to any problems encountered until the end of the phase. By doing so, users can leave the installation unattended until they return at their convenience.
- **Eliminate unnecessary pages.** If most users always just click Next on a page, consider getting rid of the page. For guidelines about eliminating certain types of pages, see Page types.
- **Eliminate unnecessary text.**
  - Remove redundant text from instructions and labels.
  - Don't explain basic Windows usage concepts, such as:

    - How to interact with controls (examples: To begin, click Next; For more options, click Options; For more information, click Help).
    - How wizards work (example: If you want to review or change any settings, click Back).
    - How setup works (example: This program will copy the program files to your hard disk...).

- **Eliminate unnecessary effort.**
    - Provide good default values:
        - Generally, select the most secure and private response to be the default.
        - If safety and privacy aren't factors, select the most likely or convenient response.



In this example, the user name and organization provided by default are obtained from the registry.

        - If an option is strongly recommended, consider selecting it by default, or adding "(recommended)" to its label.
    - Advance pages automatically when a page has no input and the task is done successfully, such as with download, installation, progress, and updates pages. Once the step is done, stay on these pages only to show problems.

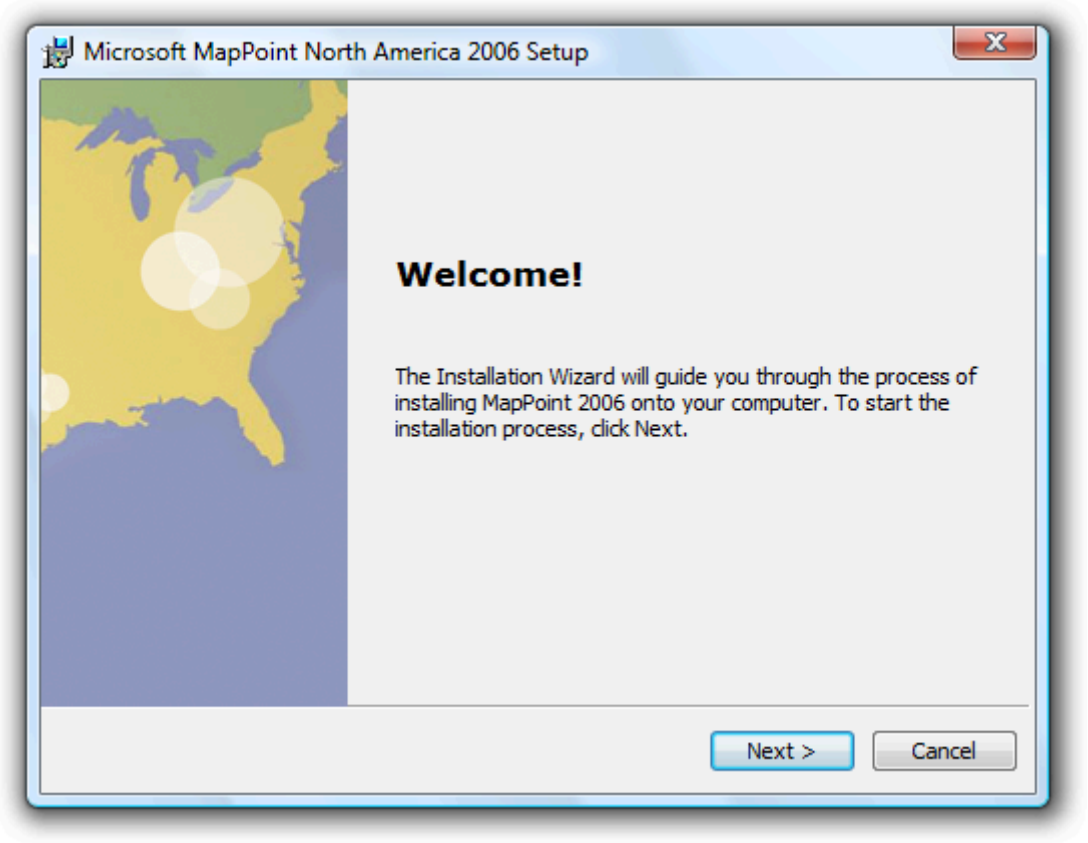
- When practical, start the program automatically when setup is done, instead of showing a Congratulations or Completion page. When setup is run interactively, assume the user is installing your program in order to run it immediately, so running the program is the best feedback to show that setup is complete. Automatically running the program isn't practical when the setup installs more than one program (for example, a suite consisting of many programs), when setup isn't run interactively, or when the installation process isn't complete after setup.

Page types

Welcome and Getting Started pages

- **Eliminate Welcome pages.** While it's great to feel welcome, users typically just click Next without reading. And because users typically skip over these pages without reading, the text does little more than state the obvious, by design.

  Incorrect:

  In this example, there's nothing for the user to do but click Next.

- **Use a Getting Started page only if you must inform users about prerequisites for installing.** Such prerequisites include installing required software or hardware, performing required system configuration changes and updates, performing a

system backup to protect against data loss, or obtaining required information that the user isn't likely to have already.

- **Whenever practical, provide the ability to perform the prerequisites directly from the setup program.** Users should have to perform the steps manually only if there isn't an alternative.

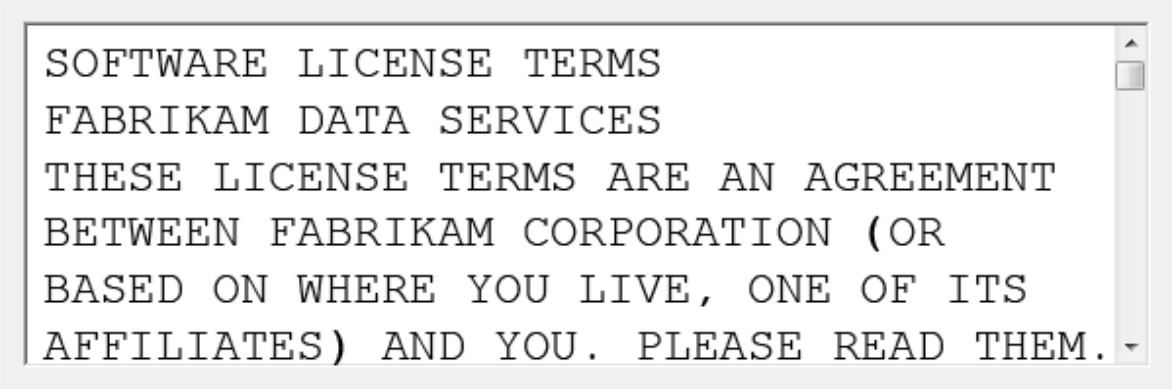
- If a Welcome page or Getting Started page isn't used, **include the program name and description on whatever is the first page of the Setup program.** You can use welcoming language as introductory text as long as the page's purpose is clear.

License terms pages

- **Write the license terms using clear, concise text.** Use plain language. Avoid "legalese."

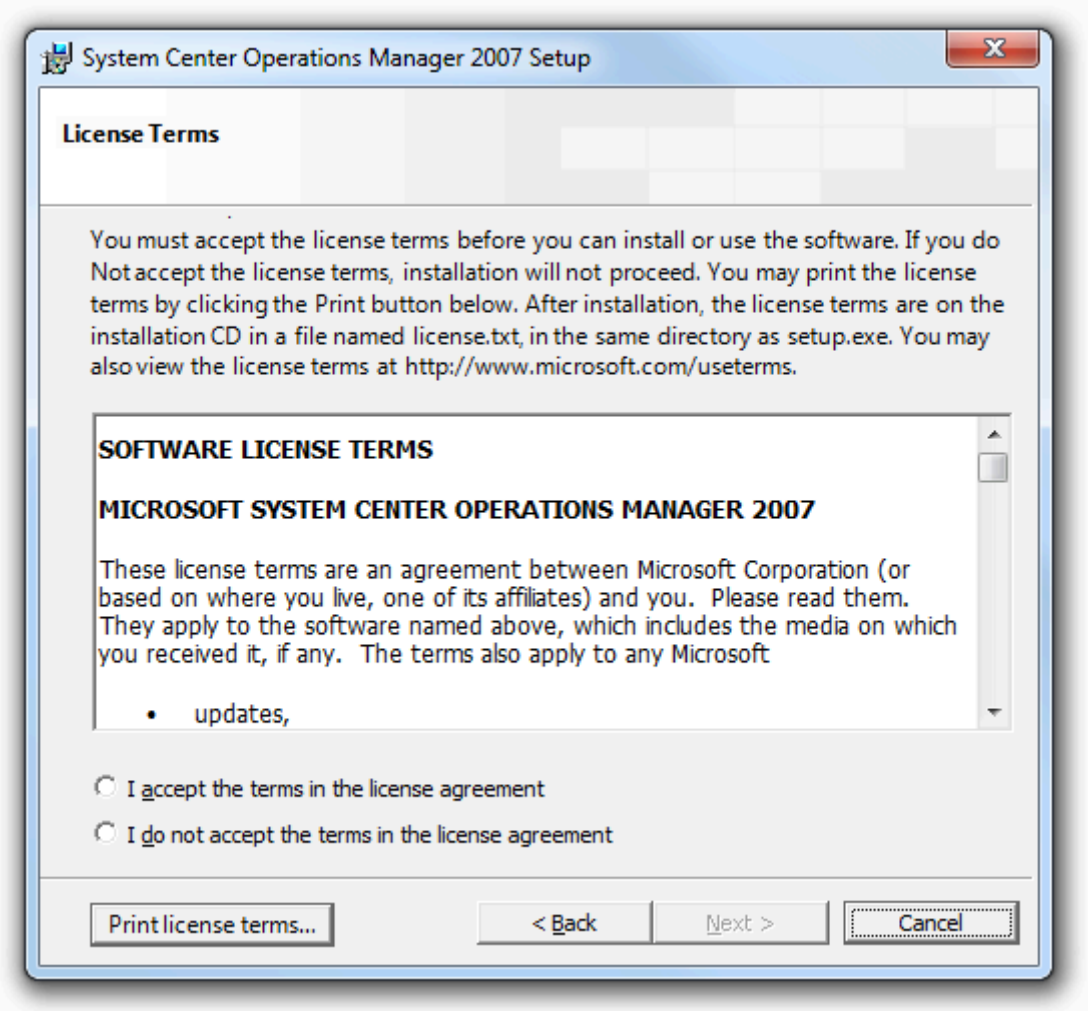
- **Present using a format that is easy to read and scan.** Don't use long passages of uppercase text.

  Incorrect:

  In this example, the uppercase text and large font size makes the terms difficult to read, forcing users to scroll more than necessary.

- **Require explicit consent to accept the license terms.** License acceptance should never be selected by default. If radio buttons are used to indicate acceptance, leave the options cleared by default and require users to accept the terms before enabling the Next button.



In this example, the Next button is disabled until users have explicitly accepted the license terms.

- **Don't require users to scroll to the bottom of the license terms text before the Next button is enabled.** This imposes an unnecessary burden on users to understand why the Next button is disabled.

- **Provide a Print command,** either with a command button or a context menu. Present the terms in a format optimized for printing.

Product registration pages

- **Require users to register only if they must in order to use the program.** Clearly explain why users must register.
- **Provide optional registration only if there is a clear user benefit,** such as to notify users of product updates. Leave this option cleared by default.
- **Allow users to register later.** Provide a maximum of three reminders and allow users to dismiss the reminders with a single click.

Scope pages (typical, custom, or minimum)

- **Prefer to eliminate this page.** Assume that most users want the typical setup experience (and design that experience so that it works well for most users).

- If you must include a scope page:
  - **Explain the differences among options in terms of functionality and disk space.** Users rely on the clarity of information on the scope page to ensure that they make the right choice.
  - **Make sure that the custom options are necessary only for a small percentage of users, while most users can safely ignore them.** If not, the options should be in the typical setup path.
  - **If users choose custom options, have the typical installation options selected by default.** Users regard the typical installation as the baseline, and want to customize by adding or removing options from that baseline.

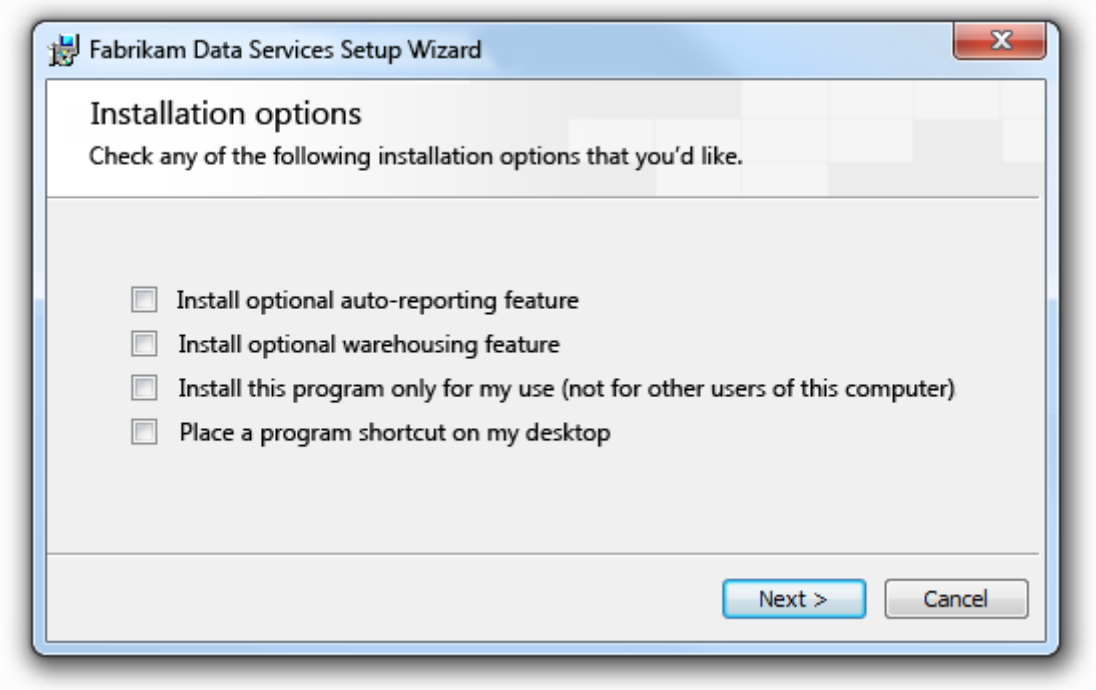
- If you must use a custom installation option, **consider using relative button sizing and placement to guide most users to the typical installation.**

  In this example, the page design visually reinforces the fact that most users should opt for the typical installation.

Input pages

- **Reduce the number of setup options by doing the right thing by default.** For ways to eliminate options see Streamlining setup.
- **Provide acceptable default values whenever possible.** Choose defaults that are secure and private, and are acceptable to most users without change.
- **Unless your program has unusual requirements, strive to have a single page of questions and options.** But if your program requires several pages of questions

and options, display them in the main wizard page flow. Don't try to reduce the number of pages technically by putting options in dialog boxes or using tabs.



- In this example, options are limited to a single page.
- **Validate input as soon as possible:**
  - Prohibit invalid characters on entry.
  - Use balloons to report problems with invalid text boxes.
  - Validate related fields on a page when users click Next.
  - Validate related fields across input pages as soon as problems can be detected.
- **Give all editable file paths a Browse button.** Allow users to specify network paths.
- **For the final input page, label the commit button Install, not Next.** Users shouldn't be surprised by when the installation starts. Before the commit point, make sure that users can easily change any settings.

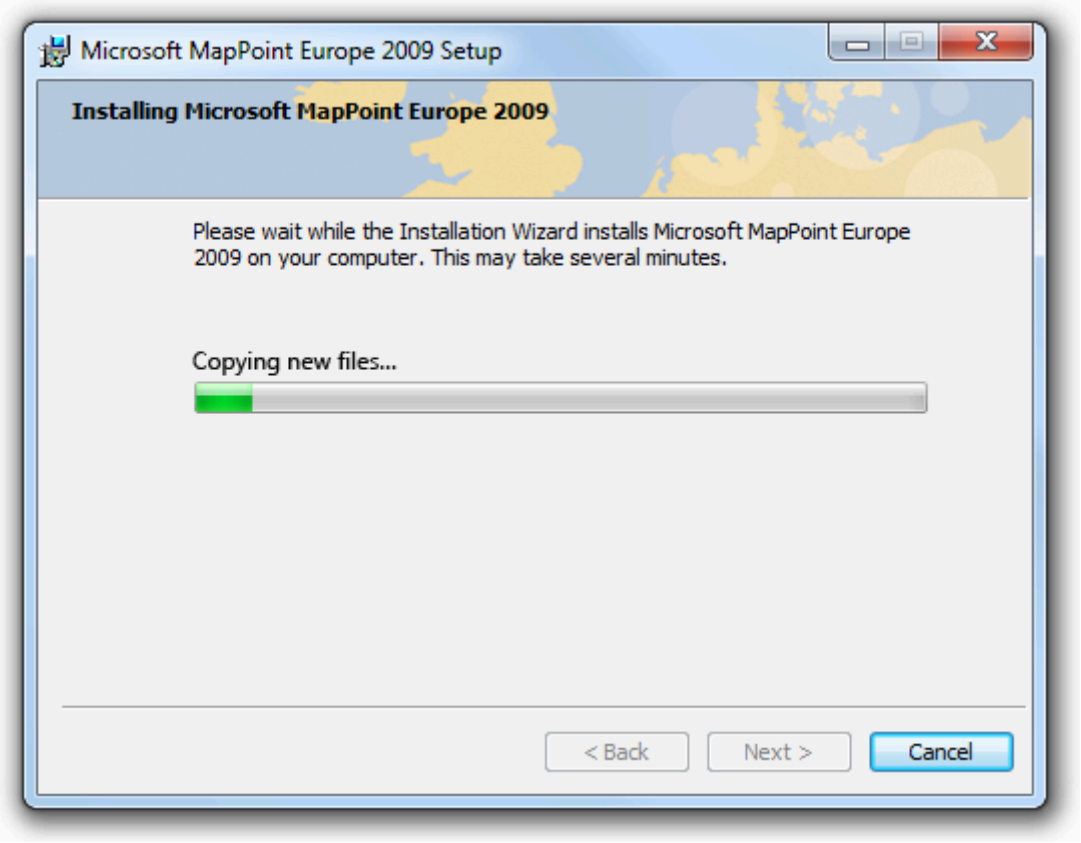
Start installation pages

- **Eliminate this page if it has no purpose other than to summarize the previous choices and begin installation.** If the input pages are clear and few in number, there should be no need to summarize them. Instead, the final input page should have the Install button, leading directly to the progress page.
- **For complex installations targeted at IT professionals, provide an Installation page with a comprehensive list of changes that the setup program will perform.** Many IT professionals have strict change management control, so they need to know the effect installing the program will have in detail.

Progress pages

- **Always provide a progress page,** even if the program installs quickly. Provide a separate progress page for the downloading phase if there is one. Disable the Back

(or Previous) and Next buttons while the setup is in progress, but leave the Cancel button enabled and responsive.



A typical progress page.

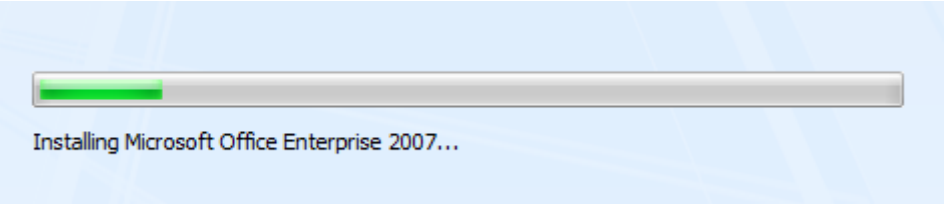
- **Use a single, determinate progress bar.** Follow the determinate progress bar guidelines, including:
  - Clearly indicate completion. Don't let a progress bar go to 100 percent unless the operation has completed.
  - Don't restart progress. A progress bar loses its value if it restarts (perhaps because a step in the operation completes) because users have no way of knowing when the operation will complete. Instead, have all the steps in the operation share a portion of the progress and have the progress bar go to completion once.

- **Provide a concise description of the current step above the progress bar.** For quick installations, such text is unnecessary; the progress bar alone is sufficient. For installations requiring a minute or longer, text can be helpful for users attending the setup.

  - **Use sentence fragments, typically beginning with a verb, and ending with an ellipsis.** Examples: Copying files..., Installing required components....

  - **Place text above the bar, not below.**

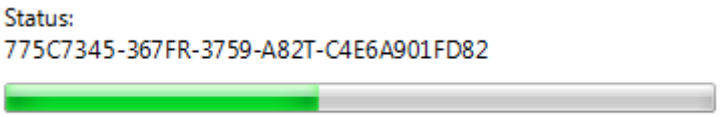
Incorrect:



In this example, the explanatory text should appear above the progress bar.

- **Refrain from cluttering the progress page with unnecessary details.** This page isn't for technical support, so there's no need to display registering GUIDs or specific files copied.

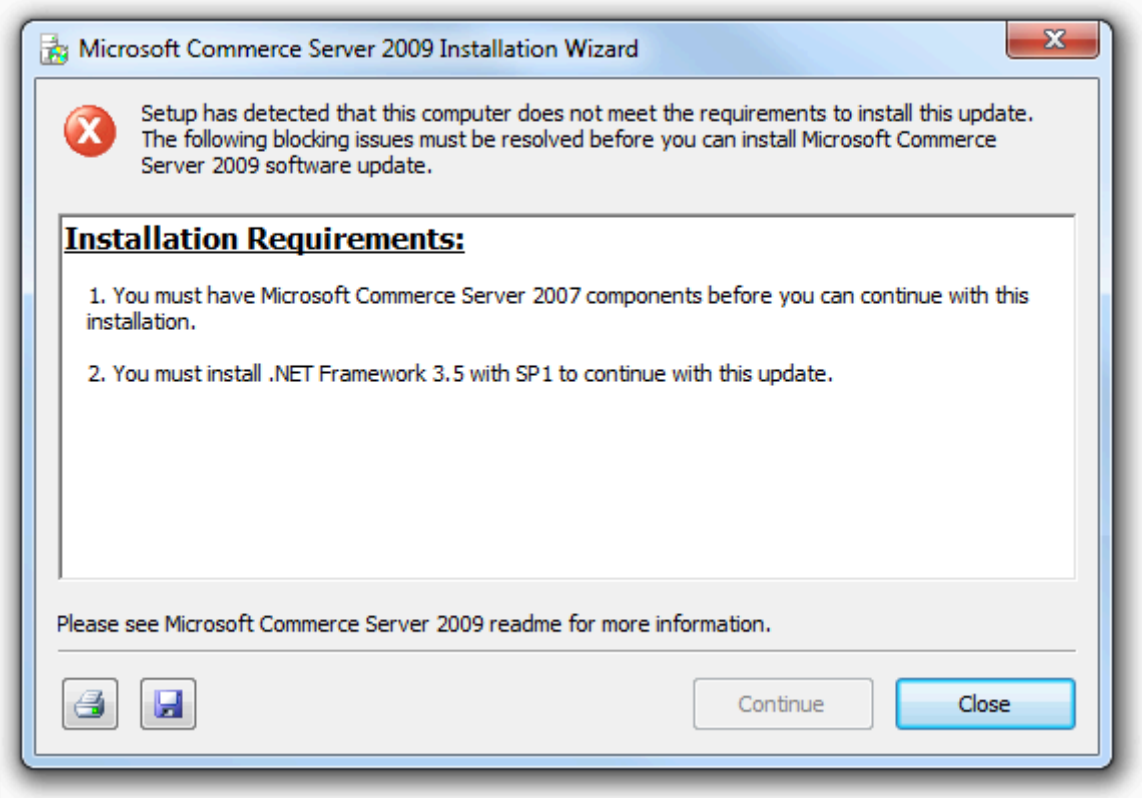
  Incorrect:



  In this example, technical details such as GUIDs are meaningless to users.

Error pages

- **If setup fails with a significant problem, display an error page that explains the problems along with practical steps to resolve them.** Display the page with an error icon. Don't use a dialog box for this purpose.



In this example, the setup failure is explained on an error page, along with some steps to resolve the problem.

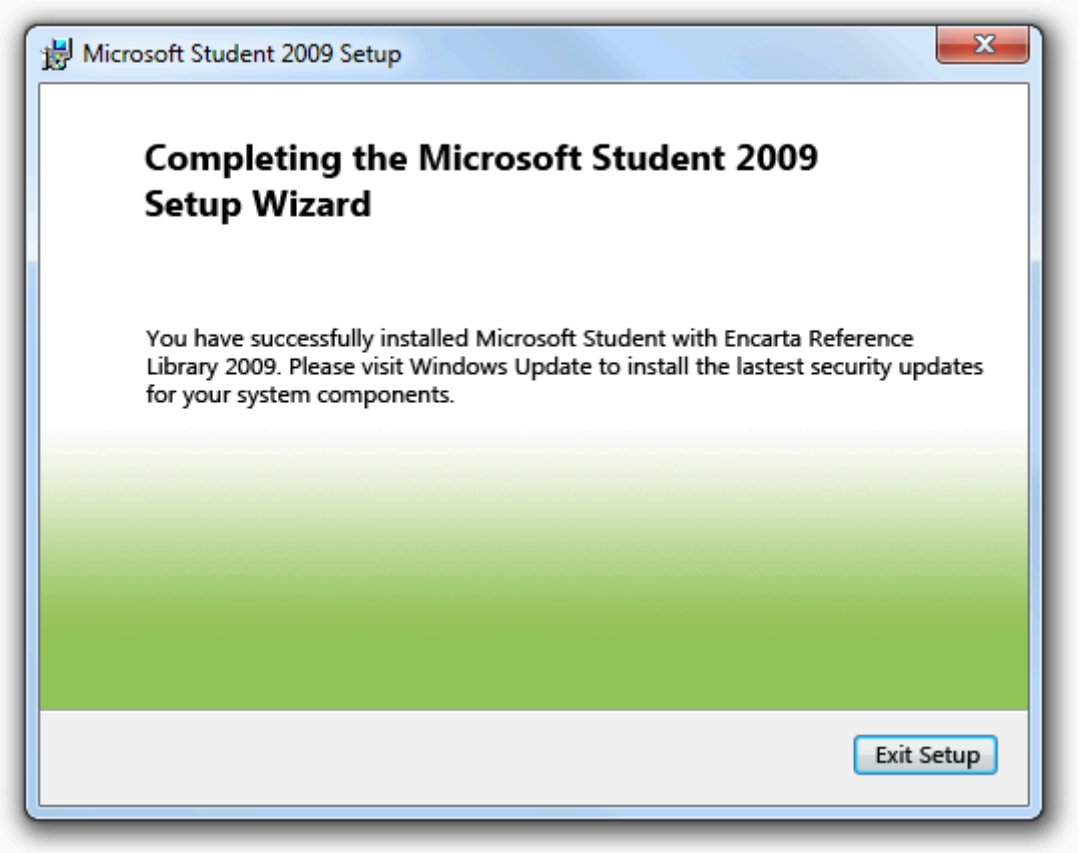
- **If setup completes with a minor recoverable problem, present the problem as an additional task instead of an error.** Use positive, success-oriented, encouraging language, not terms like error, failure, or problem. Don't use an error icon.

Congratulations/Completion pages

- **When installing a single program interactively, start the program (and close the setup wizard) to indicate successful setup, instead of displaying a completion page. Exceptions:**
  - Setups that are run from the command line should not start programs.
  - Automatic updates (for example, Windows Update) shouldn't start programs.
  - Group policy installation shouldn't start programs.
  - Any IT professional setup scenarios (because they are not installing for their own use).

- **If the setup has follow-up steps after installation, list them on a Completion page.** But to justify a Completion page, make sure users are likely to perform the steps, and that the steps genuinely need to be stated (that is, they are not obvious).

  Incorrect:



In this example, an unnecessary Completion page states the obvious. Windows Update runs automatically, so there's no reason for users to run it manually.

- **When installing a suite of programs, display a Completion page to indicate success and any follow-up steps that may be necessary.**

In this example, setup has installed multiple programs, so it doesn't make sense to start one particular program automatically. A Completion page is more appropriate.

Leaving users in control

- **Don't gather personal information, such as that used for marketing purposes.** Setup isn't an opportunity to push your own agenda, cross-sell other program offerings, or conduct market research; you can damage the trust relationship with your users this way.
- **Don't force users to opt out of installing optional features.** Allow them to opt in instead. For example, users should explicitly choose to install a Windows Desktop Gadget.
- **Allow users to add or remove optional features using the setup program after initial setup.** Users can perform this task using the **Uninstall or change a program** control panel item.
- **For customer experience improvement initiatives, explain what data is transmitted, how it is used, and how long it is kept.** Use a link to a privacy statement Help topic for this purpose.
- **Avoid using sound,** because many installation scenarios are unattended, and because sound can be unnecessarily distracting even during attended installations.

Security

- **For Internet-based setup, provide any security updates automatically during initial setup.** Users should not have to update as a separate step.
- **Avoid recommending that users turn off firewalls as a prerequisite to installing your program.**
- If a firewall must be turned off, do the following:
  - **Limit the duration of this condition to as short a time as possible.**
  - **Explicitly point out when users can turn the firewall back on again.**

Uninstall

- **Uninstall should remove all traces of a program, including the following:**
  - Program files, including the setup program.
  - Start menu entries.
  - Desktop icons and Quick Launch icons (if any).
  - Registry settings.
  - File associations.
- **Uninstall should leave behind the following:**
  - User created files, such as document files.
  - Shared dynamic-link libraries stored in the System folder.

Help and support

- **Design your setup program not to need Help by asking clear, self-explanatory questions.** Reserve Help for advanced questions that really benefit from further explanation.
- **Don't use readme files.** These files are now obsolete and users don't read them anyway. Instead, provide online content if needed.
- **Link to appropriate Help topics or troubleshooting content from setup error messages.** Make sure the Help content provides a clear path to resolving the problem. For more information, see Error Messages.
- **Create log files to capture information useful to technical support.** Don't clutter the setup UI with technical support-related details that are meaningless to most users. Use log files for this purpose instead.

Text

- **Be concise.** Setup wizards often overexplain features and options, using blocks of text that are difficult to scan quickly. **Exceptions:**

    - Spell out all acronyms. Setup is often users' first experience with your program, so don't assume they understand jargon such as acronyms.
    - Explain unfamiliar terminology and concepts, preferably in place but using Help topics if necessary.
- **Prefer a friendly, professional tone; avoid an overly-technical tone.**

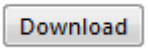
Incorrect:

Restrict installation on a per-user basis.

Correct:

Install only for me.

- Don't use now in command button labels because the immediacy of the command can be taken for granted.
    - **Exception:** When necessary, use now to differentiate commands that start a task from commands that perform a task immediately.



In this example, clicking the command button goes to a window or page that allows users to download.

In this example, clicking the command button performs the download immediately.

Only one command in a task flow should be labeled with now. So, for example, a **Download now** command should never be followed by another **Download now** command.

- Use license terms, not license agreement, licensing agreement, end user license agreement, or EULA.

For more guidelines, see Style and Tone.

Documentation

- As a verb, set up is two words; as an adjective or noun, setup is one word.
- The Setup program is capitalized and is not hyphenated.
- Use install to refer to adding hardware or software to a computer system.
- Don't use install as a noun. Use installation instead.

- Use restart, not reboot. Indicate that it is the computer, not a program, that is restarting.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

First Experience

Article • 06/04/2021

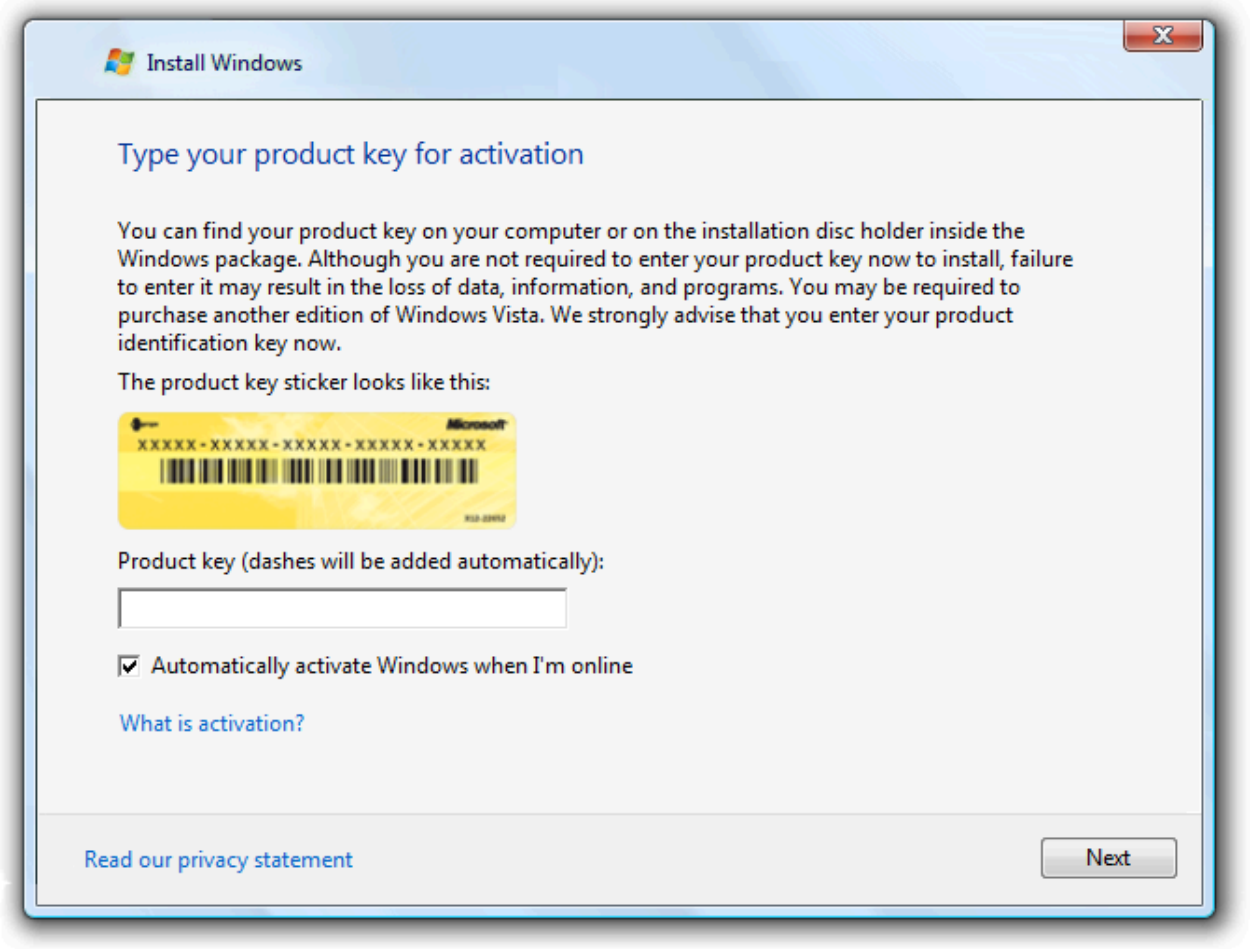
> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our **current design guidance**.

In the ideal first experience, users install your program and use it productively immediately, without answering a bunch of questions or learning a bunch of things.

A first experience user interface helps users transition from their first exposure to a new program or feature to everyday usage.

For Windows programs, the initial first experience occurs when users run the Setup program. Setup programs typically:

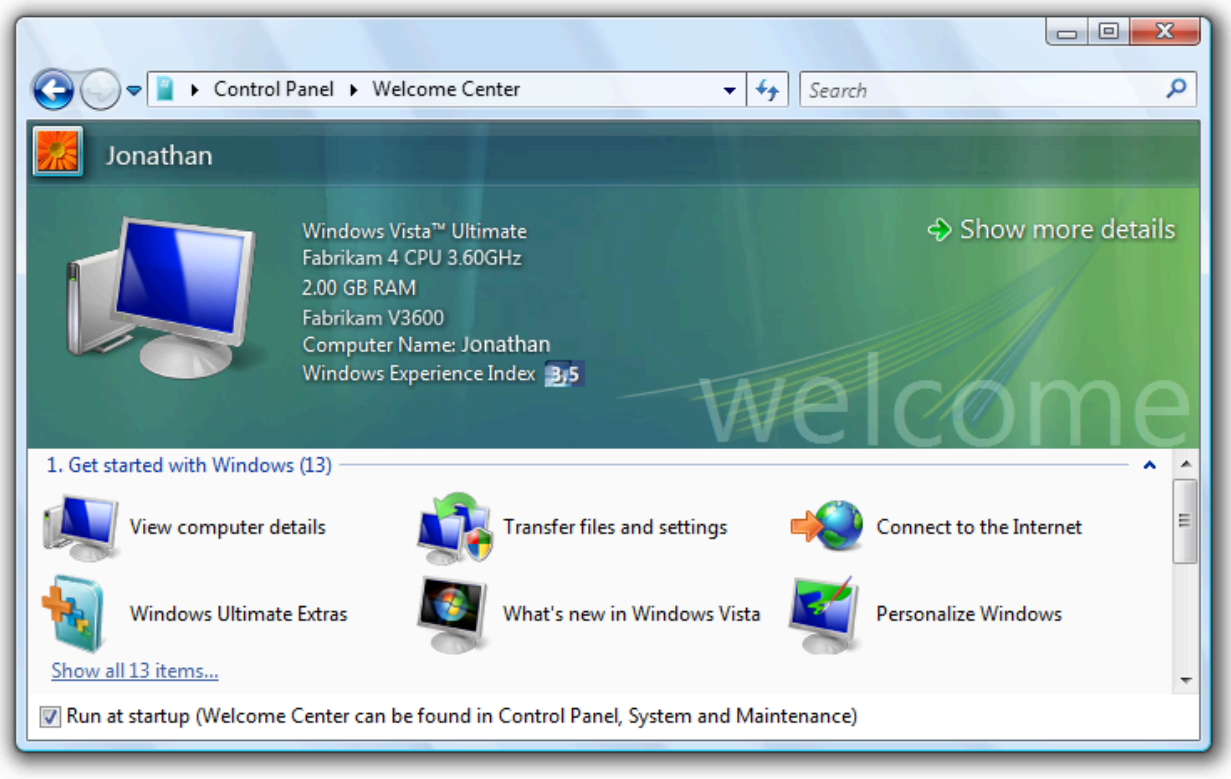
- Require the user to accept an End User License Agreement (EULA).
- Ask for a product key.
- Present required configuration-related options, including installation of optional software.
- Copy the software to the user's hard disk.
- Present program options that apply to all users.



Part of a typical Windows setup experience.

The first experience then continues to the first use of the program or feature. This first use experience can:

- Present program options that apply only to the current user.
- Offer product or feature tutorials.



The first use experience.

Note: Guidelines related to program options are presented in a separate article.

Is this the right user interface?

To decide, consider the following questions.

Setup experience

Do the following conditions apply?

- The correct settings are required to use the program, and they apply to all users.
- The settings customize a core experience, or one that is crucial to the user's personal identification with the program.
- There is no safe default, the user is likely to choose settings that aren't the default, or the default settings require user consent.
- The user is unlikely to change settings after setup.
- Changing the settings requires elevation.

If so, consider presenting the settings during the setup experience.

First use experience

Do the following conditions apply?

- The correct settings or tasks are required to use the program or feature, and they apply to individual users.
- The settings customize a core experience, or one that is crucial to the user's personal identification with the program.
- There is no safe default, the user is likely to choose settings that aren't the default, or the default settings require user consent.
- Users are likely to make better choices within the context of the program than within setup.
- The user is unlikely to change settings using Options.

If so, consider presenting the tasks and settings during the first use experience of the program or feature.

Design concepts

In the ideal first experience, users install your program (or even just start it if it doesn't require installation) and use it productively immediately without answering a bunch of questions or learning a bunch of things.

This ideal is obtainable for most programs, so you should strive for this ideal experience whenever you can. However, this goal is often not obtainable for programs that require significant system integration, have many optional features, or have privacy implications. For example, if your program has features that might reveal personal information to untrusted parties, the tenets of trustworthy computing require that you obtain user consent before enabling these features.

Questions aren't choices

Questions require responses they must be answered before users can proceed. Questions during the first experience are hurdles that users must jump over before they can use your program productively. By contrast, choices are optional. Users don't have to respond to them, or can choose to see them only when they want to.

Thus, settings presented in the main flow of a setup wizard are questions, whereas settings outside the main setup flow or in a program options dialog box are choices. Unnecessary questions make your program's first experience cumbersome and long, effectively eliminating the positive anticipation and excitement users have about getting started with your program.

Use the first experience when you must

Present settings and tasks to users during the first experiences when you must, but usually there are better alternatives:

Expand table

First experience	Alternatives
Setup questions	Select appropriate defaults. Allow users to change from program options. Provide typical vs. custom setup paths.
First use questions	Select appropriate defaults, and allow users to change from program options.
First use tasks	Present contextually instead.
First use feature advertisements	Make the most common and important tasks discoverable and contextual.
First use tutorials	Make program features self-explanatory.
Product registration	Provide command in Help menu and About box.

If you do only one thing...

Keep your first experience as simple as possible. Get your program working right away. Choose safe, secure, convenient defaults and ask questions during setup and first use only when you must.

You have only one chance to make a good first impression and that first impression is lasting.

Guidelines

General

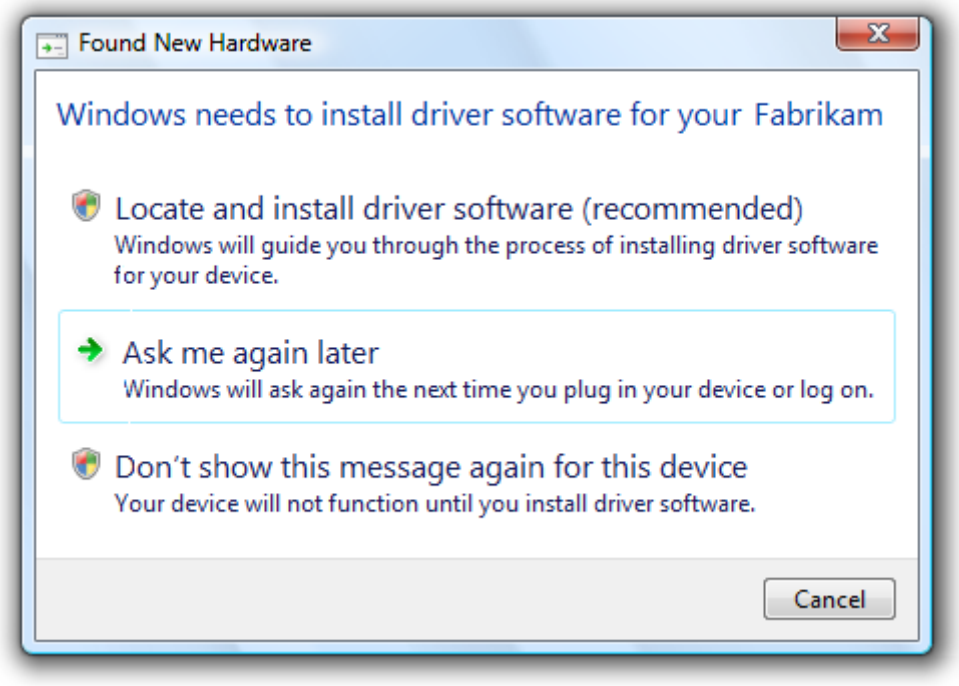
- **Limit first experiences to tasks and settings required to use a program or feature, and only include these when there is no better alternative.** See the previous table for alternatives.
  - **Exception:** Add personalization or program customization settings to the first experience if their customization is part of the core experience or crucial to the user's personal identification with the program.

Windows asks users for the computer name and choice of background during setup because these settings help form an emotional connection to the product.

- **Use the setup experience** for tasks and settings if they apply to all users or changing settings requires elevation.
- **Use the first use experience** for tasks and settings if they apply to individual users.

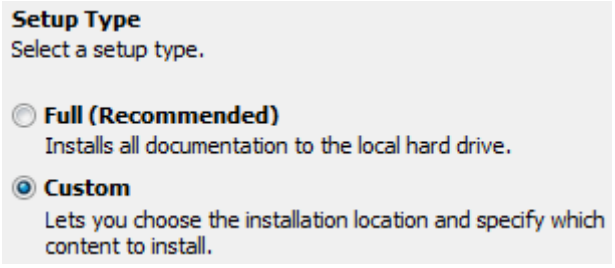
Presentation

- **Prefer optional tasks and settings to required tasks and settings.** Avoid forcing users to configure your program.



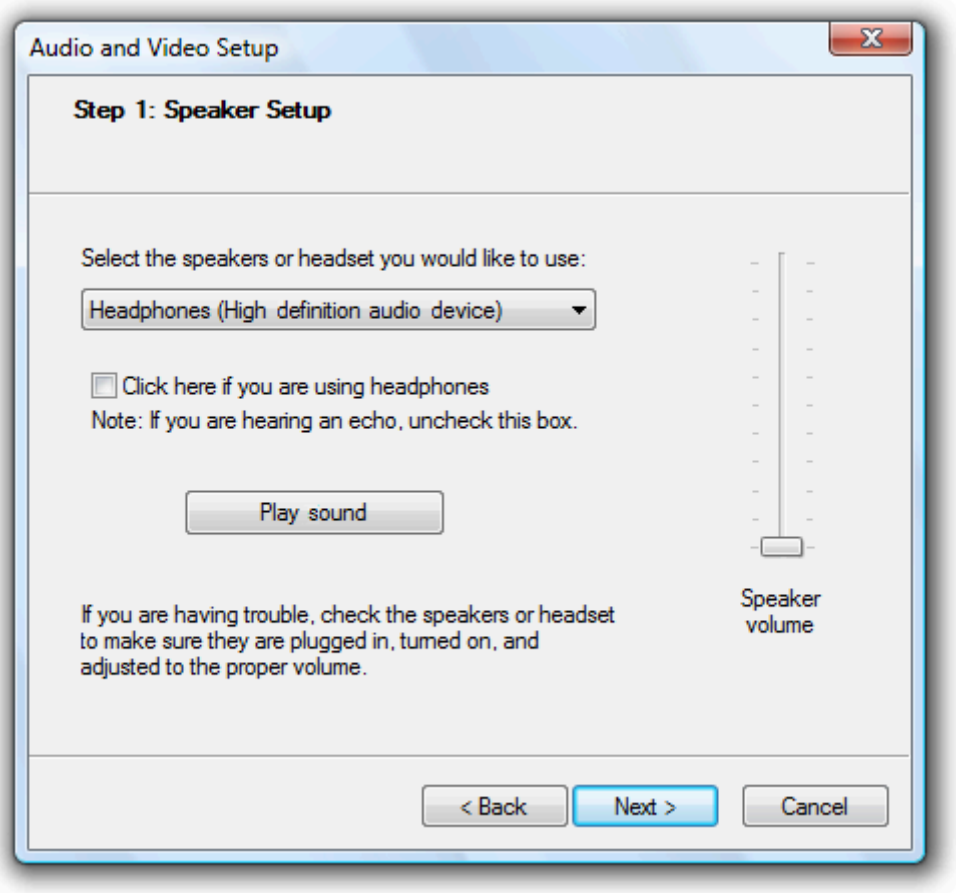
The Found New Hardware dialog box makes it optional to install driver software instead of making it a required task.

- **Take optional tasks and settings out of the main task flow whenever practical.** For example, many setup programs provide a custom installation path to remove infrequently changed settings from the main task flow.



A setup experience that facilitates main task flow if the user does not intend to customize the installation.

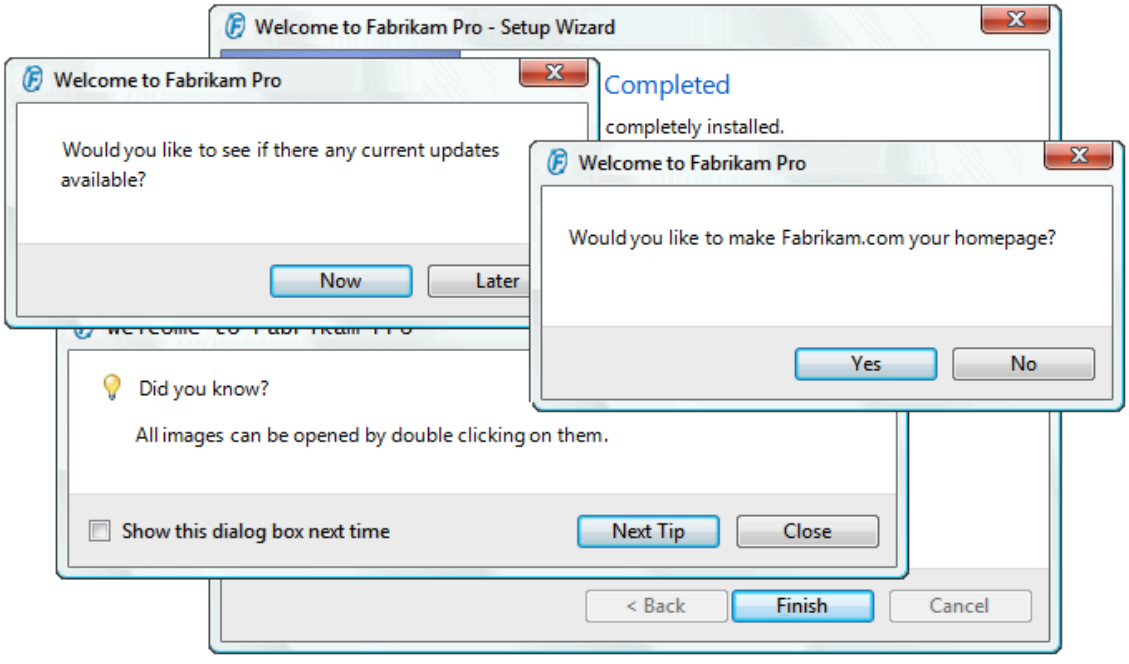
- **Don't overwhelm users with tasks and settings:**
  - **Start simple.** Begin with simple, personalization settings and progress towards more complex, technical tasks and settings. For example, Windows setup starts with personal information and ends with network configuration.
  - **Use a contextual first experience** for tasks and settings if they apply only to features that aren't fundamental to the main program.



Windows Live Messenger has a contextual setup for audio and video because they are used by secondary features.

- **Don't present everything all at once.** Consolidate to use a single UI instead of multiple UI surfaces, or display tasks at different times instead of all at once.

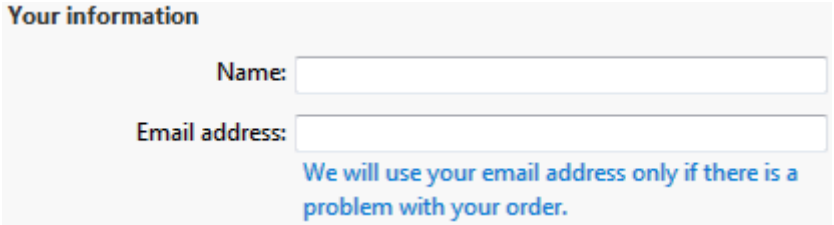
  Incorrect:



In this example, the first use experience is overwhelming.

- **Express questions and options in terms of users' goals and tasks, not in terms of technology.** Provide options that users understand and clearly differentiate. Make sure to provide enough information for users to make informed decisions.

- **If the need for personal information isn't obvious, explain why your program needs the information and how it will be used.**



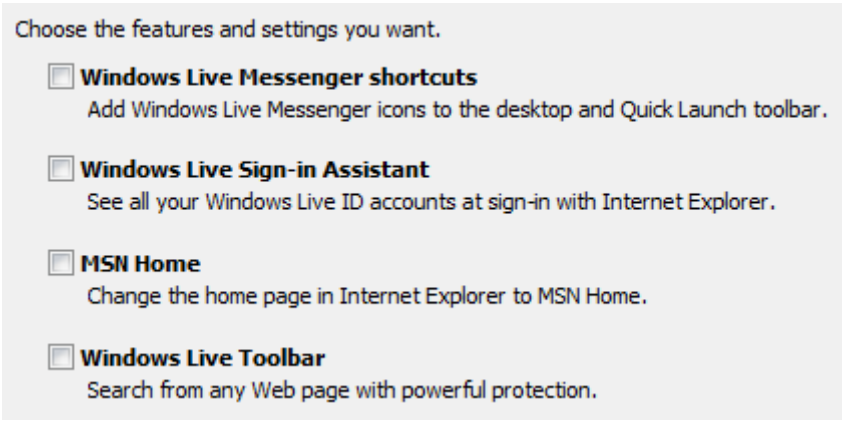
In this example, an e-commerce application explains how personal information will be used.

- **Present first experiences full screen only if users can't productively perform other tasks.** For example, Windows setup is presented full screen to discourage users from performing other tasks while Windows is being installed. Most first experiences shouldn't be full screen.

Settings

- **For all settings, select the safest (to prevent loss of data or system access), most secure and private value by default.** If safety and security aren't factors, select the most likely or convenient value. Choosing good defaults is an effective way to simplify the first experience.

- **Require users to opt in for:**
  - Settings with legal implications, such as end user licensing agreements (EULAs). Such settings can't have default selections.
  - Features that perform automatic system configuration changes, such as Windows automatic updates.
  - Features that reveal personally identifiable information (PII) or system information.
  - Changes to the user's desktop beyond adding entries to the Start menu, such as adding icons to the desktop or quick launch bar.
  - Optional software, such as product enhancements, subscriptions, and third-party products.



In this example, users opt in to product enhancements, subscriptions, and third-party products.

- **If a setting is strongly recommended, add "(recommended)" to the label.** For radio buttons and check boxes, be sure to add to the control label, not the supplemental notes.

- **If a setting is intended only for advanced users, add "(advanced)" to the label.** For radio buttons and check boxes, be sure to add to the control label, not the supplemental notes.

Tasks

- **Help users pass waiting time productively.**
  - If the waiting time is typically between one and two minutes, consider providing helpful information while users are waiting, such as a presentation of what is new during setup.
  - If the waiting time is typically longer than two minutes, make it easy for users to perform other tasks. Display the estimated wait time, recommend that users do something else in the meantime, and make task completion obvious by changing the screen significantly.
- **Reconsider presenting tutorials during the first experience.** Most likely users want to use your program right away and are interested in tutorials at a later point.
- **Don't use feature advertisement notifications in the first experience.** Instead of using a feature advertisement notification, design the feature to be easier to discover in contexts where it is needed, or don't do anything special and let users discover the feature on their own.
- **Don't use any notifications during the initial Windows experience.** To improve its first experience, Windows 7 suppresses all notifications displayed during the first few hours of usage. Design your program assuming users won't see any such notifications.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Printing (Design basics)

Article • 08/26/2021

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

Printing is the user experience on paper. It's easy to overlook, but it is an important part of the overall user experience.

In this article, *printing* refers to the user experience on paper, where output is directed to paper instead of the screen display. *Printer-friendly format* refers to modifications the program can make to screen display output that make it more suitable for paper output.

Despite the prediction that computing would result in the "paperless office," surprisingly enough we print now as much as ever. We distribute hard copies of Microsoft PowerPoint presentation decks, we print articles we discover online but wish to research more carefully later, we print important e-mails or resumes we have received in electronic form, and so on. While it's easy to overlook printing when designing a user interface, remember that printing is an important part of the overall user experience.

Note: Guidelines related to common dialogs are presented in a separate article.

Is this the right user interface?

To decide if your program needs to support printing, consider these questions:

- **What type of program are you designing?** The program type is a good indicator of the appropriate level of printing support. Document and image creation, viewing, and browsing programs need excellent printing support, whereas other types of programs may only need printing support to a lesser degree. (For a list of program types, see the Printing patterns section of this article.)
- **Is the program used in scenarios that benefit from direct paper output?** If so, it's more convenient to add printing support to your program than to require users to copy the data to another program to print.

Design concepts

Design your program to eliminate unnecessary printing

There are many reasons why users need to print some which are good, some which are less so. Users should print because they want to, not because they must. Requiring users to print can be a sign of missing features. For example, in the past users had to print documents in order to make comments and suggest revisions, but now users can do these tasks directly within Microsoft Word documents. **Review your program's scenarios that involve printing, and to the best extent possible, make sure that the need to print is optional and not the result of missing features.**

It's also worth remembering that conserving resources like paper and ink is helpful environmentally and saves organizations money in the long run.

Understand the differences between screen display and print

While there are many similarities between display output and printing, there are many differences as well. Print output:

- **Has a high dpi.** Display output is usually at 96 or 120 dots per inch (dpi), whereas printer output is usually at 600 dpi or greater.
- **Has different optimal fonts.** While well-designed fonts work well for both display and print, serif fonts are more readable at high resolutions for large amounts of text than sans serif fonts. Thus, large amounts of text primarily intended for print should use a serif font, whereas text primarily intended for display should use a sans serif font. For more information, see Fonts (Segoe UI).
- **Has aspect ratio.** Display usually has a low aspect ratio (4:3 or 5:4), whereas print uses a high aspect ratio (8.5:11 or 1:1.4142 based on the standard page sizes). This is because portrait mode printing is more common than landscape mode.
- **Has pages.** Consequently, print output:
  - Has standard page sizes. The standard in the United States and Canada is 8.5"x11" paper; the standard everywhere else is A4 paper.
  - Has page breaks.
  - Has page margins.
  - Has headers and footers.
  - Has single- or double-sided output.
  - May have multiple copies.
  - May be printed out of order or selectively.
- **Has many options.** Users may want to choose a printer and paper size, printer options (such as print quality, double-sided printing, and stapling), number of copies, page ranges, collation, and print format.

- **Takes time and money.** It can take a significant amount of time to print a large document or a high-quality photo, and the cost of the paper and ink adds up over time. By contrast, display output is instantaneous and essentially free.
- **May be black and white.** Many printers today are black and white, whereas few displays are monochrome.
- **Is not interactive.** Users can't scroll pages or controls to see more content. They can't click links or buttons, or hover over controls. Interactive content has no value when printed.
- **Can run out of paper, ink, or toner, or be offline.** Consequently, paper output needs more error handling and troubleshooting.

These differences may affect your printing design. Creating a good print experience requires more than just directing your program's output to the printer.

WYSIWYG and the evolving requirements of screen display

Historically, the most fundamental principle for the printing user experience is known as WYSIWYG ("what you see is what you get"). This principle suggests that there should be a strong relationship between what is seen on the display and what is printed. Before WYSIWYG became a standard practice, there was often no relationship between the display and print versions of a document. Users had to print in order to see what the document really looked like on paper. Using WYSIWYG was a great improvement in productivity because most programs at the time were primarily designed for document creation and printing.

Today, it is common for Web sites to optimize for the display, and their printer-friendly format may appear much different. Furthermore, we have diverse computing devices (for example, smart phones and personal digital assistants) that often need output optimized for small displays. While WYSIWYG is still the best approach for document creation programs, for other programs it often makes more sense to optimize for a variety of target devices. For such programs, what you see on a PC display may be different from what you see on other device displays, which may be different from what you get on the printed page.

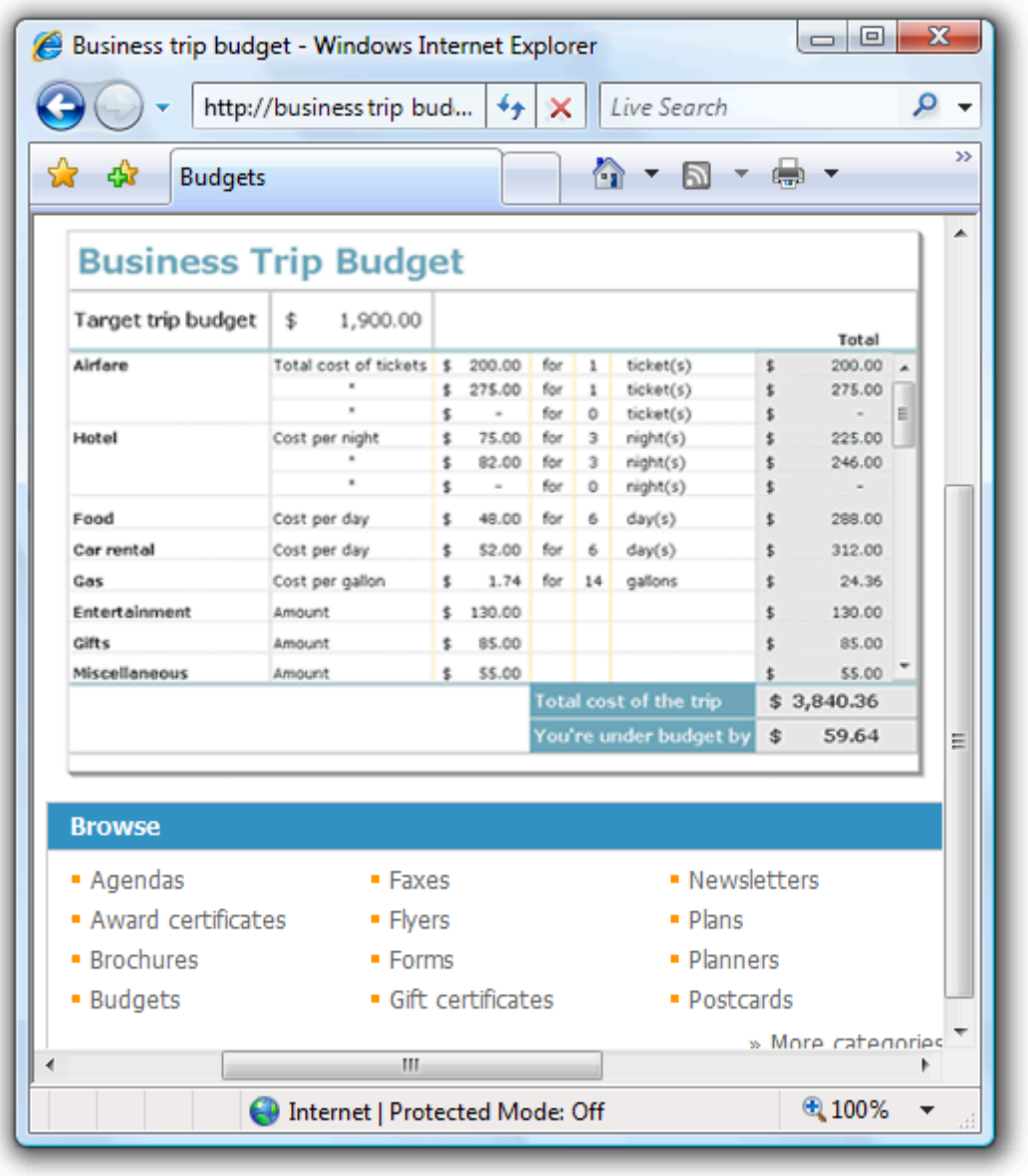
Optimize for printing

Programs that don't employ a strict WYSIWYG print experience can still optimize for printing in the following ways:

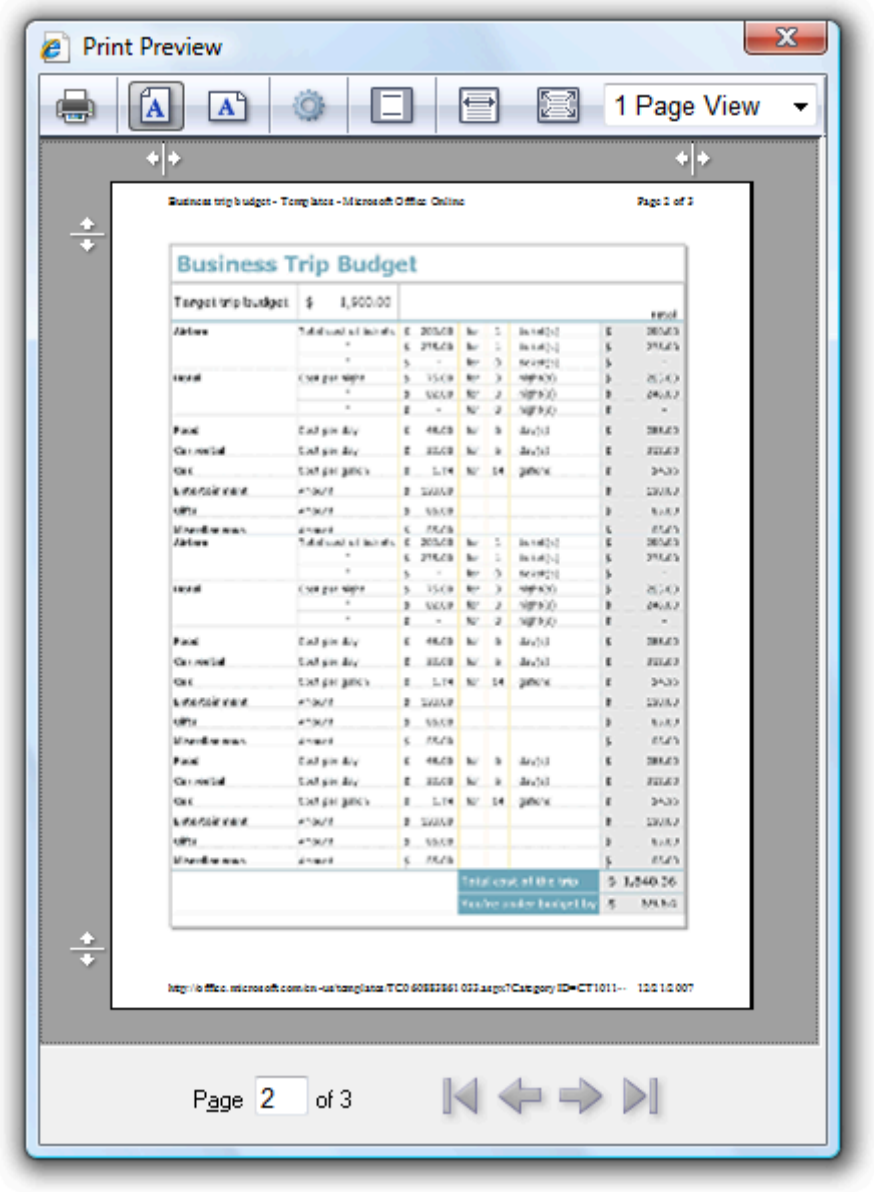
- Reformat the layout for the target page size.

- Provide print preview, preferably with easy customization options allowing users to experiment directly in the print dialog (for example, dragging margins).
- If appropriate, provide a printer-friendly format option.
    - Consolidate separate partial documents into a single document.
    - Remove backgrounds and other design elements such as ads, especially if they are unsuitable for a black and white printer.
    - Remove interactive elements, such as navigation controls and command buttons.
    - Make sure that all data is visible without scrollbars or hovering.

      Display version:



      Printer-friendly version:



In the printer-friendly version, all the data is visible on the printed page, and the interactive elements are removed.

- Replace links with their text equivalent.

  Acceptable:

  For more information, see UX Guide.

  Optimized for printing:

  For more information, see UX Guide (https://msdn.microsoft.com/windowsvista/uxguide).

- Convert light text on a dark background to dark text on a white background.

Include the right print options

The Print options common dialog provides options to:

- Select the printer and paper size.
- Set printer properties.
- Select the page range, number of copies, and collation.
- Use both sides of the paper.

Your program may require additional options, such as document content options (which content to print), format options (how to print, including print quality, picture sizes, fitting to frame), and color options. If you need to provide additional options, do so by extending the Print options common dialog. Don't create a custom Print dialog box.

When designing the Print options, consider the experience when printing multiple documents. Chances are the next print job will be very similar to the last print job. Optimize the default settings for reprints and similar print jobs don't make users start over completely each time.

Design print preview for performance and usability

An incorrect print job wastes time and money. For document creation programs, users should be able to evaluate the results before doing the actual printing. A print preview should allow users to:

- Evaluate margins, page breaks, page orientation, headers, and footers.
- Browse through all the pages.
- Print directly from the print preview.

Some complex documents (such as computer-aided design [CAD] drawings) can take a long time to render. The performance of the preview is important a print preview can become quite tedious if it takes awhile to render each page. **Consequently, it's better to have a print preview that renders quickly and is accurate enough to allow users to evaluate the print results than to have a completely accurate preview that renders slowly.**

When designing the print preview, consider the whole task of preparing to print. What are users going to be looking for? What are they going to change? Document creation programs should provide an interactive print preview so that users can adjust frequently changed settings like margins and line breaks within the preview.

However, to the best extent possible, your program should do the right thing by default. When necessary, warn about printing situations that are unlikely to be what the user intended. Don't rely on users finding problems using the print preview. For example, suppose a spreadsheet has too many columns to print on a single page in portrait

mode. While the program could present a confirmation dialog box, a better solution is to print in landscape mode automatically.

If you do only five things...

1. Design a printing experience appropriate for your program type.
2. Review your program's scenarios that involve printing and to the best extent possible, make the need to print optional.
3. Provide useful printing extensions by customizing the Print common dialog. Don't create a custom Print dialog box for this purpose.
4. Optimize the Print options for reprints and similar print jobs.
5. Provide a preview feature whenever appropriate.

Printing patterns

The type of program is the primary indicator of the appropriate printing experience:

Expand table

Label	Value
Advanced document creation Used to create, view, and print high-end documents. The ability to create high-quality printouts is one of the main reasons why the program exists. Targeted at expert users.	**User goals:** Perfect results detailed control over the print output. **Example:** Microsoft Word **Recommended printing experience:** • Output optimized for print (WYSIWYG). • Advanced document formatting features, with options to print large objects. • Advanced print options, including headers and footers. Document-related print options are saved within the document itself. • Fast, accurate, powerful print previews.
Intermediate document creation Used to create and view more complex documents. The ability to create good-quality printouts is important, but not necessarily one of the main reasons why the program exists. Targeted at intermediate users.	**User goals:** Good results with minimal effort. Some control over the print output. **Examples:** Most Microsoft Office programs, such as Outlook and Excel. **Recommended printing experience:** • Output optimized for print (WYSIWYG). • Some document formatting features, with ability to print large objects without truncation.

Label	Value
	• Some custom print options, including headers and footers. • Accurate, easy to use print previews.
Simple document creation Used to create and view simple documents. Targeted at all users.	**User goals:** Basic printing support with standard printing options. Users expect good results without any tweaking. **Examples:** WordPad, Paint. **Recommended printing experience:** • Output may be optimized for print (WYSIWYG), but this is not required. • Some document formatting features, with ability to print large objects without truncation. • Standard print options; custom print options are optional. • Simple or no print previews.
Document viewers Used to view documents. Users can't change the document content or format.	**User goals:** Basic printing support with standard printing options. Users expect good results without any tweaking. Printing problems are handled automatically because users can't modify the document. **Example:** Windows Internet Explorer **Recommended printing experience:** • Output may be optimized for print (WYSIWYG), but this is not required. • Program automatically handles page breaks, eliminates blank pages, handles large objects, and removes backgrounds and other design elements. • Standard print options; custom print options are optional. • Simple or no print previews.
Utilities or line-of-business applications Used to perform simple, specific tasks. Targeted at all users.	**User goals:** Ability to export selected data efficiently. Users expect good results without any tweaking. Often for such programs, users are pleasantly surprised to find any printing support at all. **Recommended printing experience:** • Printing support is optional depending upon supported scenarios.

Label	Value
	• Output may be optimized for print (WYSIWYG), but this is not required. • Some document formatting features. Might be acceptable if large objects are truncated. • Standard print options. • Print previews optional.

Guidelines

General

- **Don't print blank pages or pages with just headers and footers.** However, print blank pages if the headers or footers contain page numbers and those page numbers might referenced elsewhere.
- **Completely spool out any pending print jobs before shutting down a program.**

Formatting pages

- **Reformat text layout to fit within the target page size.** Never truncate text.
- If users don't control the format of the document:
  - **Automatically handle large objects by scaling, rotating, or splitting across pages.** For more guidelines about printing large objects, see Oversized objects.
  - **Optimize the page breaks to eliminate blank and nearly blank pages.**
  - **Convert light text on a dark background to dark text on a white background.**
  - **Remove backgrounds and other design elements,** especially if they are unsuitable for a black and white printer.
- **If your program presents separate partial documents, provide a printer-friendly format option to consolidate them into a single document for printing.**
- **Remove interactive elements:**
  - Remove navigation controls and command buttons.
  - Make sure that all data is visible without scrollbars.
  - Replace links with their text equivalent.

    Acceptable:

    For more information, see UX Guide.

Optimized for printing:

For more information, see UX Guide (https://msdn.microsoft.com/windowsvista/uxguide).

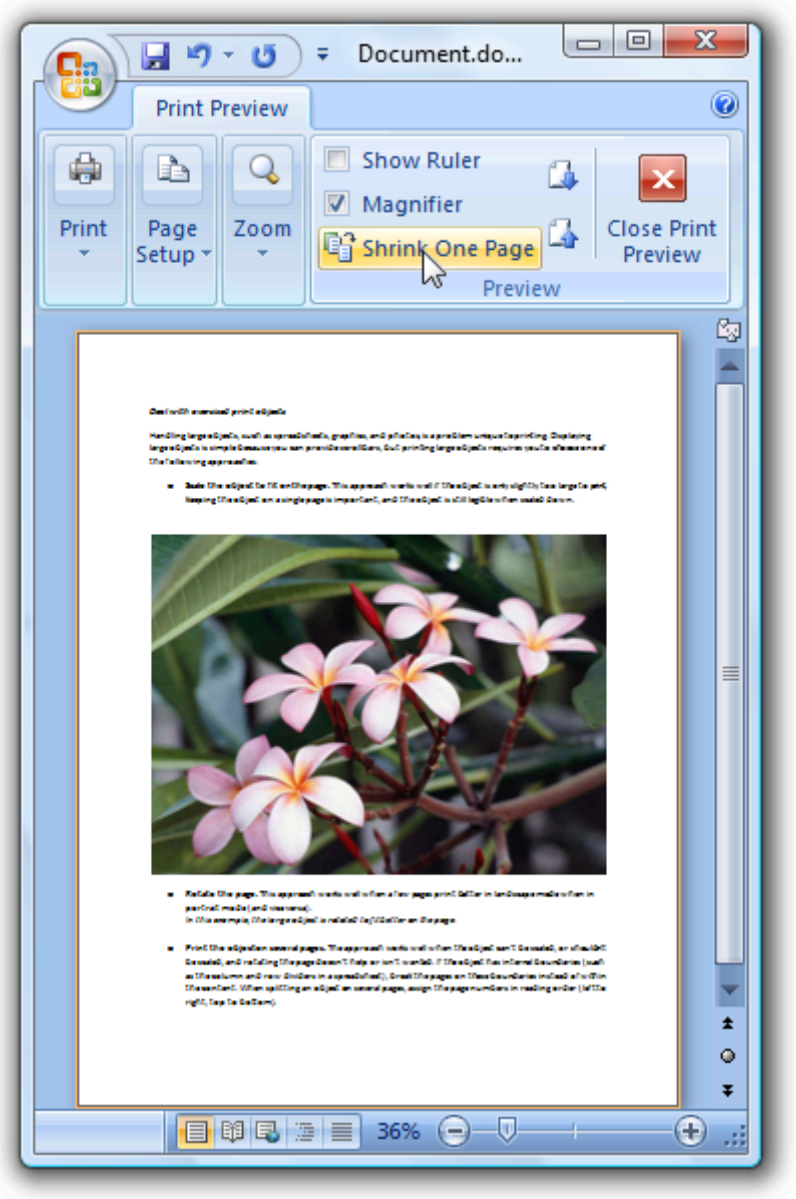
In this example, the link is replaced with its text equivalent in parentheses.

- Move useful information displayed on hover to inline.

Oversized objects

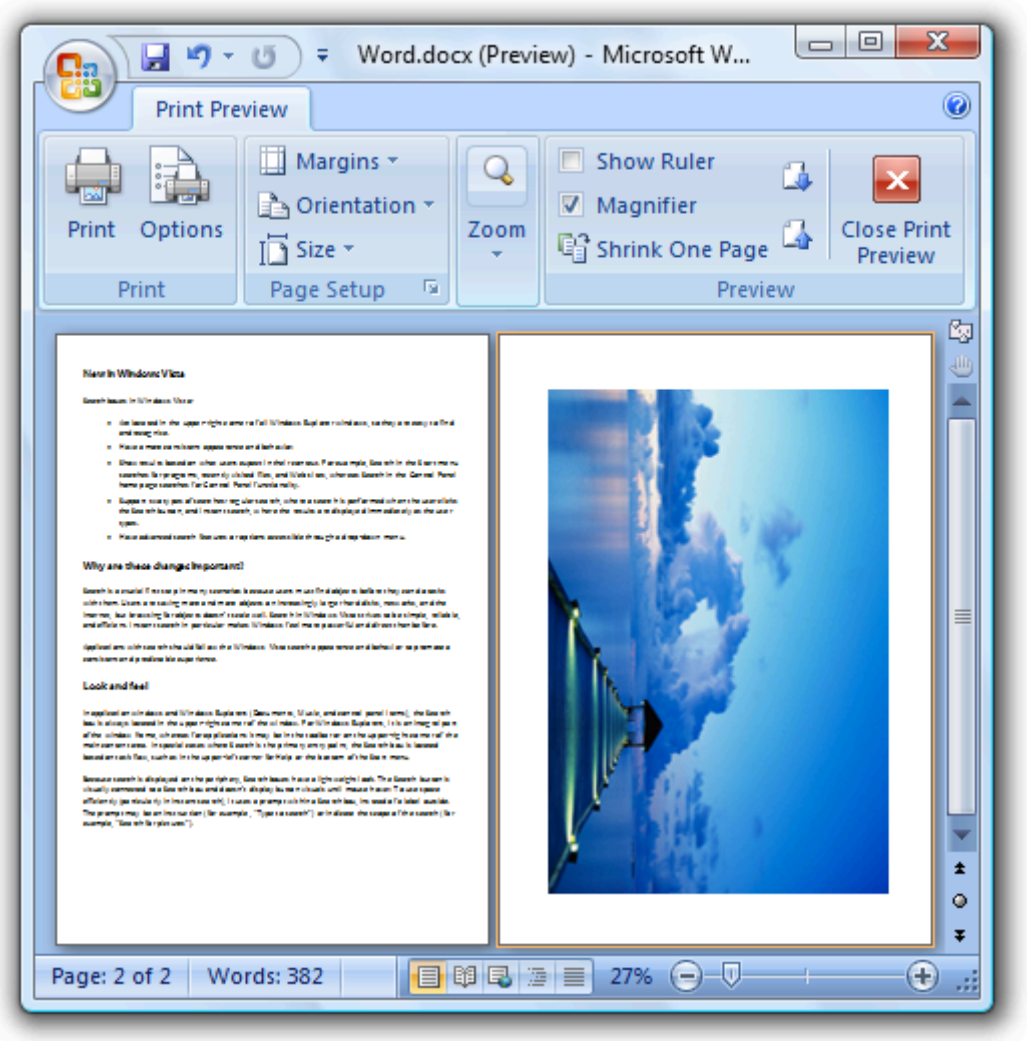
Handling large objects, such as spreadsheets, graphics, and photos, is a problem unique to printing. Choose one of the following approaches:

- **Scale the object to fit on the page.** This approach works well if the object is only slightly too large to print, keeping the object on a single page is important, and the object is still legible when scaled down.



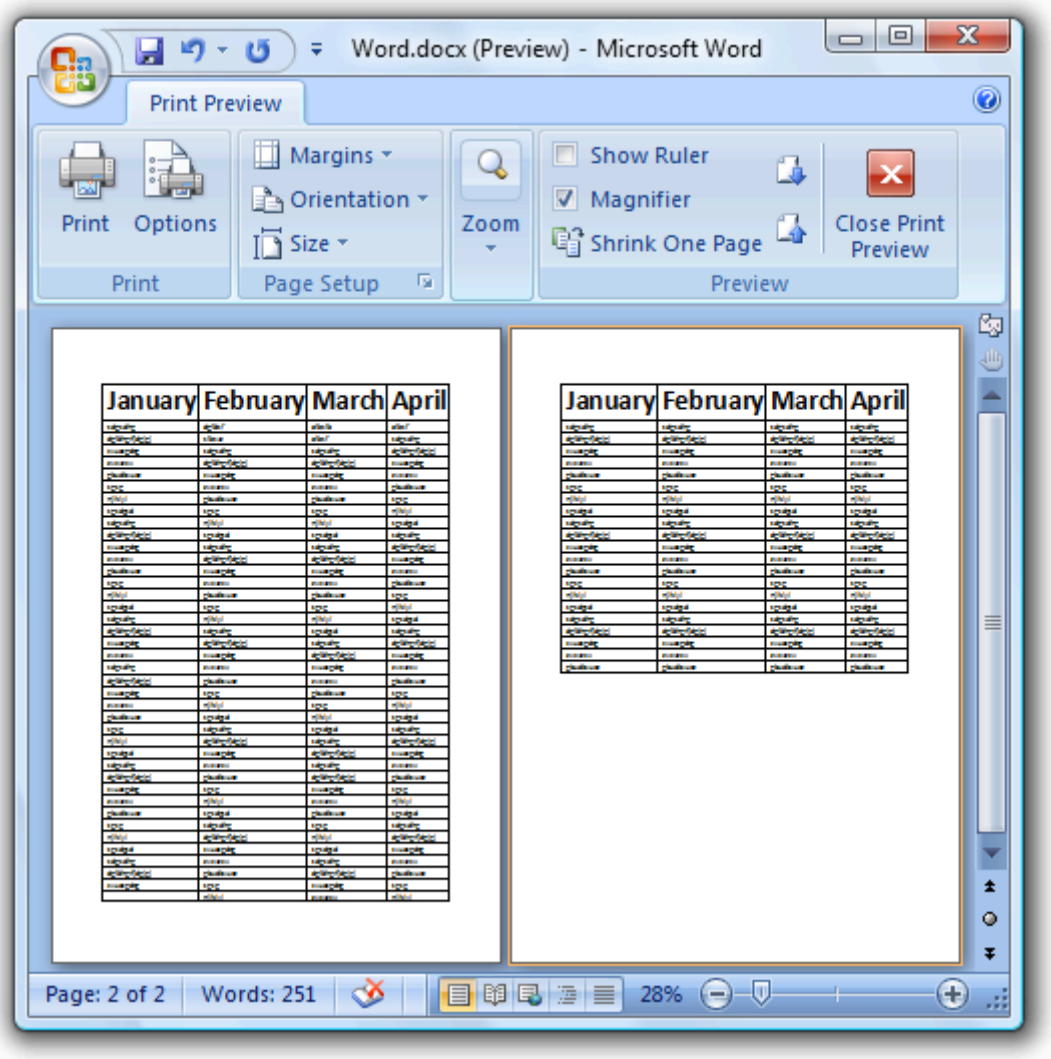
In this example, the large image is scaled to fit on the page.

- **Rotate the page.** This approach works well when a few pages print better in landscape mode when in portrait mode (and vice versa).



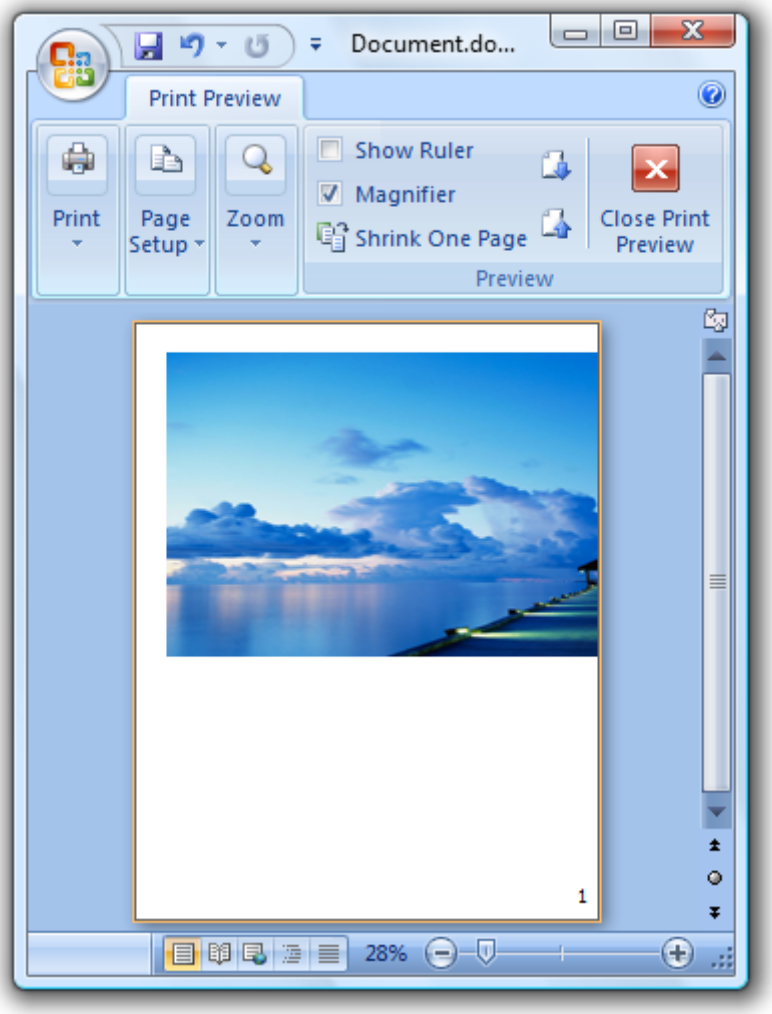
In this example, the large image is rotated to fit better on the page.

- **Print the object on several pages.** The approach works well when the object can't be scaled, or shouldn't be scaled, and rotating the page doesn't help or isn't wanted. If the object has internal boundaries (such as the column and row dividers in a spreadsheet), break the pages on these boundaries instead of within the content. Also, repeat any elements required to understand the page, such as legends or column headers. When splitting an object on several pages, assign the page numbers in reading order (left to right, top to bottom).



In this example, the large table is printed on two pages. Column headers persist from page to page to facilitate quick comprehension.

- **Truncate the object** (printing only the part of the object still visible after truncation). This approach is the simplest solution to implement, but likely to be the least acceptable. Also note that truncating is never acceptable for text.



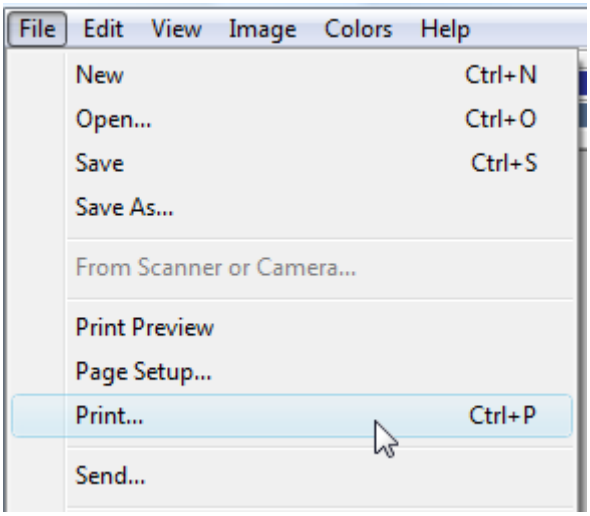
In this example, the large image is truncated.

Headers and footers

- **Provide headers and footers for advanced and intermediate document creation programs.** Consider providing headers and footers for other types of programs if they are used for multipage documents.
- **Make headers and footers customizable.** Allow users to define the left, center, and right portions.
  - For headers, put the document name on the left side by default.
  - For footers, put the document copyright or source on the left side, and the page number on the right side, by default.
- **Use friendly file path and URLs.** Display spaces as spaces, not "%20."

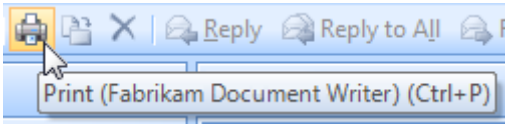
Print commands

- **For menu bars and shortcut menus, use the Print command that displays the Print options common dialog.** Use an ellipsis to indicate that additional information is required.



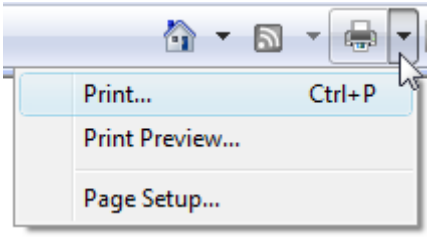
In this example, the Print command has an ellipsis to indicate that it will display the Print options common dialog to get more information.

- **For toolbars used with a menu bar, use an immediate Print command.** Clicking the button prints a single copy of the document to the default printer. Such toolbar commands should be immediate. To indicate that the command is immediate, put the default printer in the tooltip. Users can access the full Print command from the menu bar.



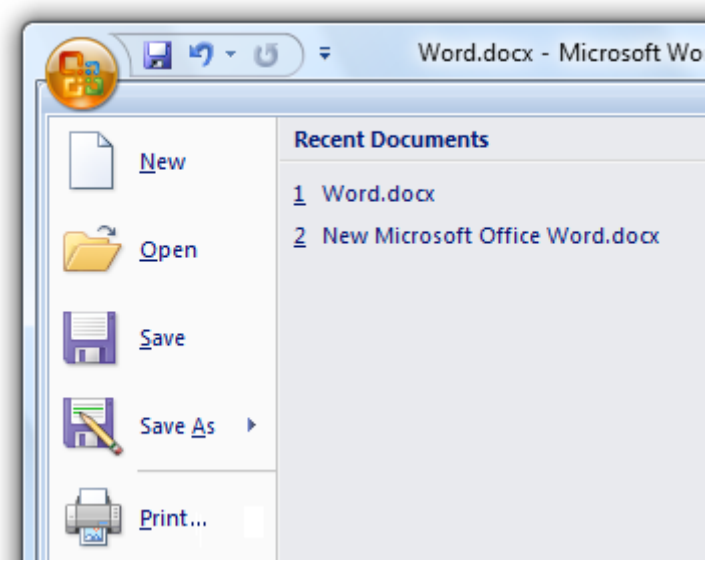
In this example, the Print command in a toolbar prints immediately instead of displaying the Print options common dialog. Putting the default printer in the tooltip provides textual reinforcement that the user is bypassing the dialog.

- **For toolbars used without a menu bar, use a Print split button.** Clicking the button prints a single copy of the document to the default printer. Clicking the arrow portion of the button drops down a menu with full Print, Print preview, and Page setup commands.



In this example, the Windows Internet Explorer toolbar uses a split button control to provide all the print commands.

- **For the ribbon command user interface, put the Print command in the application menu.**

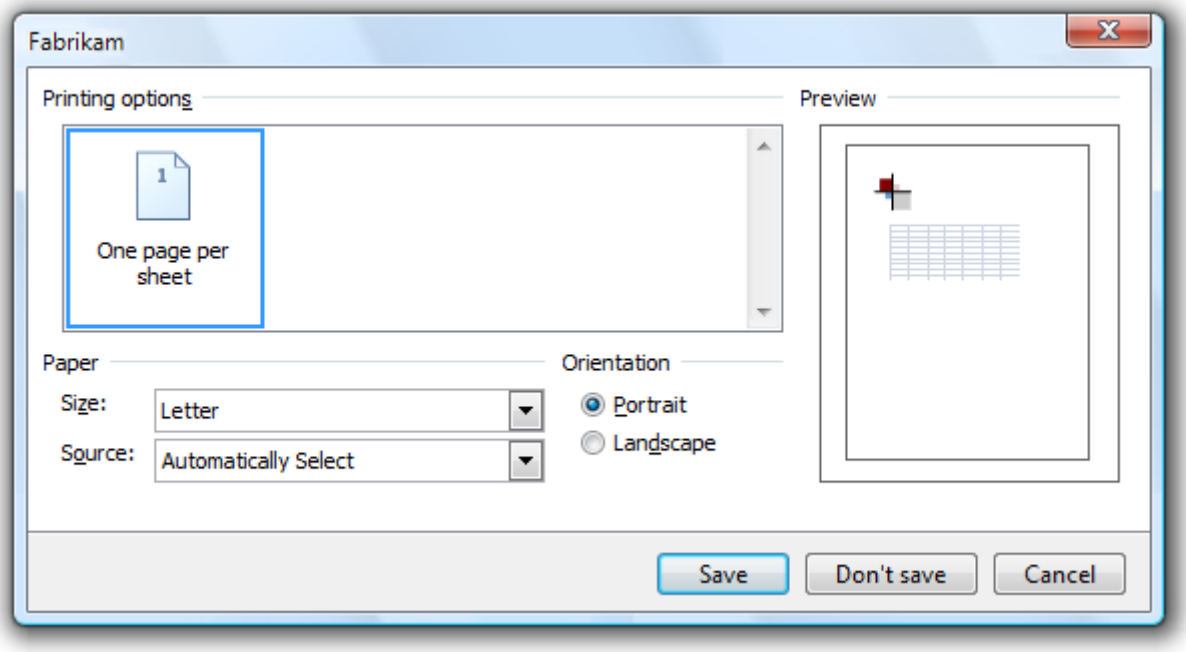


For ribbons, the Print command is accessed using the application menu.

Print options

- **Don't create a custom Print options dialog box.** If you must provide additional options, extend the Print options common dialog. Don't use a separate dialog for additional print options.

Incorrect:



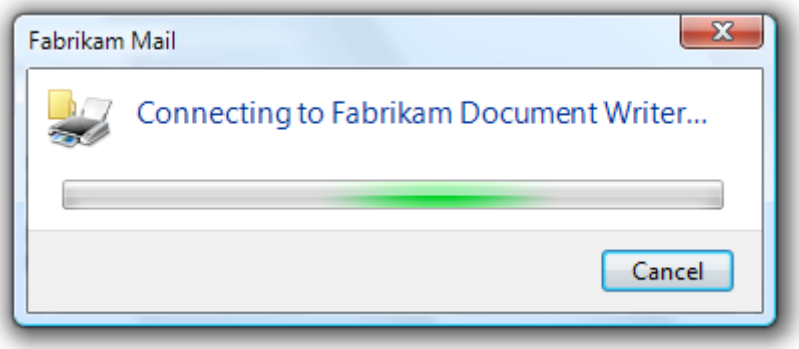
In this example, Fabrikam incorrectly uses a separate dialog for additional print options.

Developers: For information about how to extend the Print common dialog, see PRINTDLGEX Structure.

- **When extending the Print options common dialog, don't duplicate any features already provided.**
- **If users are likely to maintain settings from one print job to the next, make those settings the defaults.** For the first print job after program launch, use the standard default values, including the default printer. For subsequent print jobs in the program instance, preserve the last selected printer and paper size. Don't preserve the number of copies or page ranges, because these are far less likely to be reselected later.
- **Optimize the settings by removing options that currently don't apply.** Remove options that are inconsistent with the capabilities of the selected printer or characteristics of the current document. For example, a photo printing application could limit the combinations of paper size, paper type, and print quality that give the best results, so choosing a glossy paper option might remove envelopes from the paper formats. If for any reason users want to see all the options, you can provide this ability through a control such as a check box.

Developers: To learn how to determine the capabilities of the selected printer, see Print Schema.

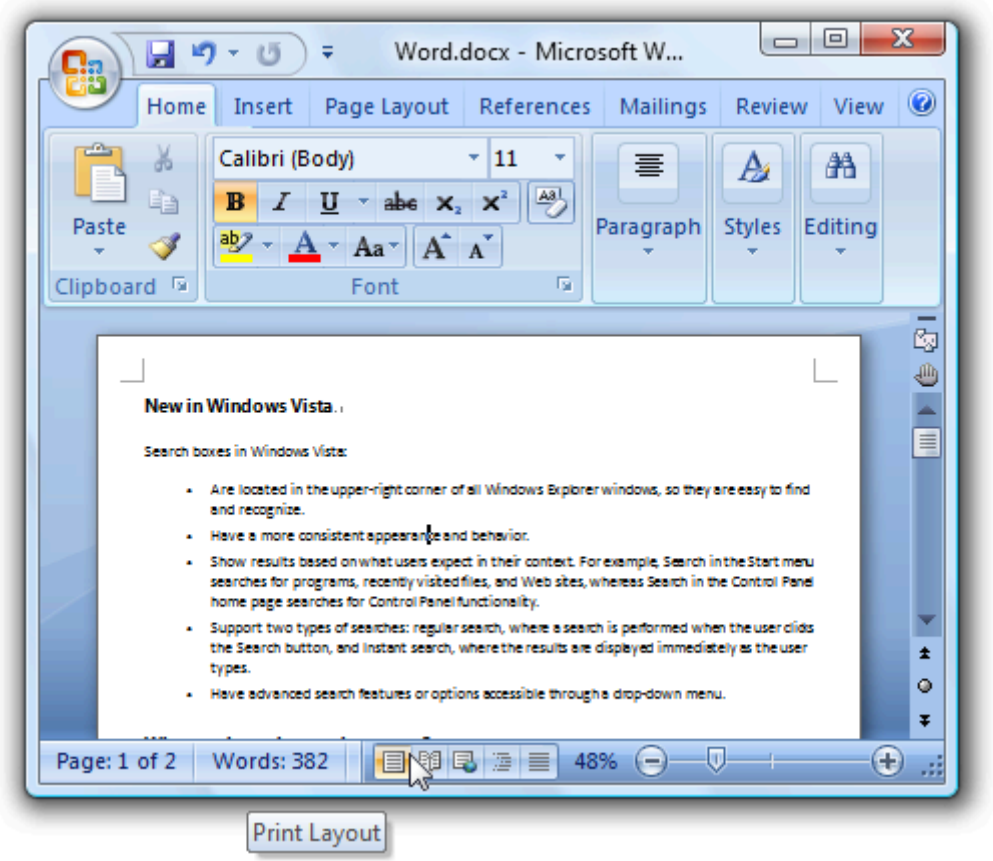
- **For advanced document creation programs, save the document-related print options within the document itself.** For these programs, the print options are an integral part of the document.
- **For other types of programs, save settings on a per-user basis.**
- **Consider selecting a non-default printer for specialized printing.** For example, a photo printing application could always select the printer last used by the program, regardless of the system default printer. Doing so assumes that the system default printer isn't likely to be a photo printer. Such programs should save the setting for the last selected printer.
- **Don't lock up your program while detecting printer capabilities.** Doing so presents a poor user experience. Instead, either:
  - Perform the printer capability detection in a separate thread.
  - Time out after 10 seconds.
  - Provide a dialog box to allow users to cancel.



In this example, the dialog box makes it easy to cancel the printer capability detection if the user decides the task is taking too long.

Print previews

- **Provide a print preview feature whenever appropriate.** All document creation programs benefit from print previews, but users don't expect them in simple document creation programs. For advanced document creation programs, consider having print preview support directly within the main program window.



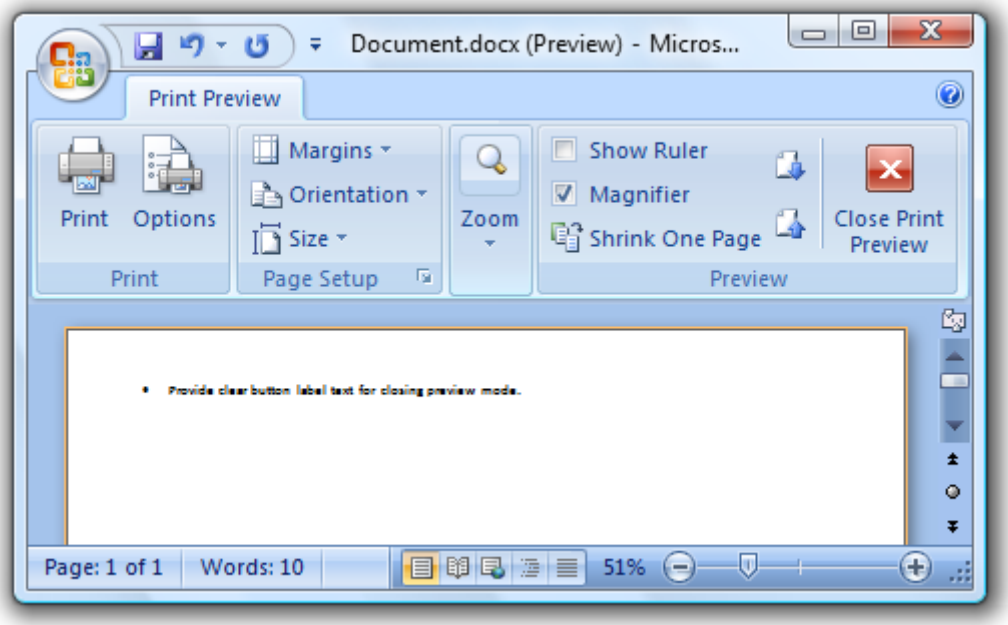
In this example, Word has print preview support within the main program window.

- Provide print preview features that allow users to:
  - Evaluate margins, page breaks, page orientation, headers, and footers.
  - Browse through all the pages.
  - Print directly from the print preview.

Consider providing an interactive print preview so that users can adjust frequently changed settings like margins and line breaks directly within the preview.

- **Have print preview pages render within one second.** It's better to have a print preview that renders quickly and is accurate enough to allow users to evaluate the print results than to have a completely accurate preview that renders slowly.

- **For advanced document creation programs, consider extending the standard Print dialog box by incorporating preview functionality directly within it,** rather than creating a separate dialog for it.
- **Provide an obvious button for closing preview mode.**



The Print Preview mode in Word has an obvious close preview command.

Printing errors

Note: Once the print job has been spooled to the printer, Windows is responsible for any subsequent errors. Your program only has to handle errors that happen before the print job is spooled.

- **Before spooling a print job, check for any potential printing problems the user can fix.** Present a clear, concise confirmation before continuing to print. Whenever possible, offer to fix the problem automatically. Doing so can prevent a waste of time and money.

Text

- For the option to print on both sides of the paper, label the option Print double-sided. Don't use the phrase Manual Duplex.

Documentation

- Use print, not print out, as a verb.
- It's acceptable to use printout to refer to the result of a print job.

- Use print queue, not printer queue.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Windows Environment

Article • 02/09/2021

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

The Windows environment is the onscreen work area provided by Windows, analogous to a physical desktop, and the operating system's core extension points. Learn how to leverage the desktop, taskbar, notification area, control panels, help, and user account control for your app.

In this section

Expand table

Topic	Description
Desktop	The desktop is the user's work area for their programs. It's not a way to promote awareness of your program or its brand. Don't abuse it!
Taskbar	The taskbar is the access point for programs displayed on the desktop. With the new Windows 7 taskbar features, users can give commands, access resources, and view program status directly from the taskbar.
Notification Area	The notification area provides notifications and status. Well-designed programs use the notification area appropriately, without being annoying or distracting.
Control Panels	Use control panel items to help users configure system-level features and perform related tasks. Programs that have a user interface should be configured directly from their UI instead.
Help	Use Help as a secondary mechanism to help users complete and better understand tasks the primary mechanism being the UI itself. Apply these guidelines to make the content truly helpful and easy to find.
User Account Control	A well designed User Account Control experience helps prevent unwanted system-wide changes in a way that is predictable and requires minimal effort.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

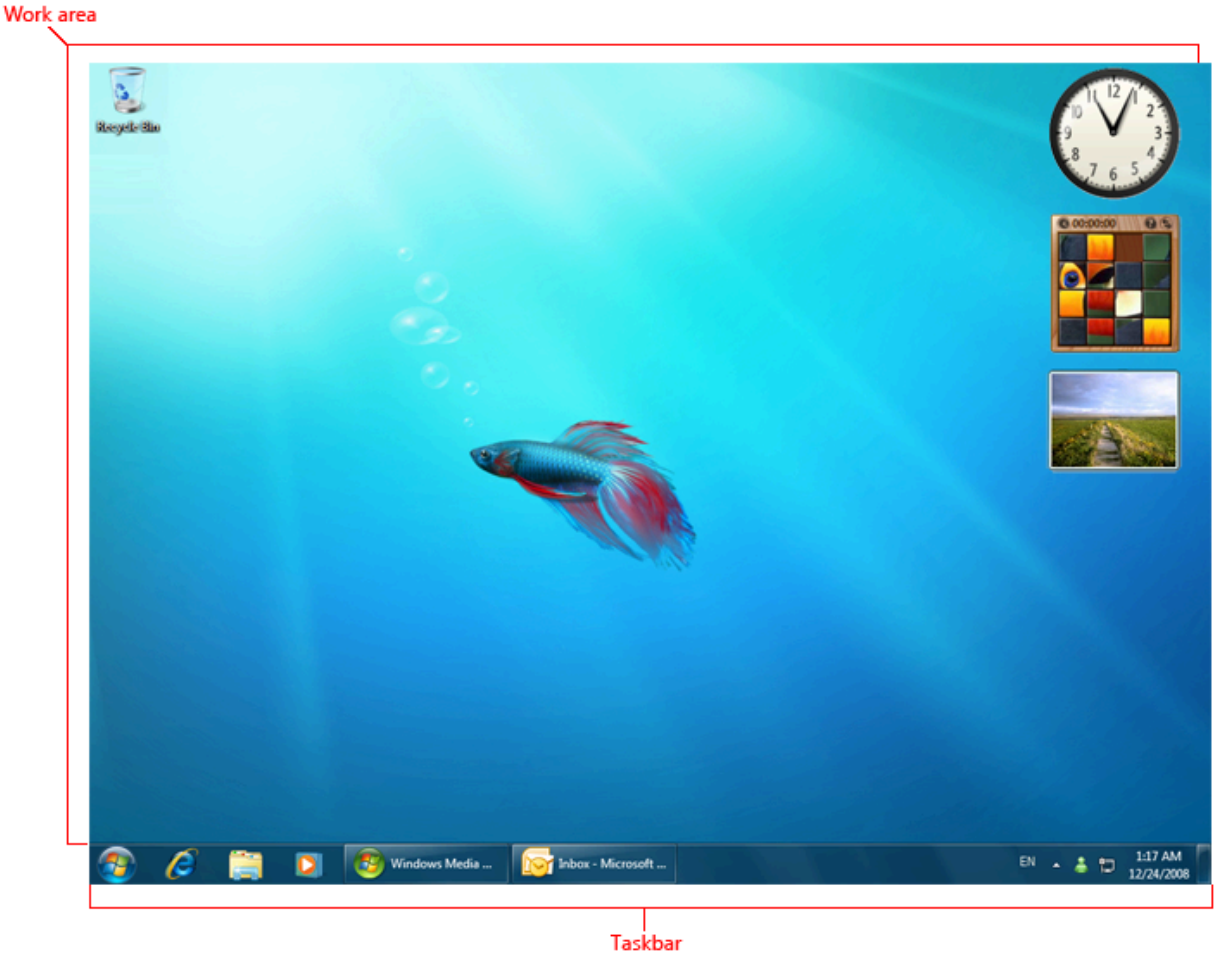
The Windows 7 Desktop

Article • 02/08/2022

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

The Windows 7 desktop is the user's work area for their programs. It's not a way to promote awareness of your program or its brand. Don't abuse it!

The desktop is the onscreen work area provided by Microsoft Windows, analogous to a physical desktop. It consists of a work area and taskbar. The work area may span multiple monitors.



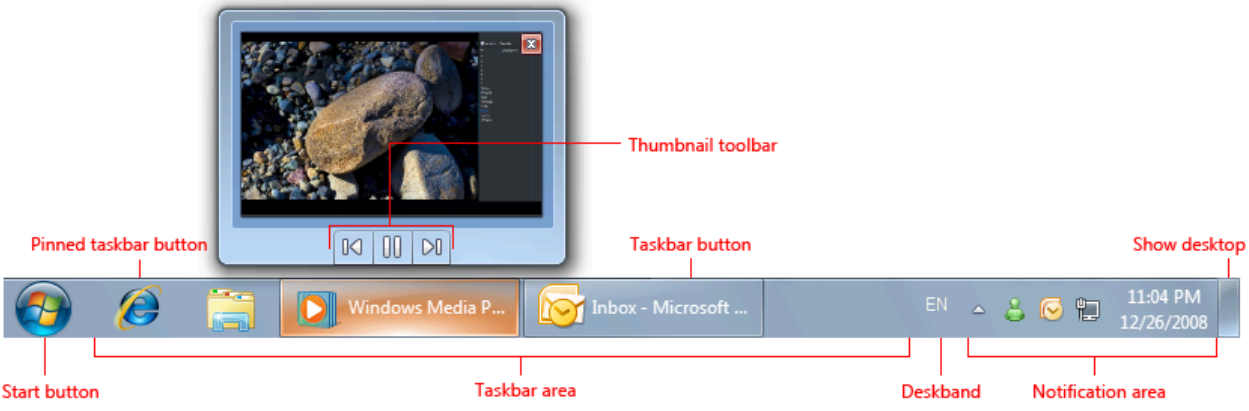
A typical Windows desktop.

The active monitor is the monitor where the active program is running. The default monitor is the one with the Start menu, taskbar, and notification area.

Design concepts

The Windows desktop has the following program access points:

- **Work area.** The onscreen area where users can perform their work, as well as store programs, documents, and their shortcuts. While technically the desktop includes the taskbar, in most contexts it refers just to the work area.
- **Start button.** The access point for all programs and special Windows places (Documents, Pictures, Music, Games, Computer, Control Panel), with "most recently used" lists for quick access to recently used programs and documents.
- **Quick Launch. Removed from Windows 7.** A direct access point for programs selected by the user.
- **Taskbar.** The access point for running programs that have desktop presence. While technically the taskbar spans the entire bar from the Start button to the notification area, in most contexts taskbar refers to the area in between, containing the taskbar buttons. This area is sometimes referred to as the taskband.
- **Deskbands. Not recommended.** Minimized functional, long-running programs, such as the Language Bar. Programs that minimize to deskbands don't display taskbar buttons when minimized.
- **Notification area.** A short-term source for notifications and status, as well as an access point for system- and program-related features that have no presence on the desktop.



The Windows desktop access points include the Start button, taskbar, and notification area. Note the thumbnail feature of the taskbar button.

The Windows desktop is a limited, shared resource that is the user's entry point to Windows. Leave users in control. You should use its areas as intended—any other usage should be considered an abuse. Never view them as ways to promote awareness of your program or its brand.

For Windows 7, Original Equipment Manufacturers (OEMs) and Independent Hardware Vendors (IHVs) can use Device Stage to design a customized, branded UI for the computer and devices, without cluttering users' desktops. OEMs in particular can use Device Stage PC to feature custom programs, service offerings, and support. For more information, see the Microsoft Device Experience Development Kit.

If you do only one thing...

- Don't abuse the desktop—keep users in control. If your target users are likely to use your program frequently, provide an option during setup to put a shortcut on the desktop, unselected by default.

Guidelines

- **If your users are very likely to use your program frequently, provide an option during setup to put a program shortcut on the desktop.** Most programs won't be used frequently enough to warrant offering this option.
- **Present the option unselected by default.** Requiring users to select the option is important because once undesired icons are on the desktop, many users are reluctant to remove them. This can lead to unnecessary desktop clutter.
- **If users select the option, provide only a single program shortcut.** If your product consists of multiple programs, provide a shortcut only to the main program.
- **Put only program shortcuts on the desktop.** Don't put the actual program or other types of files.

  Correct:

  Incorrect:

  In the incorrect example, the program, not a shortcut, is copied to the desktop.

- **Choose a label that can be displayed without truncation.** Users shouldn't see an ellipsis.

  Correct:

  Incorrect:

  In the incorrect example, the program shortcut label is so long that it is truncated.

Documentation

- When referring to the desktop, use desktop, uncapitalized.
- When referring to desktop shortcuts, use shortcut, uncapitalized.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

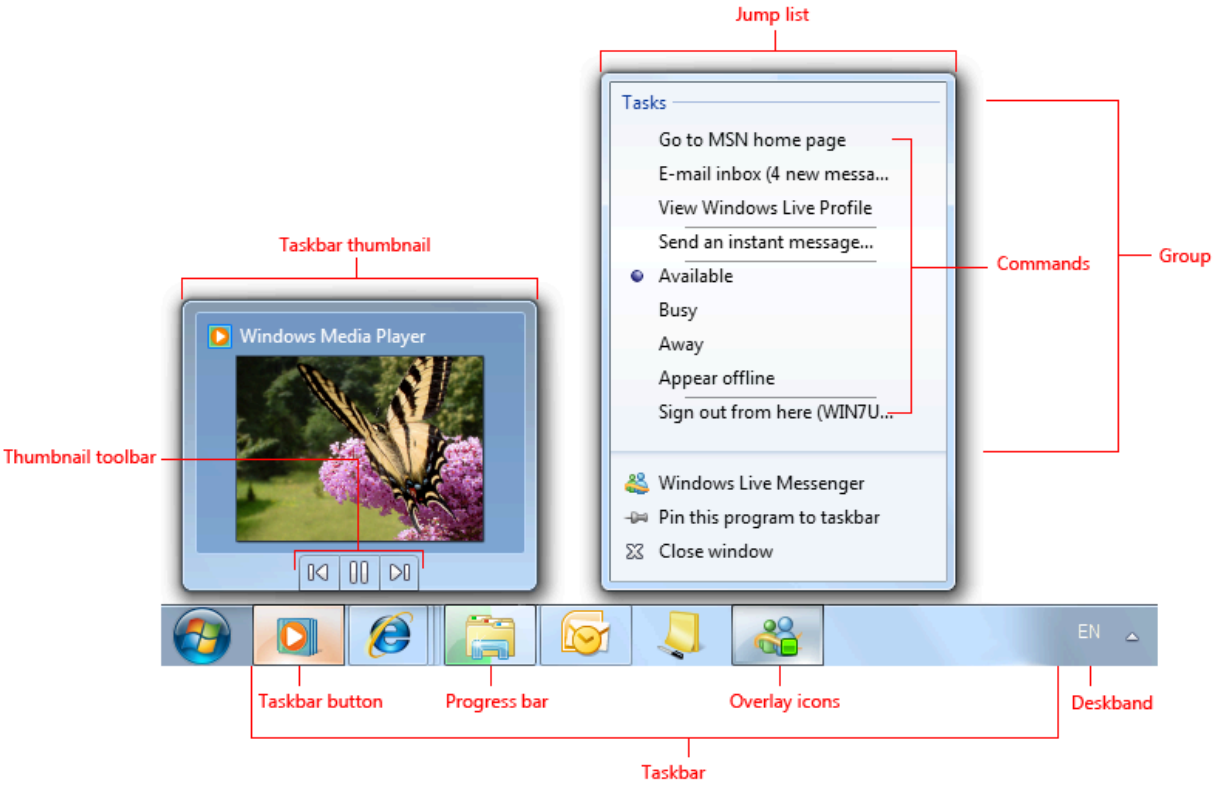
Taskbar

Article • 06/03/2021

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

The taskbar is the access point for programs displayed on the desktop. With the new Windows 7 taskbar features, users can give commands, access resources, and view program status directly from the taskbar.

The taskbar is the access point for programs displayed on the desktop, even if the program is minimized. Such programs are said to have desktop presence. With the taskbar, users can view the open primary windows and certain secondary windows on the desktop, and can quickly switch between them.



The Microsoft Windows taskbar.

The controls on the taskbar are referred to as taskbar buttons. When a program creates a primary window (or a secondary window with certain characteristics), Windows adds a taskbar button for that window and removes it when that window closes.

Programs designed for Windows 7 can take advantage of these new taskbar button features:

- Jump Lists provide quick access to frequently used destinations (like files, folders, and links) and commands through a context menu accessible from the program's taskbar button and Start menu item even if the program isn't currently running.
- Thumbnail toolbars provide quick access to frequently used commands for a particular window. Thumbnail toolbars appear in the taskbar button's thumbnail.
- Overlay icons show change of status on the program's taskbar button icon.
- Progress bars show progress for long-running tasks on the program's taskbar button.
- Sub-window taskbar buttons allow users to use taskbar button thumbnails to switch directly to window tabs, project windows, multiple-document interface (MDI) child windows, and secondary windows.
- Pinned taskbar buttons allow users to pin program buttons to the taskbar to provide quick access to programs even when they aren't running.

Technically, the taskbar spans the entire bar from the Start button to the notification area; more commonly, however, the taskbar refers only to the area containing the taskbar buttons. For multiple monitor configurations, only one monitor has a taskbar, and that monitor is the default monitor.

Note: Guidelines related to desktop, notification area, and window management are presented in separate articles.

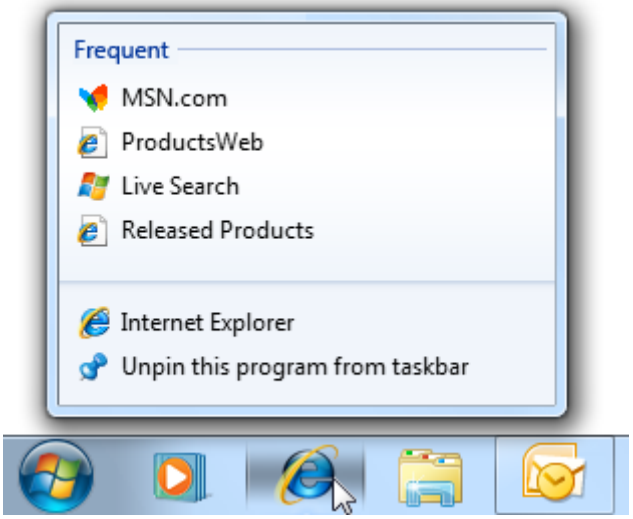
Is this the right user interface?

Programs designed for Windows 7 can take advantage of these taskbar button features. Ask yourself the following key questions to determine whether or not to use them:

Jump Lists

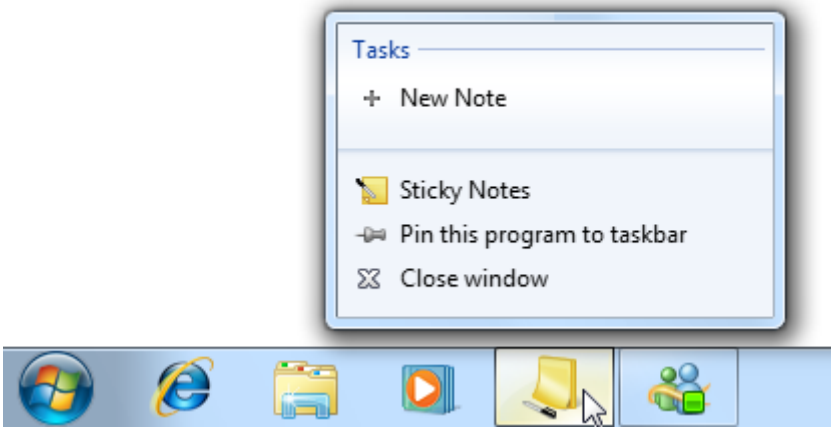
- **Do users often need to start new tasks using your program?** If so, consider providing a Jump List. While Jump Lists can be used for other purposes, most scenarios involve starting a new task.

- **Do users often need to access recently or frequently used files, folders, links, or other resources?** If so, consider providing a Jump List to access these useful resources.



In this example, Windows Internet Explorer uses a Jump List to present frequently visited pages.

- **Do users often need quick access to a small number of your program's commands while using other programs, even if your program isn't running?** If so, consider providing a Jump List with these frequently used commands. These commands must work even if your program isn't running, and must apply to the entire program, not a specific window. As an alternative, consider providing a thumbnail toolbar for commands that apply to a specific window.



In this example, the Sticky Notes accessory allows users to create a new note quickly while using other programs.

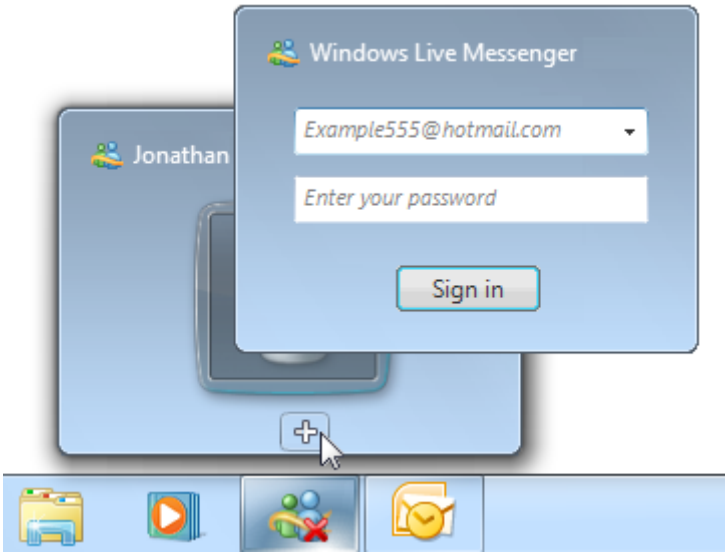
- **Are you promoting new, single use, or hard to find features?** If so, don't use Jump Lists because they aren't intended for this purpose. Instead, improve the discoverability of such commands directly in the program.

Thumbnail toolbars

Do all of the following conditions apply?

- **Do the commands apply to a specific window?** Thumbnail toolbars are for commands that apply to existing tasks, whereas Jump List commands are for starting new tasks.
- **Do users need to interact with a running task quickly while using other programs?** If so, thumbnail toolbars are a good choice. Thumbnail toolbars can present a maximum of seven commands, but a maximum of five commands is generally preferred.
- **Are the commands immediate?** That is, do they not require additional input? Thumbnail toolbars need to have immediate commands to be efficient, whereas Jump Lists work better with commands that require additional input.

  Incorrect:

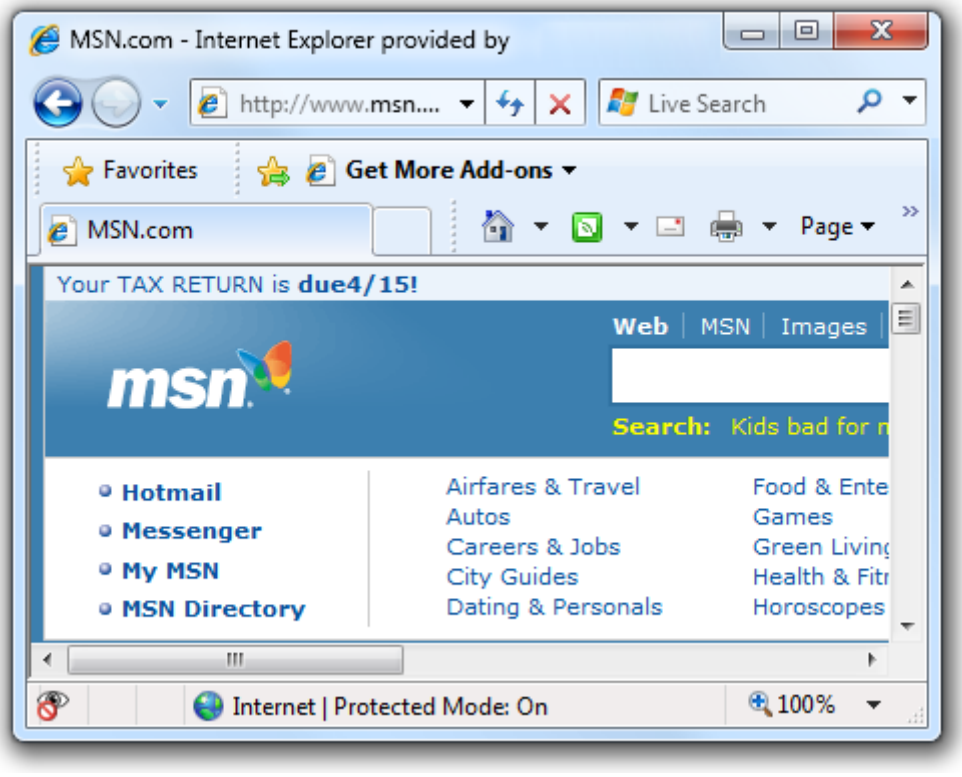
  Commands that require additional input don't work well on thumbnail toolbars.
- **Are the commands direct?** That is, can users interact with them using a single click? Toolbars need to have direct commands to be efficient.
- **Are the commands well represented by icons?** Thumbnail toolbar commands are presented using icons not text labels, whereas Jump List commands are represented by text labels.

  Incorrect:

In this example, the command isn't well represented by icons.

Overlay icons

- **Does the program have "desktop presence"?** If not, use a notification area icon instead. If so, consider using an overlay icon instead of putting status on the notification area icon for programs designed for Windows 7. Doing so ensures that the icon will always be visible (when large icons are used), and consolidates the program with its status in one place.
- **Is the overlay icon displayed temporarily to show a change of status?** If so, an overlay icon may be appropriate, depending on the following factors:
    - **Is the status useful and relevant while using other programs?** If not, display the information in the program's status bars or other program status area.



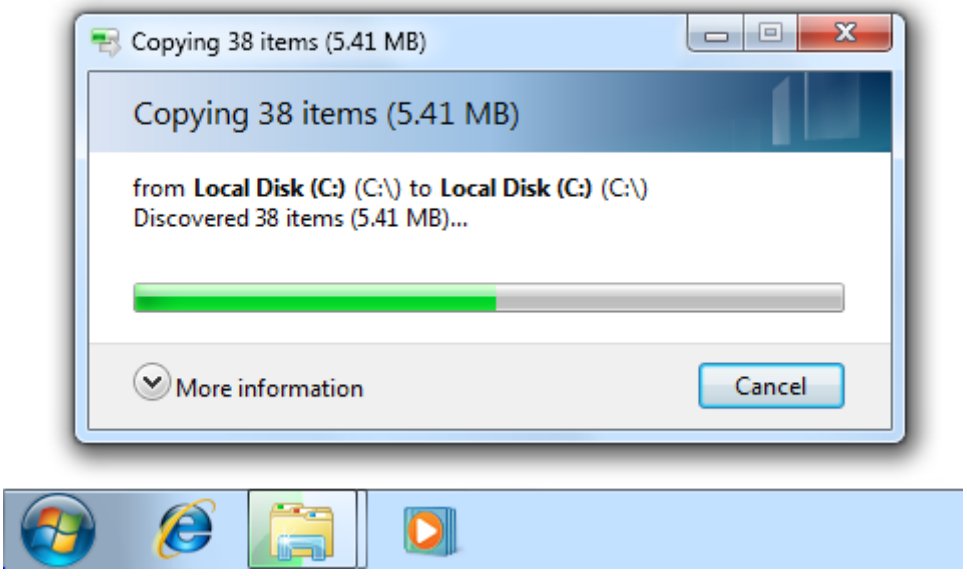
In this example, the status bar is used because the status isn't useful when using other programs.

- **Is the status showing progress?** If so, use a taskbar button progress bar instead.
- **Is the status critical? Is immediate action required?** If so, display the information in a way that demands attention and cannot be easily ignored, such as a dialog box.

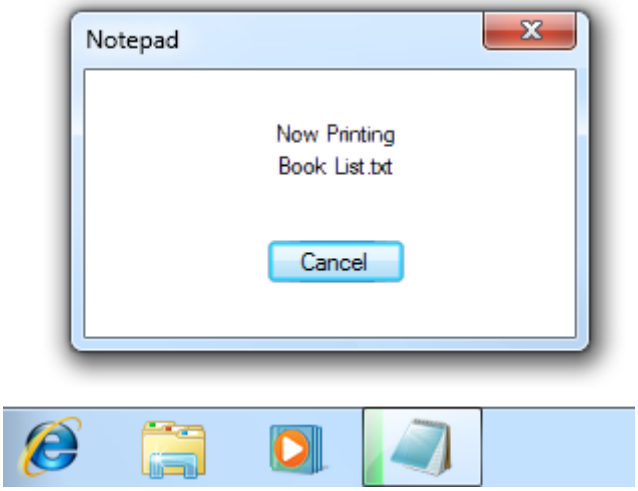
Progress bars

- **Is the progress feedback useful and relevant while using other programs?** That is, are users likely to monitor the progress while using other programs, and change their behavior as a result? Such useful and relevant status is usually displayed using a modeless progress dialog box or a dedicated progress page, but not with a busy pointer, activity indicator, or progress bar on a status bar. If the status isn't useful when using other programs, just display the progress feedback directly in the program itself.

  Correct:



  Incorrect:



In the incorrect example, the taskbar button progress bar isn't very useful.

- **Is the task continuous?** If the task never completes, there's no need to show its progress. Examples of continuous tasks include antivirus scans that aren't initiated by users, and file indexing.

  Incorrect:

  In this example, a continuous task doesn't need to show progress.

Sub-window taskbars

- **Does your program contain tabs, project windows, MDI child windows, or secondary windows that users would often want to switch to directly?** If so, giving these windows their own taskbar button thumbnails may be appropriate.

Design concepts

Using Jump Lists and thumbnail toolbars effectively

Jump Lists and thumbnail toolbars help users access resources and perform commands more efficiently. However, when designing how your program supports these features, don't take improved efficiency for granted. If users can't accurately predict which feature has the command they need, or they have to check multiple places, eventually users will become frustrated and stop using these features.

Jump Lists and thumbnail toolbars work together most effectively when they are:

- **Clearly differentiated.** Users know when to look for a destination or command in a Jump List, and when to look in a thumbnail toolbar. There is a clear purpose for each, so users rarely confuse the contents of the two. Generally, Jump Lists are used to start new tasks, whereas thumbnail toolbars are used to interact with running tasks while using other programs.
- **Useful.** The destinations and commands offered are the ones that users need. If users aren't likely to need something, it isn't included. Don't use the maximum number of items if they aren't needed.
- **Predictable.** The destinations and commands offered are the ones that users expect to find. Users rarely have to look in more than one place.
- **Well organized.** Users are able to find what they are looking for quickly. They use descriptive yet concise labels, and suitable icons to aid recognition.

Be sure to do user research to make sure you've got it right. If you ultimately find that you can't design Jump Lists and thumbnail toolbars together that achieve these goals, consider providing only one of them. It's better to have one predictable way to give commands than two confusing ones.

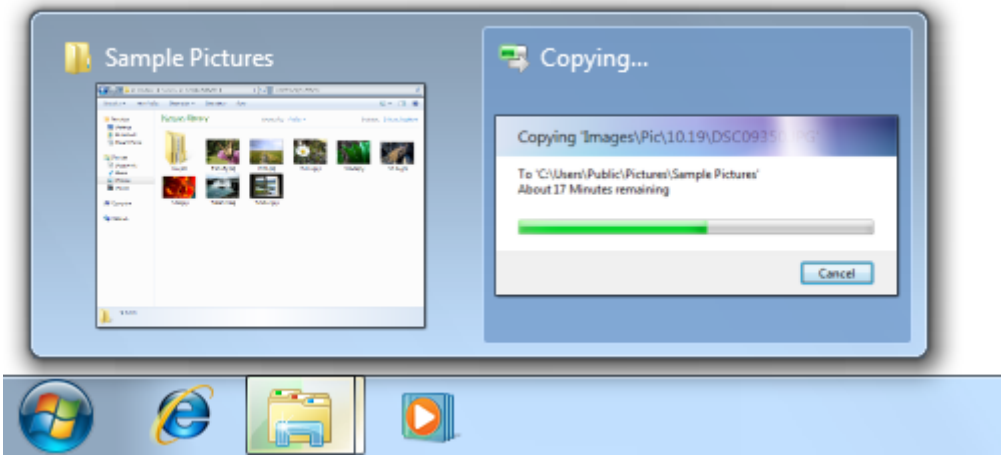
Guidelines

Taskbar buttons

- **Make the following window types appear on the taskbar (for Windows 7, by using a taskbar button thumbnail):**
  - Primary windows (which includes dialog boxes without owners)
  - Property sheets
  - Modeless progress dialog boxes
  - Wizards

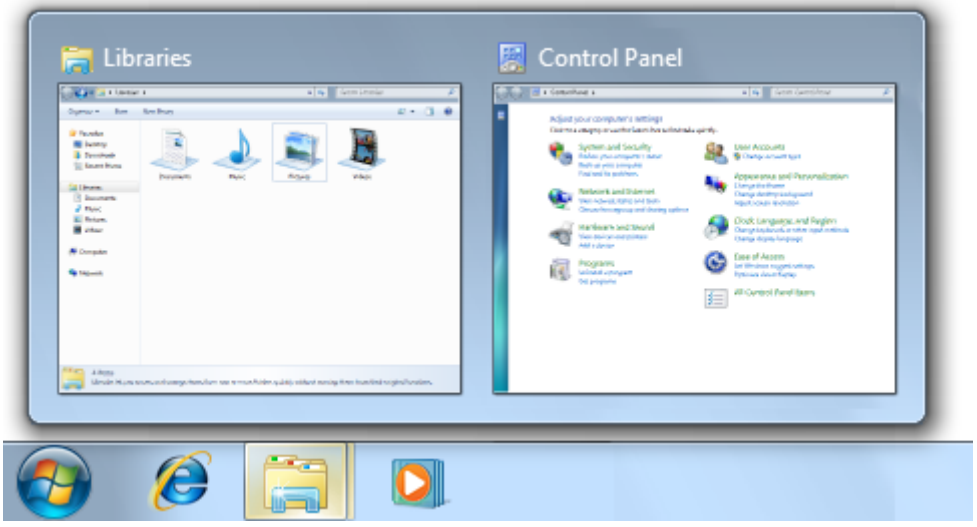
- **For Windows 7, use taskbar button thumbnails to group the following window types with the primary window taskbar button it was launched from.** Each program (specifically, each program perceived as a separate program) should have a single taskbar button.
  - Secondary windows
  - Workspace tabs
  - Project windows
  - MDI child windows

  Correct:



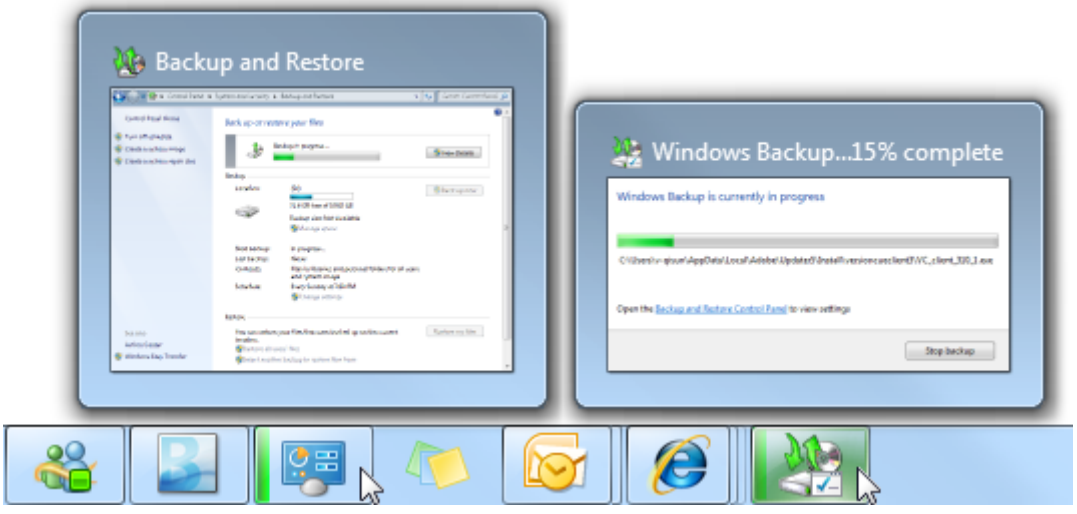
In this example, a secondary window is grouped with its primary window's taskbar button.

Incorrect:



In this example, Control Panel is incorrectly grouped with Windows Explorer. Users perceive these as separate programs.

Incorrect:

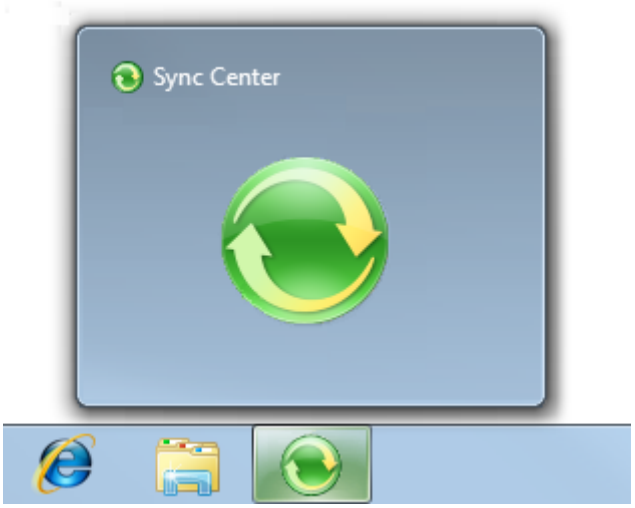


In this example, Windows Backup incorrectly uses two taskbar buttons for a single program.

- **Restoring a primary window should also restore all its secondary windows,** even if those secondary windows have their own taskbar buttons. When restoring, place secondary windows on top of the primary window.

- **For Windows 7, programs that normally have desktop presence may temporarily display a taskbar button to show status.** Do so only if your program is normally displayed on the desktop and users frequently interact with it. A program that normally runs without desktop presence should use its notification area icon instead, even though it might not always be visible.

  Incorrect:



  In this example, Windows Sync Center incorrectly uses a temporary taskbar button to display status. It should use its notification area icon instead.

Icons

- **Design your program icon to look great on the taskbar.** Ensure it is meaningful, and reflects its function and your brand. Make it distinct, make it special, and ensure it renders well in all icon sizes. Spend the time necessary to get it right. Follow the Aero-style icon guidelines.

- **If your program uses overlay icons, design your program's base icon to handle overlays well.** Overlay icons are displayed in the lower right corner, so design the icon so that area can be obscured.

In this example, the program's taskbar button icon doesn't have important information in the lower right area.

- **Don't use overlays in your program's base icon,** whether your program uses overlay icons or not. Using an overlay in the base icon will be confusing because users will have to figure out that it's not communicating status.

  Incorrect:

  In this example, the program's base icon looks like it is showing status.

For general icon guidelines and examples, see Icons.

Overlay icons

- **Use overlay icons to indicate useful and relevant status only.** Consider the display of an overlay icon to be a potential interruption of the user's work, so the status change must be important enough to merit a potential interruption.

  Incorrect:

  In these examples, the overlay icon isn't important enough to merit a potential interruption.

- **Use overlay icons for temporary status.** The overlay icons lose their value if displayed constantly, so normal program status should not show an icon. Remove the overlay icon when the icon:
  - **Is for a problem:** Remove the icon once the problem has been resolved.
  - **Alerts that something is new:** Remove the icon once the user has activated the program.

  Exception: Your program can constantly display an overlay icon if users always need to know its status.

  In this example, Windows Live Messenger always displays an overlay icon so that users can always check their reported presence.

- **Don't display an icon to indicate that a problem has been solved.** Instead, simply remove any previous icon indicating a problem. Assume that users normally expect your program to run without problems.
- **Display either overlay icons or notification area icons, but never both.** Your program may support both mechanisms for backward compatibility, but if your program displays status using overlay icons, it shouldn't also use notification area icons for status.

  Incorrect:

  In this example, the new mail icon is displayed redundantly.
- **Don't flash the taskbar button to draw attention to a status change.** Doing so would be too distracting. Let users discover overlay icons on their own.
- **Prefer standard overlay icons to indicate status or status changes.** Use these standard overlay icons:

  Expand table

Overlay	Status
	Warning
	Error
	Disabled/Disconnected
	Blocked/Offline

- **For custom overlay icons, choose an easily recognizable design.** Use high-quality 16x16 pixel, full color icons. Prefer icons with distinctive outlines over square or rectangular shaped icons. Apply the other Aero-style icon guidelines as well.
- **Keep the design of custom overlay icons simple.** Don't try to communicate complex, unfamiliar, or abstract ideas. If you can't think of a suitable custom icon, use a standard icon error or warning icon instead when appropriate. These icons can be used effectively to communicate many types of status.
- **Don't change status too frequently.** Overlay icons shouldn't appear noisy, unstable, or demand attention. The eye is sensitive to changes in the peripheral field of vision, so status changes need to be subtle.

- **Don't change the icon rapidly.** If underlying status is changing rapidly, have the icon reflect high-level status.

  Incorrect:

  In this example, the rapidly changing overlay icon demands attention.

- **Don't use animations.** Doing so is too distracting.

- **Don't flash the icon.** Doing so is too distracting. If an event requires immediate attention, use a dialog box instead. If the event otherwise needs attention, use a notification.

Taskbar button flashing

- **Use taskbar button flashing sparingly to demand the user's immediate attention to keep an ongoing task running.** It's hard for users to concentrate while a taskbar button is flashing, so assume that they will interrupt what they are doing to make it stop. While flashing a taskbar button is better than stealing input focus, flashing taskbar buttons are still very intrusive. Make sure the interruption is justified, such as to indicate that the user needs to save data before closing a window. Inactive programs should rarely require immediate action. Don't flash the taskbar button if the only thing the user has to do is activate the program, read a message, or see a change in status.
- **If immediate action isn't required, consider these alternatives:**
  - Use an action success notification to indicate that a task has completed.
  - Do nothing. Just wait for users to attend to the issue the next time they activate the program. This is often the best choice.
- **If an inactive program requires immediate attention, flash its taskbar button to draw attention and leave it highlighted.** Don't do anything else: don't restore or activate the window and don't play any sound effects. Instead, respect the user's window state selection and let the user activate the window when ready.
- **For secondary windows that have a taskbar button, flash its button instead of the primary window's taskbar button.** Doing so allows users to attend to the window directly.
- **For secondary windows that don't have a taskbar button, flash the primary window's taskbar button and bring the secondary window on top of all the other windows for that program.** Secondary windows that require attention must be topmost to ensure that users see them.

- **Flash only one taskbar button for one window at a time.** Flashing more than one button is unnecessary and too distracting.
- **Remove the taskbar button highlight once the program becomes active.**
- **When the program becomes active, make sure there is something obvious to do.** Typically, this objective is accomplished by displaying a dialog box that asks a question or initiates an action.

Quick Launch shortcuts

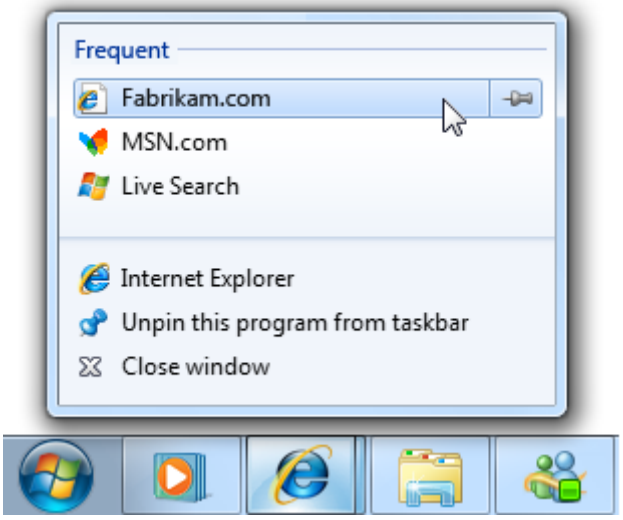
- **Put program shortcuts in the Quick Launch area only if users opt in.** Because Quick Launch was removed from Windows 7, programs designed for Windows 7 shouldn't add program shortcuts to the Quick Launch area or provide options to do so.

Jump Lists

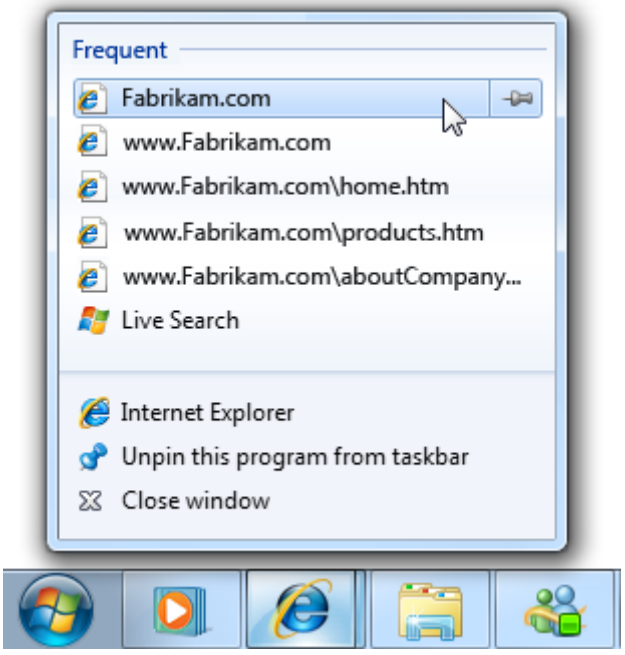
Design

- **Design Jump Lists to satisfy your users' goals for their everyday tasks.** Consider:
    - **Your program's purpose.** Think about what users are most likely to do next. For document creation programs, users are likely to return to recently used documents. For programs that show existing content, users may want access to resources they use frequently. For other programs, users might be likely to do tasks they haven't done before, such as read new messages, watch new videos, or check their next meeting.
    - **What users care about most.** Think about why users would use the Jump List instead of other means. For example, users are more likely to care about destinations they explicitly identified as important (such as Web addresses users placed on their links bar or in Favorites, or typed in). They are less likely to care about those obtained indirectly or with little effort (such as Web addresses visited through redirection or by clicking links).

      Correct:



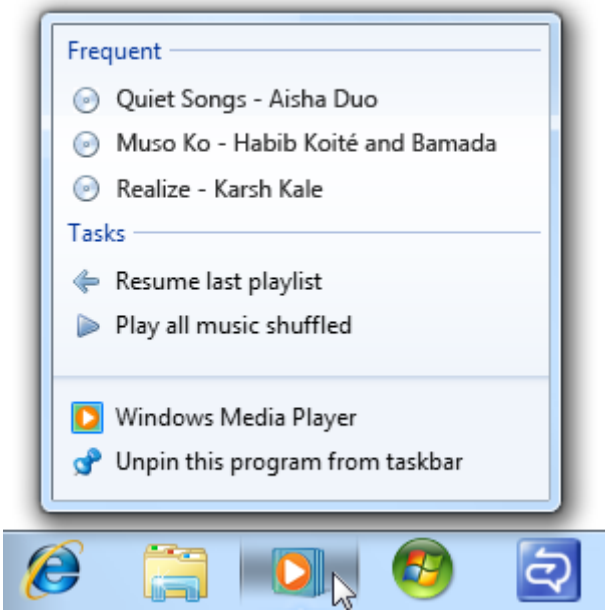
Incorrect:



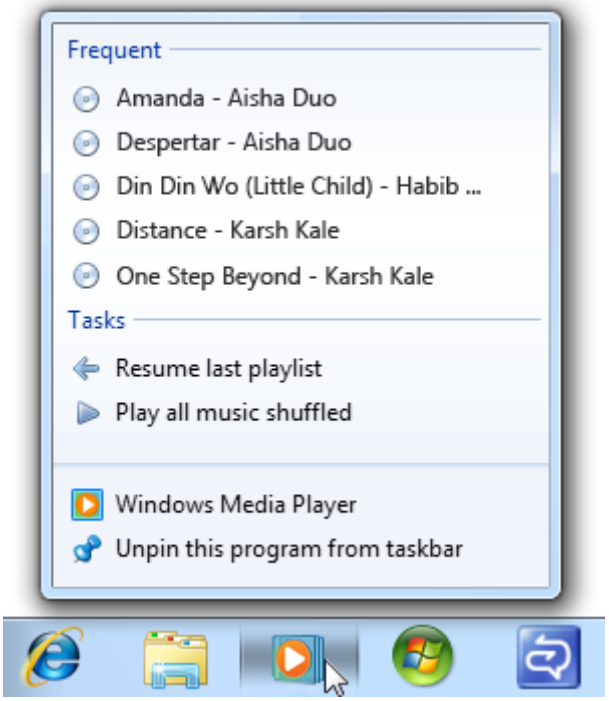
In the incorrect example, the Jump List contains many destinations that users aren't likely to care about.

- **Don't make destinations too granular.** Making destinations too narrow and specific can result in redundancy, with several ways to go to the same place. For example, instead of listing individual Web pages, list top-level home pages instead; instead of listing songs, list albums.

  Correct:

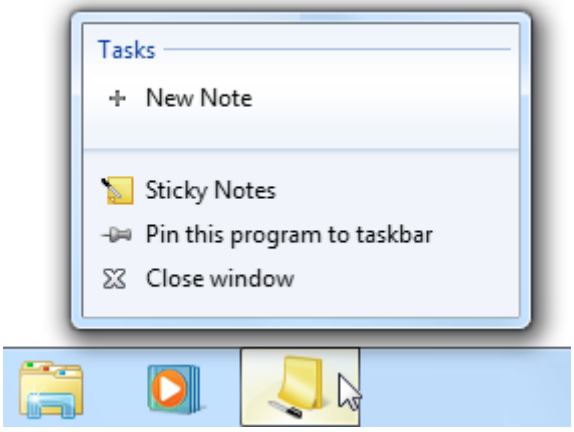


Incorrect:



In the incorrect example, listing songs in a Jump List will fill it with a single album.

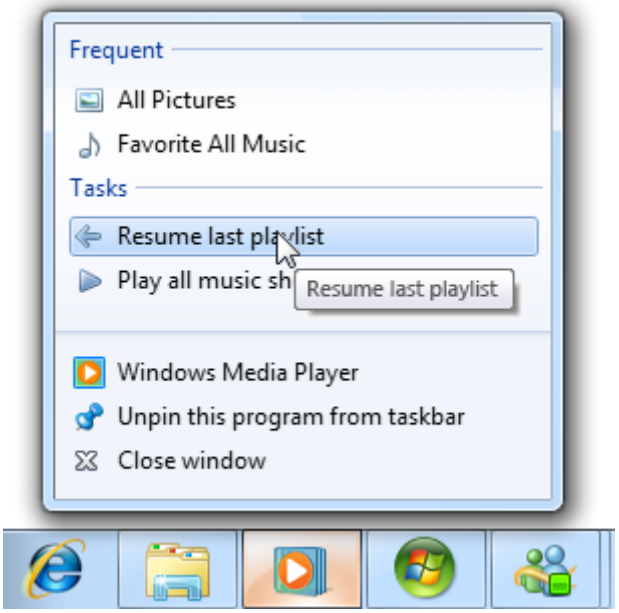
- **Don't fill all the available Jump List slots if you don't need to.** Focus Jump List content on the most useful items if your program has only three useful items, provide only three. The more items in a Jump List, the more effort required to find any specific item.



In this example, the Sticky Notes accessory provides a single Jump List command, because that's all that is needed.

- **Provide tooltips only when needed to help users understand Jump List items.** Avoid redundant tooltips because they are an unnecessary distraction. For more tooltip guidelines, see Tooltips and Infotips.

  Incorrect:



  In this example, the Jump List tooltip is redundant.

Jump List features vs. program features

- **Don't make destinations and commands available only through Jump Lists.** The same destinations and commands should be available directly from the program itself.
- **Use consistent names for destinations and labels for commands.** Jump List items should be labeled the same as the equivalent items accessed directly from the program.

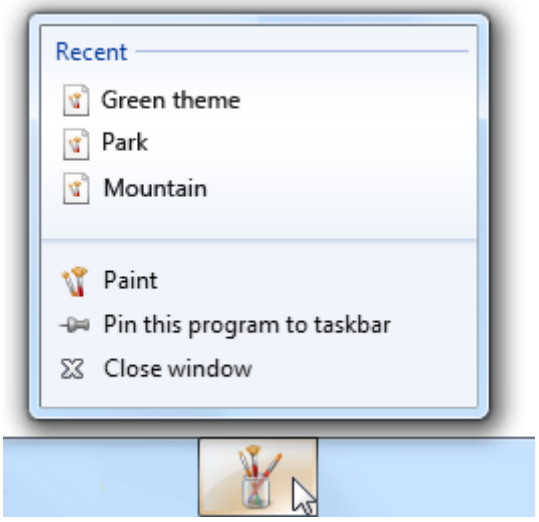
- **Enable your program to handle destinations and commands even when the program isn't running.** Doing so is necessary for a consistent, dependable, and convenient experience.

Grouping

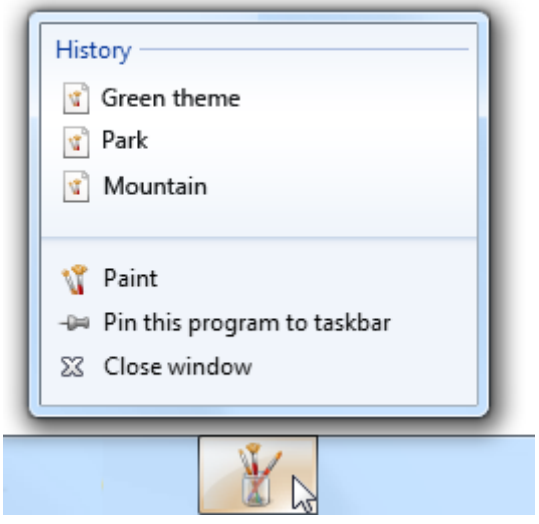
- **Provide at least one and at most three groups.** Jump List items are always grouped to label their purpose. Having more than three groups makes items harder to find.
- **Use standard group names when appropriate.** Standard group names are familiar and easier for users to understand.

  Commands are given the Tasks group name, which is assigned by Windows and therefore can't be changed.

  Correct:



  Incorrect:



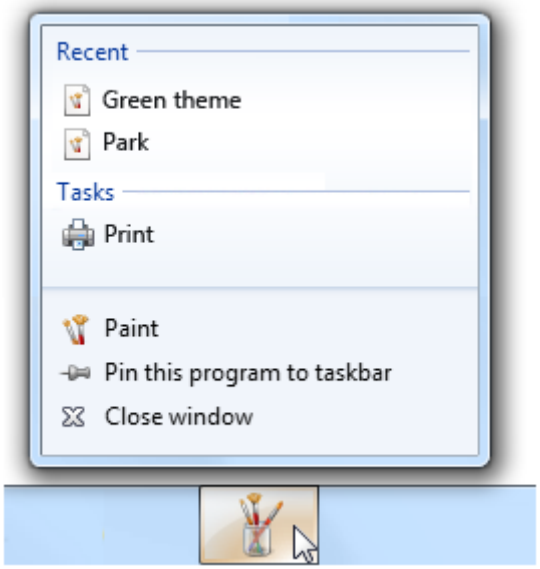
Recent is the better group name because it is familiar, and the subtle distinction between history and recent isn't worth making.

Commands

- **Provide a fixed set of commands regardless of program running state, current document, or current user.** The commands should apply to the entire program, not to a specific window or document. Doing so is necessary for a consistent, dependable, and convenient experience. Commands shouldn't be removed or disabled.

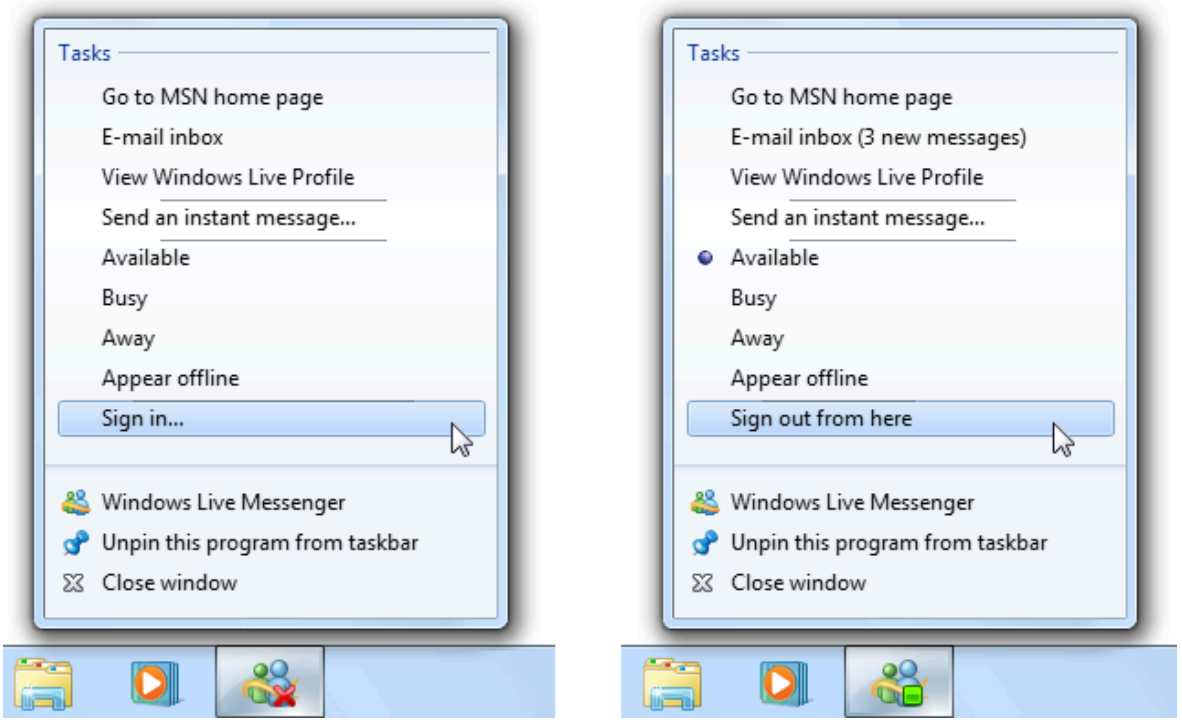
  Exceptions: You may substitute or remove commands when:
  - A set of mutually exclusive commands share a single command slot, as long as one command always applies.
  - Commands don't apply until specific features have been used, as long as the commands otherwise always apply.

  Incorrect:



  In this example, Print isn't a good Jump List command because it depends on the current document.

  Correct:



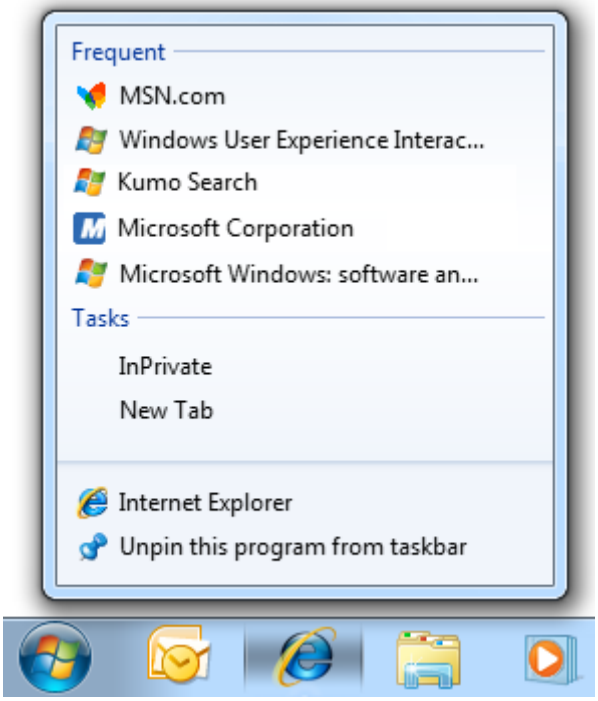
In this example, Sign in and Sign out are mutually exclusive commands. Also, separators are used to group related commands.

- **Use the following standard command labels when appropriate.** Standard command labels are easier for users to understand.
- **Present the commands in a logical order.** Common orders include by frequency of use or order of use. Place highly related commands next to each other. Within the Tasks group, put separators between groups of related commands as needed.
- **Don't provide commands for opening or closing the program.** These commands are built into all Jump Lists.

Command icons

- **Within the Tasks group, provide a command icon only when it helps users understand, recognize, or differentiate commands,** especially when there is an established icon for the command used within the program.
  - **Exception:** If your program uses both destinations (which always have icons) and commands, consider providing icons for all commands if not doing so would look awkward.

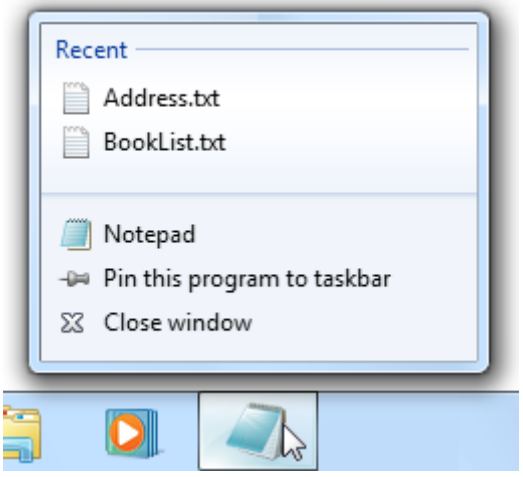
Incorrect:



In this example, Internet Explorer should provide icons for all commands to avoid an awkward appearance.

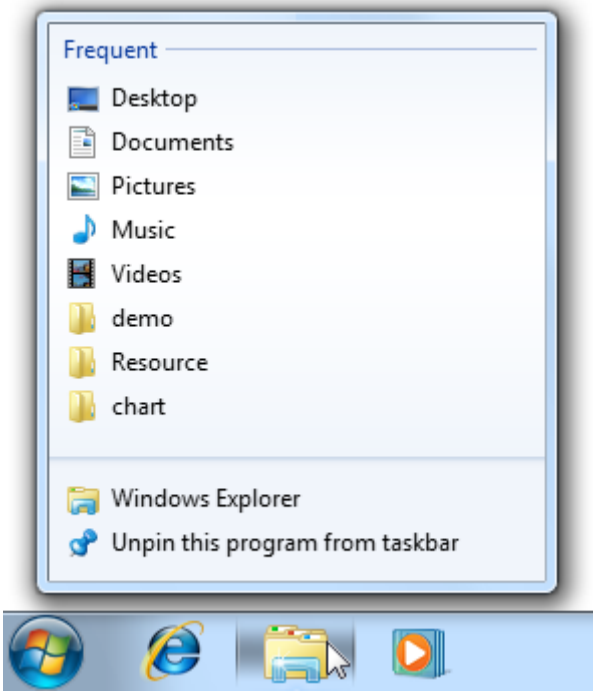
Destinations

- **Provide a dynamic set of destinations that are specific to the current user, but independent of the program running state or current document.** As mentioned previously, make sure they fit your program's purpose, are what users care about the most, and have the right level of specificity.
- **When suitable, use an "automatic" destination list.** Automatic destinations are managed by Windows, but your program controls the specific destinations that are passed on.
    - Consider using Recent for document creation programs where users are likely to return to recently used destinations.



In this example, Windows Notepad uses Recent destinations.

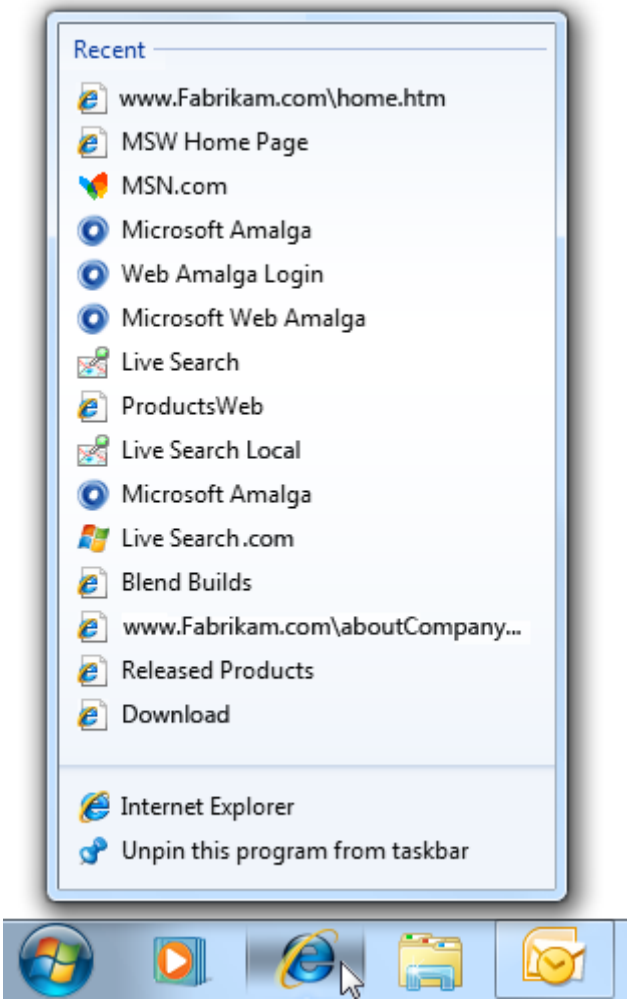
- Consider using Frequent for programs that show existing content, where users are likely to return to items that they use often. Frequent destinations are sorted in order of frequency, most frequent first.



In this example, Windows Explorer uses Frequent destinations.

- Use Frequent if Recent would result in many useless destinations. Frequent lists are more stable, and the better choice when users go to many different destinations, but aren't likely to return to rarely used ones.

Incorrect:



Using Recent in Windows Internet Explorer would result in many useless destinations.

  - If Recent or Frequent are equally suitable choices, use Recent because that approach is easier for users to understand and is more predictable.

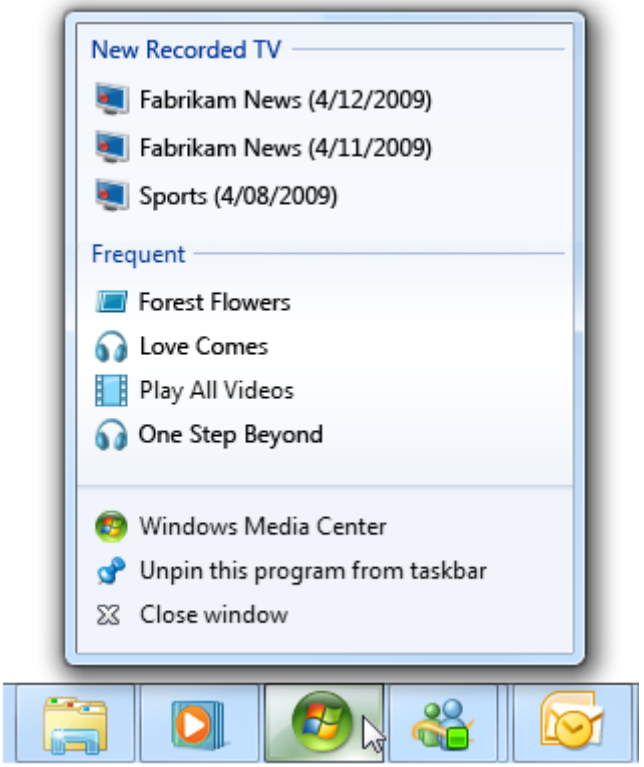
  - If using Recent, and the program has an equivalent in the File menu, make the lists have the same contents in the same order. To users, these should appear to be the same lists.

- **When necessary, use a custom destination list.** Your program has complete control over a custom destination list's contents and sort order, and therefore can base the list on any factors.

  - Create custom versions of Recent or Frequent if those are suitable, but the automatic management doesn't work well for your program. For example, your program may need to track a variety of factors beyond open file commands. In this case, use the same name (Recent or Frequent) and sort order because users won't be aware of the difference.

  - Otherwise, use a different type of destination to better satisfy your user's goals. Often, these lists help users perform tasks that they haven't done before, such

as read new messages, watch new videos, or check their next meeting.

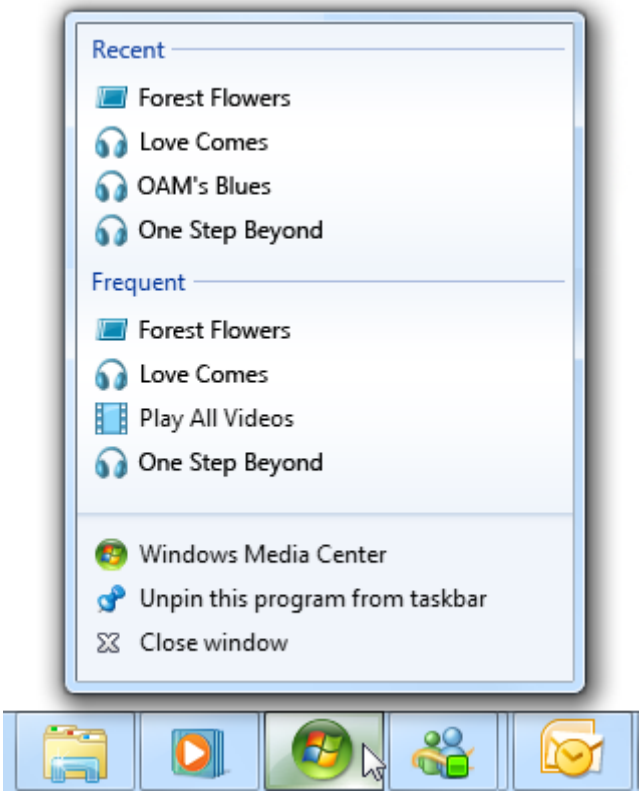


In this example, Windows Media Center lists the recently recorded shows that the user hasn't seen yet.

  - Choose a sort order that corresponds to the user's mental model of the list. For example, a to-do style list would have the next thing to do listed first. If there is no clear mental model, sort the destination list in alphabetical order.

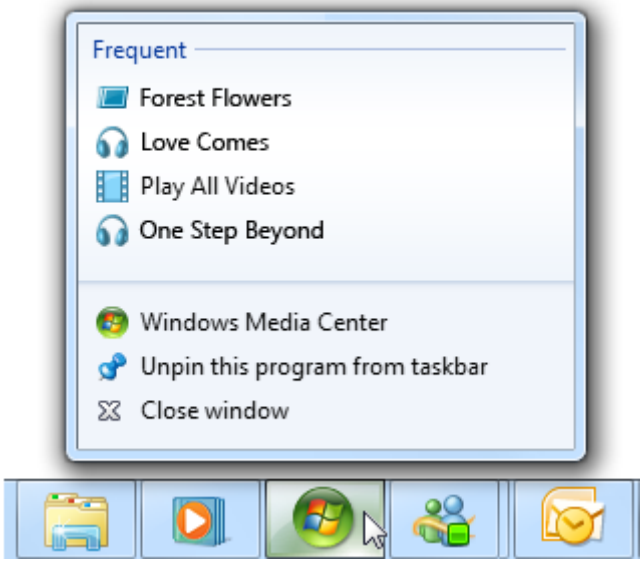
- **Don't use multiple destination lists that give different views of the same data.** Rather, multiple destination lists should have mostly different data to support difference scenarios. For example, you can provide a Recent list or a Frequent list, but not both. Doing so is wasteful if overlapping items are present, but confusing if overlapping items are removed.

  Incorrect:



In this example, providing different views of the same destinations is wasteful.

Correct:



In this example, the destination lists have different data for different tasks.

- **If your program has a command to clear data for privacy, clear the Destinations lists as well.** Destination lists may contain sensitive data.

Thumbnail toolbars

Interaction

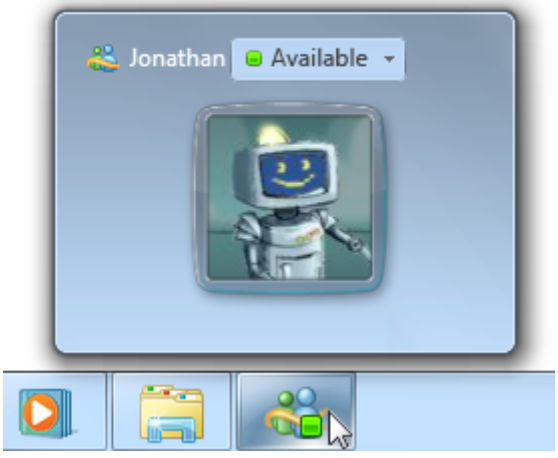
- **Provide up to seven of the most important, frequently used commands that apply to the window shown in the thumbnail.** Don't feel obligated to provide as many commands as you can if your program has only three important, frequently used commands, provide only three.

  Incorrect:

  In this example, the thumbnail toolbar has commands that aren't important.

- **Use commands that are direct and immediate.** These commands should have an immediate effect clicking the command should not display a drop-down menu or dialog box for more input.

  Incorrect:

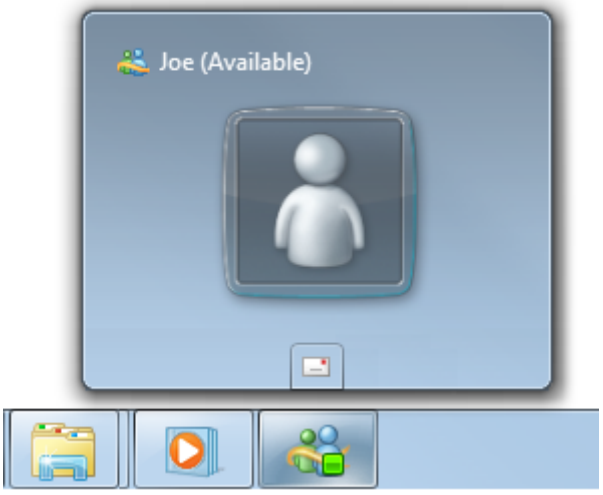
  

  Thumbnail toolbar commands must have an immediate effect.

- **Disable commands that don't apply to the current context, or that would directly result in an error.** Don't hide such commands because doing so makes the toolbar presentation unstable.

- **Don't dismiss the thumbnail when users click a command if they are likely to review the results or immediately click another command.** Remove the thumbnail for commands that indicate that the user is finished for now, such as with commands that display other windows.

In this example, clicking Next in Windows Media Player continues to display the thumbnail because users might want to give other commands.



In this example, clicking Chat in Windows Live Messenger dismisses the thumbnail because users are most likely to send a message.

Presentation

- **Make sure thumbnail toolbar icons conform to the Aero-style icon guidelines.** For each command, provide high-quality 16x16, 20x20, and 24x24 pixel, full color icons. The larger versions are used in high-dpi display modes.
- **Make sure the icons are clearly visible against the toolbar background color in both normal and hover states.** Always evaluate icons in context and in the high-contrast modes.

- **Choose command icon designs that clearly communicate their effect.** Well-designed command icons are self-explanatory to help users find and understand commands efficiently.
- **Choose icons that are recognizable and distinguishable.** Make sure the icons have distinctive shapes and colors. Doing so helps users find the commands quickly, even if they don't remember the icon symbol. After initial use, users shouldn't have to rely on tooltips to distinguish between the commands.
- **Provide a tooltip to label each command.** A good tooltip labels the unlabeled control being pointed to. For guidelines and examples, see Tooltips and Infotips.

Progress bars

- **Follow the general progress bar guidelines,** including not restarting or backing up progress, and using a red progress bar to indicate a problem.
- **Avoid using indeterminate progress bars.** Indeterminate progress bars show activity, not progress. Reserve indeterminate progress bars for those rare situations where users don't take activity for granted.

For more guidelines, see Progress Bars.

Text

Window titles

When choosing window titles, consider the title's appearance on the taskbar:

- Optimize titles for display on the taskbar by concisely placing the distinguishing information first.
- For modeless progress dialog boxes, first summarize the progress. Example: "66% Complete."
- Avoid window titles that have awkward truncations.

  Incorrect:

  In this example, the truncated window title has unfortunate results.

Jump List commands

- **Start commands with a verb.**
- **Use sentence-style capitalization.**

For more command label guidelines, see Menus.

Documentation

When referring to the taskbar:

- Refer to the entire bar as the taskbar (a single compound word in lowercase).
- Refer to items on the taskbar specifically by their label, or generally as taskbar buttons.
- When possible, format the taskbar labels using bold text. Otherwise, put the label in quotation marks only if required to prevent confusion.
- Refer to overlay icons as taskbar button icons. Don't refer to them as notifications, even if their purpose is to notify users. However, you can say that these icons notify users of specific events.

Example: The New Mail taskbar button icon notifies you that a new e-mail message has arrived.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Notification Area

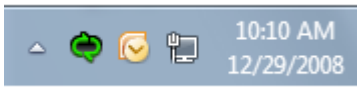
Article • 08/26/2021

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

The notification area provides notifications and status. Well-designed programs use the notification area appropriately, without being annoying or distracting.

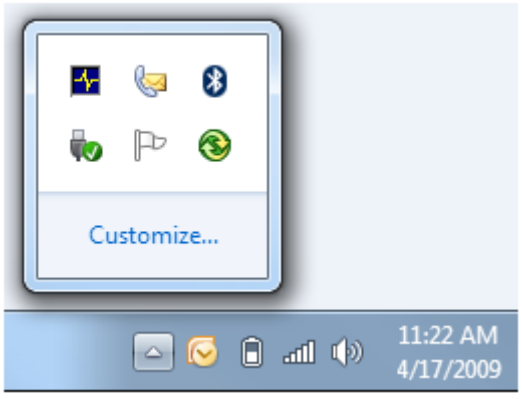
The notification area is a portion of the taskbar that provides a temporary source for notifications and status. It can also be used to display icons for system and program features that have no presence on the desktop.

Items in the notification area are referred to as notification area icons, or simply icons if the context of the notification area is already clearly established.



The notification area.

To give users control of their desktop in Windows 7, not all notification area icons are displayed by default. Rather, icons are displayed in the notification area overflow unless promoted to the notification area by the user.



The notification area overflow.

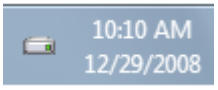
Note: Guidelines related to the taskbar, notifications , and balloons are presented in separate articles.

Is this the right user interface?

To decide, consider these questions:

- **Does your program need to display a notification?** If so, you must use a notification area icon.
- **Is the icon displayed temporarily to show a change of status?** If so, a notification area icon may be appropriate, depending upon the following factors:
  - **Is the status useful and relevant?** That is, are users likely to monitor the icon and change their behavior as a result of this information? If not, either don't display the status, or put it in a log file.

    Incorrect:

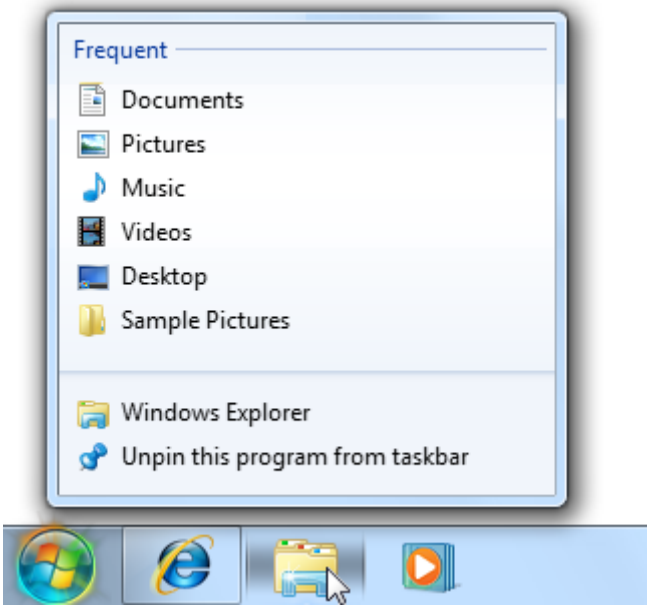
    

    In this example, the disk drive activity icon is inappropriate because users are unlikely to change their behavior based on it.
  - **Is the status critical? Is immediate action required?** If so, display the information in a way that demands attention and cannot be easily ignored, such as a dialog box.

  Programs designed for Windows 7 can use overlay icons on the program's taskbar button to show change of status, as well as taskbar button progress bars to show progress of long-running tasks.
- **Does the feature already have "desktop presence"?** That is, when run, does the feature appear in a window on the desktop (possibly minimized)? If so, display status in the program's status bar, other status area, or, for Windows 7, directly on the taskbar button. If the feature doesn't have desktop presence, you can use an icon for program access and to show status.
- **Is the icon primarily to launch a program or access its features or settings quickly?** If so, use the Start menu to launch programs instead. The notification area isn't intended for quick program or command access.

  In this example from Windows Vista, Quick Launch is used to launch Windows Explorer and Windows Internet Explorer quickly.

For programs designed for Windows 7, users can pin taskbar buttons for quick program access. Programs can use a Jump List or thumbnail toolbar to access frequently used commands directly from a program's toolbar button. The Quick Launch area isn't displayed by default in Windows 7.



In this example, a Jump List is used for quick command access.

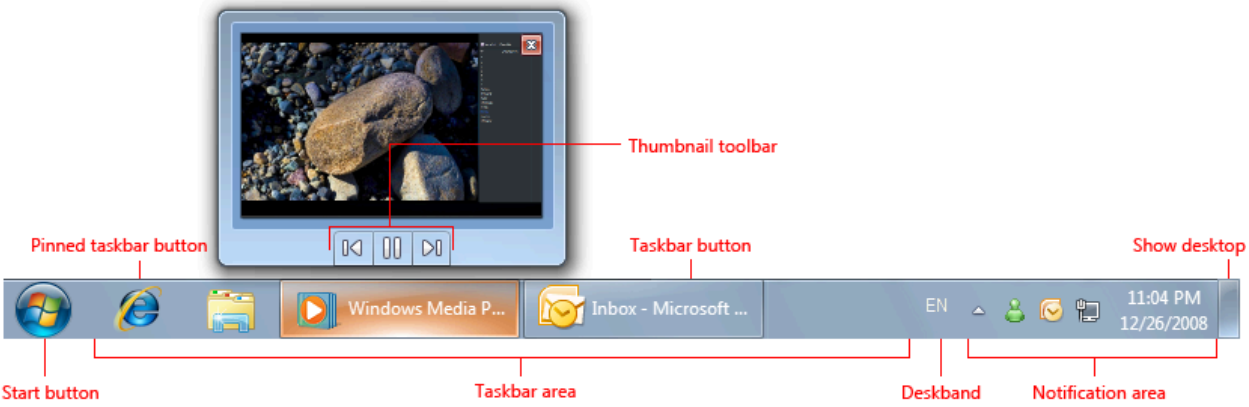
Design concepts

The Windows desktop

The Windows desktop has the following program access points:

- **Work area.** The onscreen area where users can perform their work, as well as store programs, documents, and their shortcuts. While technically the desktop includes the taskbar, in most contexts it refers just to the work area.
- **Start button.** The access point for all programs and special Windows places (Documents, Pictures, Music, Games, Computer, Control Panel), with "most recently used" lists for quick access to recently used programs and documents.
- **Quick Launch.** A direct access point for programs selected by the user. Quick Launch was removed from Windows 7.
- **Taskbar.** The access point for running programs that have desktop presence. While technically the taskbar spans the entire bar from the Start button to the notification area, in most contexts taskbar refers to the area in between, containing the taskbar buttons. This area is sometimes referred to as the taskband.
- **Deskbands. Not recommended.**

- **Notification area.** A short-term source for notifications and status, as well as an access point for system- and program-related features that have no presence on the desktop.



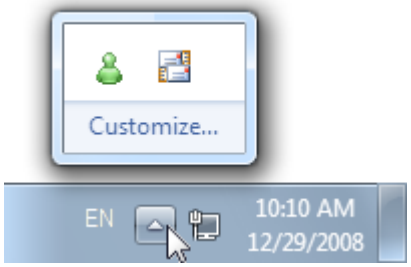
The Windows desktop access points include the Start button, taskbar, and notification area. Note the thumbnail feature of the taskbar button.

The desktop is a limited, shared resource that is the user's entry point to Windows. Leave users in control. You should use the desktop areas as intended any other usage should be considered an abuse. For example, never view desktop areas as ways to promote your program or its Brand.

Using the notification area appropriately

The notification area was originally intended as a temporary source for notifications and status. Its efficiency and convenience has encouraged developers to give it other purposes, such as launching programs and executing commands. Unfortunately over time, these additions made the notification area too large and noisy, and confused its purpose with the other desktop access points.

Windows XP addressed the scale problem by making the area collapsible and hiding the unused icons. Windows Vista addressed the noise by removing unnecessary, annoying notifications. Windows 7 has gone a step further by focusing the notification on its original purpose of being a notification source. **Most icons are hidden by default in Windows 7, but can be promoted to the notification area manually, by the user. To keep users in control of their desktops, there is no way for your program to perform this promotion automatically.** Windows still displays notifications for hidden icons by promoting them temporarily.



In Windows 7, most notification area icons are hidden by default.

In addition, Windows 7 supports many features directly in the taskbar buttons. Specifically, you can use:

- Jump Lists and thumbnail toolbars to quickly access frequently used commands.
- Overlay icons to show status for running programs.
- Taskbar button progress bars to show progress for long-running tasks.

In short, if your program has desktop presence, take full advantage of the Windows 7 taskbar button features for these purposes. **Keep the notification area icons focused on displaying notifications and status.**

Keeping users in control

Keeping users in control extends beyond using the notification area correctly. Depending on the nature of your icon, you may want to let users do the following:

- **Remove the icon.** Your icon may provide relevant, useful status, but even so, users may not want to see it. Windows allows users to hide icons, but this feature isn't easily discoverable. To keep users in control, provide a **Display icon in notification area** option on the icon's context menu. Note that removing an icon doesn't have to affect the underlying program, feature, or process.
- **Select types of notifications to display.** Your notification must be useful and relevant, but there may be notifications that users don't want to see. This is especially true for FYI notifications. Let users choose to enable the less important ones.
- **Suspend optional features.** Icons are used to display status for features without desktop presence. Such features tend to be long-running, optional background tasks, such as printing, indexing, scanning, or synchronizing. Users may want to suspend such features to increase system performance, reduce power consumption, or because they are offline.
- **Quit the program.** Provide the more suitable of these options:
  - **Temporarily quit program.** The program is stopped and restarted when Windows is restarted. This approach is suitable for important system utilities

such as security programs.
    - **Permanently quit program.** The program is stopped and not restarted when Windows is restarted (unless the user chooses to restart later). Either the user no longer wants to run the program, or wants to run the program on demand perhaps to improve system performance.

Although it's a good idea to provide most of these settings on the icon's context menu, the program's default experience should be suitable for most users. Don't turn everything on by default and expect users to turn features off. Rather, turn the important features on by default, and let users enable additional features as desired.

If you do only four things...

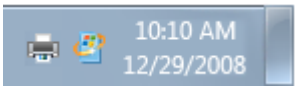
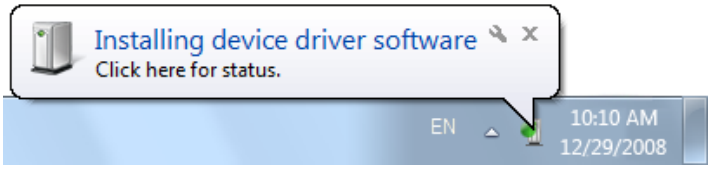
1. Don't abuse the notification area. Use it only as a source for notifications and status, and for features without desktop presence.
2. Keep users in control. Provide appropriate options to control the icon, its notifications, and the underlying features.
3. Present a default experience that is suitable for most users. Let users enable desired features rather than expect them to disable undesired ones.
4. Take full advantage of the Windows 7 taskbar button features to show status and make your program's most frequently performed tasks efficient.

Usage patterns

Notification area icons have several usage patterns:

Expand table

Label	Value
System status and access Displayed continuously to show important but not critical system status, and to provide access to relevant features and settings.	System features that need notification area icons have no persistent desktop presence. Can also be used as a notification source. In this example, the battery, network, and volume icons are displayed continuously when applicable.
Background task status and access Displayed while a background task is running to show status and provide access to features and settings.	Background processes need notification area icons when they have no desktop presence. Can also be used as a notification source.

Label	Value
	In this example, the Action Center icon allows users to check its status even when it has no desktop presence.
Temporary event status Programs with desktop presence can display icons temporarily to show important events or changes in status.	 In this example, icons for printing and installing updates are displayed temporarily to show important events or changes in status.
Temporary notification source Displayed temporarily to show a notification. Removed after a timeout, or when the underlying problem is addressed or task performed.	Temporary icons are preferred for pure notification sources. Don't display an icon that doesn't provide useful, relevant, dynamic status just because a feature might need to display a notification in the future.  In this example, the plug-and-play icon is displayed while a new hardware detected notification is shown.
Minimized single-instance application To reduce taskbar clutter, a single-instance, long-running application can be minimized to a notification area icon instead.	 In this example from Windows Vista, Outlook and Windows Live Messenger are single-instance applications that minimize to notification area icons. Consider using this pattern only if all of the following apply: • The application can have only a single instance. • The application is run for an extended period of time. • The icon shows status. • The icon can be a notification source. • Doing so is optional and users must opt in. If all these conditions apply, minimizing to an icon eliminates having two access points when only one is necessary. **Note:** This icon pattern is no longer recommended for Windows 7. Use regular taskbar buttons instead if your program has desktop presence. In this example from Windows 7, a regular taskbar button takes little space, but benefits from the Windows 7 taskbar button features, including Jump Lists, overlay icons, and rich thumbnails.

Guidelines

General

- **Provide only one notification area icon per component.**
- **Use an icon with 16x16, 20x20, and 24x24 pixel versions.** The larger versions are used in high-dpi display modes.

When to show

- For the temporary notification source pattern:
  - Windows displays the icon when the notification is displayed.
  - Remove the icon based on its notification design pattern:

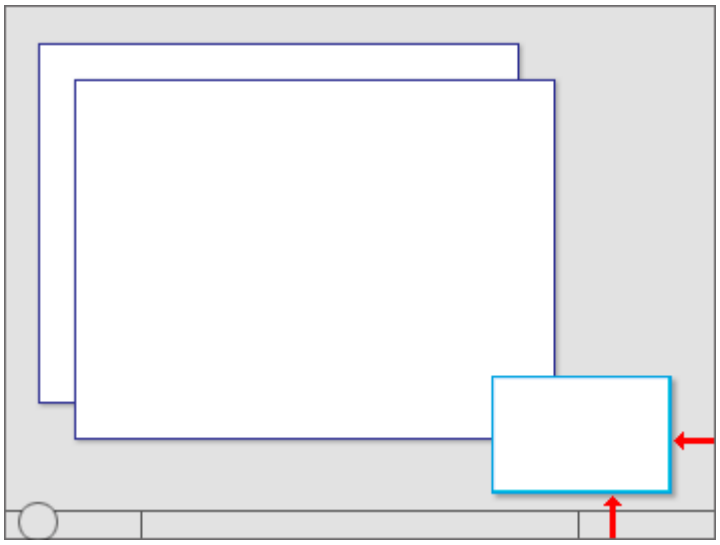
Expand table

Pattern	When to remove
Action success	When notification is removed.
Action failure	When problem is resolved.
Non-critical system event	When problem is resolved.
Optional user task	When task is done.
FYI	When notification is removed.

- **For the temporary event status pattern, display the icon while the event is happening.**
- For all other patterns, **display the icon when the program, feature, or process is running and the icon is relevant** unless the user has cleared its **Display icon in notification area** option (for more information, see Context menus). Most icons are hidden by default in Windows 7, but can be promoted to the notification area by the user.
- **Don't display icons meant for administrators to standard users.** Record the information in the Windows event log.

Where to show

- **Display windows launched from notification area icons near the notification area.**



Windows launched from notification area icons are displayed near the notification area.

Icons

- **Choose the icon based on its design pattern:**

Expand table

Pattern	Icon type
System status and access	System feature icon
Background task status and access	Program or feature icon
Temporary notification source	Program or feature icon
Temporary event status	Program or feature icon
Minimized single-instance application	Program icon

In this example, Outlook uses an e-mail feature icon for a temporary notification source and its application icon for the minimized application.

- **Choose an easily recognizable icon design.** Prefer icons with unique outlines over square or rectangular shaped icons. Keep the designs simple prefer symbols over realistic images. Apply the other Aero-style icon guidelines as well.

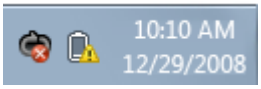
- **Use icon variations or overlays to indicate status or status changes.** Use icon variations to show changes in quantities or strengths. For other types of status, use

the following standard overlays. Use only a single overlay, and locate it bottom-right for consistency.

Expand table

Overlay	Status
	Warning
	Error
	Disabled/Disconnected
	Blocked/Offline

In this example, the wireless and battery icons show changes in quantities or strengths.



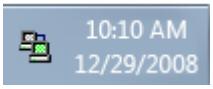
In this example, overlays are used to show error and warning states.

- **Avoid swaths of pure red, yellow, and green in your base icons.** To avoid confusion, reserve these colors to communicate status. If your branding uses these colors, consider using muted tones for your base notification area icons.
- For progressive escalation, **use icons with a progressively more emphatic appearance as the situation becomes more urgent.**

In these examples, the appearance of the battery icon becomes more emphatic as the urgency increases.

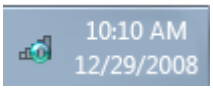
- **Don't change status too frequently.** Notification area icons shouldn't appear noisy, unstable, or demand attention. The eye is sensitive to changes in the peripheral field of vision, so status changes need to be subtle.
  - **Don't change the icon rapidly.** If underlying status is changing rapidly, have the icon reflect high-level status.

    Incorrect:



In this example, the modem icon displays blinking lights (as a hardware modem does), but those state changes aren't significant to users.

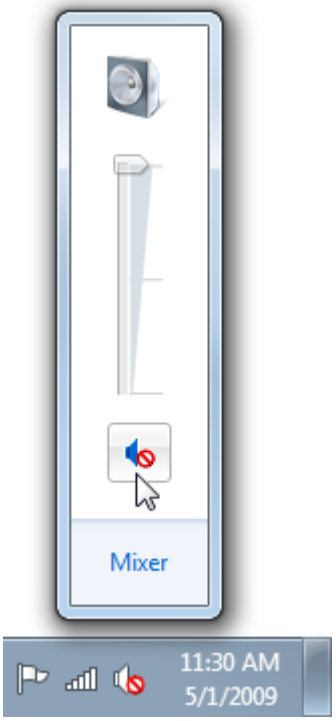
- **Don't use long-running animations to show continuous activities.** Such animations are a distraction. An icon's presence in the notification area sufficiently indicates continuous activity.
- **Brief, subtle animations are acceptable to show progress during important temporary, transitive status changes.**



In this example, the Wireless icon displays an activity indicator to show that work is in progress.

- **Don't flash the icon.** Doing so is too distracting. If an event requires immediate attention, use a dialog box instead. If the event otherwise needs attention, use a notification.

- **Don't disable notification area icons.** If the icon doesn't currently apply, remove it. However, you can show an enabled icon with a disabled status overlay if users can enable from the icon.



In this example, users can enable sound output from the icon.

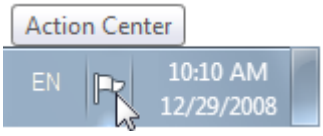
For general icon guidelines and examples, see Icons.

Interaction

Note: The following click events should occur on mouse up, not mouse down.

Hover

- **Display a tooltip or infotip that indicates what the icon represents.**

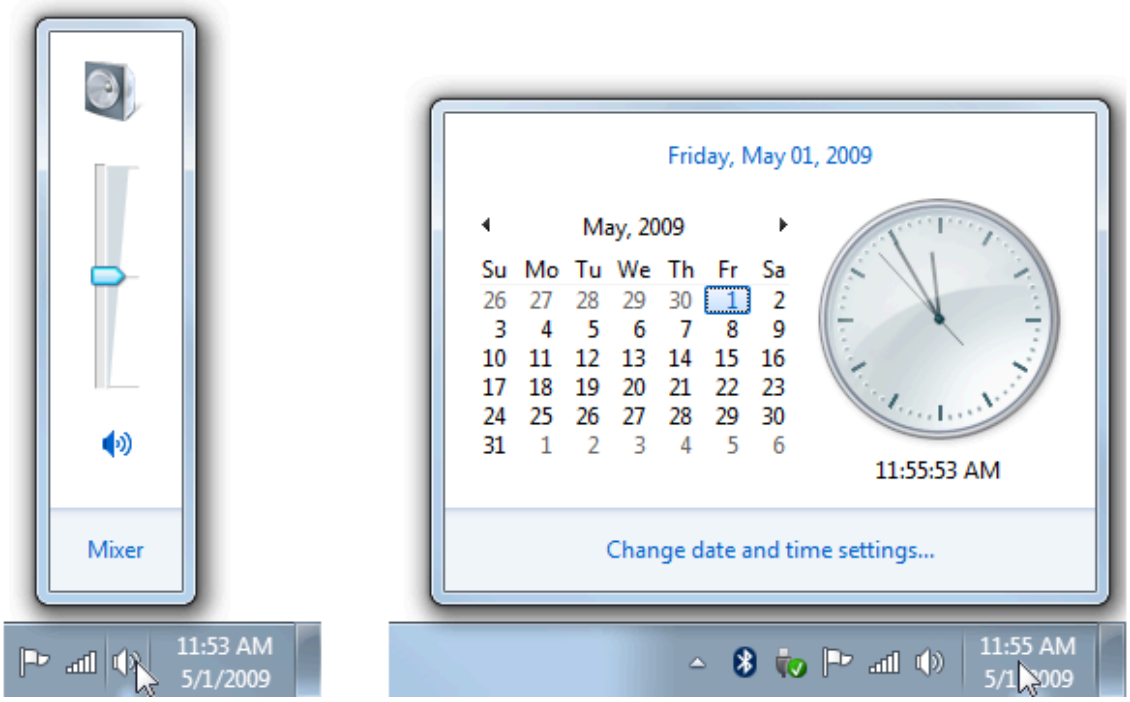


In this example, a tooltip is used to describe the icon on hover.

For infotip text guidelines, see the Text section of this article.

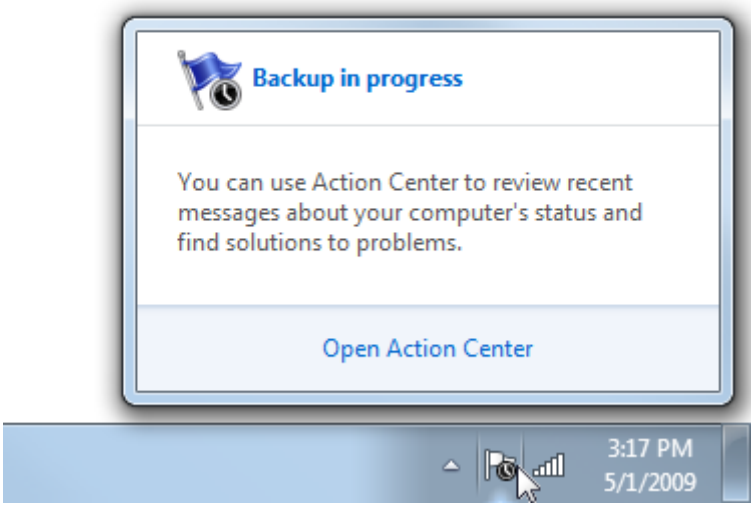
Left single-click

- **Display whatever users most likely want to see**, which may be:
  - A flyout window, dialog box, or program window with the most useful settings and commonly performed tasks. For presentation guidelines, see Notification area flyouts.



In these examples, left clicking displays popup windows with the most useful settings.

    - A status flyout.



    In this example, left clicking displays the status flyout.
    - The related control panel item.
    - The context menu.

    Users expect left single-clicks to display something, so not displaying anything makes a notification area icon appear unresponsive.

- **Display a context menu only if the other choices don't apply**, with the default command in bold. In this case, display the same context menu that is shown on right-click to avoid confusion.

- **Prefer using a popup window over a dialog box** for a more lightweight feel. Show only the most common settings and have them take immediate effect for a simpler interaction. Dismiss the popup window if the user clicks anywhere outside the window.

- **Display small windows near the associated icon.** However, large windows such as control panel items can be displayed in the center of the default monitor.

Left double-click

- **Perform the default command on the context menu.** Typically, this displays the primary UI associated with the icon, such as the associated control panel item, property sheet, or program window.
- **If there is no default command, perform the same action as a left single-click.**

Right-click

- **Display the context menu**, with the default command in bold.

Context menus

- **Display the context menu near its associated icon**, but away from the taskbar.
- **The context menu may include the following items**, as appropriate, in the listed order (exact text is in quotes):

Primary commands

Open (default, list first, in bold)

Run

Secondary commands

< separator>

Suspend/resume enable/disable command (check mark)

"Minimized to notification area" (check mark)

Opt in to notifications (check mark)

"Display icon in notification area" (check mark)

< separator>

"Options"

"Exit"

- **Remove rather than disable any context menu items that don't apply.**
- For Open, Run, and Suspend/Resume commands, **be specific about what is being opened, run, suspended, and resumed.**

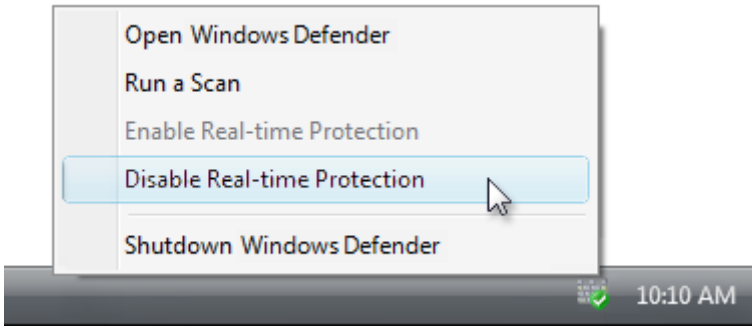
In this example, Windows Defender has specific Open and Run commands.

- **Use Suspend/Resume running background tasks, Enable/Disable for everything else.**

- **Use check marks to indicate state.** List and enable all states and place the check mark next to the current state. Don't disable options or change option labels to indicate the current state.

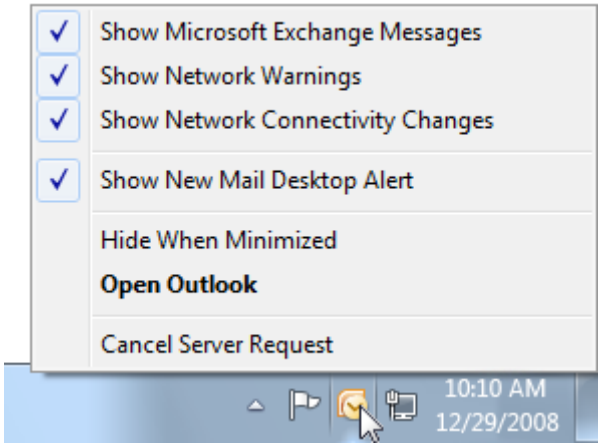
Correct:

Incorrect:



In the incorrect example, Windows Defender should use a check mark to indicate the current state.

- **All background tasks must have a Suspend/Resume command.** Choosing the command should temporarily suspend the task. Users may want to temporarily suspend background tasks to increase system performance or reduce power consumption. Suspended background tasks are restarted when resumed by the user or when Windows is restarted.
- **Allow users to opt into or out of different notification types** if your program has notifications that some users might not want to see. The FYI notification pattern requires users to opt in, so these notifications must be disabled by default.



In this example, Outlook allows users to choose the notifications they receive from the icon.

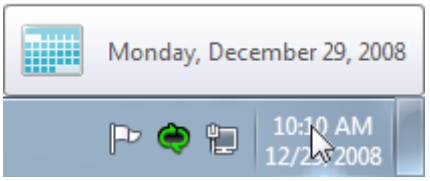
- **Clearing the "Display icon in notification area" option removes the icon from the notification area**, but doesn't affect the underlying program, feature, or process. Users can redisplay the icon from the program's Options dialog box. Don't automatically re-display the icon when Windows is restarted.
- **The Exit command quits the program for the current Windows session and removes the icon.** Don't have an Exit command if program can't be shut down. The program is restarted when Windows is restarted. Users can permanently quit the program from the Options dialog box.
- **Don't have an About command.** Such information should be communicated by the icon, its infotip, and the context menu. If users want more information, they can view the primary UI.
  - **Exception:** You may provide an About command if the icon is for a program that doesn't have desktop presence.

For general context menu guidelines and examples, see Menus.

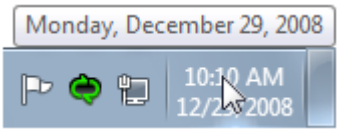
Rich tooltips

- **Use rich tooltips only to make the information easier to understand.** Don't use rich tooltips just to decorate the feature. If you can't use richness to make the information easier to understand, use a plain tooltip instead.

  Incorrect:



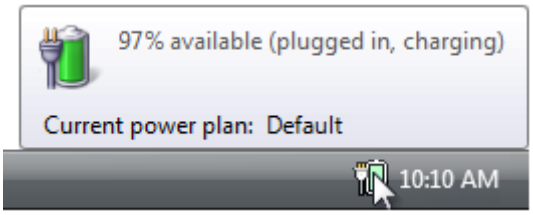
Correct:



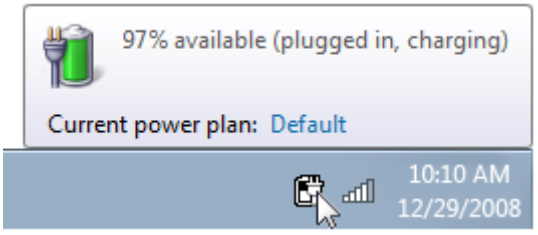
In the incorrect example, the calendar icon doesn't make the date easier to understand.

- **Use a concise presentation.** Use concise text and a concise layout with a 32x32 pixel icon. Spacious tooltips risk being distracting, especially when displayed unintentionally.
- **Don't put controls or elements that appear interactive in a rich tooltip.** Tooltips aren't interactive and therefore shouldn't appear interactive. Don't use blue or underlined text.

  Correct:

  Incorrect:

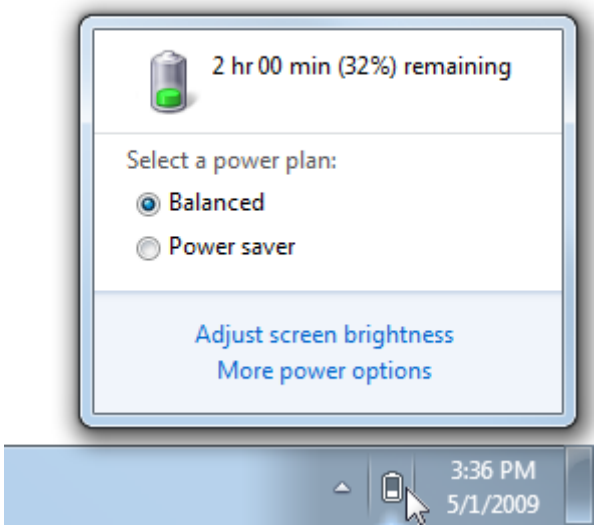
  

  In the incorrect example, the current power plan appears to be a link, but it is impossible to click.

Notification area flyouts

- When appropriate, present notification area flyouts with three sections:
  - **Summary.** Display the same information that is displayed in the icon's tooltip or infotip, possibly with more detail. For consistency, use the same text and icons, and generally the same layout (if using a rich tooltip). Unlike the infotips, this information is accessible when using touch.

- **Common tasks.** Present the most commonly performed tasks directly in the flyout.
- **Related links.** Provide at most one of each type of the following optional links:
  - A link to the most frequently performed task in Control Panel. Provide if there is a frequently performed task that can't be presented in the common tasks section.
  - A link to the related Control Panel item. This Control Panel item should allow users to perform any tasks that can't be performed in the common tasks section.
  - A link to a specific, relevant Help topic. Follow the standard Help link guidelines.



This example shows a notification area flyout using the recommended presentation.

Options dialog box

- Options not accessible directly from the context menu must be in the Options dialog box. This dialog could be the feature's control panel.
- **The Options dialog box may include the following items** as appropriate (exact text is in quotes):
  - Enable [feature name] (check box)
    - Clearing this option permanently quits the program. Program can be restarted from its control panel item. The Exit command in the context menu quits the program only for the current Windows session.
  - "Display icon in notification area" (check box)
    - Removing the icon from the notification area doesn't affect the underlying feature.
    - Selecting this option allows user to restore the icon, which of course can't be done from the icon itself.

- Disable features that are rarely used, or potentially annoying or distracting. Let users opt in to such features.

For general Options dialog box guidelines and examples, see Property Windows.

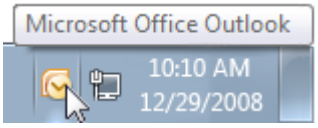
Minimizing programs to the notification area

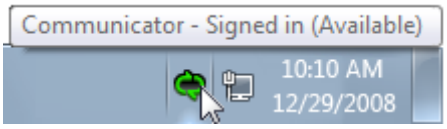
Note: Minimizing program windows to the notification area is no longer recommended for Windows 7. Use regular taskbar buttons instead. Your program may support both mechanisms for backward compatibility.

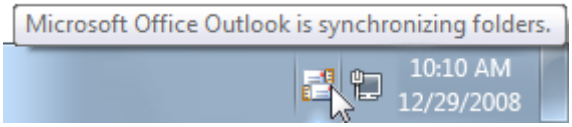
- To reduce taskbar clutter, consider providing the ability to minimize programs to the notification area only if all of the following apply:
  - The program can have only a single instance.
  - The program is run for an extended period of time.
  - The icon shows status.
  - The icon can be a notification source.
  - Doing so is optional and users must opt in.
- Use the Minimize button on the application's title bar, not the Close button.

Text

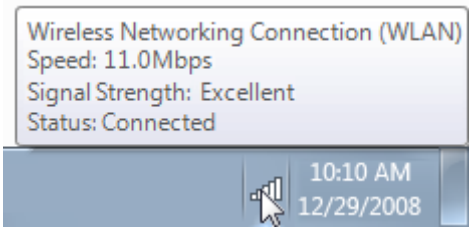
Infotips

- The icon infotip should have one of the following formats (where the company name is optional):
  - (Company name) Feature, program, or device name
  - 

  - (Company name) Feature, program, or device name - Status summary
  - 

  - (Company name) Feature, program, or device name status statement.
  - 

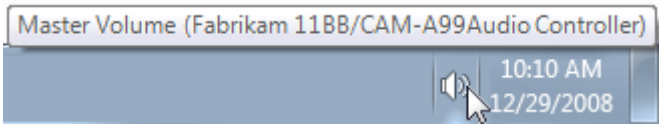
  - (Company name) Feature, program, or device name
  - Status list with each item on a separate line

- 

Infotip phrasing:

- Focus on the most useful information. Display other information on left single-click.
- Be concise. Use sentence fragments or simple statements.
- Don't use ending punctuation unless tip is phrased as a complete sentence.
- Omit unnecessary words. Don't include the software version or other extraneous information.

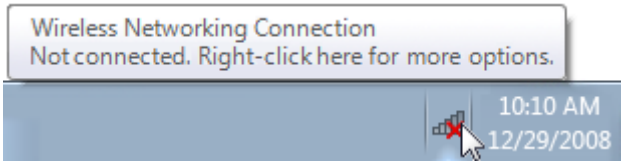
  Incorrect:

  In this example, the infotip has extraneous information.

- Don't explain how to interact with the icon.

  Incorrect:

  In this example, the Wireless Network Connection icon gives right-click instructions.

Documentation

When referring to the notification area:

- Refer to the notification area as the notification area, not the system tray.

When referring to a notification area icon:

- Refer to the icon using the exact name given in its infotip, including its capitalization, followed by icon.
- For the first reference, also refer to the notification area.
- When possible, format the heading text using bold. Otherwise, put the heading in quotation marks only if required to prevent confusion.

Example: To check the network status quickly, click the **Network** icon in the notification area.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Control Panels

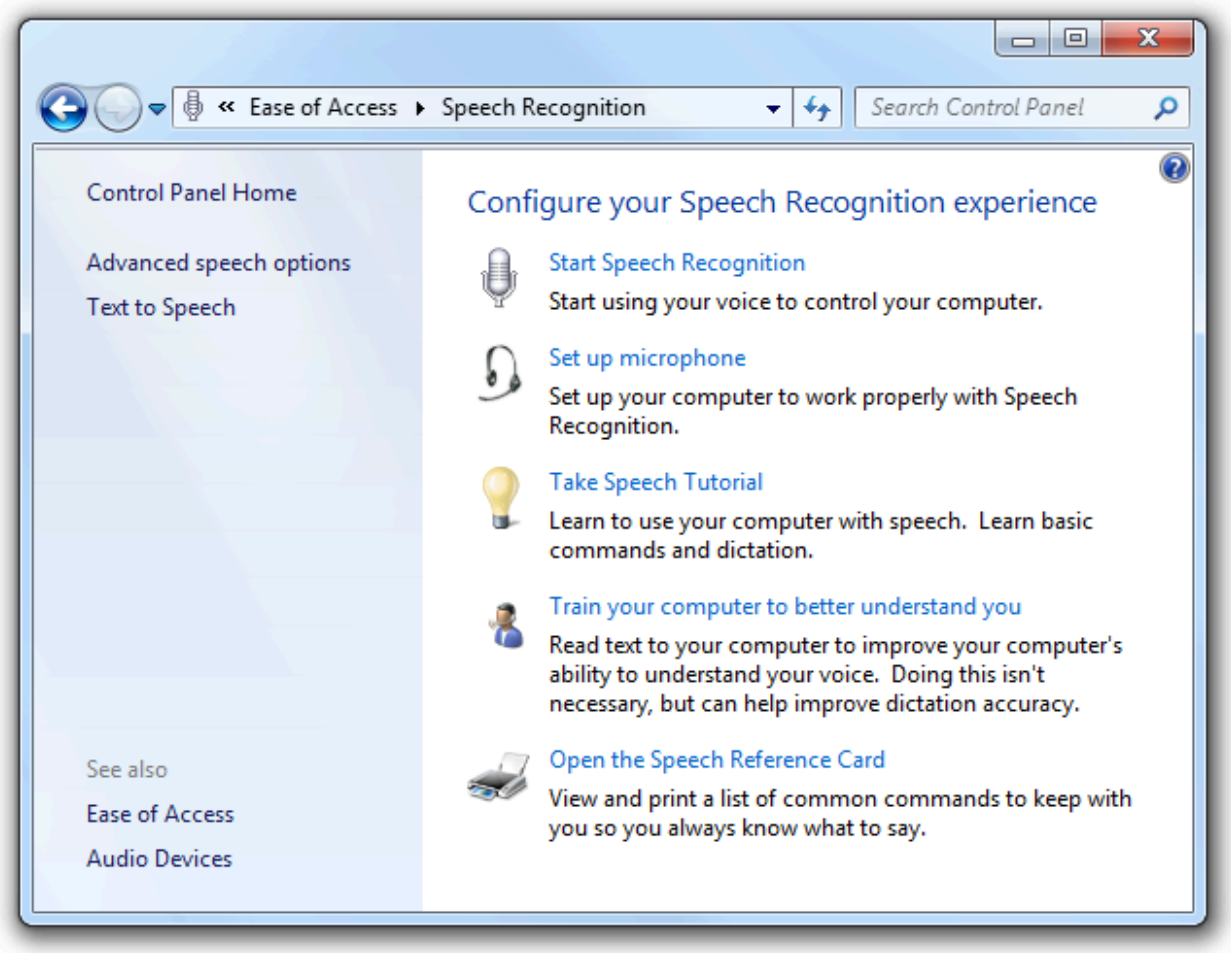
Article • 04/29/2022

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

Use control panel items to help users configure system-level features and perform related tasks. Programs that have a user interface should be configured directly from their UI instead.

With Control Panel in Microsoft Windows, users can configure system-level features and perform related tasks. Examples of system-level feature configuration include hardware and software setup and configuration, security, system maintenance, and user account management.

The term Control Panel refers to the entire Windows Control Panel feature. Individual control panels are referred to as control panel items. A control panel item is considered top-level when it is directly accessible from the control panel home page or a category page.



A typical control panel item.

The control panel home page is the main entry point for all control panel items. It lists the items by their category, along with the most common tasks. It is displayed when users click Control Panel in the Start menu.

A control panel category page lists the items within a single category, along with the most common tasks. It is displayed when users click a category name on the home page.

Control panel items are implemented using task flows or property sheets. For Windows Vista and later, task flows are the preferred user interface (UI).

Developers: To learn how to create control panel items, see Control Panel Items.

Note: Guidelines related to property sheets are presented in a separate article.

Is this the right user interface?

To decide, consider these questions:

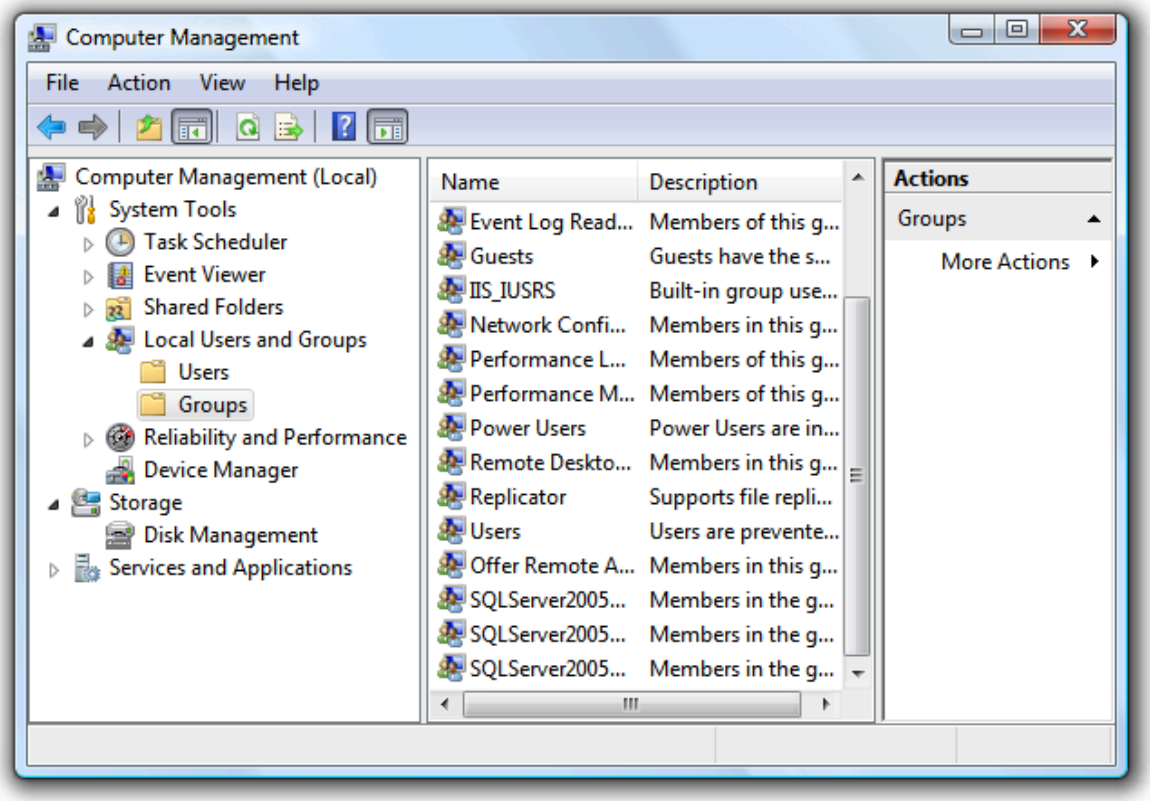
- **Is the purpose to configure system-level features?** If not, use another integration point. Make your application features configurable directly from the UI using

options dialog boxes, instead of using Control Panel. For utilities that aren't used for setup, configuration, or related tasks (like troubleshooting), use the Start menu as the integration point.

- **Does the system-level feature have its own UI?** If so, that UI is where users should go to make changes. For example, a system backup utility should be configured from its program options instead of from Control Panel.

- **Will users need to change the configuration often?** If so (say, several times a week), consider alternative solutions, perhaps in addition to using Control Panel. For example, the Windows master volume setting can be configured directly from its icon in the notification area. Some settings can be configured automatically. In Windows Explorer, for example, the Compatibility tab for application properties allows an application to be run in 256 color mode instead of requiring users to change the video mode manually.

- **Are the target users IT professionals?** If so, use a Microsoft Management Console (MMC) snap-in instead, which is designed specifically for system management tasks. In some cases, the best solution is to have both a control panel item for general users and an MMC snap-in for IT professionals.

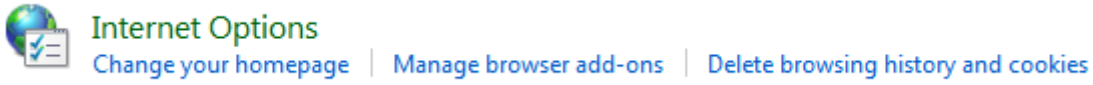


In this example, the Local Users and Groups MMC snap-in provides user management targeted at IT professionals. Other users are more likely to use the User Accounts item in Control Panel.

- **Is the feature an OEM feature used only during initial system configuration?** If so, use the Windows Welcome Center as the integration point.

Control panel items are necessary because many system-level features don't have a more obvious or direct integration point. Yet Control Panel shouldn't be viewed as the "one place" for all configuration settings. **Programs that have a user interface should be configured directly from their UI instead of using control panel items.**

Incorrect:



In this example, Windows Internet Explorer shouldn't be represented in Control Panel, because its own UI is a better integration point.

Create a new control panel item or extend an existing one?

To decide, consider these questions:

- **Can the functionality be expressed as tasks that can plug into an existing, extensible control panel item?** The following control panel items are extensible: Bluetooth Devices, Display, Internet, Keyboard, Mouse, Network, Power, System, Wireless (infrared).
- **Do the properties and tasks replace the features of the existing extensible control panel item?** If so, you should extend the existing control panel item because that results in a simpler user experience. If not, create a new control panel item.

Design concepts

The Control Panel concept is based on a real-world metaphor. A real-world control panel is a collection of controls (knobs, switches, gauges, and displays) used to monitor and control a device. Users of such control panels often need training to understand how to use them.

Unlike their real-world counterparts, **Windows control panel designs are optimized for first-time users.** Users don't perform most control panel tasks very often, so they usually don't remember how to do them and effectively have to relearn them every time.

To design a control panel item that is useful and easy to use:

- Make sure the properties are necessary.
- Present properties in terms of user goals instead of technology.
- Present properties at the right level.
- Design pages for specific tasks.
- Design pages for Standard users and Protected administrators.

When designing and evaluating items to include in Control Panel, determine the common tasks that users perform and make sure there is a clear path to perform those tasks. Users typically perform the following types of tasks with control panel items:

- Initial configuration
- Infrequent changes (for most settings)
- Frequent changes (for a few important settings)
- Rolling back settings to an initial or previous state
- Troubleshooting

If you do only one thing...

Design control panel pages for specific tasks, and present them in terms of user goals instead of technology.

Usage patterns

For control panel items, you can use a task flow or a property sheet. Here are their usage patterns:

Task flow patterns

Task flow items use a hub page to present the high-level choices, and spoke pages to perform the actual configuration.

Hub pages

- Task-based hub pages. These hub pages present the most commonly used tasks. They are best used for a few commonly used or important tasks where users need more guidance and explanation. Hub pages don't have commit buttons. Hybrid task-based hub pages also have some properties or commands directly on them. Hybrid hub pages are strongly recommended when users are most likely to use Control Panel to access those properties and commands.
- Object-based hub pages. These hub pages present the available objects using a list view control. They are best used when there could be several objects. Hub pages don't have commit buttons.

Spoke pages

- Task pages. These spoke pages present a task or a step in a task with a specific, task-based main instruction. They are best used for tasks that benefit from additional guidance and explanation.
- Form pages. These spoke pages present a collection of related properties and tasks based on a general main instruction. They are best used for features that have many properties and benefit from a direct, single-page presentation, such as advanced properties.

Property sheet patterns

- Property sheets are best used in legacy items with many settings targeted at advanced users. New items can achieve the same effect with a task flow using the form page pattern.

Guidelines

Property sheet control panel items

- **Don't use property sheets for new control panel items.** Instead, use task flows to create a seamless experience and make full use of the categorization and search functionality of the control panel home page.

Task flow control panel items

General

- **Keep the most important content and controls visible without scrolling.** Users won't scroll to see page content unless they have a reason to. You can make commit buttons always visible by placing them in a command area instead of the content area. Don't break up pages just to avoid scrolling.
  - **You can vertically scroll long pages,** as long as the most important controls are visible without scrolling.
  - **Don't use horizontal scrolling.** Instead, redesign the page content and use vertical scrolling. Pages may have horizontal scrollbars only when made very narrow.
- **To navigate between pages:**
  - Use task links to start a task.
  - Use task links or a Next button to navigate to the next page in a multi-step task.

    - Use commit buttons to complete a task.
    - Use the Back button in the menu bar to return to previously viewed pages. Do not add a Back button to the command area.
    - Use the Address bar to return directly to the control panel home page.
    - Use See also links in the task pane to navigate to pages in other control panel items. Otherwise, navigation should stay within a single control panel item.
- **Put only the control panel home page in the Address bar.** Clicking that link returns to the control panel home page, abandoning any work in progress without a confirmation .
- **Don't put a Close command button on control panel pages.** Users can close a control panel window using the Close button on the title bar.

Task links and buttons

- **When a page has a small set of fixed options, use task links instead of a combination of radio buttons and a Next button.** This allows users to select a response with a single click.

- You can put task links and buttons in the following places (in order of discoverability):
    - The command area (for command buttons on spoke pages only).
    - The content area:
        - Command buttons
        - Task links
        - Other links
    - Links in the task pane (hub pages only).

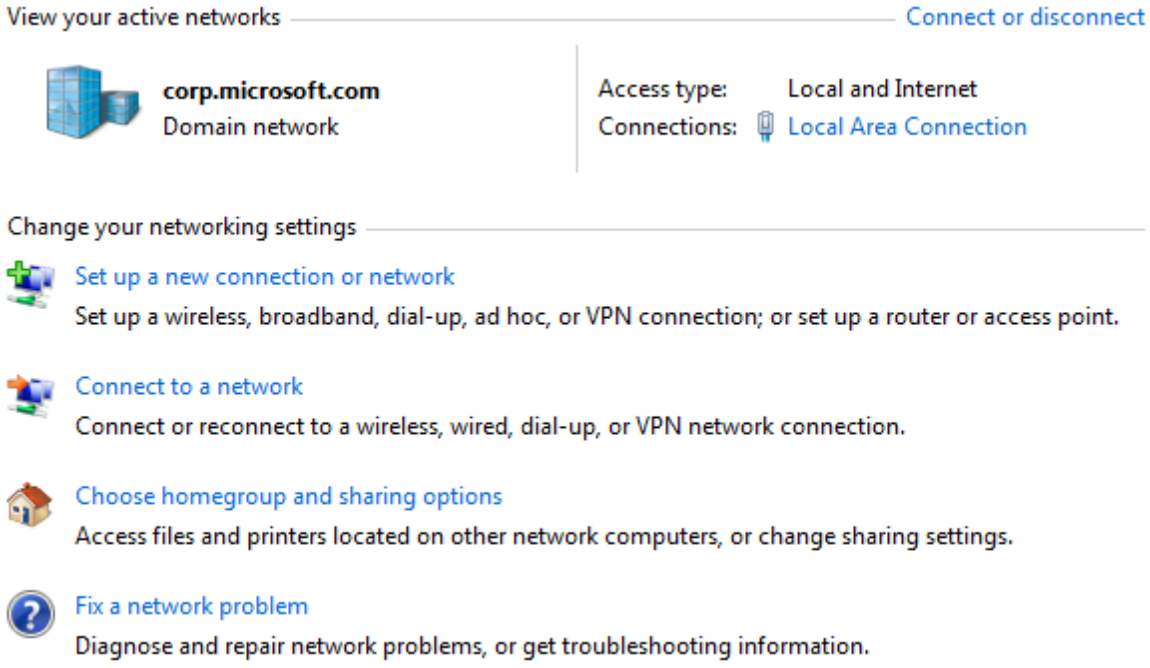
- **Base the location of task links and buttons on importance and need for discoverability.**
    - **Put only commit buttons in the command area.**
    - **Put essential tasks in the content area.** Command buttons tend to draw the most attention, so reserve them for commands users must see. Task links also draw attention, but less than command buttons.
    - **Reserve the task pane and plain links for secondary (less important) tasks.** The task pane is the least discoverable area of a task page, and plain links aren't as visible as command buttons and task links.

- For task links presented in the content area:
    - **If there are more than seven links, group the links into categories.** Provide headings for each of the groups.
    - **For fewer than seven links, present the links in a single group without a heading.**

- **Present task links and buttons in a logical order.** List task links vertically, command buttons horizontally.

- Within categories, **divide the commands into related groups.** Present the task groups by placing the most commonly used first, and within each group, place the most commonly used tasks first. **The resulting order should roughly follow the likelihood of use, but also have a logical flow.**
  - **Exception:** Task links that result in doing everything should be placed first.

- **If there are many task links, give the most important tasks a more prominent appearance** by using a 24x24 pixel icon and two lines of text. For less important tasks, use a 16x16 pixel icon, or no icon, and a single line of link text.



  In this example, important commands are given a more prominent appearance.

- **Have clear physical separation between frequently used commands and destructive commands.** Otherwise, users might click destructive commands accidentally. You may need to reorder your commands somewhat to put destructive commands together.

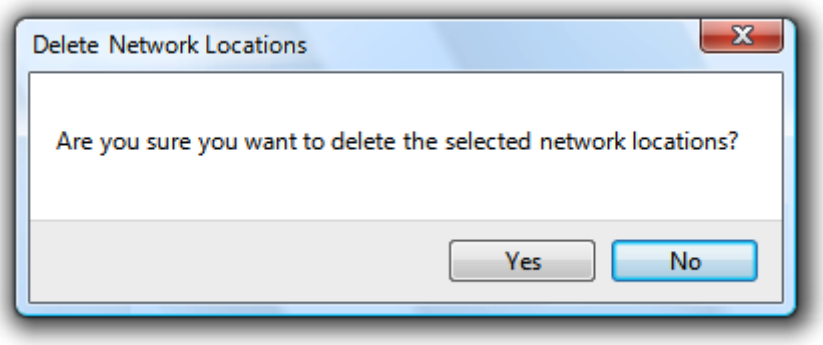
- **Provide the mechanism to undo commands directly on the page.** Users shouldn't have to navigate somewhere else to undo a mistake.

- **For task links, use either all default task link icons or all custom icons.** Don't mix them. Consider using custom icons only if:
  - They aid users in comprehending the tasks.
  - They comply with the Aero icon standards.
  - They have an unobtrusive appearance.

Dialog boxes

When using task flows, you generally want a task to flow from page to page within a single window, but the following circumstances are exceptions. Use dialog boxes in the following circumstances:

- To prompt users for an administrator user name and password. Always use the credential manager dialog box for this purpose. (Should be modal.)
- To confirm an in-place command using a task dialog or message box. (Should be modal.)
- To get input for in-place commands, such as for New, Add, Save As, Rename, and Print commands.

  In this example, the Delete command is performed in a dialog box instead of a separate page.

- To perform secondary, stand-alone tasks. Using a modeless, secondary window allows such tasks to be performed independently and outside the main task flow.

Hub pages

General

- Use task-based hub pages when:
  - **There are a small number of commonly used or important tasks.**
  - **The configuration involves one or two objects** (examples: monitors, keyboard, mouse, game controllers).
  - **The configuration applies system-wide** (examples: date and time, security, power options).
- Use object-based hub pages when:
  - **The configuration could involve several objects** (examples: user accounts, network connections, printers).
  - **The configuration applies only to the selected object**.

- **Don't use a hub page if the control panel item has a single page** that contains all the tasks and properties involved.

Object lists

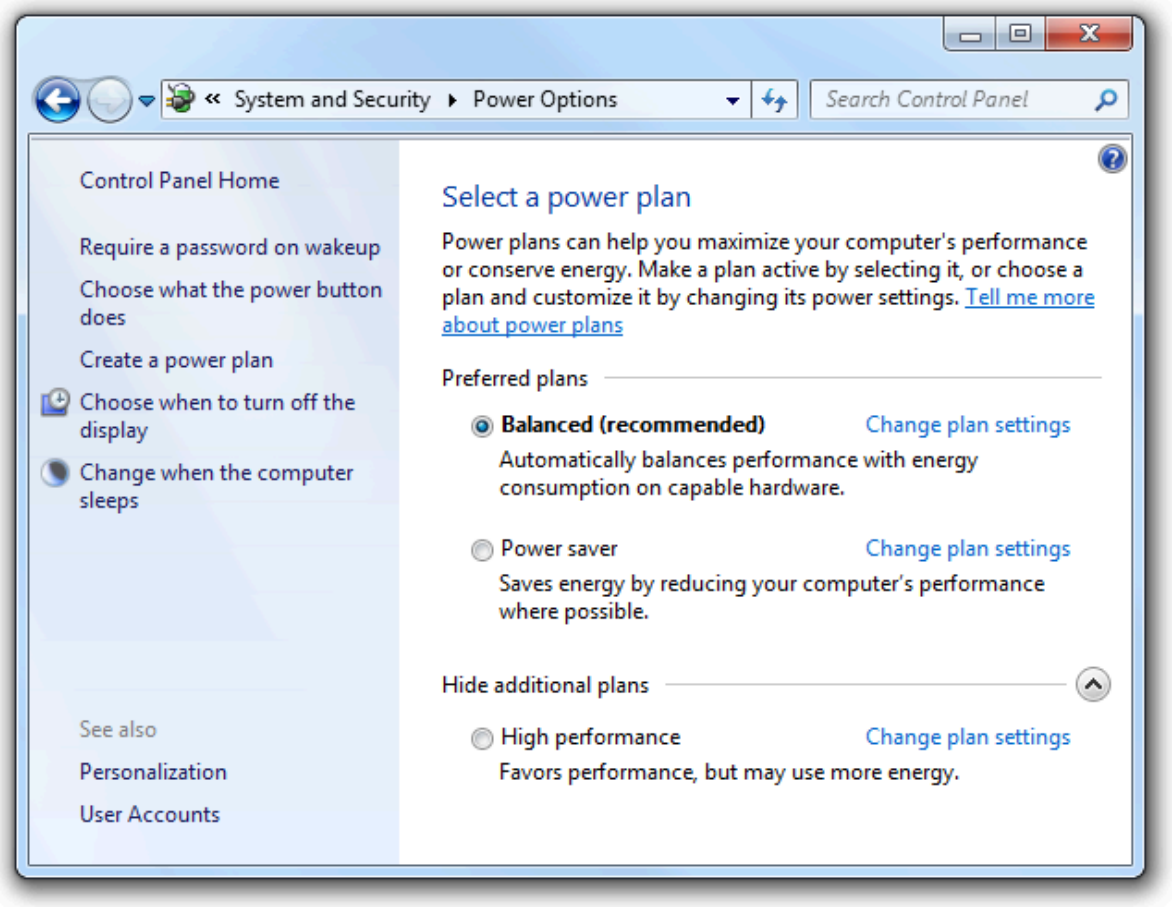
- **List items in a logical order.** Sort named objects in alphabetical order, numbers in numeric order, and dates in chronological order.
- For object-based hubs, **provide object view commands in the task pane if the ability to change the view is important to the tasks**. The ability to change views is important if there are many objects and the default presentation doesn't work well for all scenarios. Users can change the list view even if there aren't explicit commands through the list view context menu, but it's less discoverable.

For more guidelines about presenting object lists, see List Views.

Interaction

- **Don't put commit buttons on hub pages.** Hub pages are fundamentally launch points. Users never "commit" hub pages they are never done with them. And commit buttons on hub pages make any tasks initiated from a hub confusing (users will wonder if those tasks need to be committed).
  - **Exception:** If changing a setting requires elevation, provide an Apply button with a security shield icon. Disable the commit button once changes have been applied.

- **Consider putting the most useful properties directly on hub pages.** Such hybrid hub pages are strongly recommended when users are most likely to use Control Panel to access those properties.

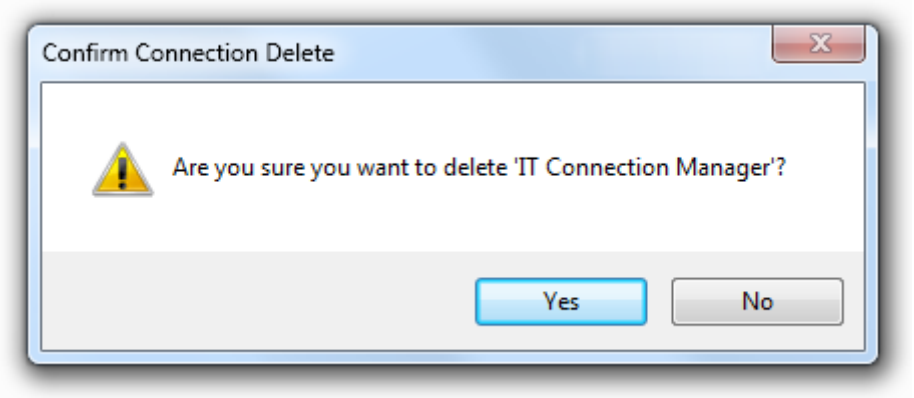


In this example, the Power Options control panel item has the most useful settings directly on the hub page.

- **Use an immediate commit model for any settings on hybrid hub pages so that changes are made immediately.** Again, users never commit a hub page. If a setting requires a commit button, don't put it on a hub page.

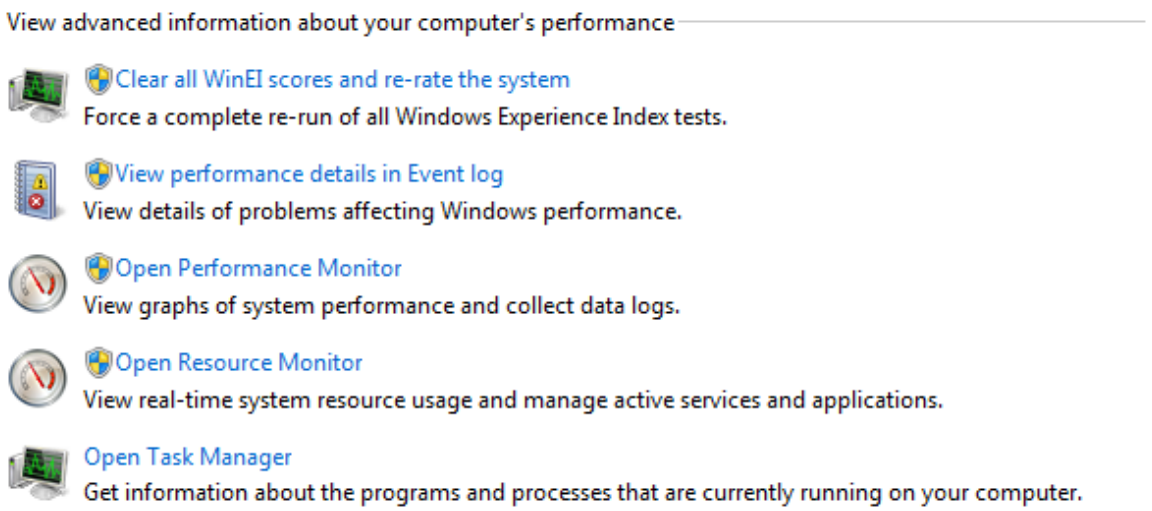
- **Consider putting simple, "one-step" commands directly on hub pages instead of using navigation links.**

- **Confirm in-place commands whose effects cannot be easily undone.** Use a task dialog or message box.



In this example, the Delete command is confirmed with a dialog box.

- **For task-based hub pages, identify each task with a task link and an icon.** You can also provide an optional description for each link. However, try to make the task links self-explanatory and provide optional descriptions only to links that really need them.



  In this example, each task has a task link and an icon.

- **For object-based hub pages, single-clicking selects objects, and double-clicking selects an object and navigates to its default page.** The default page is typically a property page or a task-based hub page.

- **An object-based hub page may navigate to a task-based hub for the selected objects.** However, such secondary hubs should be avoided because they make a control panel item feel too indirect.

Task panes

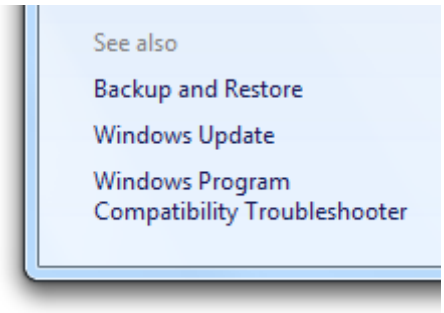
Use task panes to present links to commands, views, and related control panel items.

- For task panes in task-based hubs, present links in the following order:
  - **Secondary commands**. Present primary tasks only in the content area. Use the task pane for secondary, optional tasks. Consider a task primary if users must discover it in important scenarios; secondary if it's acceptable for users not to discover it.
  - **See also**. The optional links that navigate to related control panel items.
- For task panes in object-based hubs, present links in the following order:
  - **Object views**. The optional links used to control the presentation of the objects.
  - **Fixed commands**. The commands that are independent of the currently selected objects.
  - **Contextual commands**. The commands that depend on the currently selected objects, and are therefore not always displayed.
  - **See also**. The optional links that navigate to related control panel items.

- **Don't use task panes in spoke pages.** Unlike hub pages, spoke pages should be focused on completing the task. You don't want to encourage users to navigate away before completion.

See also links

- **Provide See also links in the task pane to help users find related control panel items, or the right control panel item if they have the wrong one.** Link to items users are likely to associate with your control panel item.

  In this example, the Action Center control panel item links to related control panel items.

- **Link to a specific task page if that is what users are more likely to recognize.** Otherwise, link to the entire control panel item. Use the control panel name without adding the phrase, control panel.

Spoke pages

General

- **Use task pages for commonly used or important tasks where users need more guidance and explanation.**
- **Use form pages for features that have many settings and benefit from a direct, single-page presentation.** The ideal tasks for such pages typically involve obvious changes to a few simple properties.
- **Don't use task panes in spoke pages.**

Interaction

- **Try to limit main tasks to a single page.** If more than one page is required, you can:
  - **Use intermediate spoke pages for additional or optional steps.** Intermediate spoke pages are committed by the final spoke page.
  - **Use independent windows for independent auxiliary tasks.** Independent windows are committed on their own, and independently of the main task.

Doing so keeps the meaning of the commit buttons for the main task clear and unambiguous. Users should always be confident in understanding what they are committing to.

- **Don't use See Also links within a task flow.** These link to related, but different, control panel items. Although navigating to a different item is acceptable in hub pages, it is not in spoke pages, because doing so interrupts the task.
- **Don't use spoke pages for simple input or confirmations.** Use modal dialog boxes instead.

Interaction (intermediate spoke pages)

- **Use task links or a Next button to navigate to the next page.** The way to proceed to the next step should always be obvious.
- **You can have navigation links to optional task steps.** To avoid confusion when users commit to the task, those extra pages should be intermediate pages within the same control panel item. They shouldn't have commit buttons, but should be committed when the main task is committed.

Interaction (final spoke pages)

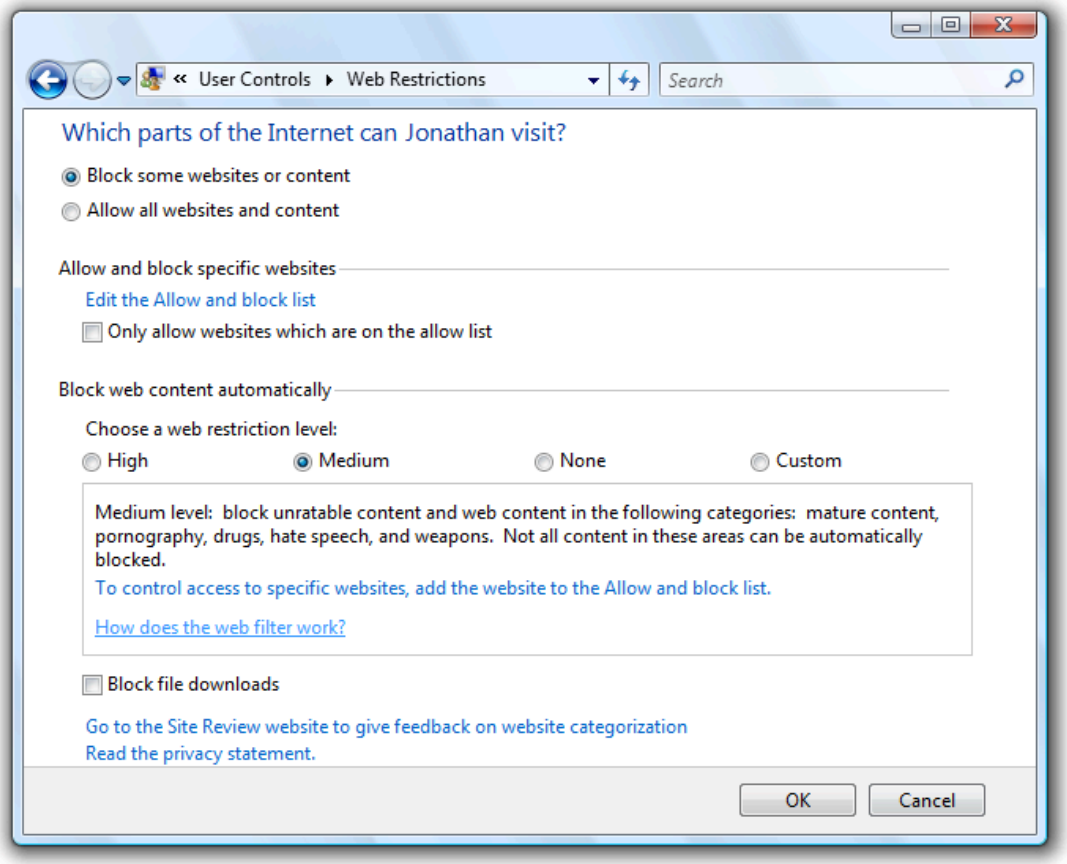
- **Use commit buttons to complete a task.** Use a delayed commit model for spoke pages, so that changes aren't made until explicitly committed (if users navigate away using Back, Close, or the Address bar, changes are abandoned). The commit buttons are a visual clue that the user is about to complete a task. Don't use links for this purpose.
- **Don't confirm commit buttons (including Cancel).** Doing so can be annoying. Exceptions:
  - The action has significant consequences and, if incorrect, not readily fixable.
  - The action may result in a significant loss of the user's time or effort.
  - The action is clearly inconsistent with other actions.
- **Don't confirm if users abandon changes** by navigating away using Back, Close, or the Address bar. However, you may confirm if a potentially unintended navigation may result in a significant loss of the user's time or effort.
- **Don't use command or navigation links** (including See also links). On final spoke pages, users should explicitly complete or cancel the task. Users should not be encouraged to navigate somewhere else, because doing so would likely cancel the task implicitly.
- **When users complete or cancel a task, they should be sent back to the hub page from which the task was launched.** If there is no such page, close the control panel window instead. Don't assume that spoke pages are always launched from another page.

- **Remove the stale "committed" pages from the Windows Explorer Back stack** when you return users back to the page that the task was launched from. Users should never see the pages that they have already committed to when clicking the Back button. Users should always make additional changes by completely redoing the task instead of clicking Back to modify stale pages.
    - **Developers:** You can remove these stale pages using the ITravelLog::FindTravelEntry() and ITravelLogEx::DeleteEntry() APIs.

Commit buttons

Note: Cancel buttons are considered to be commit buttons.

- **Confirm tasks using commit buttons that are specific responses to the main instruction, instead of generic labels such as OK.** The labels on commit buttons should make sense on their own. Avoid using OK because it isn't a specific response to the main instruction, and therefore easier to misunderstand. Furthermore, OK is typically used with modal dialog boxes and incorrectly implies closing the control panel item window.
    - **Exceptions:**
        - Use OK for pages that don't have settings.
        - Use OK when the specific response is still generic, such as Save, Select, or Choose, as when changing a specific setting or a collection of settings.
        - Use OK if the page has radio buttons that are responses to the main instruction. To maintain the delayed commit model, you can't use task links on a final spoke page.



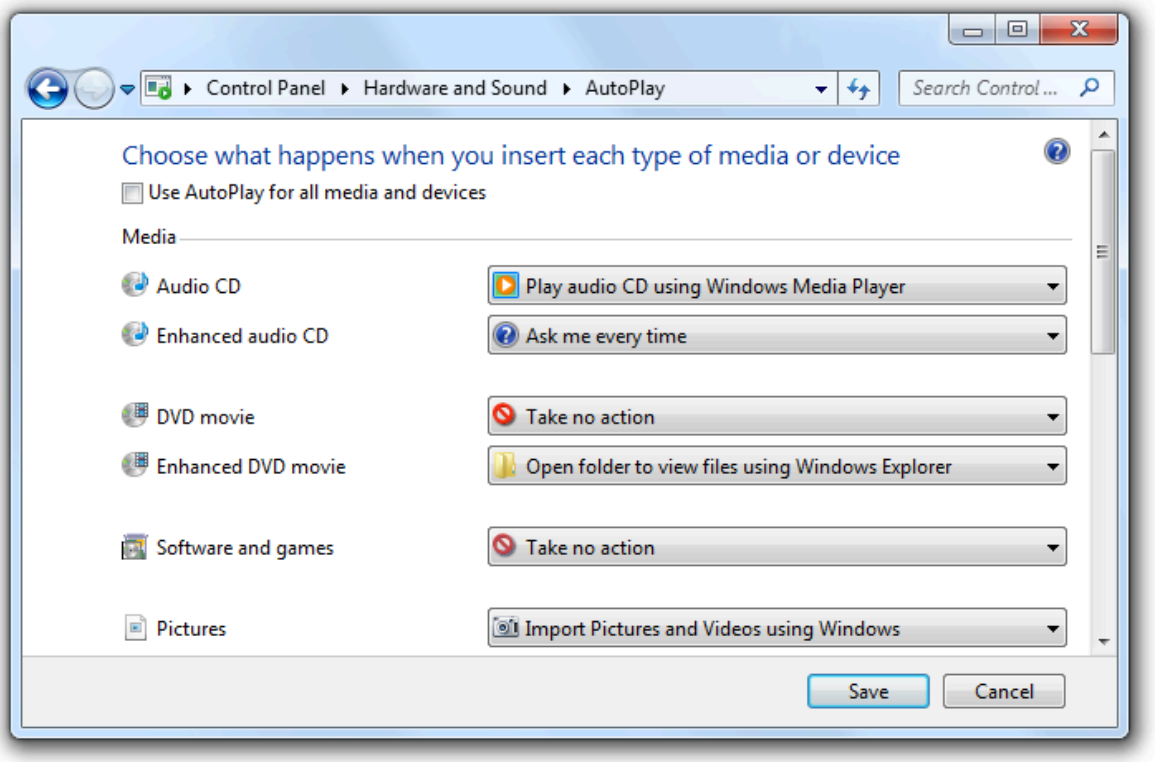
In this example, the radio buttons, not the commit buttons, are responses to the main instruction.

- **Provide a Cancel button to let users explicitly abandon changes.** While users can implicitly abandon a task by not confirming their changes, providing a Cancel button allows them to do so with greater confidence.
  - **Exception:** Don't provide a Cancel button for tasks where users can't make changes. The OK button has the same effect as Cancel in this case.

- **Don't use Close, Done, or Finish commit buttons.** These buttons are typically used with modal dialog boxes and incorrectly imply closing the control panel item window. Users can close the window using the Close button on the title bar. Also, Done and Finish are misleading because users are returned to the page where the task was launched from, so they aren't really done.

- **Don't disable commit buttons.** Otherwise users have to deduce why the commit buttons are disabled. It's better to leave commit buttons enabled, and give a helpful error message whenever there is a problem.

- **Make sure the commit buttons appear on the page without scrolling.** If the page is long, you can make commit buttons always visible by placing them in a command area, instead of in the content area.



In this example, content area size is unbounded, so the commit buttons are placed in the command area.

- Right-align the commit buttons and use this order (from left to right): positive commit buttons, Cancel, and Apply.

Preview buttons

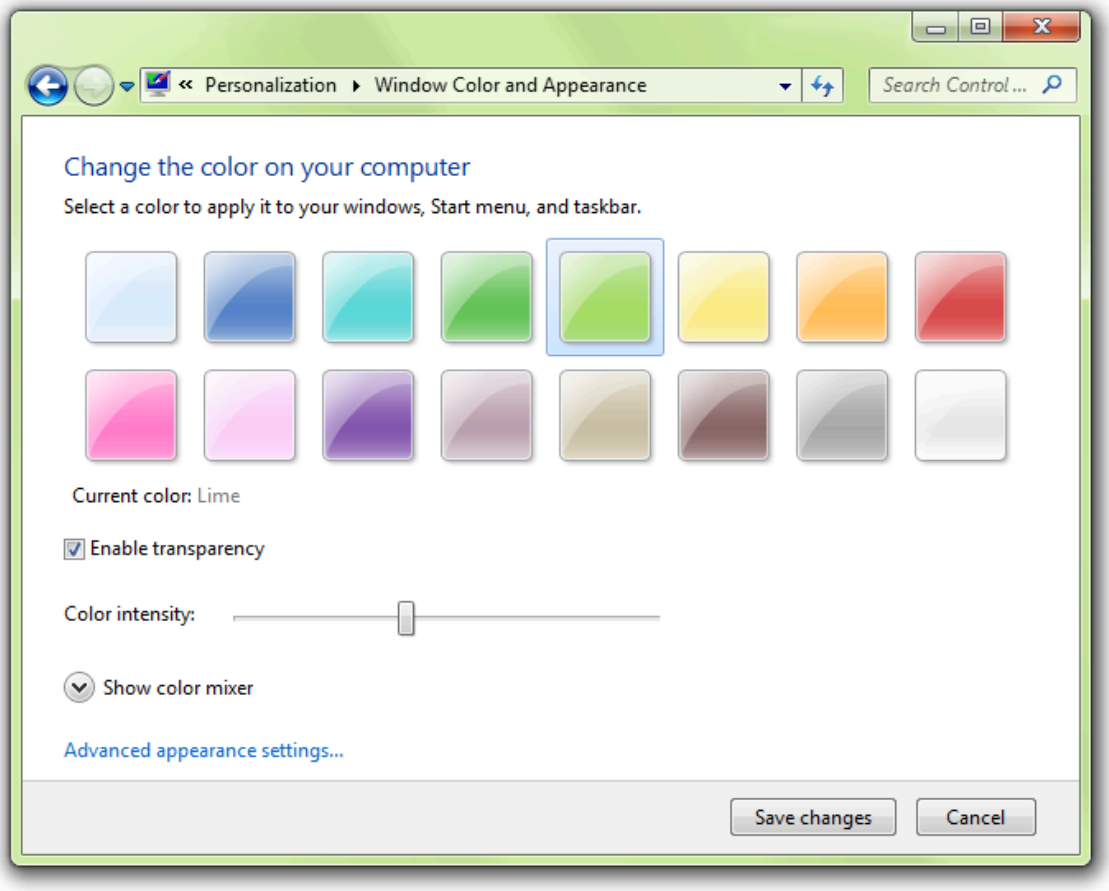
- Make sure **the Preview button means to apply the pending changes now but restore the current settings if users navigate away from the page without committing to the changes.**
- **You can use Preview buttons on any spoke page.** Hub pages don't need Preview buttons because they use an immediate commit model.
- **Consider using a Preview button instead of an Apply button for control panel pages.** Preview buttons are easier for users to understand and can be used on any spoke page.
- **Provide a Preview button only if the page has settings (at least one) with effects that users can see.** Users should be able to preview a change, evaluate the change, and make further changes based on that evaluation.
- **Always enable the Preview button.**

Live previews

A control panel item has live preview when the effect of changes on a spoke page can be seen immediately.

- **Consider using live preview for display settings when:**

- The effect is obvious, typically because it applies to the entire monitor.
- The effect can be applied without significant delay.
- The effect is safe and can be undone easily.



In this example, the effect of the Windows Color and Appearance settings is seen immediately. Doing so allows users to make changes with minimal effort.

- **Use Save changes and Cancel for the commit buttons.** "Save changes" keeps the current settings, whereas Cancel reverts to the original settings. "Save changes" is used instead of OK to make it clear that any previewed changes haven't been applied yet.
- **Don't provide an Apply button.** The live preview makes Apply unnecessary.
- **Restore any changes if users navigate away** using Back, Close, or the Address bar. To preserve changes, users must commit them explicitly.

Apply buttons

- Make sure **the Apply button means apply the pending changes (made since the task was started or the last Apply), but remain on the current page.** Doing so allows users to evaluate the changes before moving on to other tasks.

- **Use Apply buttons only on final spoke pages.** Apply buttons should not be used on intermediate spoke pages to maintain an immediate commit model.
  - **Exception:** You can use Apply buttons on a hybrid hub page if changing a setting requires elevation. For more details, see hub page interaction.
- **Provide an Apply button only if the page has settings (at least one) with effects that users can evaluate in a meaningful way.** Typically, Apply buttons are used when settings make visible changes. Users should be able to apply a change, evaluate the change, and make further changes based on that evaluation.
- **Enable the Apply button only when there are pending changes;** otherwise, disable it.
- **Assign "A" as the access key.**

Control panel integration

To integrate your control panel item with Windows, you can:

- **Register your control panel item (including its name, description, and icon)**, so that Windows is aware of it.

- If your control panel item is top level (see below):
  - Associate it with the appropriate **category page**.
  - **Provide task links (including their name, description, keywords, and command line)** to indicate primary tasks and allow users to navigate directly to the tasks.

- **Provide search terms** to help users find your task links using the Control Panel search feature.

  Note that you can provide this information only for individual control panel items you can't add or change this information for existing control panel items that you extend.

Category pages

- **Add your control panel item to a category page only if:**
  - Most users need it. Example: Network and Sharing Center
  - It is used many times. Example: System
  - It provides important functionality that isn't exposed elsewhere. Example: Printers

  Control panel items that meet these criteria are referred to as top level.

- **Don't add your control panel item to a category page if:**
  - It is rarely used or used for one-time setup. Example: Welcome Center

    - It is targeted at advanced users or IT professionals. Example: Color Management
    - It doesn't apply to the current hardware or software configuration. Example: Windows SideShow (if not supported by current hardware).

  Removing such control panel items from the category pages makes the top-level items easier to find. Given their usage, these control panel items are sufficiently discoverable through search or contextual entry points.

- **Associate your top-level control panel item with the category under which users are most likely to look for it.** This decision should be based on user testing.

- **Consider associating your top-level control panel item with the second most likely category as well.** You should associate a control panel item with two categories if users are likely to look for its main tasks in more than one place.

- **Don't associate your control panel item with more than two categories.** The value of the categorization is undermined if control panel items appear in several categories.

Task links

- **Associate your control panel item with its primary tasks.** You can display up to five tasks on a Category page, but additional tasks are used for control panel searching. Use the same phrasing as you do for task links, possibly removing some words to make the task links more succinct.
- **Prefer to have task links lead to different places in your control panel item.** Having multiple links to the same place can be confusing.

Search terms

- **Register search terms for your control panel item that users are most likely to use to describe it.** These search terms should include:
    - The features or objects configured.
    - The primary tasks.

  These search terms should be based on user testing.

- **Also include common synonyms for these search terms.** For example, monitor and display are synonyms, so both words should be included.

- **Include alternative spellings or word breaks.** For example, users might search for either web site and website. Consider providing common misspellings as well.

- **Consider singular vs. plural noun forms.** The control panel search feature doesn't automatically search for both forms; supply the forms for which users are likely to

search.

- **Use simple present tense verbs.** If you register connect as a search term, the search feature won't automatically look for connects, connecting, and connected.
- **Don't worry about case.** The search feature is not case-sensitive.

Standard users and Protected administrators

Many settings require administrator privileges to change. If a process requires administrator privileges, Windows Vista and later requires Standard users and Protected administrators to elevate their privileges explicitly. Doing so helps prevent malicious code from running with administrator privileges.

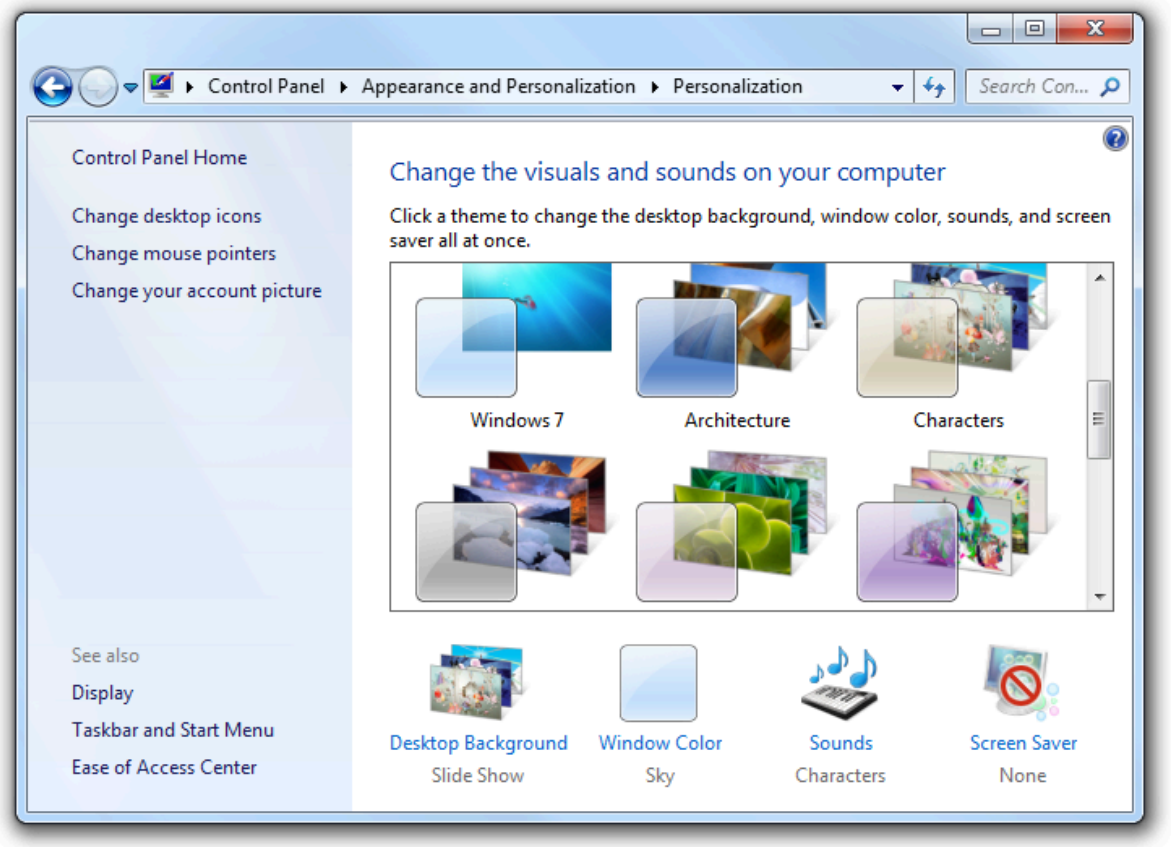
For more information and examples, see User Account Control.

Schemes and themes

A scheme is a named collection of visual settings. A theme is a named collection of settings across the system. Examples of schemes and themes include Display, Mouse, Phone and Modem, Power Options, and Sound and Audio Options.

- **Allow users to create schemes when:**
  - **Users are likely to change the settings.**
  - **Users are most likely to change settings as a collection.**

  Schemes are useful when users are in a different environment, such as a different physical location (country/region, time zone); using their computer in a different situation (on batteries, docked/undocked); or using their computer for a different function (presentations, video playback).
- **Provide at least one default scheme.** The default scheme should be well named and apply to most users in most circumstances. Users shouldn't have to create a scheme of their own.
- **Provide a preview** or other mechanism so that users can see the settings within the scheme.



In this example, the Personalization control panel item shows a preview of the desktop and appearance settings.

- **Provide Save As and Delete commands.** A rename command isn't necessary users can rename schemes by saving under the desired name and deleting the original scheme.
- If the settings can't be applied without a scheme, **don't allow users to delete all the schemes.** Users shouldn't have to create a scheme of their own.
- If the schemes are not completely independent (for example, power schemes depend upon the current laptop mode of operation), **make sure there is an easy way to change settings that apply across all schemes.** For example with power schemes, make sure that users can set what happens when a portable computer's lid is closed in a single location.

Miscellaneous

- **Use Control Panel extensions for features that replace or extend existing Windows functionality.** The following control panel items are extensible: Bluetooth Devices, Display, Internet, Keyboard, Mouse, Network, Power, System, Wireless (infrared).

Default values

- **The settings within a control panel item must reflect the current state of the feature.** Doing otherwise would be misleading and possibly lead to undesired results. For example, if settings reflect the recommendations but not the current state, users might click Cancel instead of making changes, thinking that no changes are needed.
- **Choose the safest (to prevent loss of data or system access) and most secure initial state.** Assume that most users won't change the settings.
- **If safety and security aren't factors, choose the initial state that is most likely or convenient.**

Text

Item names

- **Choose a descriptive name that clearly communicates and differentiates what the control panel item does.** Most names describe the Windows feature or object being configured, and are displayed in the Classic View of the control panel home page.

- **Don't include the words "Settings," "Options," "Properties," or "Configuration" in the name.** This is implied, and leaving it off makes it easier for users to scan.

  Incorrect:

  Accessibility Options

  Modem Settings

  Power Options

  Regional and Language Options

  Correct:

  Accessibility

  Modem

  Power

  Regional Formats and Languages

  In the correct examples, unnecessary words are removed.

- **If the control panel item configures related features, list only those features required to identify the item, and list the features the most likely to be recognized or used first.**

  Incorrect:

  Folder Options

  Phone and Modem Options

  Correct:

  Files and Folders

  Modem

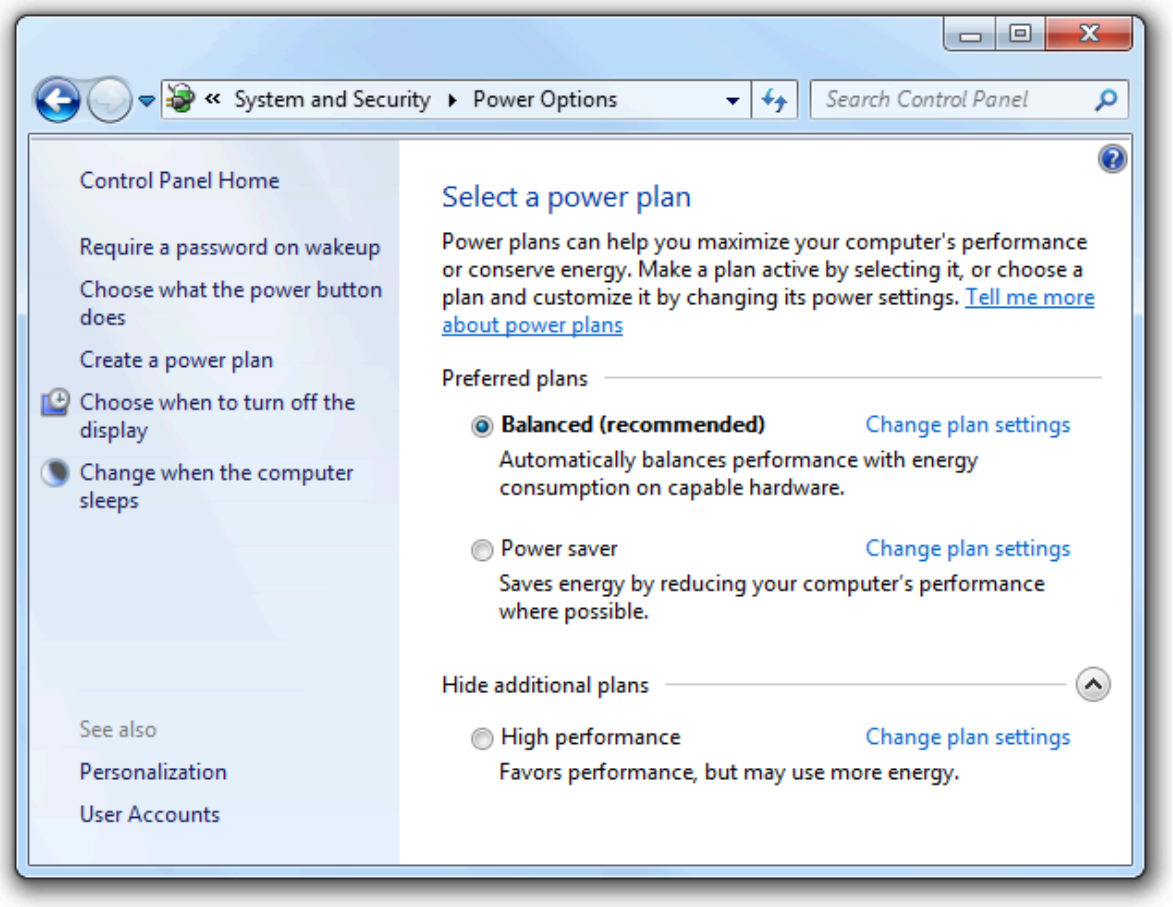
  In the correct examples, the primary control panel item features are given emphasis.

- Use title-style capitalization.

Page titles

Note: As with all Explorer windows, control panel page titles are displayed on the address bar, but not the title bar.

- **For hub pages, use the control panel item name.**
- **For spoke pages, use a concise summary of the page's purpose.** Use the page's main instruction if it is concise; otherwise use a concise restatement of the main instruction.



In this example, Power Options is used for the page title instead of the main instruction.

- Use title-style capitalization.

Task link text

The following guidelines apply to links to task pages, such as Category page task links and See also links.

- **Choose concise task names that clearly communicate and differentiate the task.** Users shouldn't have to figure out what the task really means or how it differs from other tasks.

  Incorrect:

  Adjust display settings

  Correct:

  Screen resolution

  In the correct example, the wording conveys more precision.

- **Retain similar language between task links and the pages they point to.** Users shouldn't be surprised by the page that is displayed by a link.
- **For task pages, design the main instruction, commit buttons, and task links as a related set of text.**

Expand table

Example	Instruction
Task link:	Connect to a wireless network
Main instruction:	Choose a network to connect to
Commit button:	Connect
Task link:	Set up parental controls
Main instruction:	Choose a user and set up parental controls
Commit button:	Apply parental control
Task link:	Resolve your sync conflicts
Main instruction:	How do you want to resolve this conflict?
Commit button:	Resolve

These examples show the relationship of the task link text, main instruction, and commit button text.

- While tasks often start with verbs, **consider omitting the verb on Category pages if it is a generic, configuration-related verb that doesn't help communicate the task.**

  Specific, helpful verbs:

  Add

  Check

  Connect

  Copy

  Create

  Delete

Disconnect

Install

Remove

Set up

Start

Stop

Troubleshoot

Generic, unhelpful verbs:

Adjust

Change

Choose

Configure

Edit

Manage

Open

Pick

Set

Select

Show

View

- **If the task configures several related features, list only the features that are representative of the set.** Omit details that can be readily inferred.

  Incorrect:

  Speaker volume, mute, volume icon

  Speakers, microphones, MIDI, and sound schemes

  Correct:

Speaker volume

Speakers and microphones

In the correct examples, only the representative features are given in the task link.

- **You should phrase tasks in terms of technology only if target users would do so as well.**

  Incorrect:

  Connectoids

  Pixel depth

  dpi

  Correct:

  Printers

  Scanners

  Mouse

  The correct examples are technology-based terms that target users are likely to use, whereas the incorrect examples are not.

- Use plural nouns only if the system can support more than one.
- Use sentence-style capitalization.
- Don't use ending punctuation.

Main instructions

- **For the hub page, use the main instruction to explain the user's objective with the control panel item.** The main instruction should help users determine if they have selected the right control panel item. Keep in mind that users might have selected your control panel item in error and are actually looking for settings that are part of another control panel item.

  Examples:

  Keep your computer secure and up-to-date

  Optimize your computer so it's easier to see, hear, and control

- **For spoke pages, use the main instruction to explain what to do on the page.** The instruction should be a specific statement, imperative direction, or question. Good instructions communicate the user's objective with the page rather than focusing purely on the mechanics of manipulating it.

  Incorrect:

  Pick a notification task

  Correct:

  Indicate how to handle incoming information

  The correct version better communicates the goal achieved by the page.

- **Use specific verbs whenever possible.** Specific verbs are more meaningful to users than generic ones.
- Use sentence-style capitalization.
- **Don't include final periods if the instruction is a statement.** If the instruction is a question, include a final question mark.

Supplemental instructions

- **For the hub page, use the optional supplemental instruction to further explain the purpose of the control panel item.**
- **For spoke pages, use the optional supplemental instruction to present additional information helpful to understanding or using the page.** You can provide more detailed information and define terminology.
- Use complete sentences and sentence-style capitalization.

Page text

- **Don't restate the main instruction in the content area.**
- **Use the word "page" to refer to the page itself.**
- **Use the second person (you, your) to tell users what to do** in the main instruction and content area. Often the second person is implied.

  Example:

  Choose the pictures you want to print.

- **Use the first person (I, me, my) to let users tell the control panel item what to do** in the content area that responds to the main instruction.

  Example:

  Print the photos on my camera.

Task links

- **Choose concise link text that clearly communicates and differentiates what the task link does.** It should be self-explanatory and correspond to the main instruction. Users shouldn't have to figure out what the link really means or how it differs from other links.
- **Always start task links with a verb.**
- Use sentence-style capitalization.
- Don't use ending punctuation.
- **If the task link requires further explanation, provide the explanation in a separate text control** using complete sentences and ending punctuation. However, add such explanations only when needed don't add explanations to all task links because another task link needs one.

For more information and examples, see Links.

Commit buttons

- **Use specific commit button labels that make sense on their own and match the main instruction.** Ideally users shouldn't have to read anything else to understand the label. Users are far more likely to read command button labels than static text.
- **Always start commit button labels with a verb.**
- **Don't use Close, Done, or Finish.** These button labels are better suited for other types of windows.
- Use sentence-style capitalization.
- Don't use ending punctuation.
- Assign a unique access key.
  - **Exception:** Don't assign access keys to Cancel buttons, because Esc is its access key. Doing so makes the other access keys easier to assign.

Documentation

When referring to the control panel home page or category pages:

- In user documentation, refer to Control Panel, using title-style capitalization and omitting a preceding definite article the.

  Example:

  In Control Panel, open **Network and Sharing Center**.

- In programming and other technical documentation, refer to control panel home page and control panel category page, without capitalizing any of the words. A preceding definite article is optional.

For control panel items:

- When referring to an individual control panel item, use "[control panel item name] in Control Panel" or generally "Control Panel item" in user documentation. Don't use applet, program, or control panel to refer to individual control panel items.
- When referring to a control panel item's hub page, use "main [control panel item name] page."
- When possible, format the control panel name using bold text. Otherwise, put the name in quotation marks only if required to prevent confusion.

Examples:

- In Control Panel, open **Parental Controls**.
- Return to the main **Parental Controls** page.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Help

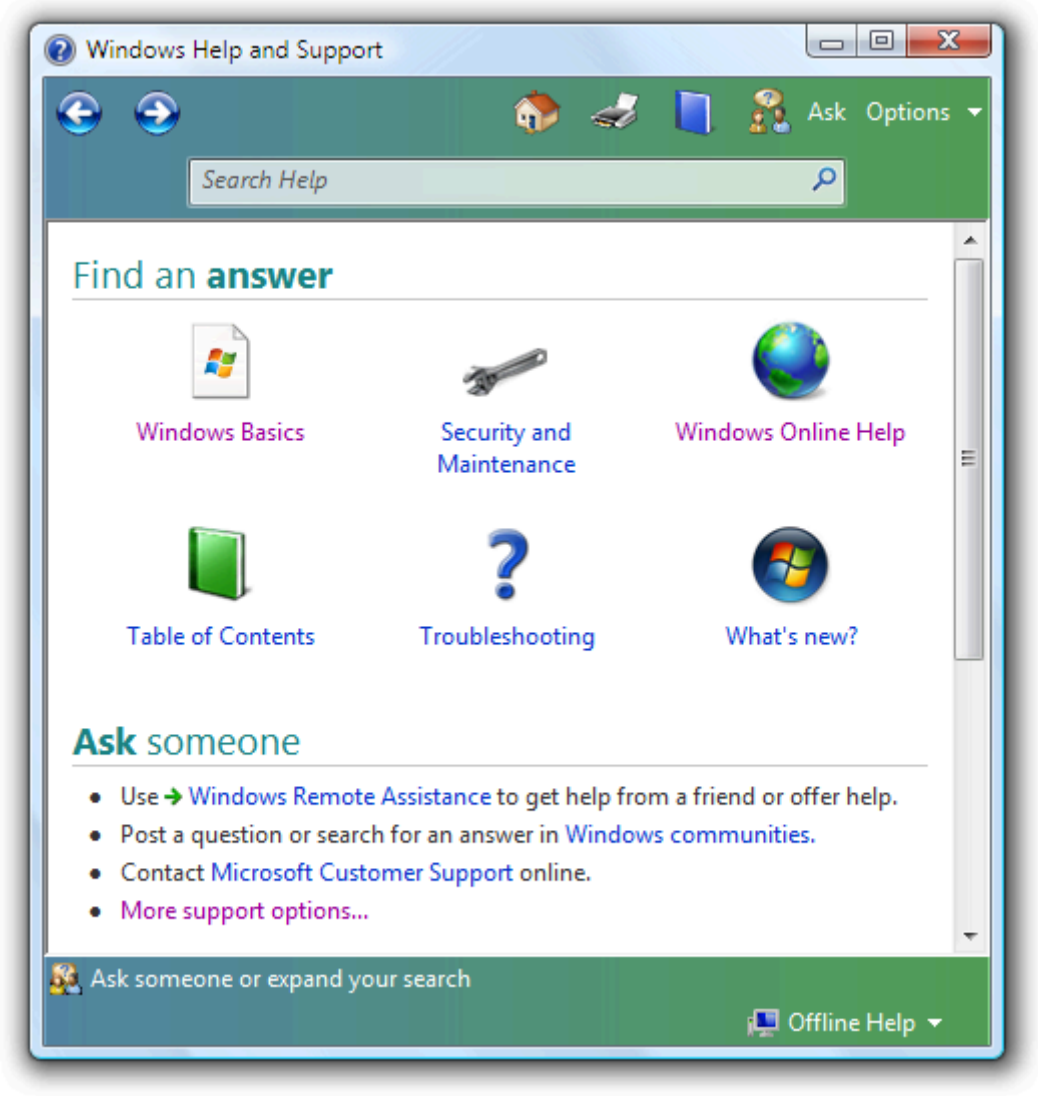
Article • 06/03/2021

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our **current design guidance**.

Use Help as a secondary mechanism to help users complete and better understand tasks the primary mechanism being the UI itself. Apply these guidelines to make the content truly helpful and easy to find.

A Help system is composed of various types of content designed to assist users when they are unable to complete a task, want to understand a concept in more detail, or need more technical details than are available in the UI.

In this article, we refer to Help as secondary to UI. The UI is primary because that is where users first try to solve their problems. They consult the Help system only if they can't accomplish their task with the UI.



The Windows Help and Support home page, available from the Start menu.

Note: Guidelines related to style and tone are presented in a separate article.

Is this the right user interface?

To decide, consider these questions:

- **How motivated are your target users?** The more highly motivated they are to discover functionality of your program and become intermediate or even advanced users of it, the more willing they will be to research answers to their questions by consulting Help topics.
- **Are you using Help to fix a bad UI?** The better your UI, the less users will seek additional help. If your program has very clear, helpful primary UI (such as jargon-free error messages, well-written wizards, and unambiguous dialog boxes), you may not need a secondary Help system at all.
- **Is your program relatively simple?** If so, consider incorporating all necessary assistance content into your primary UI surfaces. Users are more likely to seek additional help in programs that perform complex tasks.

- **Is your application intended for developers, IT professionals, or other software experts?** These users tend to expect reference Help for programming language conventions and in-depth conceptual Help for mastery of features.

Design concepts

If you decide to include Help in your program, integrate it into your overall design. The Help interface should be simple, efficient, and relevant; it should enable users to get help easily and then return to their task. Think of your Help system in terms of users' time: minimize disruption first by anticipating where they will encounter problems in your program, then solving those problems by incorporating fundamental assistance directly into your UI, and creating clear and consistent entry points into your more detailed Help.

Windows assistance was designed according to these principles. Here are some of the design changes to the Windows Help user experience:

- More discoverable entry points to Help from the primary UI (especially new Help links from UI surfaces such as dialog boxes, error messages, and wizards). Help links take you directly to the pertinent topic in Help.
- A Help button icon is available in the upper-right corner of most Control Panel hub pages, as well as shell folders.
- Users can choose to get the most up-to-date Help content from Windows Online Help and Support when they're online.
- Help topics are now task-based rather than feature-focused, so that users can accomplish their tasks quickly and efficiently.
- Help topics are now largely based on known top user scenarios.
- Help topics have a more relaxed and informal tone, using real-world language.
- Help topics are designed for effective scanning, as users rarely read the content word-for-word.

An analogy for Help

To think in greater depth about designing your Help system, consider an analogy from everyday life. You are lost in an unfamiliar city. What do you do? Many would react like this:

- Get oriented; look for landmarks, street signs (names and pointers to places).
- Look for maps.
- Finally, as a last resort, ask for directions or call a friend.

The design of the city's "interface" affects your need for assistance. Well-labeled streets, explicit directions (pointers to hospitals, airports, museums, and the post office), and clear landmarks such as prominent geographical features or buildings help you find your way.

You ask for help as a last resort. It's an indication that the city's "interface" has failed by being poorly designed and confusing. You are more likely to ask for help in a place that has a specific label that suggests helpfulness. For example, you are more likely to ask for help in a place labeled "Directions" or "Information" than a general place like the City Hall even though just about anyone at City Hall could give you directions.

When you ask for help, chances are you are frustrated and just want to get to your intended destination. You probably aren't in the mood to spend time taking a tour of the city or learning about its history. Further, your motivation depends on the importance of the task. If you are trying to find your hotel room, you will do whatever it takes. However, if your goal is to find a place of minor importance, most likely you will just give up after a modest effort.

All of these aspects of finding your way in real space correspond to how users typically find their way in the virtual space of your program. Seeking help beyond the primary UI is by its very nature disorienting; do your best to mitigate such an experience by well-designed UI and intelligent "street signs" to direct users to the answers they need.

Designing UI so that Help is unnecessary

Try to make Help unnecessary in the first place, by:

- Making common tasks easy to discover and perform.
- Providing clear main instructions.
- Providing clear, concise control labels that are goal- and task-oriented.
- Providing supplemental instructions and explanations where needed.
- Anticipating avoidable problems by using controls constrained to valid choices, providing suitable default values, handling all input formats, and preventing errors.
- Writing error messages that provide a clear solution or action for the user to take.
- Avoiding confusing UI designs, such as tasks with poor flow or using controls that are disabled for no apparent reason.
- Working with writers and editors early in the development cycle to create high-quality, consistent UI text throughout your program.

Users shouldn't have to go somewhere else to figure out how to use your UI. Add essential information directly into the primary UI, rather than forcing users out of their immediate context and into the Help pane. If important information exists only in a Help

topic, there's a good chance that users won't see it. For information that is optional and more explanatory, use Help links from the primary UI to the relevant Help topic for supplemental assistance.

Considering user motivation

For most users, speed and efficiency are among the paramount virtues of good programs. Users want to get their work done. Generally, they are not interested in learning about the program and the technology for its own sake; their patience extends only insofar as that program serves their own interests and solves problems at hand.

Design your Help system to match your users' motivation. For example, consider a user who has strolled up to a kiosk at a museum. If she cannot figure out how to perform the task quickly, she is likely just to give up and walk away. She is unlikely to spend time using Help. Alternatively, a highly-motivated user has a higher tolerance for time spent researching your Help system for answers. A business user who must balance the books, for example, is probably willing to consult Help content to get the most out of that new accounting application.

Writing content for scanning

Write Help topics knowing that they will be scanned for specific information, not read word-for-word. Write concisely, getting to the point quickly, and providing information that users can act on.

- Write "how-to" topics using numbered steps in a consistent format so that users recognize they are getting procedural assistance.
- Write reference topics with ease-of-scanning in mind, using tables, for example, to present UI options or language syntax.
- Write conceptual topics that are logically organized by subheadings, so that the user can skip whole sections of lesser interest.

In all Help content, it is easier to scan bulleted lists than standard paragraph blocks of text; use bullet lists judiciously, though, not as a crutch for unorganized material.

Creating content that matters

Given that no Help system can anticipate every question that every user might have, focus most of the content on answering the top questions in the top scenarios for your target users. For example, effective searching and how to establish network connectivity (among other tasks) may be highly sought-after topics. Also, focus on tasks within your

top user scenarios, rather than documenting a feature or technology exhaustively for its own sake.

Tip: Technical support is a good source for Help content. Help desks often keep records of frequently asked questions about particular programs or tasks that users are trying (and failing) to accomplish.

It isn't necessary to provide help for every feature in the UI. **Quite often, unhelpful Help results from trying to create Help for everything.** If the UI is well designed, most of these Help topics won't be very helpful; they will just restate the obvious.

If there is more than one way to perform a task, in most cases you can just document the most common way used by inexperienced users. Exceptions to this include accessibility considerations (documenting keyboard equivalents of mouse actions, for example), and platform considerations (documenting for the tablet form factor, for example, or for server environments in which the command line can substitute for the graphical user interface).

Remember that users often don't think of the problems they are encountering in exactly the same terms as you do. For example, users may find it strange to think of themselves as an "account." Be sure to design your search and indexing functionality, then, to account for likely terminology variations and synonyms.

Between the primary UI and the Help system, though, terms should be very similar if not identical. Users can be confused when the Help system language doesn't correlate very closely to what they are seeing on the screen.

Writing compelling Help link text

When you link to a Help topic from your primary UI, be sure to write compelling Help link text. Clear and specific language inspires confidence. Users tend to believe that generic Help links (a button with the word "Help" or "Learn more" on it) will not lead to the right information without a significant investment of time.

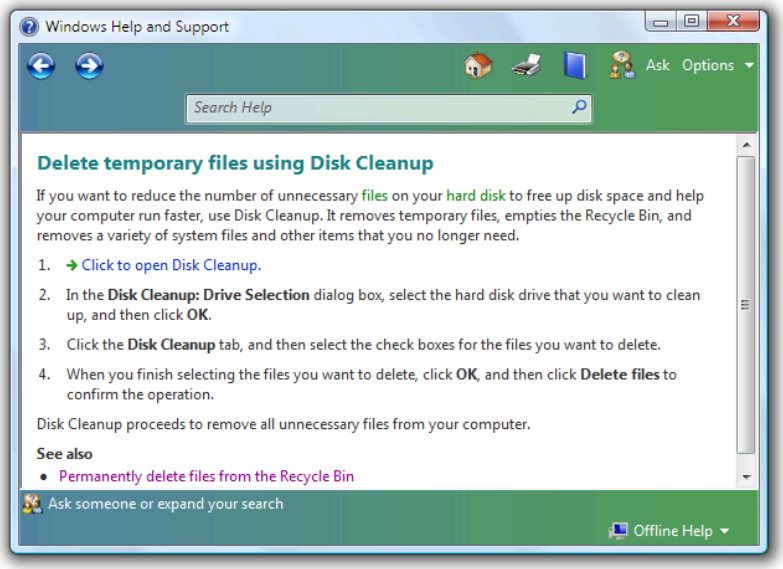
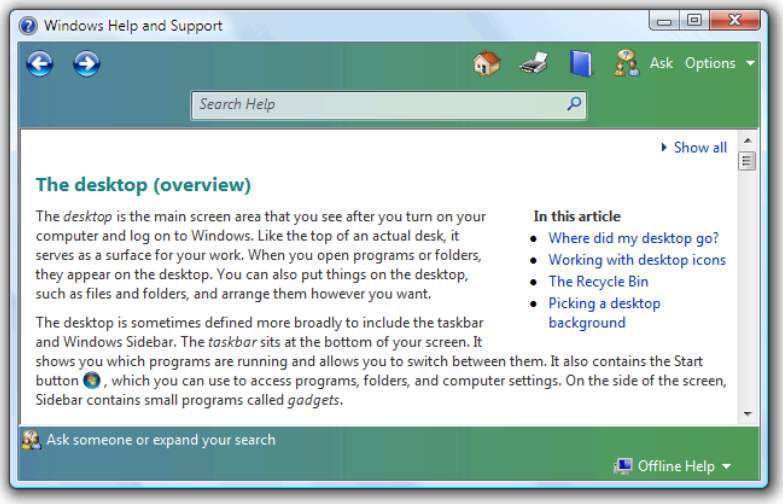
If you do only five things...

1. Design your UI so that users don't need Help.
2. Make your Help helpful by focusing the content on the top questions in the top scenarios for your target users.
3. Present the Help content so that it is easy to scan.
4. Understand that you don't have to provide help for every feature in the UI.
5. Make the Help entry points discoverable and compelling.

Usage patterns

Different kinds of content serve different purposes.

Expand table

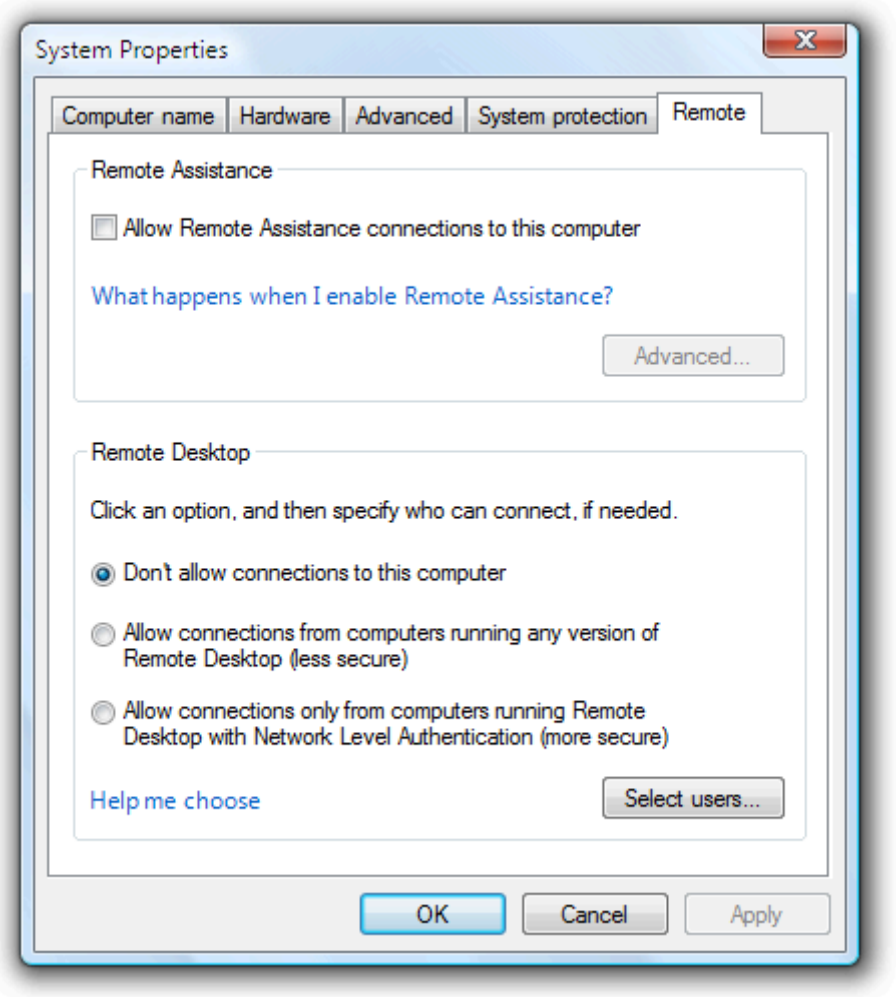
Content Type	Example
Procedural Help provides the steps for carrying out a task.	Procedural help should focus on "how" information rather than "what" or "why."  In this example, the Help topic describes how to use a feature of the Disk Cleanup utility, providing steps to follow in sequence.
Conceptual Help provides background information, feature overviews, or processes.	Conceptual help should provide "what" or "why" information beyond that needed to complete a task.  In this example, the Help topic defines what the desktop is, and provides additional detail about what it contains and why users interact with it.
Reference Help serves as an online reference book.	You can use reference help to document a programming language or programming interfaces.

Content Type	Example
	Notational Conventions **Convention** / **Meaning** **bold** — In syntax, characters that you type exactly as shown, including commands and parameters. In text, menu names and menu commands are also bold. bold monospace — Commands that you must type exactly as shown to get the results being discussed. *italic* — Variables for which you supply a specific value. For example, *Filename.ext* represents any valid file name. Initial Capitals (Filename.ext) — Names of files should begin with an initial capital letter, for example, Filename.ext. Paths and folders can be uppercase, lowercase, or mixed, according to how they actually appear in a standard installation of the application or the operating system. ALL CAPITALS — Used for acronyms. In this example, the Help topic lists typographic conventions in use for this particular language or application, providing the information in an easy-to-scan table.

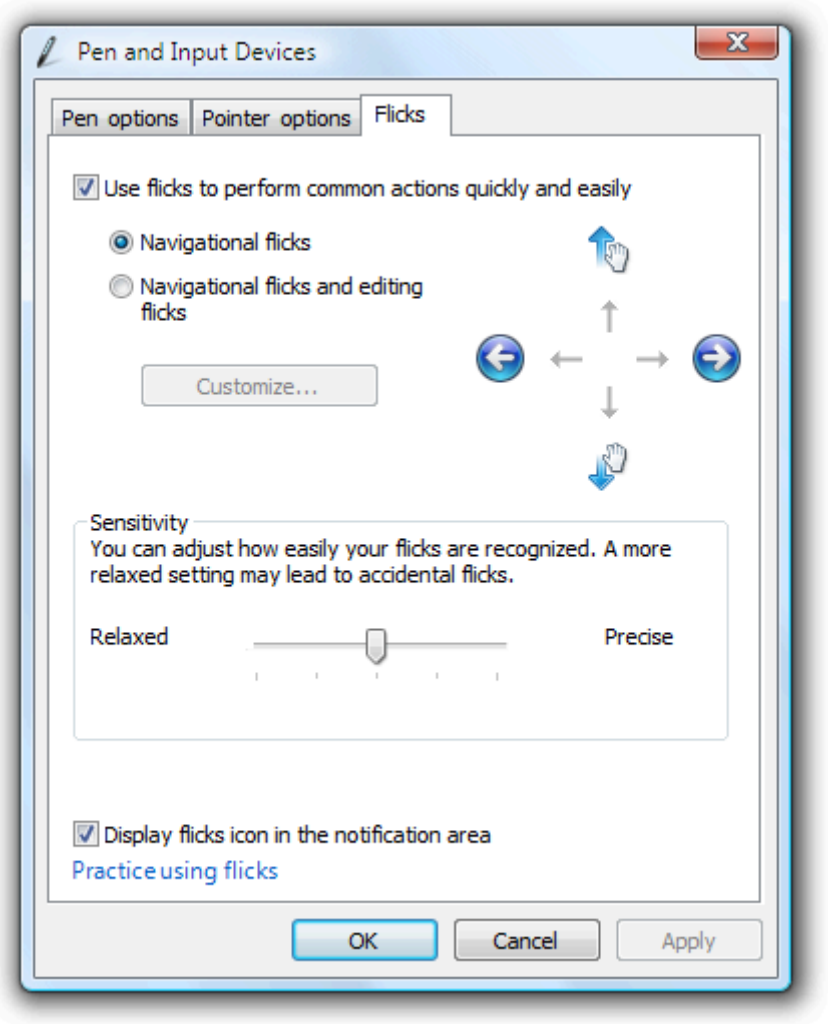
Guidelines

Entry points

- **Link to specific, relevant Help topics.** Don't link to the Help home page, the table of contents, a list of search results, or a page that just links to other pages. Avoid linking to pages structured as a large list of frequently asked questions, because it forces users to search for the one that matches the link they clicked. Don't link to specific Help topics that aren't relevant and helpful to the task at hand. Never link to empty pages.
- **Don't put Help links on every window or page for the sake of consistency.** Providing a Help link in one place doesn't mean that you have to provide them everywhere.
- **Use Help links for dialog boxes, error messages, wizards, and property sheets.** If the Help link applies to specific controls, place it under them, left-aligned. If the Help link applies to the entire window, place it at the bottom left corner of the window's content area.



In this example, the second Help link applies to a group of controls.



In this example, the Help link applies to the entire window.

- **Use Help links instead of generic textual references to Help whenever technically possible.**

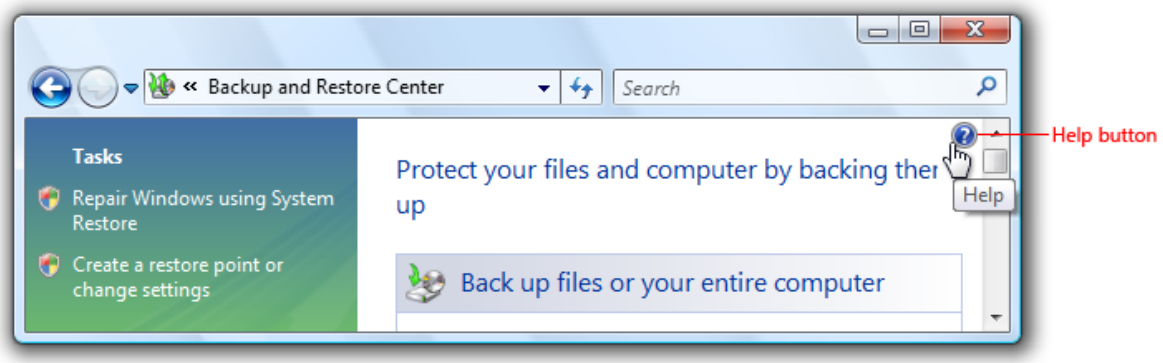
  Correct:

  How can I repair disk errors?

  Incorrect:

  For more information about repairing disk errors, see Help and Support.

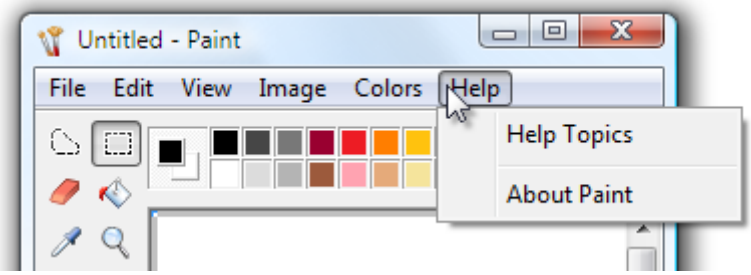
- **Use a Help button with the Help icon for the hub pages of control panel items.** Place it in the upper-right corner. These buttons don't have a label, but have a tooltip that reads Help.



A control panel item with a Help button.

- **F1 Help is optional.** Users have grown accustomed to finding Help information related to the immediate context of the UI on the screen by pressing the F1 key, which is labeled Help on standard keyboards. You can include F1 Help if, for example, usability studies show that your users expect to find it, or your program UI is complex enough to benefit from contextual assistance.

- **Programs with menu bars can have a Help menu category.** For Help menu guidelines, see Menus.



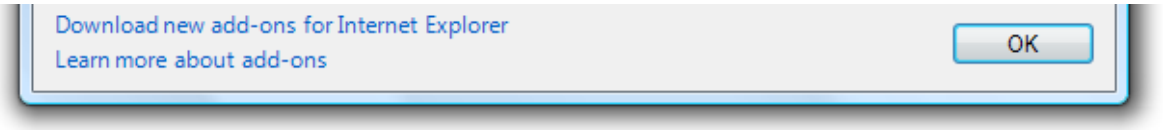
In this example, the Windows Paint accessory has a Help menu category.

- **For keyboard accessibility, provide tab stops for Help buttons and links.**

- Help button and link behavior should be as follows: Help pane opens and a dedicated Help topic is displayed; the UI that invoked the Help pane should remain open to preserve the contextual experience.

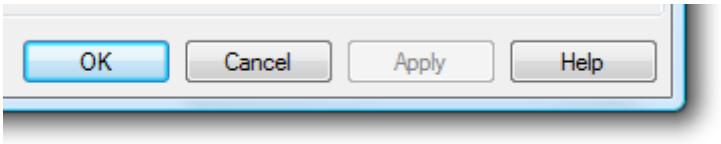
- **Don't use the following obsolete Help entry point styles: "Learn more" or "Learn more about..." links, generic Help buttons, and context-sensitive Help buttons on the title bar.** Although they have been used in the past, usability studies have determined that users tend to ignore them. Use links to specific Help topics instead. For guidelines about writing good Help links, see Help links.

  Incorrect:



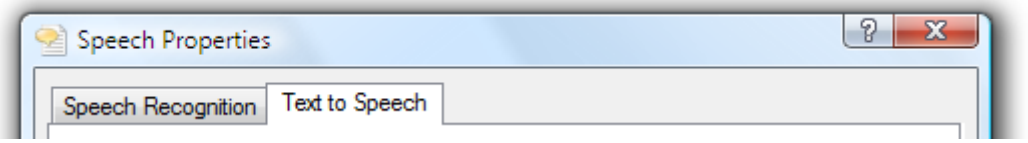
Don't use "Learn more" or "Learn more about..." links.

Incorrect:



Don't use generic Help buttons.

Incorrect:



Don't use context-sensitive Help buttons on the title bar.

Content

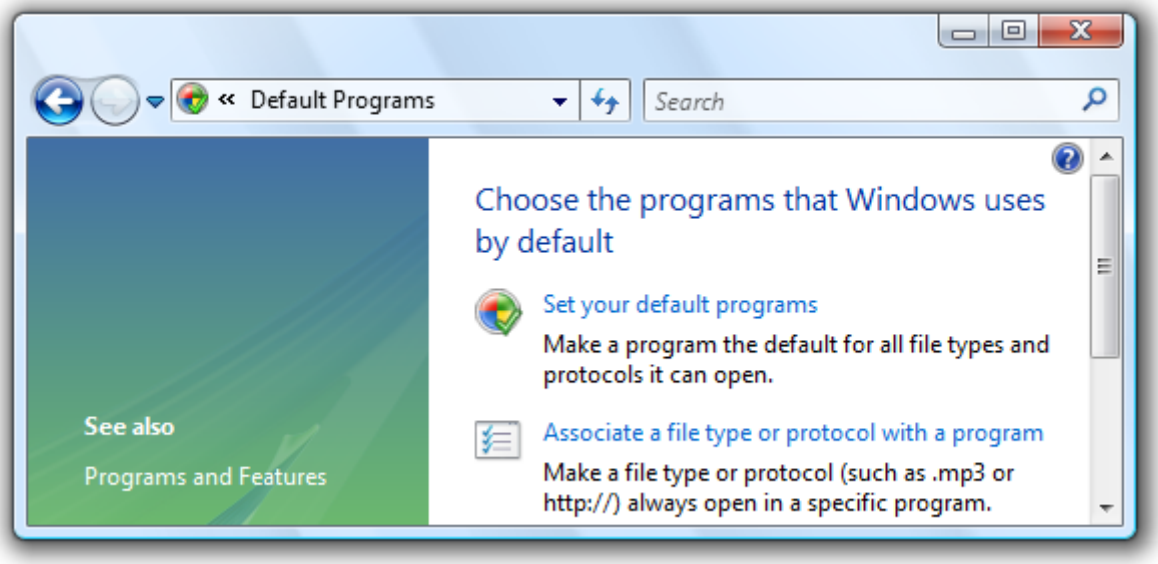
- **Don't create obvious content.** Help topics that repeat what is in the primary UI don't add value.
- **Don't create content that the user can't act on in some way.**
  - **Exception:** Some conceptual content delivers important background information without necessarily leading to user action.
- **Avoid vague resolutions to problems.** For example, "contact your system administrator" or "reinstall the application" tend to frustrate users.
  - **Exception:** Recommend contacting the system administrator if that is the only practical solution, and system administrators expect to be contacted for the problem.
- **Avoid content that addresses highly unlikely user scenarios.** Develop your main Help content for what you anticipate will be normal usage; note important exceptions to expected usage, but treat this content as a lower priority.
- **Gather feedback from your users about how helpful your Help topics are.** Allow users to rate individual topics. Conduct usability studies on your documentation to ascertain problems involving quality and discoverability of content.
  - **Tip:** User feedback is also a great way to generate more task-based content, focused on what users are actually doing with your program, as opposed to feature-based content, focused simply on a description of the technology.

- **Provide multiple ways of accessing your content.** A table of contents, an index, and a search mechanism are three of the most common methods of improving discoverability.
- **If there is more than one way to perform a task, in most cases you can just document the most common way used by inexperienced users.**

Icons

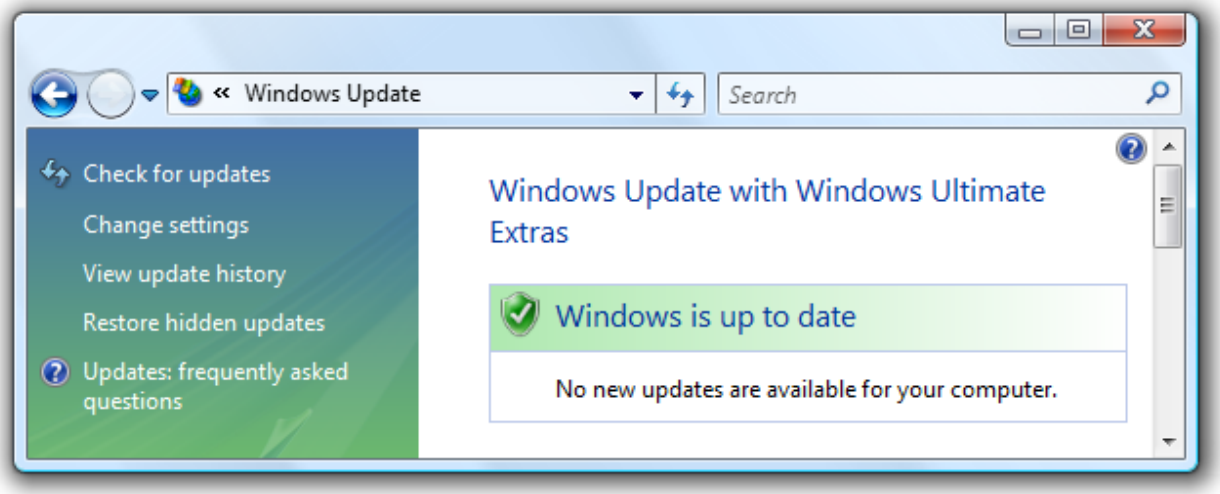
- Use the Help icon only for Explorer windows and the hub pages of control panel items. Don't use the Help icon with Help links.

Correct:



In this example, a Windows Explorer window uses a Help icon to provide access to Help.

Incorrect:



In this example, the Help icon on the lower-left is used incorrectly with a Help link.

Text

Help links

- Provide specific information about the content of the Help topic, using as much relevant, concise text as necessary. Users often ignore generic Help links. Make sure the results of the link are predictable users shouldn't be surprised by the results.
  - **Exception:** You can use "More information" to supplement instructions that are directly in the UI, especially if providing specific information in the Help link leads to unnecessary repetition or makes the link less compelling.

  Incorrect:

  A strong password has at least six mixed-case letters, numbers, and symbols. What is a strong password?

  Correct:

  A strong password has at least six mixed-case letters, numbers, and symbols. More information

  In the incorrect example, the Help link is repetitive. It asks a question that is really already answered.

- Whenever possible, phrase Help link text in terms of the primary question answered by the Help content. Don't use "Learn more about," "Tell me more about," or "Get help with this" phrasing.

  Incorrect:

  Learn more about adding exceptions

  Correct:

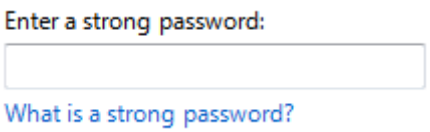
  What are the risks of allowing exceptions?

  How do I add exceptions?

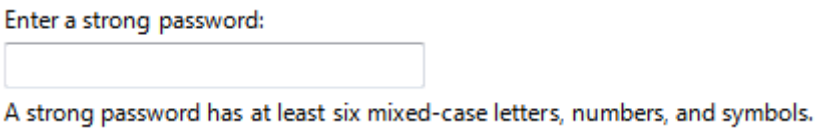
  In the correct examples, the link is phrased in terms of the primary question answered by the Help topic.

- If the most relevant information can be summarized succinctly, put the summary directly in the UI instead of using a Help link. However, you can use a Help link to provide supplemental information.

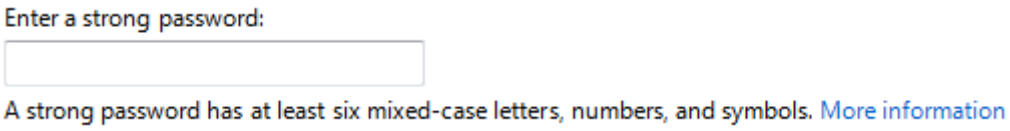
Incorrect:



Correct:



Better:



The correct example summarizes the Help information succinctly, greatly improving the likelihood that users will read it. The better example provides a Help link for more information on this complex subject.

- Phrase Help links to clearly indicate assistance. Help links should never read like action links.
- Use the entire Help link for the link text, not just the keywords.

  Correct:

  What are the risks of allowing exceptions?

  Incorrect:

  What are the risks of allowing exceptions?

  In the correct example, the entire Help link sentence is used for the link text.
  - **Exception:** Help links to external Web sites should simply use the name of the site or page as the link. Any text introducing the name of the site need not be included in the link itself.
- Help links don't have to match Help topic headings exactly, but there should be a strong and obvious connection between the two. Design links and headings in pairs for this reason.

  Correct:

How can I improve the performance of this feature? (link text)

Configuring this feature for optimal performance (topic heading)

Incorrect:

How can I improve the performance of this feature? (link text)

Complete overview of this feature (topic heading)

In the incorrect example, the Help topic heading substantially differs in scope from the Help link text, and could be disorienting.

- If the Help content is online, make that clear in the link text. Doing so helps make the result of the links predictable.

  Correct:

  For additional formats and tools, go to the Microsoft Web site.

  Incorrect:

  Where can I find additional formats and tools?

- Use complete sentences.
- Don't use ending punctuation, except for question marks.
- Don't use ellipses for Help links or commands.

Help content

- Format UI elements using bold to make them easy to identify. This is especially useful for procedural Help topics, allowing users to scan through procedures and quickly see pertinent UI elements.
- Format captions using italic. This applies to tables, art, screenshots, and other graphic elements that benefit from brief textual explanation.
- Refer to Help simply as Help. Generally, don't use the phrase online Help unless you are in fact referring to content on your Web site.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

User Account Control

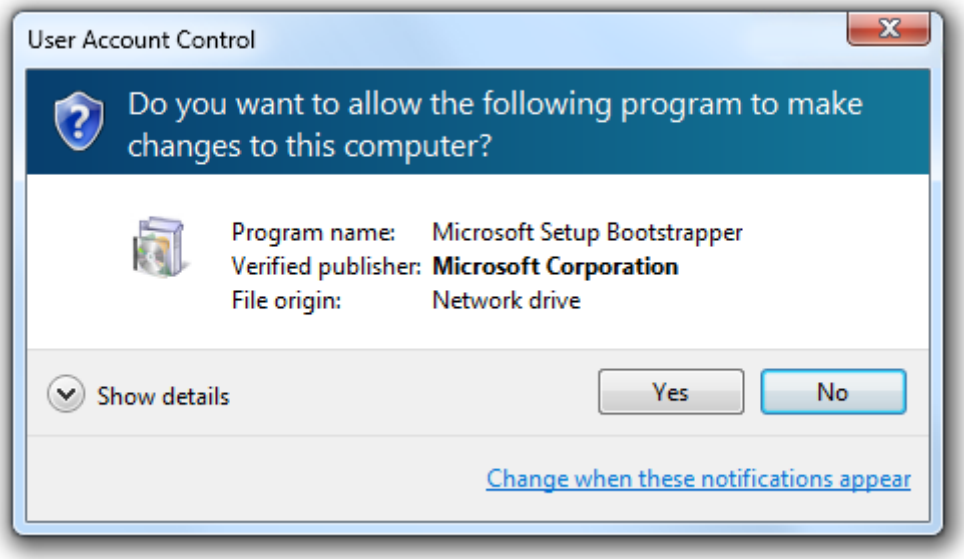
Article • 06/03/2021

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

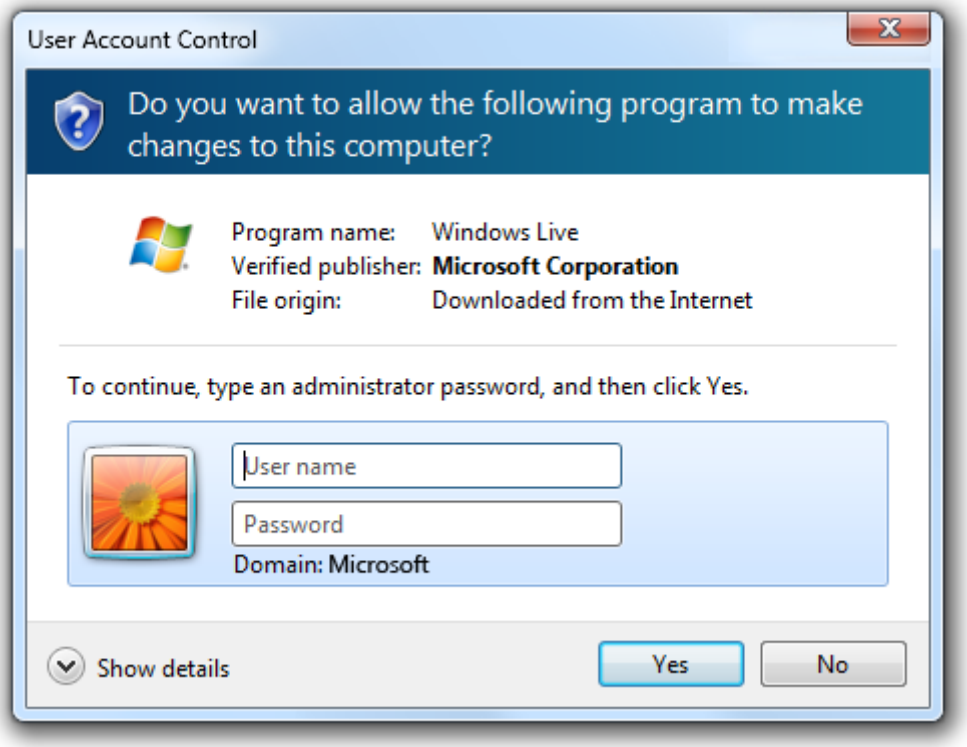
A well designed User Account Control experience helps prevent unwanted system-wide changes in a way that is predictable and requires minimal effort.

With User Account Control (UAC) fully enabled, interactive administrators normally run with least user privileges, but they can self-elevate to perform administrative tasks by giving explicit consent with the Consent UI. Such administrative tasks include installing software and drivers, changing system-wide settings, viewing or changing other user accounts, and running administrative tools.

In their least-privileged state, administrators are referred to as Protected administrators. In their elevated state, they are referred to as Elevated administrators. By contrast, Standard users can't elevate by themselves, but they can ask an administrator to elevate them using the Credential UI. The Built-in Administrator account doesn't require elevation.



The Consent UI, used to elevate Protected administrators to have administrative privileges.



The Credential UI, used to elevate Standard users.

UAC provides the following benefits:

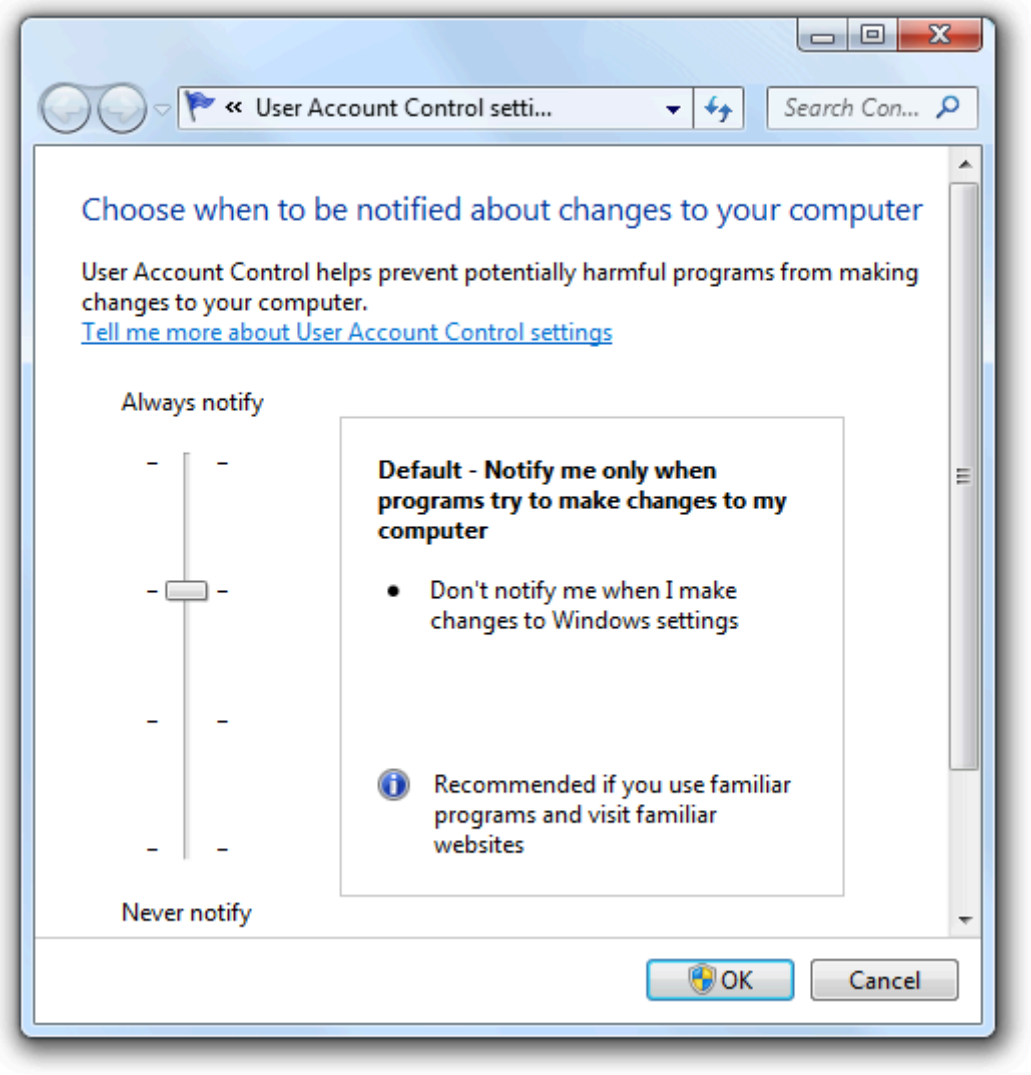
- It reduces the number of programs that run with elevated privileges, therefore helping to prevent users from accidentally changing their system settings, and helping to prevent "malware" from gaining system-wide access. When elevation is denied, malware is only able to affect the current user's data. Without elevation, malware can't make system-wide changes or affect other users.
- For managed environments, well designed UAC experiences allow users to be more productive when running as Standard users by removing unnecessary restrictions.
- It gives Standard users the ability to ask administrators to give them permission to perform administrative tasks within their current session.
- For home environments, it enables better parental control over system-wide changes, including what software is installed.

Developers: For implementation information, see Redesign Your UI for UAC Compatibility.

In Windows Vista, Protected administrators can choose to be notified about all system changes or none. The UAC default setting is to notify about all changes, no matter what their origin. When you're notified, your desktop will be dimmed, and you must either approve or deny the request in the UAC dialog box before you can do anything else on your computer. The dimming of your desktop is referred to as the secure desktop because other programs can't run while it's dimmed.

Windows 7 introduces two intermediate UAC settings for Protected administrators, in addition to the two from Windows Vista. The first is to notify users only when a program is making the change, so administrators are automatically elevated when they make a change themselves. This is the UAC default setting in Windows 7, and it also makes use of the secure desktop.

The second intermediate setting in Windows 7 is the same as the first except that it doesn't use the secure desktop.



Windows 7 introduces two intermediate UAC settings.

Note: Guidelines related to writing code to support User Account Control are presented in a separate article.

Design concepts

Goals

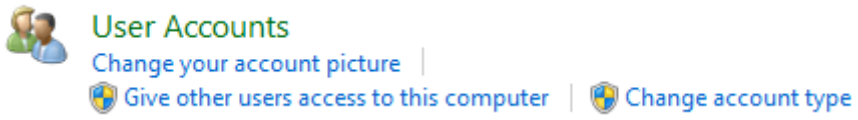
A well designed User Account Control experience has the following goals:

- **Eliminate unnecessary elevation.** Users should have to elevate only to perform tasks that require administrative privileges. All other tasks should be designed to eliminate the need for elevation. Often legacy software requires administrator privileges unnecessarily by writing to the HKLM or HKCR registry sections, or the Program Files or Windows System folders.
- **Be predictable.** Standard users need to know which tasks require an administrator to perform or cannot be performed at all in managed environments. Administrators need to know which tasks require elevation. If they can't predict the need for elevation accurately, they are more likely to give consent for administrative tasks when they shouldn't.
- **Require minimal effort.** Tasks that require administrative privileges should be designed to require a single elevation. Tasks that require multiple elevations quickly become tedious.
- **Revert to least privileges.** Once a task that requires administrative privileges is complete, the program should revert to the least privilege state.

Elevation task flow

When a task requires elevation, it has the following steps:

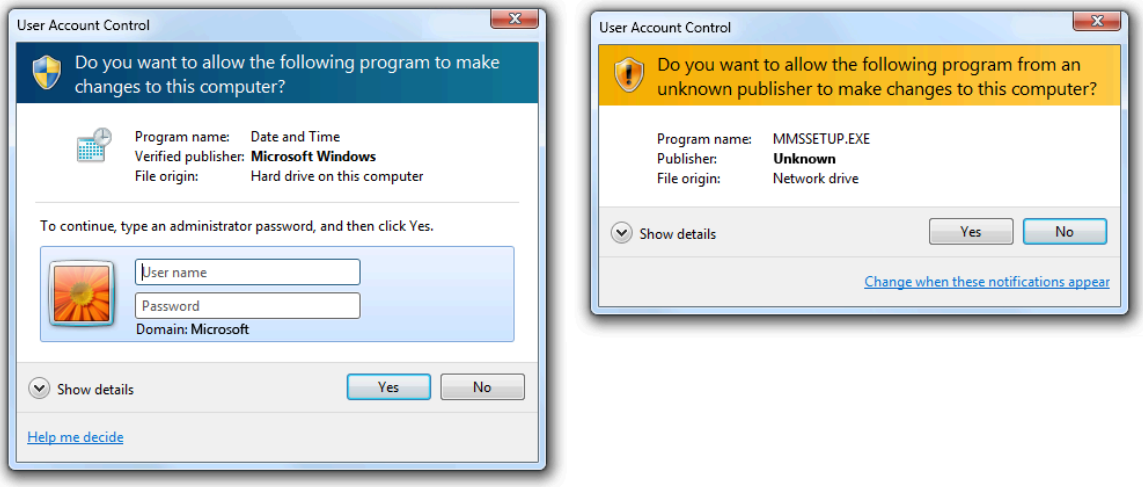
1. **Entry point.** Tasks that require immediate elevation when UAC is fully enabled have entry points marked with the UAC shield. In this case, users should expect to see an Elevation UI immediately after clicking such commands and they should be extra cautious when they see Elevation UI from tasks that don't have a shield.

   In this example, the parental control and user accounts control panel items require elevation.

   When UAC is partially enabled or turned off completely, the UAC shield is still displayed to indicate that the task involves system-level changes and therefore requires elevation, even if the user might not see Elevation UI. Always displaying the UAC shield for tasks that require elevation keeps the UI simple and predictable.

2. **Elevation.** For Protected Administrators, the task requests consent using the Consent UI. For Standard users, the task requests administrator credentials using the Credential UI.



These examples show the Credential UI and the Consent UI.

3. **Separate elevated process.** Internally, a new elevated process is created to perform the task.

4. **Revert to least privilege.** If necessary, revert to least privilege to complete any steps that don't require elevation.

Note that tasks don't "remember" elevated states. For example, if the user navigates back and forth over an elevation entry point in a wizard, the user must elevate each time.

Usage patterns

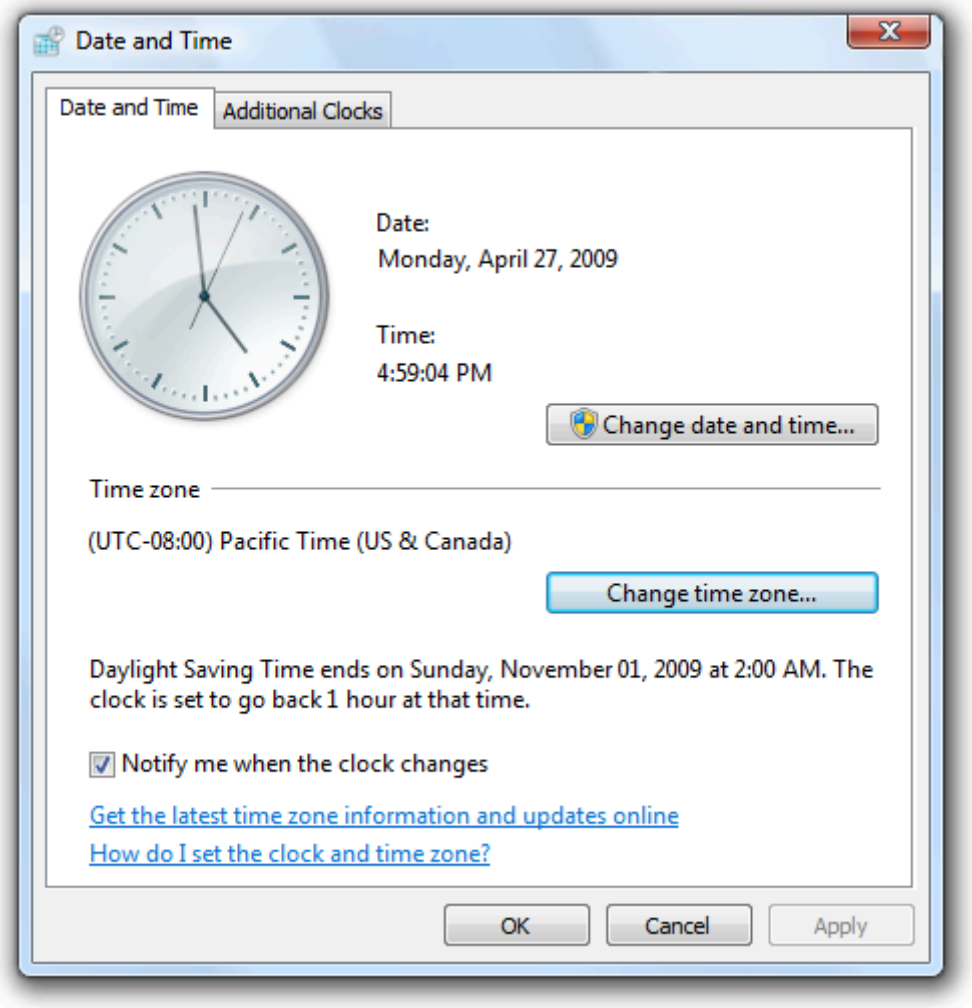
User Account Control has several usage patterns (in order of preference):

1. **Work for Standard users.** Design the feature for all users by limiting its scope to the current user. By limiting settings to the current user (as opposed to system-wide), you eliminate the need for an Elevation UI entirely, and enable users to complete the task.

   Incorrect:

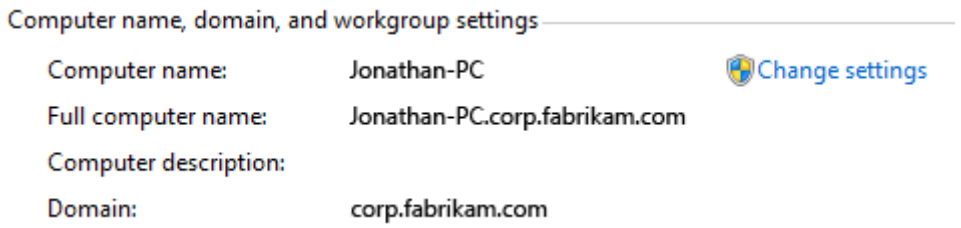
   In this example, Windows XP users had to have administrative privileges to view or change the current time zone.

Correct:



In this example, the time zone feature was redesigned in Windows 7 and Windows Vista to work for all users.

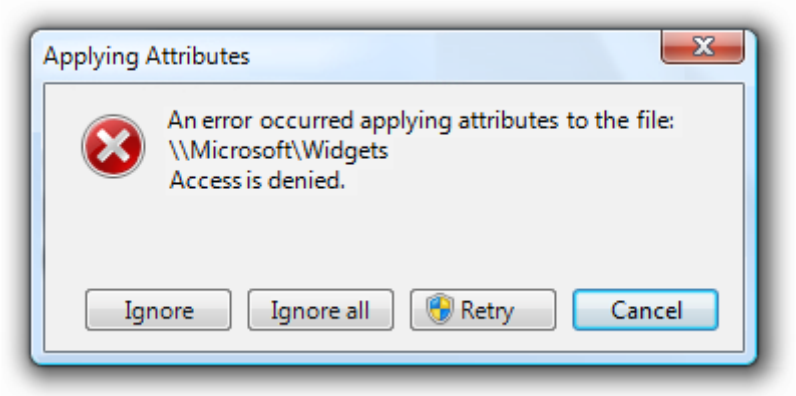
2. **Have separate UI elements for Standard users and administrators.** Clearly separate Standard user tasks from administrative tasks. Give all users access to useful read-only information. Clearly identify administrative tasks with the UAC shield.



In this example, the System control panel item shows its state to all users, but changing the system-wide settings requires elevation.

3. **Allow Standard users to attempt task, and to elevate on failure.** If Standard users can view the information and are able to make some changes without elevation,

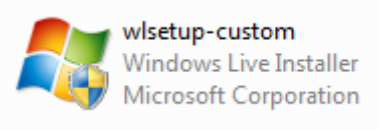
allow them to access the UI and have them elevate only if the task fails. This approach is suitable when Standard users have limited access, such as with properties of their own files in Windows Explorer. It is also suitable for settings on Control Panel hybrid hub pages.



In this example, the user attempted to change program file properties but didn't have sufficient privileges. The user can elevate and try again.

4. **Work for administrators only.** Use this approach only for administrator features and programs! If a feature is intended only for administrators (and has no navigation paths or useful read-only information for Standard users), you can prompt for administrator credentials at the entry point before showing any UI. Use this approach for lengthy wizards and page flows when all paths require administrative privileges.

   If the entire program is for administrators only, mark it to prompt for administrator credentials in order to launch. Windows displays such program icons with the UAC shield overlay.

   In this example, the program requires administrative privileges to launch.

Guidelines

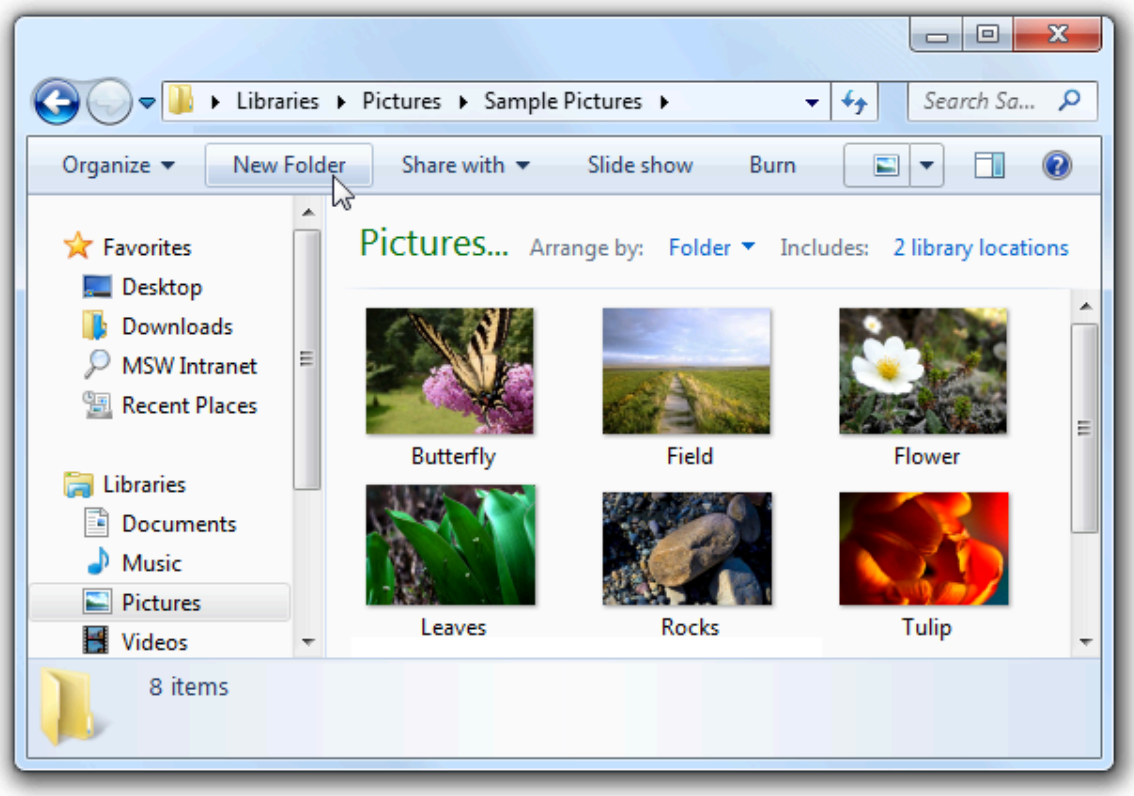
UAC shield icon

- **Display controls with the UAC shield to indicate that the task requires immediate elevation when UAC is fully enabled,** even if UAC isn't currently fully enabled. If all paths of a wizard and page flow require elevation, display the UAC shield at the

task's entry point. Proper use of the UAC shield helps users predict when elevation is required.

- **If your program supports multiple versions of Windows, display the UAC shield if at least one version requires elevation.** Because Windows XP never requires elevation, consider removing the UAC shields for Windows XP if you can do so consistently and without harming performance.

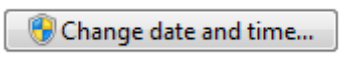
- **Don't display the UAC shield for tasks that don't require elevation in most contexts.** Because this approach will sometimes be misleading, the preferred approach is to use a properly shielded contextual command instead.



Because the New folder command requires elevation only when used in system folders, it is displayed without a UAC shield.

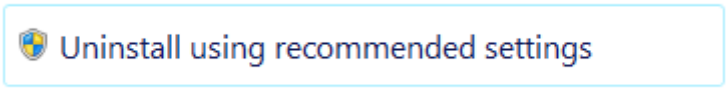
- The UAC shield can be displayed on the following controls:

  Command buttons:

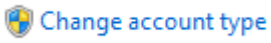
  A command button that requires immediate elevation.

  Command links:



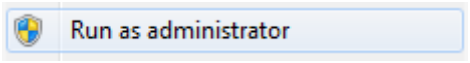
A command link that requires immediate elevation.

Links:



A link that requires immediate elevation.

Menus:



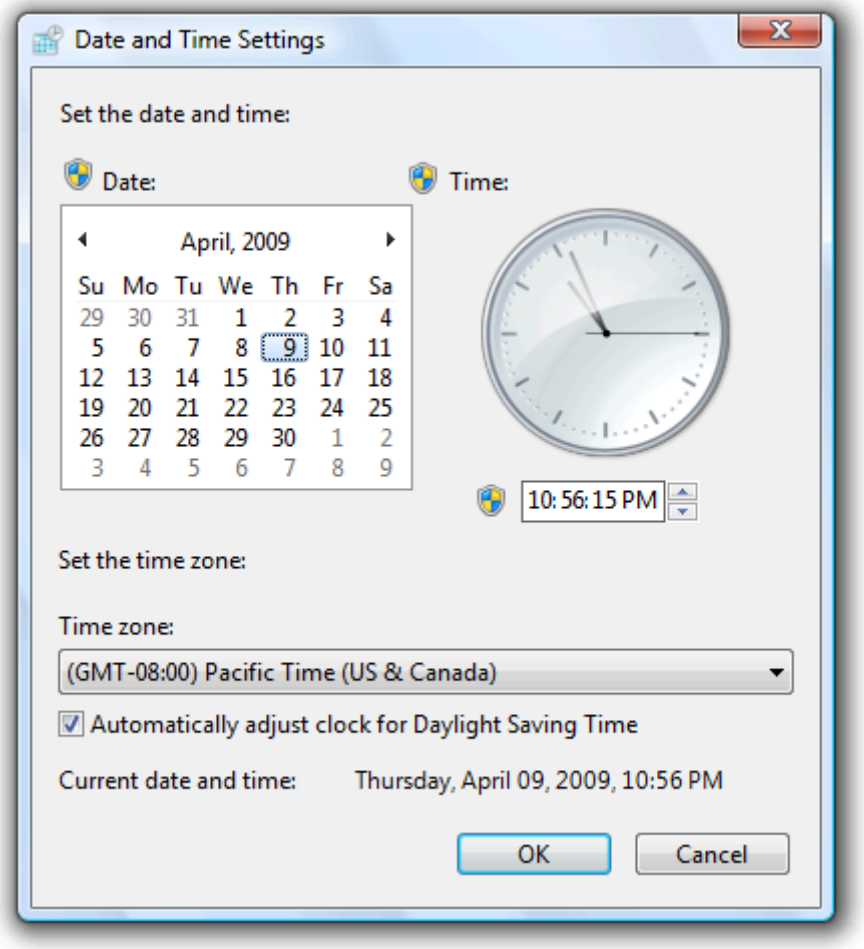
A drop-down menu that requires immediate elevation.

- Because tasks don't remember elevated states, **don't change the UAC shield to reflect state.**
- **Display the UAC shield even if User Account Control has been turned off or the user is using the Built-in Administrator account.** Consistently displaying the UAC shield is easier to program, and provides users with information about the nature of the task.

Elevation

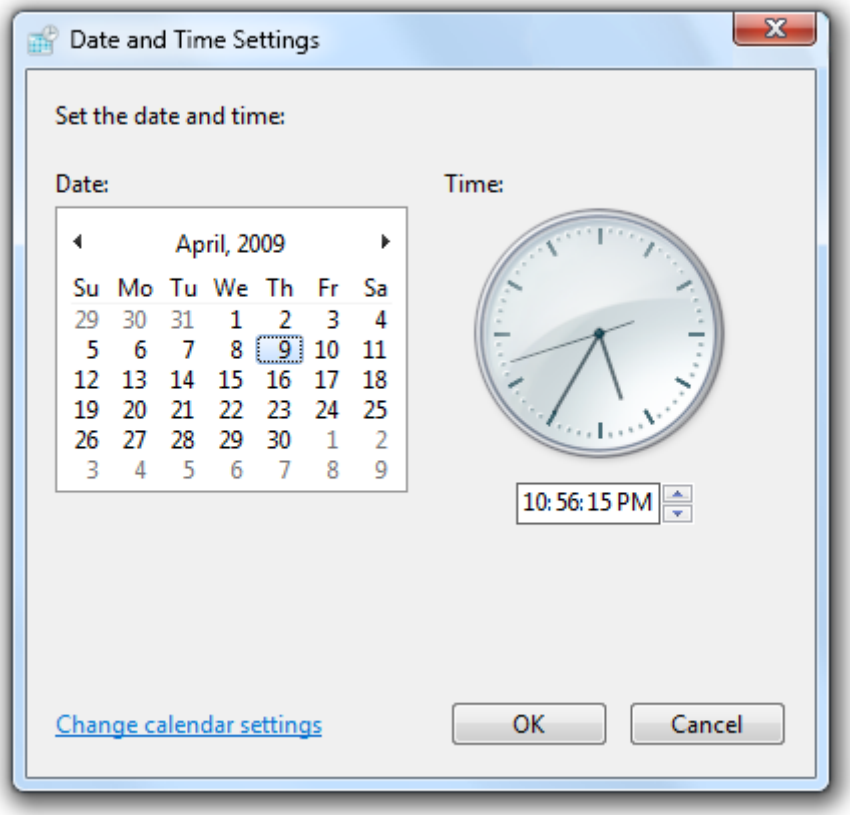
- **Whenever possible, design tasks to be performed by Standard users without elevation.** Give all users access to useful read-only information.
- **Elevate on a per task basis, not on a per setting basis.** Don't mix Standard user settings with administrative settings in a single page or dialog box. For example, if Standard users can change some but not all settings, split those settings out as a separate UI surface.

  Incorrect:



In this example, Standard user settings are incorrectly mixed with administrative settings.

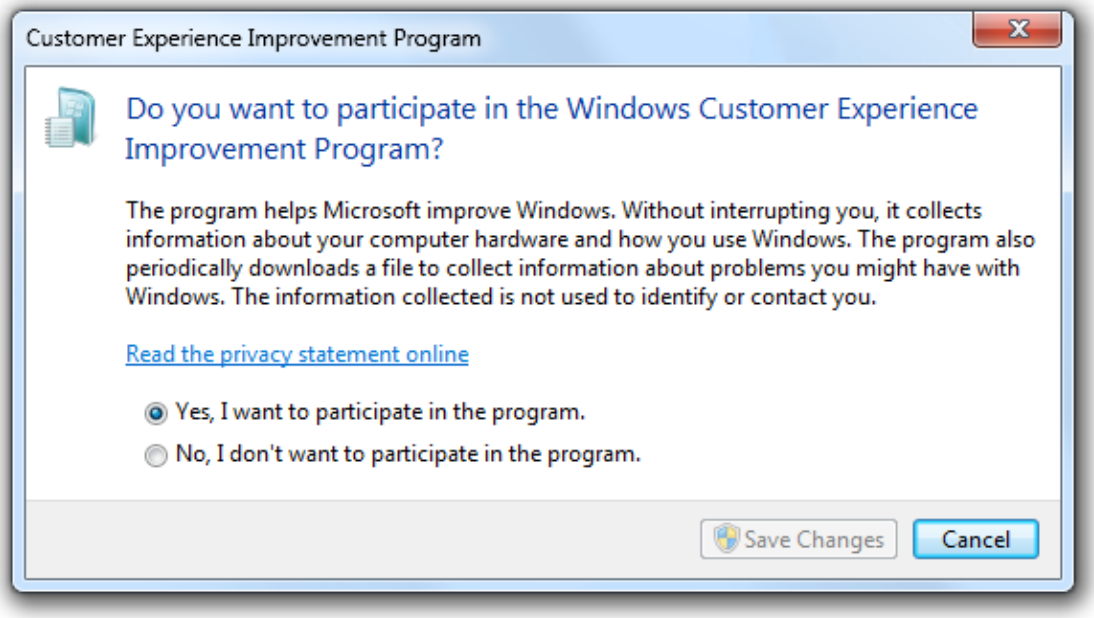
Correct:



In this example, the settings for changing the date and time are in a separate dialog box, available only to administrators. The time zone settings are available to Standard users, and are not mixed with administrative settings.

- **Don't consider the need to elevate when determining if a control should be displayed or disabled.** This is because:
  - In unmanaged environments, assume that Standard users could elevate by asking an administrator. Disabling controls that require elevation would prevent users from having administrators elevate.
  - In managed environments, assume that Standard users can't elevate at all. Removing controls that require elevation would prevent users from knowing when to stop looking.

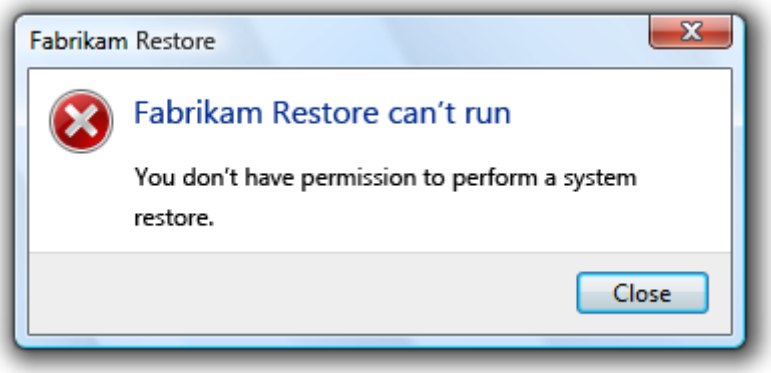
- **To eliminate unnecessary elevation:**
  - **If a task might require elevation, elevate as late as possible.** If a task needs a confirmation, display the elevation UI only after the user has confirmed. If a task always requires elevation, elevate at its entry point.
  - **Once elevated, stay elevated until elevated privileges are no longer necessary.** Users shouldn't have to elevate multiple times to perform a single task.
  - **If users must elevate to make a change but choose not to make any changes, leave the positive commit buttons enabled but handle the commit as a cancel.** Doing so eliminates users having to elevate just to close a window.
  - **Incorrect:**



    - In this example, the Save Changes button is disabled to avoid an unnecessary elevation, but becomes enabled when users change the selection. However, the disabled commit button makes it look like users really don't have a choice.

- **Don't display an error message when tasks fail because users chose not to elevate.** Assume that users intentionally chose not to proceed, so they won't regard this situation as an error.

  Incorrect:

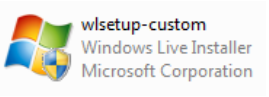
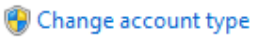
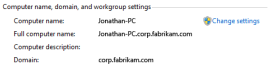
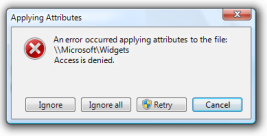
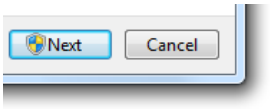
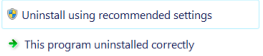
  

  In this example, Fabrikam Restore incorrectly gives an error message when the user decides to not elevate.

- **Don't display warnings to explain that users might need to elevate their privileges to perform tasks.** Let users discover this fact on their own.

- **Display the UAC shield and elevation UI based on the following table:**

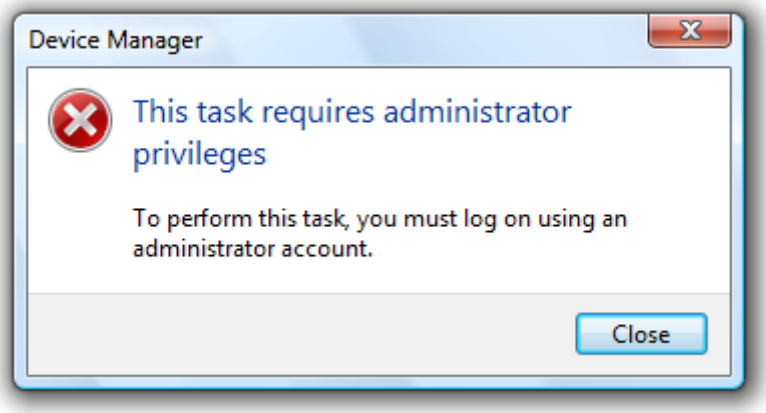
Expand table

Object	Circumstance	Where to put UAC shield	When to elevate
Program	Entire program is for administrators only.	 UAC shield overlay on program icon.	Display elevation UI at launch.
Command	Entire command is for administrators only.	 UAC shield on command button or link.	Display elevation UI when command button or link is clicked, but after any confirmations.
Command	Command displays useful read-only information appropriate for all users, but changes require administrative privilegeges.	 UAC shield on command button or link to make changes.	Display elevation UI when command button is clicked, but after any confirmations.
Command	Standard users can view the information and possibly make some changes without elevation. allow standard users to attempt, and to elevate on failure.	 Don't show the UAC shield for the command, but show it for the elevation entry point if the command fails.	Display elevation UI when user retries the command.
Task step	All subsequent steps require elevation.	 UAC shield on Next button (or equivalent).	Display elevation UI when Next or other commit button is clicked.
Task step	Some branches require elevation.	 UAC shield on command links that require elevation.	Display elevation UI when command links with UAC shield are clicked.

Elevation UI

- **If the user provides an account that isn't valid (name or password) or doesn't have administrator privileges, just redisplay the Credential UI.** Don't display an error message.

- **If the user cancels the Credential UI, return the user back to the original UI.** Don't display an error message.
- If User Account Control has been turned off and a Standard user attempts to perform a task that requires elevation, provide an error message that states "This task requires administrator privileges. To perform this task, you must log on using an administrator account."



In this example, User Account Control has been turned off so an error message explains that the user must use an administrator account.

Wizards

- **Don't elevate multiple times.** Once a wizard is elevated, it should stay elevated.
- If the task is performed within the wizard, put a UAC shield on the Commit page's "Next" button (which should be given a more specific label). When the user commits:
  - If the next page is a Progress page, advance to that page and modally display the elevation UI. After successful elevation, perform the task.
  - If the next page is a Completion page, advance to that page (but temporarily replace its contents with "Waiting for permission...") and modally display the elevation UI. After successful elevation, perform the task, and then display the Completion page contents.
  - If the user cancels the elevation UI, return to the Commit page. Doing so allows the user to try again.
- If the task is performed after the wizard completes, put a UAC shield on the Commit page's "Finish" button (which should be given a more specific label). When the user commits:
  - Remain on the Commit page and modally display the elevation UI. After successful elevation, close the wizard.
  - If the user cancels the elevation UI, return to the Commit page. Doing so allows the user to try again.

- For lengthy wizards intended only for administrators, you can prompt for administrator credentials at the entry point before showing any UI.

Text

- **Don't use an ellipsis just because a command requires elevation.** The need to elevate is indicated with the UAC shield.

Documentation

When referring to User Account Control:

- Refer to the feature as User Account Control (on first mention) or UAC (on subsequent mention), not Least-privileged User Account or LUA.
- Refer to non-administrators as Standard users.
- Refer to built-in computer administrators as Built-in administrators.

In user documentation:

- Refer to the act of giving consent to perform an administrative task as giving permission.

In programming and other technical documentation:

- Refer to the act of giving consent to perform an administrative task as elevation.
- In the context of UAC, refer to administrators as Protected administrators when not elevated, and Elevated administrators after elevation.
- Refer to the dialog box used to enter passwords as the Credential UI. Refer to the dialog box used to give consent as the Consent UI. Refer to both generally as Elevation UI.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Visual Index

Article • 06/03/2021

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

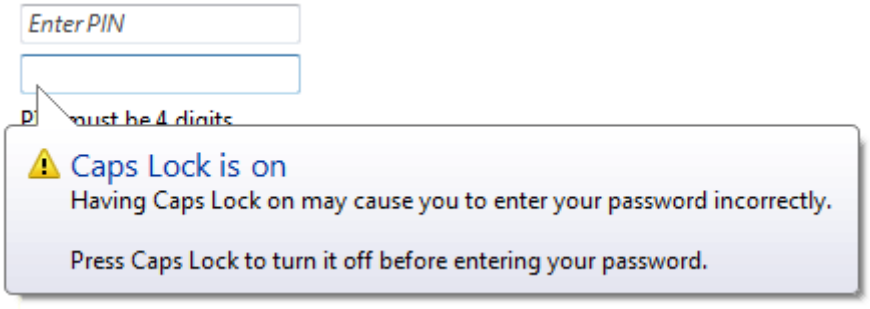
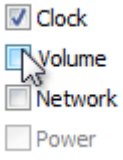
Here are visual examples of many user interface elements discussed in UX Guide.

This index is divided into the following sections:

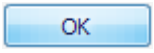
- Controls
- Commands
- Pointers
- Windows
- Windows Environment
- Visuals

Controls

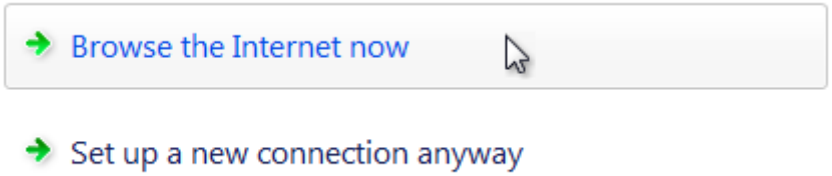
Expand table

balloons 
check boxes 

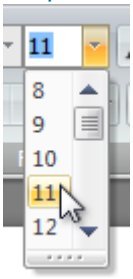
command buttons



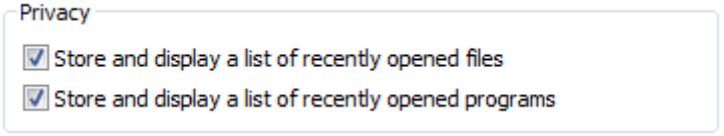
command links



drop-down lists and combo boxes



group boxes

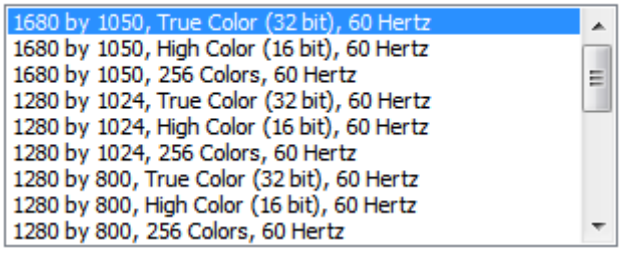


links

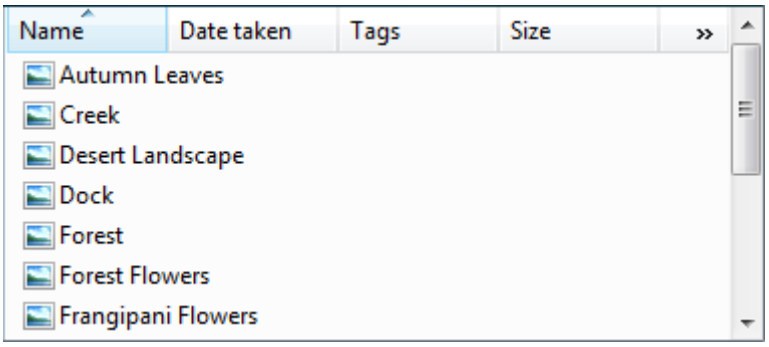
Use the computer without a display

list boxes

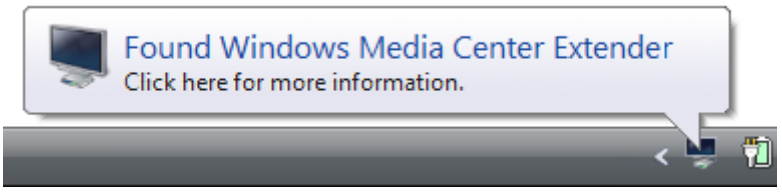
List of valid modes:



list views



notifications

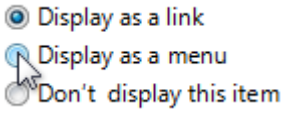


progress bars

from **Pictures** (D:\User...\Pictures) to **Pictures** (D:\User...\Pictures)
Calculating time remaining...

progressive disclosure controls

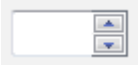
radio buttons



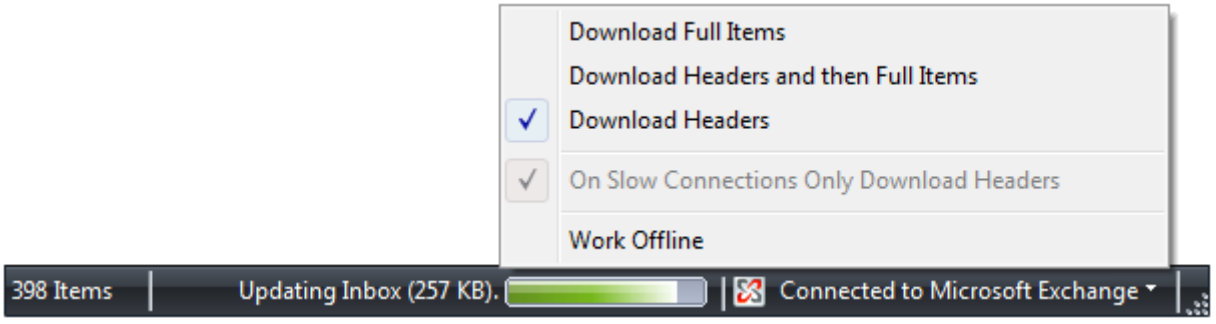
search boxes

sliders

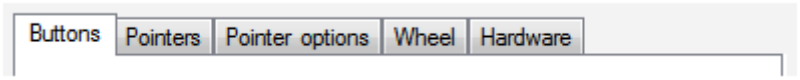
spin controls



status bars



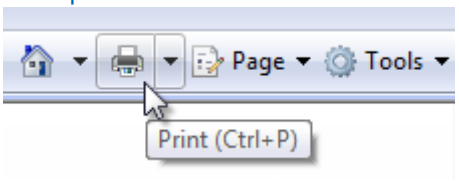
tabs



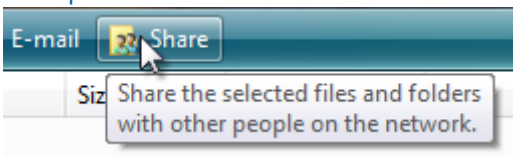
text boxes

Display name:

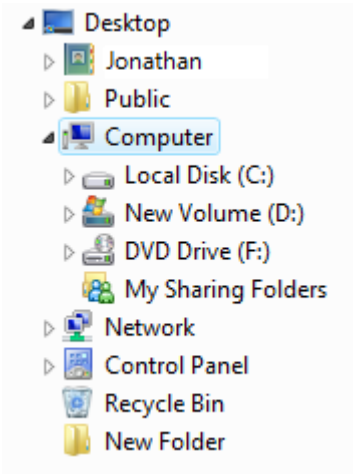
tooltips



infotip



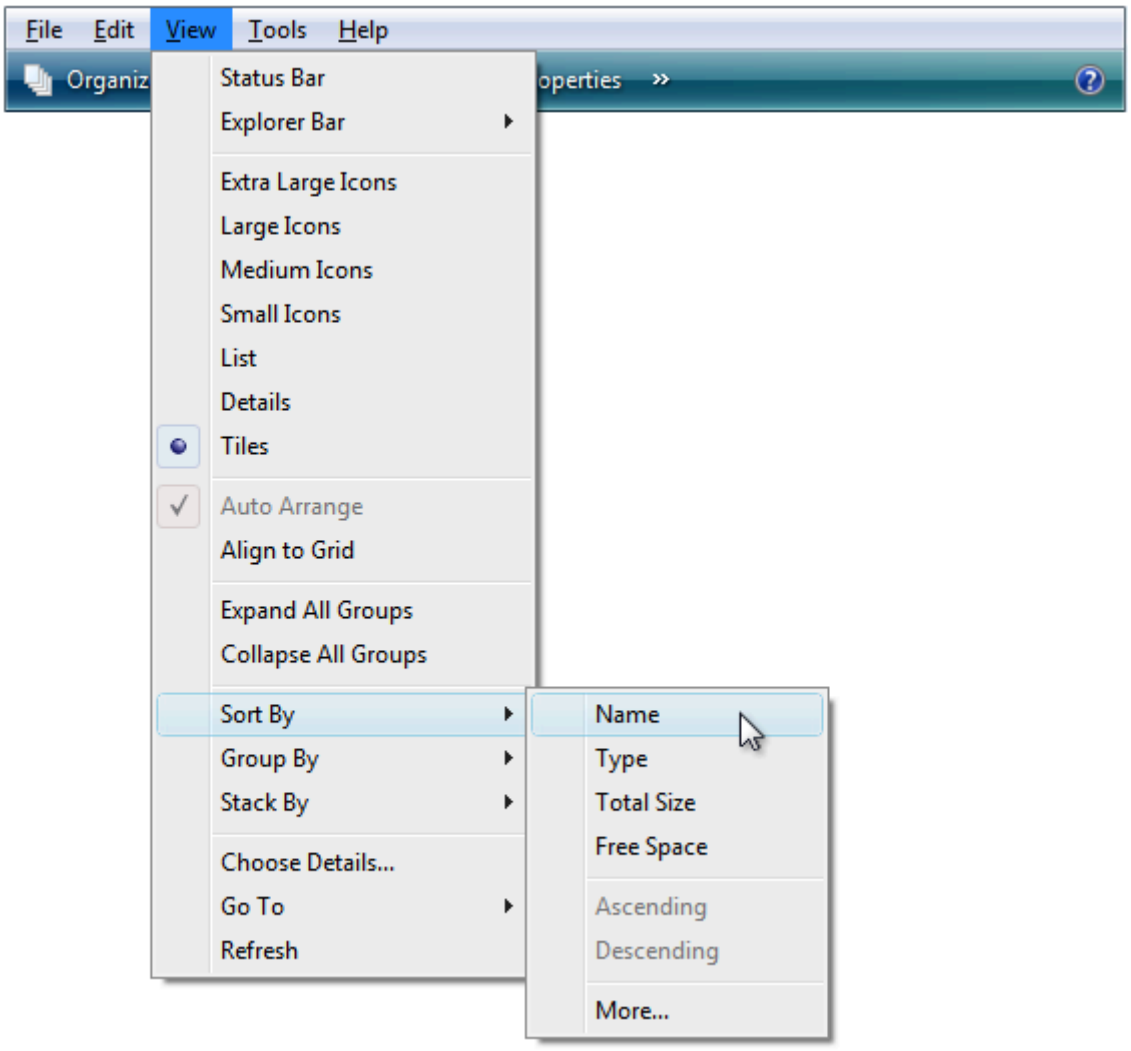
tree views



Commands

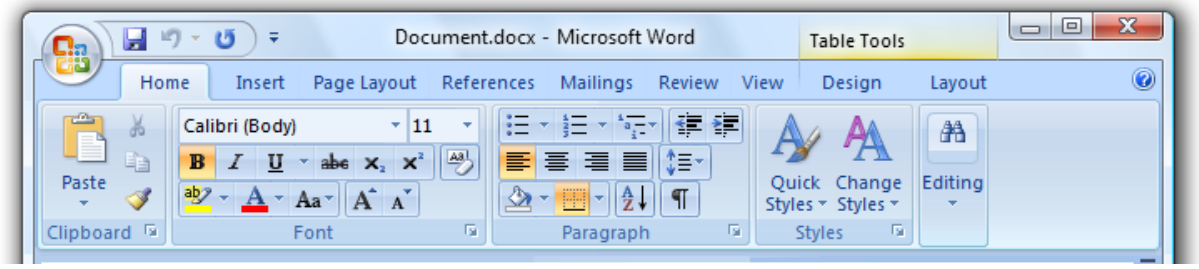
Expand table

menus



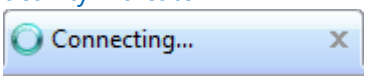
toolbars

ribbons



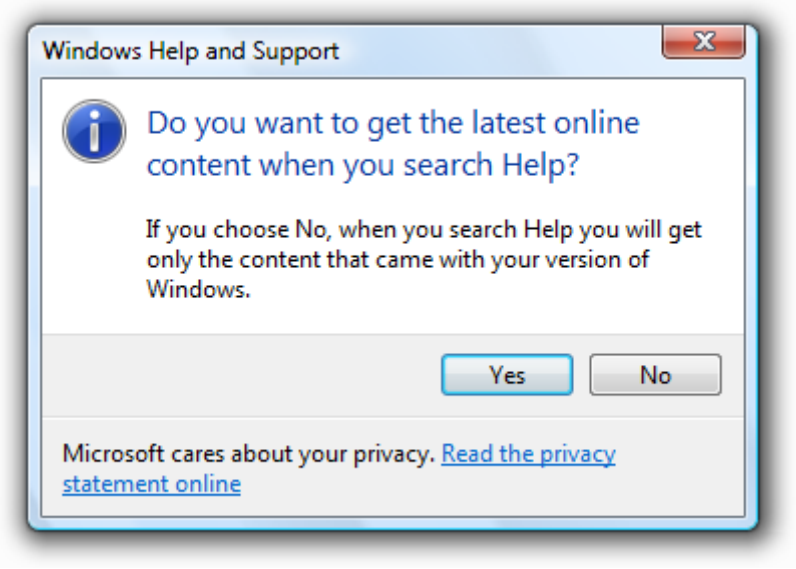
Pointers

Expand table

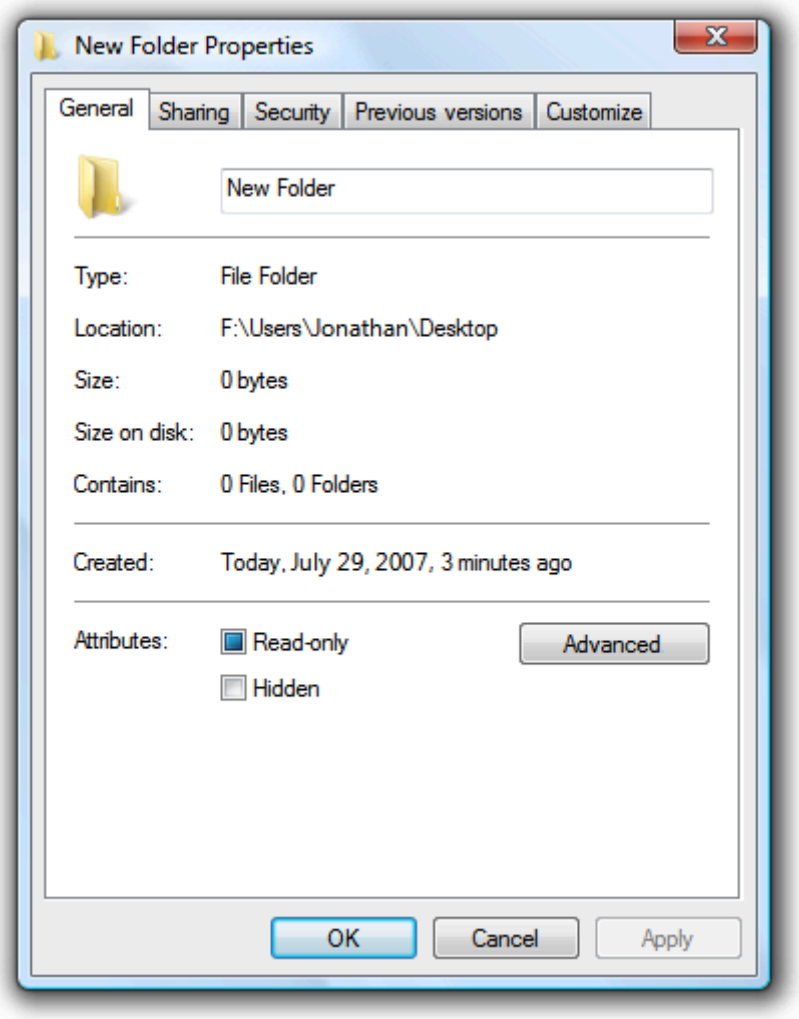
working in background
busy
activity indicator 

Windows

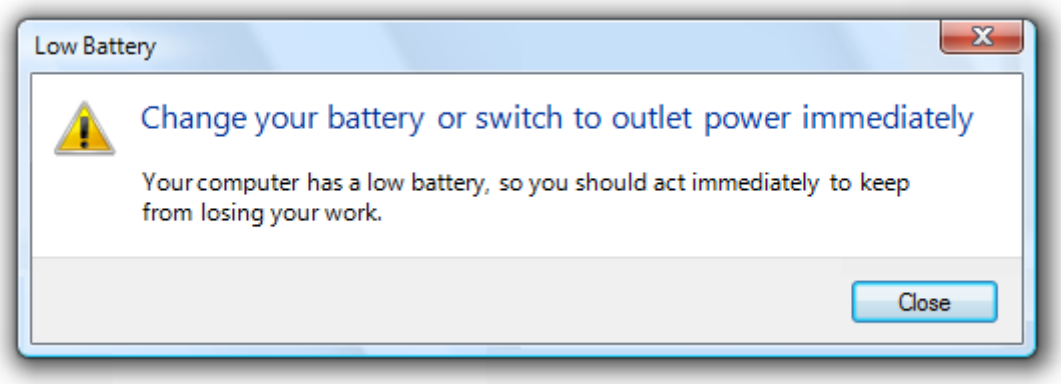
Expand table

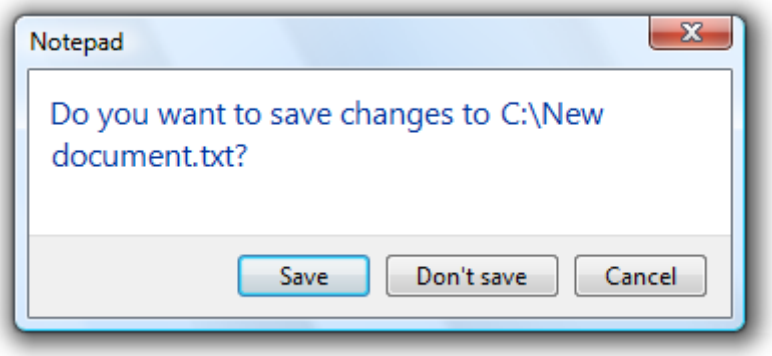
dialog boxes 

property windows



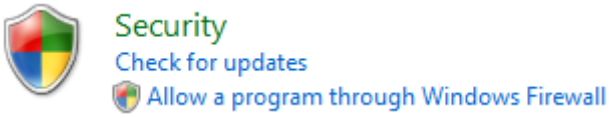
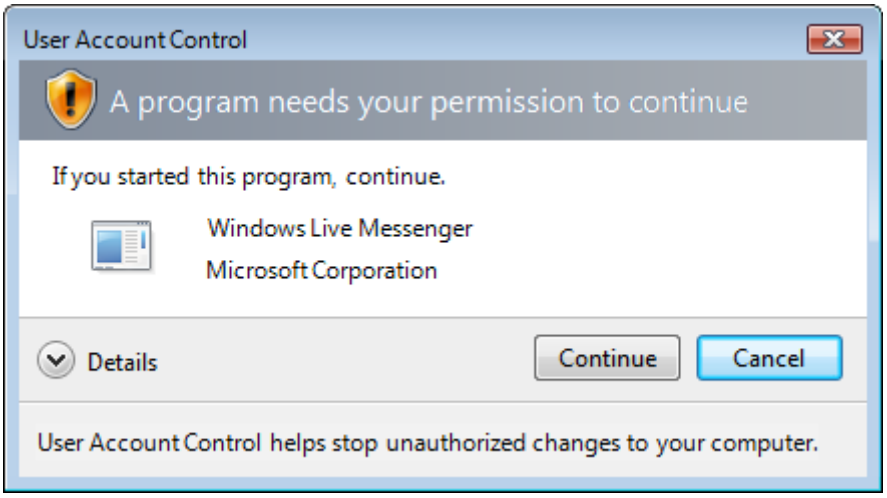
warning messages

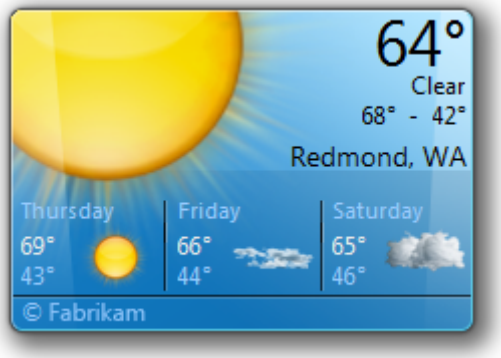
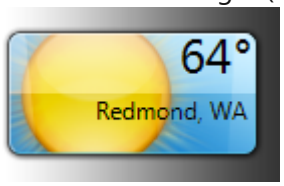


confirmations 

Windows Environment

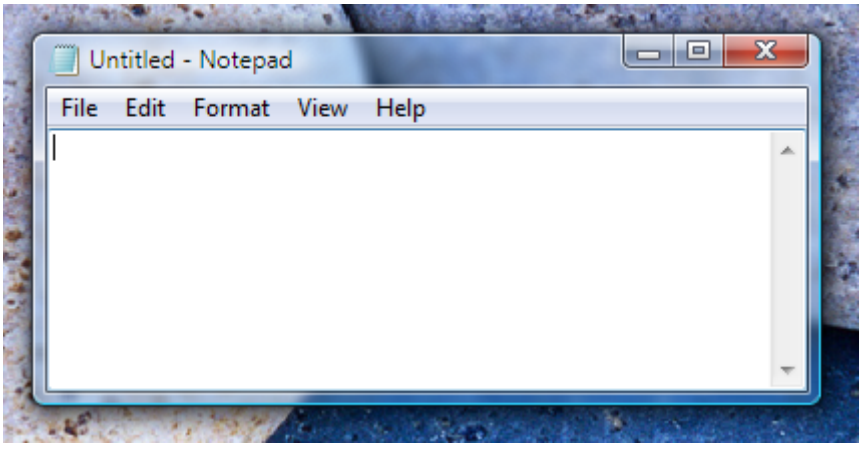
Expand table

taskbar
user account control entry points 
user account control consent ui 

DEPRECATED: Gadget (detailed/floating) 
DEPRECATED: Gadget (concise/docked) 

Visuals

Expand table

window frames 
fonts abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ

standard icons

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Glossary (Design basics)

Article • 03/05/2021

> **Note**
>
> This design guide was created for Windows 7 and has not been updated for newer versions of Windows. Much of the guidance still applies in principle, but the presentation and examples do not reflect our current design guidance.

A | B | C | D | E | F | G | H | I | J | K | L | M |

N | O | P | Q | R | S | T | U | V | W | X | Y | Z

A

About box

A dialog box providing general program information, such as version identification, copyright, licensing agreements, and ways to access technical support.

above the fold

A screen layout metaphor taken from newspaper journalism. The content above the fold of the newspaper must be particularly engaging to spur sales. Similarly, in screen layout, the most important content must be visible without scrolling. Users must be motivated to take the time to scroll past the content they encounter initially "above the fold."

access key

An alphanumeric key that, when combined with the Alt key, activates a control. Access keys are indicated by underlining one of the characters in the control's label. For example, pressing Alt+O activates a control whose label is "Open" and whose assigned access key is "O". Access keys aren't case sensitive. The effect of activating a control depends on the type of control.

In contrast to shortcut keys, which are intended mostly for advanced users, access keys are designed to improve accessibility. Because they are documented directly within the user interface (UI) itself, they can't always be assigned consistently and they aren't intended to be memorized.

active monitor

The monitor where the active program is running.

address bar

A navigational element, usually appearing at the top of a window, that displays, and allows users to change, their current location. See also: breadcrumb bar.

affordance

Visual properties of an object that indicate how it can be used or acted on. For example, command buttons have the visual appearance of real-world buttons, suggesting pushing or clicking.

application

A program used to perform a related set of user tasks; often relatively complex and sophisticated. See also: program.

application key

A keyboard key with a context menu graphic on it. This key is used to display the context menu for the selected item.

application menu

A control that presents a menu of commands that involve doing something to or with a document or workspace, such as file-related commands.

Displays the context menu for the current selection (the same as pressing Shift+F10).

application theming

Using related visual techniques, such as customized controls, to create a unique look or branding for an application.

aspect ratio

An expression of the relation between the width of an object and its height. For example, high definition television uses a 16:9 aspect ratio.

auto-complete

A type of list used in text boxes and editable drop-down lists in which the likely input can be extrapolated and populated automatically from having been entered previously. Users type a minimal amount of text to populate the auto-complete list.

auto-exit

A text box in which the input focus automatically moves to the next related text box as soon as a user types the last character.

B

balloon

A common Windows control that informs users of a non-critical problem or special condition.

breadcrumb bar

A navigational element, usually appearing at the top of a window, that displays, and allows users to change, their current location. "Breadcrumb" refers to breaking the current location into a series of links separated by arrows that users can interact with directly. Use address bar instead. See also: address bar.

C

check box

A common Windows control that allows users to decide between clearly differing choices, such as toggling an option on or off.

chevron

A small control or button that indicates there are more items than can be displayed in the allotted space. Users click the chevron to see the additional items.

child window

A window, such as a control or pane, that is contained completely within another window, referred to as the parent window. See also: parent window, owned window.

cleared

In a check box, indicates that the option is not set. See also: selected, mixed state.

combo box

A common Windows control that combines the characteristics of a drop-down list or standard list box, and an editable text box. See also: list box, drop-down list.

command area

The area in a window where the commit buttons are located. Typically, dialog boxes and wizards have a command area. See also: commit button.

command button

A common Windows control that allows users to initiate an action immediately.

command link

A control used to make a choice among a set of mutually exclusive, related choices. In their normal state, command links have a lightweight appearance similar to hyperlinks, but their behavior is more similar to command buttons.

commit button

A command button used to commit to a task, proceed to the next step in a multi-step task, or cancel a task. See also: command area.

commit page

A type of wizard page in which users commit to performing the task. After doing so, the task cannot be undone by clicking Back or Cancel buttons.

completion page

A wizard page used to indicate the end of a wizard. Sometimes used instead of Congratulations pages. See also: congratulations page.

congratulations page

A wizard page used to indicate the end of a wizard. These pages are no longer recommended. Wizards conclude more efficiently with a commit page or, if necessary, a follow-up or completion page. See also: commit page, completion page, follow-up page.

Consent UI

A dialog box used by User Account Control (UAC) that allows protected administrators to elevate their privileges temporarily.

constraint

In controls that involve user input, such as text boxes, input constraints are a valuable way to prevent errors. For example, if the only valid input for a particular control is numeric, the control can use appropriate value constraints to enforce this requirement.

content area

The portion of UI surfaces, such as dialog boxes, control panel items, and wizards, devoted to presenting options, providing information, and describing controls. Distinguished from the command area, task pane, and navigation area.

contextual tab

A tab containing a collection of commands that are relevant only when the user has selected a particular object type. See also: Ribbon.

Control Panel

A Windows program that collects and displays for users the system-level features of the computer, including hardware and software setup and configuration. From Control Panel, users can click individual items to configure system-level features and perform related tasks. See also: control panel item.

control panel item

An individual feature available from Control Panel. For example, Programs and Ease of Access are two control panel items.

Credential UI

A dialog box used by User Account Control (UAC) that allows standard users to request temporary elevation of their privileges.

critical

The highest degree of severity. For example, in error and warning messages, critical circumstances might involve data loss, loss of privacy, or loss of system integrity.

custom icon

A pictorial representation unique to a program (as opposed to a Windows system icon).

custom visuals

Graphics, animations, icons, and other visual elements specially developed for a program.

D

default command button or link

The command button or link that is invoked when users press the Enter key. The default command button or link is assigned by the developer, but any command button or link

becomes the default when users tab to it.

default monitor

The monitor with the Start menu, taskbar, and notification area.

delayed commit model

The commit model used by control panel item spoke pages where changes aren't made until explicitly committed by users clicking a commit button. Thus, users can abandon a task, navigating away using the Back button, Close, or the Address bar. See also: immediate commit model.

desktop

The onscreen work area provided by Windows, analogous to a physical desktop. See also: work area.

destructive command

An action that has a widespread effect and cannot be undone easily, or is not immediately noticeable.

details pane

The pane at the bottom of a Windows Explorer window that displays details (if any) about the selected items; otherwise, it displays details about the folder. For example, Windows Photo Gallery displays the picture name, file type, date taken, tags, rating, dimensions, and file size. See also: preview pane.

dialog box

A secondary window that allows users to perform a command, asks users a question, or provides users with information or progress feedback.

dialog box launcher

In a ribbon, a button at the bottom of some groups that opens a dialog box with features related to the group. See also: Ribbon.

dialog unit

A dialog unit (DLU) is the device-independent measure to use for layout based on the current system font.

direct manipulation

Direct interaction between the user and the objects in the UI (such as icons, controls, and navigational elements). The mouse and touch are common methods of direct manipulation.

docked window

A window that appears at a fixed location on the edge of its owner window. See also: floating window.

drop-down arrow

The arrow associated with drop-down lists, combo boxes, split buttons, and menu buttons, indicating that users can view the associated list by clicking the arrow.

drop-down list

A common Windows control that allows users to select among a list of mutually exclusive values. Unlike a list box, this list of available choices is normally hidden.

E

effective resolution

The physical resolution of a monitor normalized by the current dpi (dots per inch) setting. At 96 dpi, the effective resolution is the same as the physical resolution, but in other dpis, the effective resolution must be scaled proportionately. Generally, the effective resolution can be calculated using the following equation:

Effective resolution = Physical resolution x (96 / current dpi setting)

See also: relative pixels, physical resolution.

elevated administrator

In User Account Control, elevated administrators have their administrator privileges. Without elevating, administrators run in their least-privileged state. The Consent UI dialog is used to elevate administrators to elevated status only when necessary. See also: protected administrator, standard user.

enhanced tooltip

A pop-up window that concisely explains the command being pointed to. Like regular tooltips, enhanced tooltips may provide the shortcut key for the command. But unlike regular tooltips, they may also provide supplemental information, graphics, and an

indicator that Help is available. They may also use rich text and separators. See also: tooltip.

error

A state in which a problem has occurred. See also: warning.

expandable headings

A progressive disclosure chevron pattern where a heading can be expanded or collapsed to reveal or hide a group of items. See also: progressive disclosure.

extended selection

In list views and list boxes, a multiple selection mode where selection of a single item can be extended by dragging or with Shift+click or Ctrl+click to select groups of contiguous or non-adjacent values, respectively. See also: multiple selection.

F

flick

A quick, straight stroke of a finger or pen on a screen. A flick is recognized as a gesture, and interpreted as a navigation or an editing command.

floating window

A window that can appear anywhere on the screen the user wants. See also: docked window.

flyout

A popup window that temporarily shows more information. On the Windows desktop, flyouts are displayed by clicking on a gadget, and dismissed by clicking anywhere outside the flyout. You can use flyouts in both the docked and floating states.

follow-up page

A wizard page used to present related tasks that users are likely to do as follow-up. Sometimes used instead of congratulations pages.

font

A set of attributes for text characters.

full screen

A maximized window that does not have a frame.

G

gadget

A simple mini-application hosted on the user's desktop. See also: Sidebar.

gallery

A list of commands or options presented graphically. A results-based gallery illustrates the effect of the commands or options instead of the commands themselves. May be labeled or grouped. For example, formatting options can be presented in a thumbnail gallery.

gesture

A quick movement of a finger or pen on a screen that the computer interprets as a command, rather than as a mouse movement, writing, or drawing.

getting started page

An optional wizard page that outlines prerequisites for running the wizard successfully or explains the purpose of the wizard.

glass

A window frame option characterized by translucence, helping users focus on content and functionality rather than the interface surrounding it.

glyph

A generic term used to refer to any graph or symbolic image. Arrows, chevrons, and bullets are glyphs commonly used in Windows.

group box

A common Windows control that shows relationships among a set of related controls.

H

handwriting recognition

Software that converts ink to text.

Help

User assistance of a more detailed nature than is available in the primary UI. Typically accessed from a menu or by clicking a Help link or icon, this content may take a variety of forms, including step-by-step procedures, conceptual text, or more visually-based, guided tutorials.

high-contrast mode

A special display setting that provides extreme contrast for foreground and background visual elements (either black on white or white on black). Particularly helpful for accessibility.

hub page

In control panel items, a hub page presents high-level choices, such as the most commonly used tasks (as with task-based hub pages) or the available objects (as with object-based hub pages). Users can navigate to spoke pages to perform specific tasks. See also: spoke page.

hybrid hub page

In control panel items, a hybrid hub page is a hub page that also has some properties or commands directly on it. Hybrid hub pages are strongly recommended when users are most likely to use the control panel item to access those properties and commands.

I

immediate commit model

The commit model used by hybrid hub pages where changes take effect as soon as users make them. Commit buttons aren't used in this model. See also: delayed commit model.

in-place message

A message that appears in the context of the current UI surface, instead of a separate window. Unlike separate windows, in-place messages require either available screen space or dynamic layout.

indirect dialog box

A dialog box displayed out of context, either as an indirect result of a task or as the result of a problem with a system or background process.

inductive user interface

A UI that breaks a complex task down into simple, easily explained, clearly stated steps with a clear purpose.

infotip

A small pop-up window that concisely describes the object being pointed to, such as descriptions of toolbar controls, icons, graphics, links, Windows Explorer objects, Start menu items, and taskbar buttons. Infotips are a form of progressive disclosure, eliminating the need to have descriptive text on screen at all times.

ink

The raw output for a pen. This digital ink can be kept just as written, or it can be converted to text using handwriting recognition software.

inline

Placement of links or messages directly in the context of its related UI. For example, an inline link occurs within other text instead of separately.

input focus

The location where the user is currently directing input. Note that just because a location in the UI is highlighted does not necessarily mean this location has input focus.

instance

A program session. For example, Windows Internet Explorer allows users to run multiple instances of the program because users can have several independent sessions running at a time. Settings can be saved across program sessions. See also: persistence.

J

K

keytip

In a ribbon, the mechanism used to display access keys. The access keys appear in the form of a small tip over each command or group, as opposed to the underlined letters typically used to display access keys. See also: access key.

L

landscape mode

A presentation option that orients an object to be wider than it is tall. See also: portrait mode.

least-privilege user account

A user account that normally runs with minimal privileges. See also: User Account Control.

list box

A common Windows control that allows users to select from a set of values presented in a list, which, unlike a drop-down list, is always visible. Supports single or multiple selections.

list view

A common Windows control that allows users to view and interact with a collection of data objects, using either single selection or multiple selection.

live preview

A preview technique that shows the effect of a command immediately on selection or hover without the user commiting the action. For example, formatting options such as themes, fonts, and colors benefit from live previews by showing users the effect with minimal effort.

localization

The process of adapting software for different countries, languages, cultures, or markets.

log file

A file-based repository for information of various kinds about activity on a computer system. Administrators often consult log files; ordinary users generally do not.

M

main instruction

Prominently displayed text that concisely explains what to do in the window or page. The instruction should be a specific statement, imperative direction, or question. Good

main instructions communicate the user's objective rather than focusing just on manipulating the UI.

managed environment

A networked computer environment managed by an IT department or third-party provider, instead of by individual users. Administrators may optimize performance and apply operating system and application updates, among other tasks.

manipulation

A type of touch interaction in which input corresponds directly to how the object being touched would react naturally to the action in the real world.

maximize

To display a window at its largest size. See also: minimize, restored window.

menu

A list of commands or options available to users in the current context.

message box

A secondary window that is displayed to inform a user about a particular condition.

mini-toolbar

A contextual toolbar displayed on hover.

minimize

To hide a window. See also: maximize, restored window.

mixed state

For check boxes that apply to a group of items, a mixed state indicates that some of the items are selected and others are cleared.

modal

Restrictive or limited interaction due to operating in a mode. Modal often describes a secondary window that restricts a user's interaction with the owner window. See also: modeless.

modeless

Non-restrictive or non-limited interaction. Modeless often describes a secondary window that does not restrict a user's interaction with the owner window. See also: modal.

multiple selection

The ability for users to choose more than one object in a list or tree.

N

non-critical system event

A type of system event that does not require immediate attention, often pertaining to system status. See also: critical.

notification

Information of a non-critical nature that is displayed briefly to the user; a notification takes the form of a balloon from an icon in the notification area of the taskbar.

O

opt in

The ability for users to select optional features explicitly. Less intrusive to users than opt-out, especially for privacy and marketing related features, because there is no presumption of users' wishes. See also: opt out, options.

opt out

The ability for users to remove features they don't want by clearing their selection. More intrusive to users than opt-in, especially for privacy and marketing related features, because there is an assumption of users' wishes. See also: opt in, options.

options

Choices available to users for customizing a program. For example, an Options dialog box allows users to view and change program options. See also: properties.

out-of-context UI

Any UI displayed in a pop-up window that isn't directly related to the user's current activity. For example, notifications and the Consent UI for User Access Control are out-of-context UI.

owned window

A secondary window used to perform an auxiliary task. It is not a top-level window (so it isn't displayed on the taskbar); rather, it is "owned" by its owner window. For example, most dialog boxes are owned windows. See also: child window, owner window.

owner control

The source of a tip, balloon, or flyout. For example, a text box that has input constraints might display a balloon to let the user know of these limitations. In this case, the text box is considered the owner control.

owner window

A window from which an owned window originates. Appears beneath the owned window in Z order. See also: owned window, parent window, Z order.

P

page

A basic unit of navigation for task-based UI, such as wizards, property sheets, control panel items, and Web sites. Users perform tasks by navigating from page to page within a single host window. See also: page flow, window.

page flow

A collection of pages in which users perform a task. See also: page, task, wizard, Control Panel.

page space control

Allows users to view and interact with a hierarchically arranged collection of objects. Page space controls are like tree controls, but they have a slightly different visual appearance. They are used primarily by Windows Explorer.

palette window

A modeless secondary window that displays a toolbar or other choices, such as colors, patterns, fonts, or font attributes.

pan

To move a scene, such as a map or photo, in two dimensions by dragging it directly. This differs from scrolling in two ways: scrolled content usually has one predominant

dimension and often scrolls only along that dimension; and scrolling content conventionally appears with scroll bars that the user drags in the opposite direction of the scrolling motion.

pane

A rectangular area within a window that users may be able to move, resize, hide, or close. Panes are always docked to the side of their parent window. They can be adjacent to other panes, but they never overlap. Undocking a pane converts it to a child window. See also: window.

parent window

The container of child windows (such as controls or panes). See also: owner window.

pen

A stylus used for pointing, gestures, simple text entry, and free-form handwriting. Pens have a fine, smooth tip that supports precise pointing, writing, or drawing in ink. They may also have an optional pen button (used to perform right-clicks) and eraser (used to erase ink).

persistence

The principle that the state or properties of an object is automatically preserved.

personalization

Customizing a core experience that is crucial to the user's personal identification with a program. By contrast, ordinary options and properties aren't crucial to the user's personal identification with a program.

personas

Detailed descriptions of imaginary people. Personas are constructed out of well-understood, highly specified data about real people.

physical resolution

The horizontal and vertical pixels that can be displayed by a computer monitor's hardware.

pop-up group button

In a ribbon, a menu button that consolidates all the commands and options within a group. Used to display ribbons in small spaces.

portrait mode

A presentation option that orients an object to be taller than it is wide. See also: landscape mode.

preferences

Don't use. Use options or properties instead.

preview

A representation of what users will see when they select an option. Previews can be displayed statically as part of the option, or upon request with a Preview or Apply button.

preview pane

A window pane used to show previews and other data about selected objects.

primary command

A central action that fulfills the primary purpose of a window. For example, Print is a primary command for a Print dialog box. See also: secondary command.

primary toolbar

A collection of commands designed to be comprehensive enough to preclude the use of a menu bar. See also: supplemental toolbar.

primary window

A primary window has no owner window and is displayed on the taskbar. Main program windows are always primary windows. See also: secondary window.

program

A sequence of instructions that can be executed by a computer. Common types of programs include productivity applications, consumer applications, games, kiosks, and utilities.

progress bar

A common Windows control that displays the progress of a particular operation as a graphical bar.

progressive disclosure

A technique of allowing users to display less commonly used information (typically, data, options, or commands) as needed. For example, if more options are sometimes needed, users can expose them in context by clicking a chevron button.

progressive escalation

A sequence in which the UI used to inform users becomes progressively more obtrusive as the event becomes more critical. For example, a notification can be used for an event that users can safely ignore at first. As the situation becomes critical, a more obtrusive UI such as a modal dialog should be used.

prompt

A label or short instruction placed inside a text box or editable drop-down list as its default value. Unlike static text, prompts disappear once users type something into the control or it gets input focus.

properties

Settings of an object that users can change, such as a file's name and read-only status, as well as attributes of an object that users can't directly change, such as a file's size and creation date. Typically properties define the state, value, or appearance of an object.

protected administrator

In User Account Control, an administrator running in their least-privileged state. See also: elevated administrator, standard user.

Q

Quick Access Toolbar

A small, customizable toolbar that displays frequently used commands.

Quick Launch bar

A direct access point on the Windows desktop, located next to the Start button, populated with icons for programs of the user's choosing. Removed in Windows 7.

R

radio button

A common Windows control that allow users to select from among a set of mutually exclusive, related choices.

relative pixels

A device-independent metric that is the same as a physical pixel at 96 dpi (dots per inch), but proportionately scaled in other dpis. See also: effective resolution.

restored window

A visible, partial-screen window, neither maximized nor minimized. See also: maximize, minimize.

ribbon

A tabbed container of commands and options, located at the top of a window or work area and having a fixed location and height. Ribbons usually have an Application menu and Quick Access Toolbar. See also: menu, toolbar.

risky action

A quality of user action that can have negative consequences and can't be easily undone. Risky actions include actions that can harm the security of a computer, affect access to a computer, or result in unintended loss of data.

S

scan path

The route users are likely to take as they scan to locate things in a window. Particularly important if users are not engaged in immersive reading of text.

screen reader

An assistive technology that enables users with visual impairments to interpret and navigate a user interface by transforming visuals to audio. Thus, text, controls, menus, toolbars, graphics, and other screen elements are spoken by the computerized voice of the screen reader.

scroll bar

A control that allows users to scroll the content of a window, either vertically or horizontally.

secondary command

A peripheral action that, while helpful, isn't essential to the purpose of the window. For example, Find Printer or Install Printer are secondary commands for a Print dialog box. See also: primary command.

secondary window

A window that has an owner window and consequently is not displayed on the taskbar. See also: primary window.

secure desktop

A protected environment that is isolated from programs running on the system, used to increase the security of highly secure tasks such as log on, password changes, and UAC Elevation UI. See also: User Account Control.

security shield

A shield icon used for security branding.

selected

Chosen by the user in order to perform an operation; highlighted.

sentence-style capitalization

For sentence-style capitalization:

- Always capitalize the first word of a new sentence.
- Don't capitalize the word following a colon unless the word is a proper noun, or the text following the colon is a complete sentence.
- Don't capitalize the word following an em-dash unless it is a proper noun, even if the text following the dash is a complete sentence.
- Always capitalize the first word of a new sentence following any end punctuation. Rewrite sentences that start with a case-sensitive lowercase word.

settings

Specific values that have been chosen (either by the user or by default) to configure a program or object.

shortcut key

Keys or key combinations that users can press for quick access to actions they perform frequently. Ctrl+letter combinations and function keys (F1 through F12) are usually the best choices for shortcut keys. By definition, a shortcut key is the keyboard equivalent of

functionality that is supported adequately elsewhere in the interface. Therefore, avoid using a shortcut key as the only way to access a particular operation.

In contrast to access keys, which are designed to improve accessibility, shortcut keys are designed primarily for advanced users. Because they aren't documented directly within the UI itself (although they might be documented in menus and toolbar tooltips), they are intended to be memorized and therefore they must be assigned consistently within applications and across different applications.

Sidebar

A region on the side of the user's desktop used to display gadgets in Windows Vista. See also: gadget.

single-point error

A user input error relating to a single control. For example, entering an incorrect credit card number is a single-point error, whereas an incorrect logon is a double-point error, because either the user name or password could be the problem.

slider

A common Windows control that displays and sets a value from a continuous range of possible values, such as brightness or volume.

special experience

In programs, special experiences relate to the primary function of the program, something unique about the program, or otherwise make an emotional connection to users. For example, playing an audio or video is a special experience for a media player.

spin box

The combination of a text box and its associated spin control. Users click the up or down arrow of a spin box to increase or decrease a numeric value. Unlike slider controls, which are used for relative quantities, spin boxes are used only for exact, known numeric values.

spin control

A control that users click to change values. Spin controls use up and down arrows to increase or decrease the value.

splash screen

Transitional screen image that appears as a program is in the process of launching.

split button

A bipartite command button that includes a small button with a downward pointing triangle on the rightmost portion of the main button. Users click the triangle to display variations of a command in a drop-down menu. See also: command button.

spoke page

In control panel items, spoke pages are the place in which users perform tasks. Two types of spoke pages are task pages and form pages: task pages present a task or a step in a task with a specific, task-based main instruction; form pages present a collection of related properties and tasks based on a general main instruction. See also: hub page.

standard user

In User Account Control, standard users have the least privileges on the computer, and must request permission from an administrator on the computer in order to perform administrative tasks. In contrast with protected administrators, standard users can't elevate themselves. See also: elevated administrator, protected administrator.

static text

User interface text that is not part of an interactive control. Includes labels, main instructions, supplemental instructions, and supplemental explanations.

supplemental instructions

An optional form of user interface text that adds information, detail, or context to the main instruction. See also: main instruction.

supplemental toolbar

A collection of commands designed to work in conjunction with a menu bar. See also: primary toolbar.

system color

A color defined by Windows for a specific purpose, accessed using the GetSysColor application programming interface (API). For example, COLOR_WINDOW defines the window background color and COLOR_WINDOWTEXT defines the window text color. System colors are not as rich as theme colors. See also: theme color.

system menu

A collection of basic window commands, such as move, size, maximize, minimize, and close, available from the program icon on the title bar, or by right-clicking a taskbar

button.

T

tabbed dialog

A dialog box that contains related information on separate labeled pages (tabs). Unlike property sheets, which also often contain tabs, tabbed dialog boxes are not used to display an object's properties. See also: properties.

task

A unit of user activity, often represented by a single UI surface (such as a dialog box), or a sequence of pages (such as a wizard).

task dialog

A dialog box implemented using the task dialog API. Requires Windows Vista or later.

task flow

A sequence of pages that helps users perform a task, either in a wizard, explorer, or browser.

task link

A link used to initiate a task, in contrast to links that navigate to other pages or windows, choose options, or display Help.

task pane

A type of UI similar to a dialog box, except that it is presented within a window pane instead of a separate window. As a result, task panes have a more direct, contextual feel than dialog boxes. A task pane can contain a menu to provide the user with a small set of commands related to the selected object or program mode.

taskbar

The access point for running programs that have a desktop presence. Users interact with controls called taskbar buttons to show, hide, and minimize program windows.

text box

A control specifically designed for textual input; allows users to view, enter, or edit text or numbers.

theme color

A color defined by Windows for a specific purpose, accessed using the GetThemeColor API along with parts, states, and colors. For example, the windows part defines a FillColor and a TextColor. Theme colors are richer than system colors, but require the theme service to be running. See also: system color.

title-style capitalization

For title-style capitalization:

- Capitalize all nouns, verbs (including is and other forms of to be), adverbs (including than and when), adjectives (including this and that), and pronouns (including its).
- Capitalize the first and last words, regardless of their parts of speech (for example, The Text to Look For).
- Capitalize prepositions that are part of a verb phrase (for example, Backing Up Your Disk).
- Don't capitalize articles (a, an, the), unless the article is the first word in the title.
- Don't capitalize coordinate conjunctions (and, but, for, nor, or), unless the conjunction is the first word in the title.
- Don't capitalize prepositions of four or fewer letters, unless the preposition is the first word in the title.
- Don't capitalize to in an infinitive phrase (for example, How to Format Your Hard Disk), unless the phrase is the first word in the title.
- Capitalize the second word in compound words if it is a noun or proper adjective, an "e-word," or the words have equal weight (for example, E-Commerce, Cross-Reference, Pre-Microsoft Software, Read/Write Access, Run-Time). Do not capitalize the second word if it is another part of speech, such as a preposition or other minor word (for example, Add-in, How-to, Take-off).
- Capitalize user interface and application programming interface terms that you would not ordinarily capitalize, unless they are case-sensitive (for example, The fdisk Command). Follow the traditional capitalization of keywords and other special terms in programming languages (for example, The printf Function, Using the EVEN and ALIGN Directives).
- Capitalize only the first word of each column heading.

toolbar

A graphical presentation of commands optimized for efficient access.

tooltip

A small pop-up window that labels the unlabeled control being pointed to, such as unlabeled toolbar controls or command buttons.

touch

Direct interaction with a computer display using a finger.

U

usability study

A research technique that helps you improve your user experience by testing your UI design and gathering feedback from real target users. Usability studies can range from formal techniques in settings such as usability labs, to informal techniques in settings such as the user's own office. But the constants of such studies are: capturing information from the participants; evaluating that information for meaningful trends and patterns; and finally implementing logical changes that address the problems identified in the study.

User Account Control

With User Account Control (or UAC, formerly known as "Least-privilege User Account," or LUA) enabled, interactive administrators normally run with least user privileges, but they can self-elevate to perform administrative tasks by giving explicit consent with the Consent UI. Such administrative tasks include installing software and drivers, changing system-wide settings, viewing or changing other user accounts, and running administrative tools.

User Account Control shield

A shield icon used to indicate that a command or option needs elevation for User Account Control.

user input problem

An error resulting from user input. User input problems are usually non-critical because they must be corrected before proceeding.

user scenario

A description of a user goal, problem, or task in a specific set of circumstances.

V

W

warning

A message that describes a condition that might cause a problem in the future. Warnings aren't errors or questions. In Windows Vista and later, warning messages are typically displayed in task dialogs, include a clear, concise main instruction, and usually include a standard warning icon for visual reinforcement of the text.

welcome page

The first page of a wizard, used to explain the purpose of the wizard. Welcome pages are no longer recommended. Users have a more efficient experience without such pages.

window

A rectangular area on a computer screen in which programs and content appear. A window can be moved, resized, minimized, or closed; it can overlap other windows. Docking a child window converts it to a pane. See also: pane.

Windows logo key

A modifier key with the Windows logo on it. This key is used for a number of Windows shortcuts, and is reserved for Windows use. For example, pressing the Windows logo key displays or hides the Windows Start menu.

wireframes

A UI mockup that shows a window's functionality and layout, but not its finished appearance. A wireframe uses only line segments, controls, and text, without color, complex graphics, or the use of themes.

wizard

A sequence of pages that guides users through a multi-step, infrequently performed task. Effective wizards reduce the knowledge required to perform the task compared to alternative UIs.

work area

The onscreen area where users can perform their work, as well as store programs, documents, and their shortcuts. See also: desktop.

X

Y

Z

Z order

The layered relationship of windows on the display.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A