GNOME Human Interface Guidelines (Draft 2.1)

The GNOME Usability Project

GNOME Human Interface Guidelines (Draft 2.1)

The GNOME Usability Project

Copyright © 2002-2005 Calum Benson, Bryan Clark, Adam Elman, Seth Nickell, colin z robertson

Permission is granted to copy, distribute and/or modify this document under the terms of the *GNU Free Documentation License*, Version 1.1 or any later version published by the Free Software Foundation with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. You may obtain a copy of the *GNU Free Documentation License* from the Free Software Foundation by visiting their Web site [http://www.fsf.org] or by writing to: Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

Many of the names used by companies to distinguish their products and services are claimed as trademarks. Where those names appear in any GNOME documentation, and those trademarks are made aware to the members of the GNOME Documentation Project, the names have been printed in caps or initial caps.

Table of Contents

Introduction xiii
What's new since HIG v2.0? xiv
1. Usability Principles 1
    1. Design for People 1
    2. Don't Limit Your User Base 1
        2.1. Accessibility 1
        2.2. Internationalization and Localization 2
    3. Create a Match Between Your Application and the Real World 2
    4. Make Your Application Consistent 3
    5. Keep the User Informed 3
    6. Keep It Simple and Pretty 4
    7. Put the User in Control 4
    8. Forgive the User 4
    9. Provide Direct Manipulation 4
2. Desktop Integration 5
    1. Placing Entries in the Applications Menu 5
        1.1. Menu Item Names 5
        1.2. Menu Item Tooltips 7
    2. GConf Keys 8
    3. Mapping Document Types to Applications 9
    4. Application Startup 9
    5. Using the Status Notification Area 9
        5.1. Appropriate Uses for the Notification Area 10
        5.2. Icon Appearance 10
        5.3. Icon Animation 11
        5.4. Interaction 11
        5.5. Notification Balloons 12
3. Windows 13
    1. Parts of Windows and System Interaction 13
        1.1. Titles 13
        1.2. Icon 13
        1.3. Borders and Window Commands 13
        1.4. Modality 13
        1.5. Focus 14
        1.6. Showing and Hiding Windows 14
    2. Primary Windows 15
        2.1. Title 15
        2.2. Window Commands 16
        2.3. Relation between Documents and Windows 17
    3. Utility Windows 18
        3.1. Instant apply windows 18
        3.2. Explicit apply windows 18
        3.3. Default Buttons 19
        3.4. Property Windows 19
        3.5. Preferences Windows 19
        3.6. Toolboxes 20
    4. Alerts 21
        4.1. Alert Text 22
        4.2. Alert Buttons 23
        4.3. Spacing and Positioning Inside Alerts 23
        4.4. Information Alerts 25
        4.5. Error Alerts 25
        4.6. Confirmation Alerts 26

4.7. Authentication Alerts ... 27
5. Progress Windows ... 28
5.1. Checklist Windows ... 29
6. Dialogs ... 30
6.1. Additional Buttons ... 30
6.2. Layout ... 30
6.3. Common Dialogs ... 31
7. Assistants ... 31
7.1. Introductory Page ... 31
7.2. Content Pages ... 32
7.3. Last Page ... 32
4. Menus ... 33
1. The Menubar ... 33
2. Types of Menu ... 34
2.1. Drop-down Menus ... 34
2.2. Submenus ... 35
2.3. Popup Menus ... 35
3. Designing a Menu ... 36
3.1. Grouping Menu Items ... 36
3.2. Types of menu item ... 37
4. Standard Menus ... 38
4.1. File ... 39
4.2. Edit ... 44
4.3. View ... 47
4.4. Insert ... 49
4.5. Format ... 51
4.6. Bookmarks ... 53
4.7. Go ... 54
4.8. Windows ... 55
4.9. Help ... 56
5. Toolbars ... 57
1. Appearance and Content ... 57
1.1. Vertical Toolbars ... 58
1.2. Media Player Toolbars ... 58
2. Controlling Display and Appearance ... 58
3. Labels and Tooltips ... 59
6. Controls ... 61
1. Using Controls Effectively ... 61
2. Terminology ... 61
3. Sensitivity ... 61
3.1. Locked Controls ... 61
4. Text Entry Fields ... 62
4.1. Behavior of Return key ... 63
4.2. Behavior of Tab key ... 64
5. Spin Boxes ... 64
6. Sliders ... 65
7. Buttons ... 65
8. Check Boxes ... 66
9. Radio Buttons ... 68
10. Toggle Buttons ... 69
11. Drop-down Lists ... 70
12. Drop-down Combination Boxes ... 71
13. Scrollbars ... 72
14. Lists ... 72
14.1. Sortable Lists ... 73
15. Trees ... 74
15.1. Sortable Trees ... 75
16. Tabbed Notebooks ... 75

16.1. Status Indicators ... 76
17. Progress Bars ... 76
17.1. Time-remaining Progress Indicator ... 77
17.2. Typical-time Progress Indicator ... 77
17.3. Indeterminate-progress indicator ... 77
18. Statusbars ... 78
19. Frames and Separators ... 79
20. Disclosure Triangles ... 79
7. Feedback ... 80
1. Characteristics of Responsive Applications ... 80
2. Acceptable Response Times ... 81
3. Responding to User Requests ... 81
4. Types of Visual Feedback ... 82
4.1. Pointer Feedback ... 82
4.2. Progress Animations ... 83
5. Choosing Appropriate Feedback ... 84
6. Allowing Interruptions ... 85
8. Visual Design ... 86
1. Color ... 86
1.1. Palette ... 86
1.2. Hue, Brightness, Contrast ... 87
2. Window Layout ... 88
2.1. General ... 88
2.2. Dialogs ... 88
2.3. Size ... 89
2.4. Spacing and Alignment ... 90
3. Text Labels ... 91
3.1. Spacing and Alignment ... 91
3.2. Capitalization ... 93
4. Fonts ... 94
9. Icons ... 95
1. Style ... 95
1.1. Perspective ... 95
1.2. Lighting ... 96
1.3. Palette ... 96
2. Kinds of Icons ... 96
2.1. Document Icons ... 96
2.2. Application Icons ... 97
2.3. Toolbar Icons ... 97
2.4. Menu Icons ... 97
3. Designing Effective Icons ... 97
3.1. Suggested Design Process For Toolbar and Menu Icons ... 98
3.2. Problems to Avoid ... 99
4. Designing Accessible Icons ... 101
4.1. High Contrast Icons ... 101
4.2. Low Contrast Icons ... 103
10. User Input ... 104
1. Mouse Interaction ... 104
1.1. Buttons ... 104
1.2. Selecting Objects ... 105
1.3. Drag and Drop ... 107
1.4. Mouse Interaction with Panel Applications (Applets) ... 109
2. Keyboard Interaction ... 109
2.1. Keyboard Navigation ... 110
2.2. Choosing Access Keys ... 111
2.3. Choosing Shortcut Keys ... 112
2.4. Standard Application Shortcut Keys ... 113
2.5. Keyboard Interaction with Panel Applications (Applets) ... 117

11. Language 118
1. Labels 118
1.1. Controls 118
1.2. Tooltips 118
1.3. Menus 119
2. Warning and Error Messages 119
3. On-line Help 120
12. Checklists 121
1. Things You Can Do Yourself 121
1.1. Before You Start 121
1.2. Keyboard Access and Focus 121
1.3. Theming, Colors and Contrast 121
1.4. Animation 122
2. Things You Can Do With Other People 122
2.1. Get Early Feedback 122
2.2. Internationalization and Localization 122
13. Credit 123
1. Active Authors 123
2. Retired/Inactive Authors 123
3. Reviewers and Contributors 123
Bibliography 124

List of Figures

2.1. The Applications menu 5
3.1. Example of a window title 13
3.2. A typical primary window (gedit) 15
3.3. A typical SDI application (Eye of GNOME) 17
3.4. A typical MDI application (epiphany) showing three open documents on tabbed pages 17
3.5. Buttons in an explicit apply window 19
3.6. Example of a property window 19
3.7. Example of a preferences window 19
3.8. An example of a toolbox 20
3.9. A large toolbox broken into categories 21
3.10. An example of an alert 22
3.11. Primary and Secondary Text Placement 22
3.12. Button ordering and placement for alerts 23
3.13. Spacing inside an alert 23
3.14. An information alert 25
3.15. An error alert 25
3.16. A confirmation alert 26
3.17. A save confirmation alert 26
3.18. An authentication alert 27
3.19. An example of a progress window 28
3.20. A progress window for a file copy operation 29
3.21. An example checklist window (In Progress) 29
3.22. An example checklist window (Ready to Start) 29
3.23. An example checklist window (Completed) 29
3.24. An example of a dialog 30
3.25. Example of the first page of an assistant 32
4.1. A typical menubar 33
4.2. A typical drop-down menu 34
4.3. A drop-down menu with a submenu 35
4.4. A popup menu for a mail folder 35
4.5. Items grouped on a menu with separators 36
4.6. A group of command items on a menu 37
4.7. A group of check box items on a menu 38
4.8. A group of radiobutton items on a menu 38
4.9. A menubar showing all the standard menu titles in their correct order 39
4.10. A generic File menu 39
4.11. A generic Edit menu 44
4.12. A generic View menu 47
4.13. A generic Insert menu 49
4.14. A generic Format menu 51
4.15. A generic Bookmarks menu 53
4.16. A generic Go menu for a browser application 54
4.17. A generic Go menu for document-based applications 55
4.18. A generic Windows menu 55
4.19. A generic Help menu 56
5.1. Example toolbar from a web browser window 57
5.2. Example View menu fragments for applications with one toolbar (left), two or three toolbars (middle), or four or more toolbars (right) 59
5.3. Toolbar with labels under all buttons 60
5.4. Toolbar with "priority text" labels beside the first few buttons only 60
6.1. Two check boxes: sensitive (top) and insensitive (bottom) 61
6.2. Example of a dialog with locked controls 62

6.3. Single and multi-line entry fields 62
6.4. Text entry field with static text prompt 63
6.5. Text entry field requiring a date as input, with a button beside it to pop up a GtkCalendar control to simplify the task 63
6.6. Example of a spin box 64
6.7. A simple slider control 65
6.8. Slider controls with linked spin boxes 65
6.9. Typical buttons in a modal dialog 65
6.10. A typical group of check boxes 66
6.11. Ambiguous check box (top), radio buttons work better in this case (bottom) 67
6.12. Check boxes (right) showing properties for a multiple selection of files in Nautilus (left) 67
6.13. A typical group of radio buttons 68
6.14. Radio buttons (right) showing properties for a multiple selection of shapes in a drawing application (left) 68
6.15. A typical group of toggle buttons 69
6.16. Toggle buttons (right) showing properties for a multiple selection of shapes in a drawing application (left) 70
6.17. A drop-down list showing current selection (left) and the list of available choices when clicked on (right) 70
6.18. A drop-down combination box before and after its dropdown list is displayed 71
6.19. A simple two column list 72
6.20. A simple check box list 73
6.21. A simple tree control with one level of hierarchy 74
6.22. A simple check box tree 75
6.23. A typical notebook control with four tabs 75
6.24. Fixed- and proportional-width tabs (preferred) 76
6.25. Use of list control where there would be too many tabs to fit comfortably in a notebook 76
6.26. A simple 'time remaining' progress bar 77
6.27. A simple 'typical time remaining' progress bar 77
6.28. A simple 'indeterminate time' progress bar; the slider moves from left-to-right and back again until the operation is complete 77
6.29. A simple statusbar 78
6.30. An interactive statusbar 78
6.31. Preferred frame style, using bold labels, spacing and indentation 79
6.32. Traditional frame style, using borders (deprecated) 79
7.1. Busy pointer (left) and Busy-Interactive pointer (right) 83
7.2. A Checklist Window 84
8.1. The basic GNOME 32-color palette 86
8.2. How the earth looks to a user with normal color vision (left), deuteranopia (middle), and tritanopia (right). (Images from http://www.vischeck.com). 88
8.3. Improved window layout 89
8.4. Layout specifications 89
9.1. Illustration of the table perspective 95
9.2. Illustration of the shelf perspective 95
9.3. A functionally suggestive icon for a word processor 98
9.4. A functionally suggestive icon for underline 98
9.5. A name suggestive icon for Nautilus 99
9.6. Text in the old GEdit icon 99
9.7. A seemingly random icon for SodiPodi 99
9.8. Extraneous information - the Evolution icon 100
9.9. Extraneous information - the old Gnumeric icon 100
9.10. Using body parts - the font selector icon 100
9.11. A better icon for the Font Selector 100
9.12. Word play - System Log Monitor icon 101
9.13. Destructive-looking Shutdown icon 101
9.14. Part of application window rendered in high contrast, large print theme 102

9.15. Simplified representation of metaphors for high contrast icons 102
9.16. Layered technique for high contrast icons 102
9.17. Part of application window rendered in high contrast inverse, large print theme ... 103
9.18. Part of application window rendered in low contrast theme 103
9.19. Levels dialog in GIMP showing correct levels for generating low contrast icons .. 103
10.1. A plethora of pointing devices: mouse, trackball, foot-operated mouse, joystick, trackpad, and a finger-mounted pointing device. 104
10.2. Examples illustrating dynamic selection highlighting during bounding box selection. In the first example, the folder color and label highlighting changes to indicate selection. In the second, selection is indicated by the addition of resizing handles to selected objects. 107
10.3. Example of copy pointer augmented by an icon representing the file being copied 107
10.4. Dialog and menu, with some of their access and shortcut keys indicated 110

List of Tables

3.1. Properties for the GtkDialog 24
3.2. Properties for the GtkVBox (included in the dialog by default) 24
3.3. Properties for the GtkHBox 24
3.4. Properties for the GtkImage 24
3.5. Properties for the GtkLabel 25
4.1. Creation and Opening operation menu items 39
4.2. Saved State Operation menu items 40
4.3. Export Operation menu items 41
4.4. Properties menu items 42
4.5. Closing Operation menu items 43
4.6. Modification History menu items 44
4.7. Selected Data Manipulation menu items 45
4.8. Search and Replace menu items 46
4.9. User Preferences menu items 47
4.10. Toolbar and Statusbar menu items 47
4.11. Content Presentation menu items 47
4.12. Insert menu items 49
4.13. Format menu items 52
4.14. Bookmark menu items 53
4.15. Go menu items for a browser application 54
4.16. Go menu items for a document-based application 55
4.17. Windows menu items 55
4.18. Help menu items 56
7.1. Maximum acceptable response times for typical events 81
7.2. Visual feedback types for operations that take at least 1 second 84
8.1. RGB and hexadecimal values for the basic palette 86
8.2. Alignment and spacing for different Text elements 91
8.3. Capitalization Style Guidelines for User Interface Elements 93
9.1. A globe in different icon styles 95
9.2. Specifications for different kinds of icons used within GNOME 96
9.3. Simulation of low vision user viewing high contrast icons 102
10.1. Effect of modifier keys on a middle button transfer operation 104
10.2. Standard mouse and keyboard selection mechanisms 106
10.3. Effect of modifier keys during a drag and drop operation 108
10.4. Mouse Pointers for Drag and Drop 109
10.5. Standard GNOME application shortcut keys and access keys - File menu 113
10.6. Standard GNOME application shortcut keys and access keys - Edit menu 113
10.7. Standard GNOME application shortcut keys and access keys - View menu 114
10.8. Standard GNOME application shortcut keys and access keys - Bookmarks menu . 114
10.9. Standard GNOME application shortcut keys and access keys - Go menu 115
10.10. Standard GNOME application shortcut keys and access keys - Format menu 115
10.11. Standard GNOME application shortcut keys and access keys - Help menu 115
10.12. Standard window manager shortcut keys and access keys 115
10.13. Standard GNOME keyboard navigation keys for widgets 116
10.14. Emacs-style navigation keys for widgets 117

List of Examples

2.1. Including functional description in menu names 6
2.2. Removing non-essential information from menu names 6
2.3. Removing technical jargon from menu names 6
2.4. Using application's name as menu name 7
2.5. Using functional description in menu names 7
2.6. Using application's name as menu name where no functional description exists 7
2.7. Example tooltips for GNOME applications 8
2.8. Example descriptions for GConf Keys from gnome-terminal 8
3.1. Using document names as window titles 15
3.2. Using application names as window titles 16
3.3. Recommended wording for overriding theme elements- replace with screenshot 20
3.4. Firewall Setup Wizard 29
6.1. Sample code fragment showing how to make a GConf-locked control insensitive 62
9.1. Distinct silhouettes from the Nautilus Crux theme 98

Introduction

This document tells you how to create applications that look right, behave properly, and fit into the GNOME user interface as a whole. It is written for interface designers, graphic artists and software developers who will be creating software for the GNOME environment. Both specific advice on making effective use of interface elements, and the philosophy and general design principles behind the GNOME interface are covered.

These guidelines are meant to help you design and write applications that are easy to use and consistent with the GNOME desktop. Following these guidelines will have many benefits:

- Users will learn to use your program faster, because interface elements will look and behave the way they are used to.
- Novice and advanced users alike will be able accomplish tasks quickly and easily, because the interface won't be confusing or make things difficult.
- Your application will have an attractive look that fits in with the rest of the desktop.
- Your application will continue to look good when users change desktop themes, fonts and colors.
- Your application will be accessible to all users, including those with disabilities or special needs.

To help you achieve these goals, these guidelines will cover basic interface elements, how to use them and put them together effectively, and how to make your application integrate well with the desktop.

The recommendations here build on design aspects that have worked well in other systems, including Mac OS, Windows, Java and KDE. At the same time they retain a uniquely GNOME flavor.

Remember...

Following the guidelines will make your job easier, not harder!

What's new since HIG v2.0?

This section highlights recent changes to the HIG that may affect your application.

The following sections are new to this version of the HIG:

- Section 4, “Application Startup”
- Section 20, “Disclosure Triangles”

The following sections include new or revised guidelines:

- Section 4, “Designing Accessible Icons”
  - contains more information on low contrast icons
- Section 2.3, “Popup Menus”
  - clarifies when to disable items and when to hide them
- Section 5.4, “Interaction”
  - specifies that single click, not double click, should be used to activate a status notification icon
  - specifies that if clicking a notification icon opens a window, clicking it again should dismiss it (unless it's an explicit apply dialog)
- Section 1.2, “Icon”
  - suggests that an icon identical or similar to an application's launcher icon be used as the window icon for every window in the application.

Chapter 1. Usability Principles

This section explains some of the basic principles behind the more specific technical guidelines recommended in this document. We believe that these principles are important for all application development.

1. Design for People

Remember that the purpose of any software application is to enable some group of people to accomplish a specific set of tasks. So, the first things to establish when designing your application are:

1. who your users are
2. what you want to enable them to do

For example, you may be designing an application that will enable engineers (software, electrical, or mechanical) to create diagrams. You may be designing an application that will enable system administrators to configure and monitor a web server. You may be designing an application that will help elementary school students to learn math.

The important thing is that you know your audience, and you understand both their goals and the tasks necessary to achieve those goals. There are a large number of professional interaction designers who write books and teach courses on design methods that can help with this process, many of which are extremely useful— see the Bibliography for a selection. Most of these methods, however, boil down to specific ways of understanding your users, understanding the tasks you want to help them accomplish, and finding ways to support those tasks in your application.

2. Don't Limit Your User Base

If you are designing an application for use by engineers, or by children, or by system administrators, be sure to create an application that can be used by *all* engineers, children, or system administrators, including those with disabilities or those who are native speakers of a language different from yours. Be aware of accessibility issues and internationalization and localization issues, many of which are addressed by the guidelines in this document.

2.1. Accessibility

Accessibility (sometimes called *a11y*) means enabling people with disabilities of some kind to participate in life's activities: in this case, specifically to use your software. For example:

- Color-blind users may not be able to use your application if you rely only on color-coding to distinguish different types of information
- Users with hearing impairments may not be able to use your application if you rely on sounds to indicate critical information
- Users with limited movement may not be able to use your application if you don't provide keyboard equivalents for commands

Your software should also be usable with voice interfaces, screen readers such as Gnopernicus [http://developer.gnome.org/projects/gap/AT/Gnopernicus/index.html], alternate input devices, and other assistive technologies. The standard GNOME libraries do most of this work for you, but with a little extra effort you can make your application every bit as useful to users who rely on those technologies as to those who don't.

GNOME has excellent inbuilt support for accessibility by means of the ATK and GAIL libraries, which in many cases can do most of the work for you. More information on accessibility in GNOME can be found at the GNOME Accessibility Project [http://developer.gnome.org/projects/gap/].

2.2. Internationalization and Localization

Internationalization means designing software so that it can function in different language environments. Localization is the process of actually translating the messages, labels, and other interface elements of an application into another language.

GNOME has excellent support for both internationalization (also referred to as i18n) and localization (also referred to as l10n). In most cases, simply using standard GNOME APIs for displaying text and messages will allow you or others to localize your application for other locales. For more information on how to make your application localizable, see the Pango project home page [http://www.pango.org] (Pango is the GNOME library for rendering internationalized text), the GNOME Translations page [http://www.gnome.org/i18n/], and the GNOME Translation Project page [http://developer.gnome.org/projects/gtp/].

Sensitivity to cultural and political issues is also an important consideration. Designing icons and sounds, and even choosing colors requires some understanding of the connotations they might have to a user from a different part of the world.

Examples of elements it is best to avoid for these reasons include:

- Pictures of flags or money
- Maps showing political boundaries or contentious location names
- Lists of countries or cities in non-alphabetical order (unless specifically requested or required by the context)
- Icons depicting animals
- Icons depicting only hands or feet

3. Create a Match Between Your Application and the Real World

Always use words, phrases, and concepts that are familiar to the user rather than terms from the underlying system. Use terms that relate to the user's knowledge of the tasks your application supports. For example, in medicine, the paper folder that contains all information about a specific patient is called a "chart." Hence, a medical application might refer to a patient record that contains the same information as a paper chart as a "patient chart" rather than as a "patient database record."

You can often take advantage of your users' knowledge of the real world by using metaphor— that is, a familiar concept from the outside world— to represent elements within your application. For

example:

- an image of a file folder suggests a container into which documents can be placed
- a waste basket suggests a container into which items can be placed when they are no longer needed

When using metaphors, however, it is important to neither take the metaphor too literally, nor to extend the metaphor beyond its reasonable use. For example, the capacity of a file folder should not be limited to the capacity of a physical file folder, which presumably could contain only a few documents before becoming unwieldy. On the other hand, a waste basket should not be used for anything other than holding discarded files. It should not be used, for example, to eject a removable disk such as a floppy or CD.

4. Make Your Application Consistent

Make your application consistent with itself and with other applications, in both its appearance and its behavior. This is one of the most important design principles, and probably the most famous, but it is also frequently ignored. While this document serves as the basis for consistency between GNOME applications, you are encouraged to look at and follow other application's conventions where this document provides no guidelines.

Consistency enables users to apply their existing knowledge of their computing environment and other applications to understanding a new application. This not only allows users to become familiar with new applications more quickly, but also helps create a sense of comfort and trust in the overall environment. Most of the recommendations in the GNOME HI Guidelines are designed to help you create applications that are consistent with the GNOME environment and other GNOME applications.

A word of caution: a misapplied or incomplete consistency is often worse than inconsistency. If your application includes an Undo menu item for consistency, but it is always disabled because your application does not actually support Undo, this will reduce the user's trust in the availability of Undo in other applications on their desktop. Either make your application support Undo, or eliminate the Undo menu item.

5. Keep the User Informed

Always let the user know what is happening in your application by using appropriate feedback at an appropriate time. The user should never have to guess about the status of the system or of your application. When the user performs an action, provide feedback to indicate that the system has received the input and is operating on it. Feedback can be visual, audio, or both. If the system will take a long time to process the request, provide as much feedback as possible about how lengthy the operation will be. Types of helpful feedback include but are not limited to: cursor changes, animated "throbbers", progress indicators, audio feedback such as a beep, and error messages. Error messages should use simple language, clearly state the problem, and provide solutions or tell the user how to get out of the current situation if possible.

It is critical that feedback be *accurate* and *precise*. If you display a determinate progress indicator to display the state of completion of a task and it is inaccurate, the user will lose faith in progress indicators, and they will find the environment less usable. If you display a generic error message that indicates that there is a problem but fails to provide enough information to diagnose or solve the problem, your users will be unable to continue with their task.

See Chapter 7, *Feedback* and Section 4, “Alerts” for more information on feedback.

6. Keep It Simple and Pretty

Your application should enable the user to concentrate on the task at hand. So, design your application to show only useful and relevant information and interface elements. Every extra piece of information or interface control competes with the truly relevant bits of information and distracts the user from important information. Hence, don't clutter your interface, and don't overload the user with buttons, menu options, icons, or irrelevant information. Instead, use progressive disclosure and other techniques to limit what the user sees at any given moment.

Finally, present your information and interface elements in an aesthetically pleasing manner. A disorganized, cluttered-looking interface with a few elements can be just as distracting as an organized interface with too much information. Make sure that dialog elements are cleanly-aligned, and do not overuse or misuse color or graphics. If you know a graphic designer, seek their advice if possible— the guidelines in this document will help you with the basics, but there is no substitute for a trained eye.

See Chapter 8, *Visual Design* and Chapter 9, *Icons* for more information on designing the visual appearance of your application.

7. Put the User in Control

Remember that computers exist to serve humans. A user should always feel in control, able to do what they want when they want. This means you should generally avoid modes; users should be able to switch between different tasks (and specifically, different windows) at any time. See Section 1.4, “Modality” for more information on modes.

The user should also be able to tailor aspects of their environment to fit personal preferences. It is very important, however, to avoid the trap of allowing too much configuration, or allowing the configuration of parameters that most users will not understand or find useful to modify. Wherever possible, inherit visual and behavioral parameters from global preferences and settings such as the current GTK+ theme.

8. Forgive the User

We all make mistakes. Whether we're exploring and learning how to use the system, or we're experts who just hit the wrong key, we are only human. Your application should therefore allow users to quickly undo the results of their actions.

If an action is very dangerous, and there is no way to undo the result, warn the user and ask for confirmation. Only do this in extreme cases, though; if frequently faced with such confirmation messages, users begin to ignore them, making them worse than useless.

In all cases, the user's work is sacrosanct. Nothing your application does should lose or destroy user's work without explicit user action. Among other techniques, this can be achieved by auto-saving backups of documents, and allowing multiple levels of undo.

9. Provide Direct Manipulation

Wherever possible, allow users to act on objects and data directly, rather than through dialogs or explicit commands. For example, it is more intuitive to drag a circle object around in a diagram rather than selecting a "Move" command from a menu while the circle is selected. Similarly, in an email application, allow the user to attach files by dragging them from the file manager and dropping them onto the message composition window if they wish.

See Chapter 10, *User Input* for more information on direct manipulation.

Chapter 2. Desktop Integration

There are two elements to basic integration with the user environment of the GNOME Desktop.

1. Place an entry for your application in the Applications menu. This is the primary mechanism by which users discover and run applications.
2. If your application can open and save files, place entries for those file types in the application database and the document type (MIME) database. This allows the file manager and other applications to automatically launch your application when they encounter files your application can handle.

Do not add launchers or other icons to the desktop when your application is installed. The desktop is the user's space, and is reserved for icons that they explicitly request or add themselves.

1. Placing Entries in the Applications Menu

Figure 2.1. The Applications menu

The Applications menu, which appears on the panel at the top of the screen by default, is the primary mechanism by which users discover and run applications. You place entries in this menu by installing an appropriate `.desktop` file.

The menu is arranged into a set of categories, such as Accessories and Games. Applications are placed in particular categories by the set of keywords they include in their `.desktop` file.

Guidelines

- Assign your application to only one category on the Applications menu
- For application suites that wrap a number of smaller sub-applications into a single window, such as Evolution or OpenOffice.org, add a menu item for each sub-application. For example, the mail, calendar, and tasklist in Evolution should each have their own menu item.

Technical details can be found in the freedesktop.org menu [http://freedesktop.org/Standards/menu-spec] and desktop entry [http://www.freedesktop.org/Standards/desktop-entry-spec] specifications.

1.1. Menu Item Names

1.1.1. Include a functional description in the menu name

In the menu item name, include a description of functionality in addition to the proper name of the application. This is especially useful novice users, and to users of systems where numerous applications are installed by default. Users are more likely to find your application if the name that appears in the menu includes a description of its functionality.

For example, user testing of MIT's Athena system [http://web.mit.edu/is/usability/aui/] revealed

that users had difficulty finding the file manager because they were unfamiliar with the name "Nautilus". Because users did not associate the word "Nautilus" with the concept "file manager" the menu item did not help them. This is an example of not using the user's language. See Section 3, “Create a Match Between Your Application and the Real World” for more on this topic.

Example 2.1. Including functional description in menu names

Original menu item	Revised menu item
Epiphany	Epiphany Web Browser

1.1.2. Only put useful information in the menu name

Do not include words like "GNOME", "X Window System", "GTK+" or other platform details in Application menu names. The user probably already knows what platform they are using, and if they don't, then application names are not the right place to inform them.

Example 2.2. Removing non-essential information from menu names

Original menu item	Revised menu item
GNOME Image Viewer	Image Viewer
GTK Blog Editor	Blog Editor

Do not include technical details when the user does not need to know them, or can infer them from context. Avoid technical jargon unless the application is to be used only by a technical audience.

For example, when both a client and a server for something are listed in the menus, remove the word "Client" from the menu name for the client.

Example 2.3. Removing technical jargon from menu names

Original menu item	Revised menu item
Gnome Batalla Naval Client	Batalla Naval
Gnome Batalla Naval Server	Batalla Naval Multiplayer Server
Gnome VideoLAN Client	VideoLAN Movie Player

Providing the right information

Try to imagine what words users will be looking for when they select your application from the Applications menu. That is the information that should be in the menu name. For example, a user wanting to play a movie will probably not be looking for the word "Client". On the other hand, a user wanting to transmit movies from their computer may well look for the word "Server". Avoid thinking of the applications menu as an ontology!

1.1.3. Menu name formats

1. If your application's proper name is already descriptive of its functionality, and not just suggestive, use the format: *`Application Name`*

Example 2.4. Using application's name as menu name

Application name	Menu name
Dictionary	Dictionary
Search Tool	Search Tool

2. If there is a succinct functional description of your application, use the format: *`ApplicationName FunctionalDescription`*

Example 2.5. Using functional description in menu names

Application name	Menu item name
The GIMP	GIMP Image Editor
Evolution email sub-application	Evolution Email
AbiWord	AbiWord Word Processor
Galeon	Galeon Web Browser
Gramps	Gramps Genealogy
AisleRiot	AisleRiot Solitaire

3. A few applications, particularly games, do not have appropriate functional descriptions (but note that many games do). In this case, use *`Application Name`* as the menu name.

Example 2.6. Using application's name as menu name where no functional description exists

Application name	Menu item name
Bomber Maze	Bomber Maze

1.2. Menu Item Tooltips

Tooltips help provide users with enough information to run the right application. Many users use tooltips to explore a new environment.

Provide a tooltip for each Application menu item you add, following these guidelines:

Guidelines

- Phrase the tooltip as an imperative verb, for example "design", "write" or "check".
- Describe the most important tasks users can accomplish with your application.
- While tooltips should not be verbose, they should be longer and more descriptive than the item's name.

Example 2.7. Example tooltips for GNOME applications

Application	Menu item tooltip
Character Map	Insert special characters into documents
Memprof	Check your applications for memory leaks
Same Gnome	Arrange long chains of similarly-colored balls to eliminate them
Gnome Batalla Naval Client	Find and sink enemy ships in this networked version of Battleship

2. GConf Keys

GConf keys are required to have long and short descriptions for each key. Many keys have no interface through the application, so for someone administering the key values from another application each description field will be the only interface available to them.

Guidelines

- Short Descriptions should be short, less than 8 words, describing the purpose of the key
- Long Description should be complete in describing the possible values of the key and the effects that those values have on the application

Example 2.8. Example descriptions for GConf Keys from gnome-terminal

Key	Short Description	Long Description
background_type	Background type	Type of terminal background. May be "solid" for a solid color, "image" for an image, or "transparent" for pseudo-transparency.
delete_binding	Effect of the Delete key	Sets what code the delete key generates. Possible values are "ascii-del" for the ASCII DEL character, "control-h" for

Key	Short Description	Long Description
		Control-H (AKA the ASCII BS character), "escape-sequence" for the escape sequence typically bound to backspace or delete. "escape-sequence" is normally considered the correct setting for the Delete key.

3. Mapping Document Types to Applications

The document type (MIME) database allows users to specify their preferred applications for opening different types of document. This is the mechanism by which Nautilus, Evolution and other applications decide which application to run when they encounter a document they cannot open themselves.

It is important for users to be able to double click on documents they see on the desktop, such as files and email messages, and have them launch in their favorite application. Therefore, your GNOME application should associate itself at install-time with all the document types it can handle. Technical details on doing this can be found in the *GnomeVFS API reference* [http://developer.gnome.org/doc/API/gnome-vfs].

4. Application Startup

Guidelines

- Do not show a splash screen, but instead make proper use of the GNOME startup notification library to provide startup feedback.
- Show the main window of your application as early as possible, even if all its controls are disabled until the application is fully initialized. If further initialization is likely to take several seconds, consider showing some kind of progress indicator within the application window.

5. Using the Status Notification Area

The status notification area on the panel is used to notify the user of non-critical events, such as the arrival of new email, and to monitor the status of background activities, such as a laptop battery charging. Recent versions of the GNOME desktop (2.12 or newer) can also pop up a small transient "balloon" attached to a notification icon, to give additional information and interaction choices related to an event.

> **Warning**
>
> The utility of the notification area decreases rapidly when more than four icons are displayed at the same time. Icons that appear only temporarily, in response to specific events, are therefore preferable.

Following the guidelines in this section will help to clarify the difference in the user's mind

between information presented in the notification area, and controls and information presented on other parts of the panel.

5.1. Appropriate Uses for the Notification Area

There are three acceptable uses for the notification area:

- Displaying a transient icon in response to an event (e.g. arrival of new mail in a monitored folder). Clicking the icon opens the most appropriate window to deal with the event (e.g. Inbox). The icon is removed when the state prior to the event is restored (e.g. no more unread mail in monitored folders).
- Displaying an icon for the duration of a background activity (e.g. while a document is being printed). The icon is removed when the activity successfully completes, or replaced with a suitable error icon if the activity fails (optional tooltip or balloon to explain problem). Clicking the error icon (or balloon) opens the most appropriate window for the user to rectify the problem.
- Displaying an ever-present icon to monitor a continuous background activity, such as a laptop battery being charged. Continous notification icon presence should always be controllable by a user preference. Only core GNOME applications may have this preference turned on by default; other applications should turn it off by default.

In particular, you should probably use an applet instead of using a notification icon if:

- your notification icon would need to be shown all the time, or would benefit from being shown all the time

and:

- clicking your notification icon would do anything other than opening a window or dialog box *directly* associated with the icon or the event that caused it to appear , or
- multiple instances of the icon would either be required, or could be considered useful (for example, a clock-- a user might want to display a separate clock for each of multiple timezones)

Because the notification area is itself a panel applet, remember that the user may not have it on their desktop at all. Above all, therefore, only use notification icons to provide redundant, non-critical information.

5.2. Icon Appearance

Guidelines

- Use table perspective for icons representing physical devices, for example a printer icon shown during printing, with the light source above and to the left of the represented object. See Section 1.1, “Perspective” for more about table perspective.
- Use a flat, unshaded image for status monitors that take the form of a chart or graph, such as a

CPU usage monitor. Clearly delimit the borders of the chart area.

- Use shelf perspective, with overhead lighting, for all other icons. For example, an envelope shown when new mail arrives. See Section 1.1, “Perspective” for more about shelf perspective.

5.3. Icon Animation

As with any part of the desktop, animation must be used sparingly to be effective, and redundantly to be accessible.

Guidelines

- To avoid distracting the user, do not update notification icons that are monitoring a background activity any more than once per second by default. For any such icons that are perpetually shown, make the exact update frequency a user preference.
- Do not animate notification icons that are not monitoring the status of a background activity.
- Do not rely on animation, or any other change of appearance within a notification icon, as a means of alerting the user to a particular event.

5.4. Interaction

All notification icons should respond to user interaction in a consistent way, similar to that for panel applets.

Guidelines

- Perform the icon's default action when the user clicks the icon, or presses the Space key when it is focused. In general, an icon's default action should be to open a relevant window or dialog box, or to raise and focus that window or dialog box if it is already open but not focused. Examples of such windows and dialogs include:
  - the printer queue window, for a "printing in progress" icon
  - the default mail application's Inbox window, for an incoming email icon
  - the message window, for an incoming instant message icon

  If the icon's associated window or dialog box is already raised and focused, close it when the user clicks the icon, or presses the Space key when the icon is focused. All windows associated with notification icons should be dismissable in this way, except for explicit apply dialog boxes. Those should always be dismissed by clicking their action or cancel buttons.
- Present a context menu, containing at least the icon's default action, when the user right clicks the icon or presses **Shift-F10** when the icon is focused.
- If the icon's properties, or the properties of its associated application or document may be altered, include a Properties menu item in its context menu, and show its property panel in response to **Alt-Enter** when the icon is focused.

5.5. Notification Balloons

Random Thoughts

-
-
-
-
-

Chapter 3. Windows

1. Parts of Windows and System Interaction

1.1. Titles

Give every window a title (with the exception of alerts and toolboxes). A good window title contains information that is relevant to the user, and distinguishes a particular window from other open windows. Omit information that does not assist in this selection, for example the application's version number or vendor name.

Figure 3.1. Example of a window title

See the description of each particular window type for title formats.

1.2. Icon

Every window has an icon, which normally appears on its menu button. If the window appears on the window list or in the **Alt-Tab** sequence, the icon will also appear there, alongside the window title.

Always provide your own icon for this purpose, as the default icon assigned by GNOME is unlikely to convey any useful information about the window to the user. In general, every window that is part of the same application should use the same icon. Ideally, this icon should be the same or similar to the icon used to launch the application.

1.3. Borders and Window Commands

Most windows have borders, except certain shaped windows and some torn-off windows. Do not attempt to draw your own window borders, but instead provide hints to the window manager for the desired border type.

Different window commands are appropriate to different types of window. See the description of each particular window type for a list of appropriate window commands. These are the possible window commands:

- **Close.** Closes the window. *Always* draw this as a button on the window border when relevant to the window type.
- **Maximize.** Causes the window to use all unused screen space.
- **Minimize.** Causes the window to be temporarily hidden. It will continue to appear on the desktop window list.
- **Roll-up/Unroll.** Shows only the title bar of the window, as if it has been "rolled up".

1.4. Modality

A **non-modal** window does not restrict the user's interaction with other open windows on the

desktop in any way. Using non-modal windows gives the user maximum flexibility to perform tasks within your application in any order and by whichever means they choose.

An **application modal** window, while it is open, prevents the user from interacting with other windows in the same application.

A **system modal** window, while it is open, prevents the user from interacting with any other window in any application, including the desktop itself.

Guidelines

- Use an application modal window only if allowing interaction with other parts of the application while the window is open could cause data loss or some other serious problem. Provide a clear way of leaving the modal window, such as a Cancel button in an alert.
- Do not use system modal windows.

1.5. Focus

Focus is the means by which the user designates which window should receive data from the keyboard, mouse or other input device. If using a screen reader or similar assistive technology, focus may also designate the window that the user wants to receive information about. The focused window is considered the window the user is currently "working with".

Ensure your application functions properly with the three different mechanisms by which windows can receive focus in GNOME:

- **Click-to-focus.** A window is focused by clicking in it.
- **Point-to-focus.** A window is focused by moving the mouse pointer into it. Sometimes known as "sloppy focus".
- **Keyboard focus.** A window is focused by using a keyboard shortcut such as **Alt-Tab**.

Special restrictions for point to focus

Note that point-to-focus places a number of restrictions on GNOME applications that are not present in environments such as MacOS or Windows. For example, utility windows shared between multiple document windows, like the toolbox in the GIMP Image Editor, cannot be context-sensitive— that is, they cannot initiate an action such as Save on the current document. This is because while moving the mouse from the current document to the utility window, the user could inadvertently pass the pointer over a different document window, thus changing the focus and possibly saving the wrong document.

1.6. Showing and Hiding Windows

How your application shows and hides windows can greatly affect the user's perception of your application, particularly its performance.

Guidelines

- Always show a window as soon as possible, but make sure your window is the correct size before displaying it. Resizing a window after it is visible is disorienting and gives an unpolished look to your application.
- If a window contains information that takes a few seconds to compute or display, it is often better not to fill it in completely before displaying the window. For example, a window containing a large text area can be shown quickly, and then the text can be filled in afterward (provided this does not result in the window resizing). This will make your application feel more responsive than if you had not shown the window until its content was complete.
- Hide a window as soon as possible after it is closed. Unless an alert might be shown, immediately hide a window that the user has closed by clicking the Close button in the window border-- your application can still perform any internal clean-up operations afterward. Besides making the system appear slow, not doing this can cause the window manager to think the application is not responding, and display an unnecessary alert to the user.

2. Primary Windows

A primary window usually presents a view of the user's data, such as a text document in a word processor application, an image in a drawing program, or calculations in a calculator or spreadsheet application. It may also be a view of something more abstract, like a game. A single instance of an application may have more than one primary window, and more than one kind of primary window.

A primary window is always shown on the panel window list.

Figure 3.2. A typical primary window (gedit)

A primary application window normally has a border, a menubar and a statusbar, and may also contain one or more toolbars.

2.1. Title

The most important element of a document-based application's window title is the name of the open document. For other applications, it usually the name of the application.

Guidelines

- Use *`Filename`* as the window title for document-based applications. Do not use the full pathname, as the filename alone is easier to distinguish amongst other open window titles, for example on the window list.

Example 3.1. Using document names as window titles

Application	Example window title
AbiWord	My Report.abw
Evolution	Inbox
Music player	U2 - Better Than the Real Thing

If the pathname is important, for example the user has opened two documents with the same name from different directories in the same application, show the full pathname in the statusbar.

- Before a new document has been saved for the first time, set the window title to *Unsaved <document type>*. For example, *Unsaved Drawing*, *Unsaved Spreadsheet*, or the more generic *Unsaved Document*.
- When a document has pending changes, insert an asterisk (*) at the beginning of the window title. For example, **Unsaved Drawing*, **AnnualReport*.
- For non-document-based applications, use *Application Name* as the window title.

Example 3.2. Using application names as window titles

Application	Window title
Dictionary	Dictionary
Calculator	Calculator

- Do not place version numbers, company names, or other information that is of no immediate use to the user in the window title. These consume space, making titles in limited spaces such as the system window list less useful, and add more text the user has to scan to find useful information. In a "beta" product, where version numbers are critical for bug information, placing version numbers can be useful, but remove them from stable releases. Place version information in the about box instead.

While document names are most pertinent to users, we understand that application developers may want to increase recognition of their application. If you plan to include your application's name in the title of a primary window, use the following format: *Document Name - Application Name*. This will ensure that the document name appears in limited space situations such as the system window list.

Warning

Including the application name in the title of a document-based application is **not** recommended.

Tip

Think about naming windows in the context of the panel window list. On a typical screen with a relatively small number of windows open, a window will have 20-30 characters of text and an icon. Consider which text will provide the most immediately obvious clues to a user looking for a particular window.

2.2. Window Commands

Close, Maximize/Restore, Minimize, Roll-up/Unroll

2.3. Relation between Documents and Windows

2.3.1. Single Document Interface (SDI)

A single document interface places each document in its own primary window. Toolboxes and other utility windows may be shared between multiple SDI documents, but closing them should have no effect on the document windows. Use SDI for your GNOME application unless there is a compelling reason not to.

Figure 3.3. A typical SDI application (Eye of GNOME)

2.3.2. Multiple Document Interface (MDI)

A multiple document interface presents a paned, tabbed or similar presentation of two documents within a single window.

Figure 3.4. A typical MDI application (epiphany) showing three open documents on tabbed pages

MDI has several inherent usability problems, so its use is discouraged in applications. It is better to open each document in a new primary window, with its own menubar, toolbars and statusbar, or allow multiple instances of your application to be run simultaneously. In either case, this leaves it for the window manager (acting on the user's preferences) rather than your application to decide how to group and present document windows from the same application.

2.3.3. Controlled Single Document Interface (CSDI)

In a typical SDI application, document windows are treated as primary. For example, when all document windows have been closed, the application (including utility windows) exits as well. In CSDI a utility window is treated as the primary window. For example, closing this utility window will close all document windows and exit the application.

Warning

Using CSDI is **not** recommended

CSDI is sometimes used because document windows might be too small to have menu bars. Typically this is not the normal use case for the application, but does represent a significant minority use case. For example, an image editor being used to edit small web page elements will often result in very small document windows that cannot accomodate a title bar.

A better way to address this problem is to allow menu bars to "collapse" into an overflow button, in much the same way toolbars operate when the window shrinks to below the toolbar width. This allows for small windows, but also provides an opportunity for people to figure out where their menus have gone.

Tip

Note that if very small documents are the *primary* use case for your application, you should consider finding a means to avoid windows altogether. Windows are not an

effective interface for dealing with large numbers of small items. Consider looking for a fixed/automated layout system for presenting the "documents". Also consider if the "documents" will be primarily used in a higher level grouping, in which case that grouping could become the document instead.

3. Utility Windows

Utility windows, such as palettes and toolboxes, normally have borders. They do not contain a menu bar, a toolbar, or a statusbar.

A utility window should not appear in the panel window list unless it is, or may be, the only window shown by an application. Otherwise, the utility window should be raised above the application when the application window itself is selected from the window list.

3.1. Instant apply windows

For windows that allow the user to change values or settings, such as property and preference windows, update those values or settings immediately to reflect the changes made in the window. This is known as "instant apply". Do not make the user press an OK or Apply button to make the changes happen, unless either:

- the change will take more than about one second to apply, in which case applying the change immediately could make the system feel slow or unresponsive, or
- the changes in the window have to be applied simultaneously to prevent the system entering a potentially unstable state. For example, the hostname and proxy fields in a network properties window.

If either these conditions affect only a few of the controls in your window, arrange those controls together into one or more groups, each with its own Apply button. Leave the rest of the controls as instant apply.

Guidelines

- Do not attempt to validate or apply changes caused by editing a text field control until the user has moved focus to a different control in the window, or the window is closed. Validating after each keypress is usually annoying and unnecessary. Exception: if the field accepts only a fixed number of characters, such as a hexadecimal color code, validate and apply the change as soon as that number of characters have been entered.
- When the user moves focus to a different control, do not indicate an invalid entry by displaying an alert or undoing the change the user made. Both of these methods are particularly disruptive for focus-follows-mouse users, for whom focus may leave the control more often than it does for a click-to-focus user.

3.2. Explicit apply windows

If most of the controls in your window are not suitable for instant apply, consider making the whole window "explicit apply". An explicit apply window has these three buttons in its button box, plus an optional Help button:

- **Apply.** Applies all the settings in the window, but does not close the window in case the user wishes to change their mind.
- **Cancel.** Resets all settings in the window to those that were in force when the window was opened. Note: this must undo the effects of all applications of the Apply since the window was opened, not just the most recent one.
- **OK.** Applies all settings in the window, and closes the window.

Figure 3.5. Buttons in an explicit apply window

3.3. Default Buttons

When designing a dialog or utility window, you can assign the **Return** key to activate a particular button in the window. GNOME indicates this button to the user by drawing a different border around it. For example, the OK button in Figure 3.5, “Buttons in an explicit apply window”.

Choose the default button to be the most likely action, such as a confirmation action or an action that applies changes in a utility window. Do not make a button the default if its action is irreversible, destructive or otherwise inconvenient to the user. If there is no appropriate button in your window, to designate as the default button, do not set one.

In particular, it is currently *not* recommended to make the Close button the default in an instant apply window, as this can lead to users closing the window accidentally before they have finished using it.

3.4. Property Windows

Property windows allow the user to view and change the characteristics of an object such as a document, file, drawing, or application launcher.

Figure 3.6. Example of a property window

Icon: Same as application

Title Format: `Object Name` Properties

Window Commands: Close, Minimize, Roll-up/Unroll

Buttons: Place a Close button in the lower right corner. A Help may be placed in the lower left corner.

3.5. Preferences Windows

Preferences windows allow the user to change the way an application looks or behaves.

Figure 3.7. Example of a preferences window

Icon: Same as application

Title Format: *`Application Name`* Preferences

Window Commands: Close, Minimize, Roll-up/Unroll

Buttons: Place a Close button in the lower right corner. A Help may be placed in the lower left corner.

3.5.1. Customizing Fonts and Colors

If your preferences window allows the user to customize fonts or colors, use the following wording and layout as a guide for these controls:

Example 3.3. Recommended wording for overriding theme elements- replace with screenshot

```
(o) Use font from theme
(o) Use this font: [ Font selector ]

(o) Use colors from theme
(o) Use these colors:
        Background: [ color selector ]
        Foreground: [ color selector ]
```

The wording of the radio buttons may be more specific where required, for example, "Use monspace font from theme", or "Use background color from theme".

3.6. Toolboxes

A toolbox provides convenient access to a set of actions and toggles through a set of small toolbar-like buttons. Toolboxes can be used to provide a specialized group of tools to augment a toolbar containing more universal items such as Save and open. A single toolbox can be shared between multiple documents to save screen space.

Figure 3.8. An example of a toolbox

Icon: Same as application

Title Format: Toolboxes have no title

Window Commands: Close, Roll-up/Unroll

Buttons: Toolboxes have no buttons

Resizing: Make toolboxes resizable, but only resize by discrete toolbox item widths. In other words, the user can resize the toolbox to be one item wide, two items wide, three items wide, etc. but not one and a half items wide.

Guidelines

- Only place buttons in a toolbox that do not open another window.
- Toolboxes are best used for modal toggle buttons that affect the operation of the mouse on the document, such as a set of buttons for choosing between paintbrush, eraser, and fill modes in a drawing application. Buttons that initiate actions upon clicking (such as a save button) are better placed in toolbars.
- Ensure that closing a toolbox does not close or otherwise alter any primary window with which it is associated.
- Do not place toolboxes in the system window list. Toolboxes should always remain above all primary windows with which they are associated.
- If all primary windows associated with a toolbox are closed or minimized, hide the toolbox as well. Show the toolbox again when one of the primary windows is opened or restored.
- Make a toolbox two items wide by default, unless it is broken into categories. Make categorized toolboxes four items wide by default.

3.6.1. Toolbox Categories

While categories may not be as visually appealing as a toolbox homogeneously filled with beautiful icons, they make an unwieldy large toolbox more manageable. Picking a small icon from more than fifteen other items is a difficult task. Additionally, categories allow users to hide sets of tool items that are not relevant to their current task.

Figure 3.9. A large toolbox broken into categories

Guidelines

- Break toolboxes with more than sixteen items into categories. The best size for a category is between four and ten items.
- Give each category a label (in title caps) and a collapsing arrow. Clicking the label or the arrow toggles the category between a collapsed and uncollapsed state.

4. Alerts

An alert provides information about the state of the application system, or asks for essential information about how to proceed with a particular task. It is distinct from other types of window in that it is not directly requested by the user, and usually contains a message or a question rather than editable controls. Since alerts are an unwelcome intrusion into the user's work, do not use them except where necessary to avoid potential data loss or other serious problems.

An alert has a border similar to that of a dialog, and is object modal.

An alert should not appear in the panel window list unless it is, or may be, the only window shown

by an application. For example, an appointment reminder alert may be shown after the main calendar application window has been closed.

Otherwise, an alert should be raised above the application when the application window itself is selected from the window list.

Figure 3.10. An example of an alert

Icon: Same as application

Title Format. Alert windows have no titles, as the title would usually unnecessarily duplicate the alert's primary text. This way, users can read and respond to alerts more quickly as there is less visual noise and confounding text.

Resizing. Alert windows are not resizable. If the user needs to resize your alert, the text is probably not concise enough.

Window Commands: None

> ### Alerts must stay above their parent
>
> Alerts do not appear in the system window list. Consequently, take care to ensure that alerts stay above their parent window. Otherwise, users will be likely to lose the alert and find your application unresponsive for no apparent reason. Modal windows should always stay above the window(s) they block.

4.1. Alert Text

An alert may contain both primary and secondary text. The primary text briefly summarizes the situation. The secondary text provides additional information.

Make both the primary and secondary text selectable. This makes it easy for the user to copy and paste the text to another window, such as an email message.

Figure 3.11. Primary and Secondary Text Placement

Primary Text. The primary text provides the user with a one sentence summary of the information or suggested action. This summary should concisely contain the essential details of the problem or suggestion. Every alert has primary text, displayed in a bold font slightly larger than the default. The primary text is punctuated in 'newspaper headline' style, that is, it has no terminating period, but it may have a terminating question mark.

Denote primary text with the pango markup:

```
<span weight="bold"
      size="larger">Primary Text</span>
```

Secondary Text. Secondary text provides a more in-depth description of the problem and suggested action, including possible side effects. Secondary text can also provide information that may be helpful in allowing the user to make an informed decision. In most situations the user

should only need the primary text to make a quick decision, but they may read the secondary text if they are unsure of the proper course of action, or require extra details. Secondary text is optional, but if used, place it one text line height beneath the primary text using the default font size and weight.

4.2. Alert Buttons

Give all alerts an affirmative button that dismisses the alert and performs the action suggested in the primary text. Provide a Cancel button for all alerts displayed in response to a user actions, such as Quit. If the alert warns of a technical problem or other situation that could result in data loss, provide a Help button that provides more information on the particular situation and explains the user's options. You may also provide buttons to perform alternate actions that provide another possible solution, fix potential problems, or launch related dialogs or programs.

Figure 3.12. Button ordering and placement for alerts

Button Phrasing. Write button labels as imperative verbs, for example Save, Print. This allows users to select an action with less hesitation. An active phrase also fits best with the button's role in initiating actions, as contrasted with a more passive phrase. For example Find and Log In are better buttons than than Yes and OK.

- **Affirmative Button.** Place the affirmative button in the lower right corner of the alert. The affirmative button accepts the action proposed by the alert, or simply dismisses the alert if no action is suggested (as is the case with an information alert).
- **Cancel Button.** If the alert was produced in response to a user action, place a Cancel button immediately to the left of the affirmative button. This provides an escape route for users to stop an action in response to new information, or just if they clicked accidentally. Clicking the Cancel button reverts the application to its state prior to the user action.
- **Help Button.** A Help button may be used to clarify alerts that present potentially destructive options. Place the Help button in the lower left corner of the alert. When clicked, launch a help window clarifying the situation, detailing the actions performed by the other buttons, and explaining any side-effects that each action may have.
- **Alternate Buttons.** Extra buttons may be used to provide alternates to the primary action proposed by the alert text. Place these buttons to the left of the Cancel button, or the affirmative button if Cancel is not present. An example of a common alternate action would be a Quit without Saving button in a save confirmation alert. This is an alternative to the primary suggested action Save and the Cancel button.

4.3. Spacing and Positioning Inside Alerts

Using clear, consistent spacing in alerts makes the message easier to digest and the available responses more obvious.

Figure 3.13. Spacing inside an alert

Guidelines

- The border around all edges of the alert, and the space between the icon and the text, is 12 pixels.
- The horizontal spacing between the buttons is 6 pixels.
- Add one line break at the standard font size below both the primary and secondary text, or 24 pixels if you are using Glade.
- Align the top of the icon with the top of the primary text.
- Left-align the message text, for western locales.

Technical Details for Proper Layout

Create a new GtkDialog window specifying the number of buttons you wish the alert to contain (and a help button if appropriate). The GtkDialog will contain a GtkVBox with an empty upper row, and a lower row containing a GtkButtonBox with buttons in it. In the empty upper row, place a new GtkHBox. In the left column of the GtkHBox place a GtkImage. In the right column of the GtkHBox place a GtkLabel. Inside the GtkLabel place `Primary Text` first (using the appropriate Pango markup, see Section 4.1, “Alert Text”), then put two linebreaks (return), then place `Secondary Text`. Now change the properties for each control according to these tables:

Table 3.1. Properties for the GtkDialog

Property	Value
Title	(none)
Border Width	6
Type	Top Level
Resizable	No
Has Separator	No

Table 3.2. Properties for the GtkVBox (included in the dialog by default)

Property	Value
Spacing	12

Table 3.3. Properties for the GtkHBox

Property	Value
Spacing	12
Border Width	6

Table 3.4. Properties for the GtkImage

Property	Value
Y Align	0.00
Icon Size	Dialog

Table 3.5. Properties for the GtkLabel

Property	Value
Use Markup	Yes
Wrap Text	Yes
Y Align	0.00

4.4. Information Alerts

Use an information alert when the user must know the information presented before continuing, or has specifically requested the information. Present less important information by other means such as a statusbar message.

Figure 3.14. An information alert

An information alert...

- uses the stock information icon.
- presents a selectable message and an OK button. The button is placed in the bottom right corner of the alert. Pressing Enter or Escape dismisses the alert.
- may present a convenience button to give access to a relevant object. For example, a Details button in an appointment reminder alert that opens the appointment's property window. Place this button to the left of the affirmative button.

Window Commands: Roll-up/Unroll, Minimize (if the alert has no parent window), Close

4.5. Error Alerts

Display an error alert when a user-requested operation cannot be successfully completed. Present errors caused by operations not requested by the user by another means, unless the error could result in data loss or other serious problems. For example, an error encountered during an email check initiated by the user clicking a toolbar button should present an error alert. However, an error encountered in an automated periodic email check would more appropriately report failure with a statusbar message.

Figure 3.15. An error alert

An error alert...

- uses the stock error icon.
- presents a selectable message and an OK button. The button is placed in the bottom-right corner of the alert. Pressing Enter may dismiss the error alert.
- may present a convenience button to allow immediate handling of the error. For example, a Format... button in a "This disk is not formatted" alert. Place this button to the left of the affirmative button.

Window Commands: Roll-up/Unroll

4.6. Confirmation Alerts

Present a confirmation alert when the user's command may destroy their data, create a security risk, or take more than 30 seconds of user effort to recover from if it was selected in error.

Figure 3.16. A confirmation alert

A confirmation alert...

- uses the stock warning icon.
- presents a selectable message and a button labeled with a verb or verb phrase describing the action to be confirmed, or labeled OK if such a phrase would be longer than three words. This button is placed in the bottom right corner of the alert.
- presents a Cancel button that will prevent execution of the user's command. This button is placed to the immediate left of the OK or equivalent button.
- may present an alternate action button or a convenience button. Place this button to the left of the Cancel button.

Window Commands: Roll-up/Unroll

4.6.1. Save Confirmation Alerts

Save confirmation alerts help ensure that users do not lose document changes when they close applications. This makes closing applications a less dangerous operation.

Figure 3.17. A save confirmation alert

Primary Text. Save changes to document *`Document Name`* before closing?

You may replace “document” with a more appropriate description, for example “image” or

“diagram” if the document in question is not primarily text.

Secondary Text. If you close without saving, changes from the last `Time Period` will be discarded

The secondary text provides the user with some context about the number of changes that might be unsaved.

Buttons. Close without Saving, Cancel, Save

When a confirmation alert is needed, present it immediately. If the user confirms closing without saving, hide the alert and the document or application window immediately, before doing any necessary internal clean-up. If the user chooses to save before closing, hide the alert immediately but show the document window until the document is saved, in case an error occurs. Then hide the document window immediately after it has been saved successfully.

4.7. Authentication Alerts

Authentication alerts prompt the user for information necessary to gain access to protected resources, such as their username or password. Authentication alerts are a special kind of alert because they are both routine and largely unavoidable. Every attempt should be made to retain information entered into an authentication alert as long as is possible within security constraints.

Figure 3.18. An authentication alert

Guidelines

- Use the stock authentication icon.
- Show a labeled field for each required item of information. Suggested fields are Username and Password (in that order) where appropriate.
- If it is secure to retain the username longer than the password, pre-fill the username field and give focus to the password field when the alert is displayed.
- Show a button labeled with a verb or verb phrase describing the authentication action, or OK if there is no appropriate phrase or such a phrase would be longer than three words. Place this button in the bottom right corner of the alert.
- Do not enable the OK or equivalent button until all fields that require input have been attended to by the user. Remember that not all fields may require input however, for example an empty password may be acceptable in some applications.
- Show a Cancel button that will prevent authentication and close the alert. Place this button to the immediate left of the OK or equivalent button.
- Place any alternative action or convenience button to the left of the Cancel button.
- When the user presses **Return** in the last field, activate the default button. When the user presses **Return** in any other field, move focus to the next field.

Window Commands: Roll-up/Unroll

5. Progress Windows

A progress window can be used to provide feedback during an operation that takes more than a few seconds. See Section 17, “Progress Bars” for more details about proper use of progress bars.

A progress window may appear in the panel window list if it is, or may be, the only window shown by an application. For example, the file download progress window of a web browser may remain after all the browser windows have been closed.

Otherwise, a progress window should be raised above the application when the application window itself is selected from the window list.

Figure 3.19. An example of a progress window

Icon: Same as application

Title Format. Progress windows should have the same title as their primary text. Unlike alerts, it is expected that progress windows will be present in the window list and may be active for extended periods.

Resizing. Progress windows should be resizable if they contain non-static information the user may want to copy (for example, the source URL in a download progress window). Otherwise they should not be resizable.

Guidelines

- It is often better to use the progress bar contained in many primary windows' statusbar rather than a progress window. See Section 4.2.1.1, “Progress Windows vs. the Statusbar” for details on choosing between the two.
- Progress windows should use primary and secondary text like an alert. See Section 4.1, “Alert Text”
- The progress bar text should provide an idea of how much work has been completed. It is better to provide specific information rather than a unit-less percentage. For example, "13 of 19 images rotated" or "12.1 of 30 MB downloaded" rather than "13% complete".
- If possible, an estimate of the time left until the operation is complete should also be included in the progress bar text. Indicate that the "time left" is an estimate using the word "about".
- Immediately beneath the progress bar, place italicized text indicating the current sub-operation being performed. This might be a step in a sequence, "Contacting control tower for permission to land", or it could be the current object being operated on in a bulk operation, "Rotating MonaLisa.png", "Rotating StarryNight.png".
- If the operation in progress is potentially hazardous (destructive, costly, etc) or heavily taxes a limited resource for more than ten seconds (network bandwidth, hard disk, CPU, etc), consider placing a Pause toggle button to the right of the Cancel button. When paused, the italicized current sub-operation text should have " (Paused)" appended. This will allow users to perform important tasks requiring that resource, or give them time to think whether they want to proceed with a dangerous operation they inadvertently triggered.

Figure 3.20. A progress window for a file copy operation

5.1. Checklist Windows

Occasionally a procedure is comprised of a series of user performable actions. In these cases, particularly when it is desirable that the user acquire some familiarity with the actions involved in a procedure, checklist windows may be used.

Example 3.4. Firewall Setup Wizard

A personal firewall setup wizard might install the firewall package, add entries for the firewall to /etc/xinetd.conf, restart the internet super-daemon, and configure the user's web browser to operate through the firewall. It may be desirable that the user is exposed the series of actions involved in setting up the firewall to increase the chances that they will be successful in making modifications later, if they so desire.

Figure 3.21. An example checklist window (In Progress)

Figure 3.22. An example checklist window (Ready to Start)

Figure 3.23. An example checklist window (Completed)

Guidelines

- If knowing the series of steps in an operation isn't that useful to the user, just use a regular progress window. Remember that you are probably more interested in the information than most users, many of whom will find the technical steps confusing rather than helpful.
- Unlike regular progress windows, checklist windows should not close automatically when the operation is complete and should require explicit user input before they begin. This is because one of their purposes is to inform the user concerning an operation's contingent steps.
- The progress bar indicates progress in the overall operation, not each step. While this is more difficult to program, it is the information most useful to the user. Just estimate how long each of the steps takes relative to each other and assign each step a fixed ratio of the progress bar's progress accordingly.
- Do *not* use a checklist window for a series of internal programmatic steps, use a regular progress window. For example "Connect to mail server", "Authenticate with mail server", "Download messages", "Disconnect" would *not* be an appropriate series of steps for a checklist window, but would be appropriate sub-operation steps for a regular progress window.

6. Dialogs

A dialog provides an exchange of information, or dialog, between the user and the application. Use a dialog to obtain additional information from the user that is needed to carry out a particular command or task.

A dialog should not appear in the panel window list. Any open dialogs should be raised above the application when the application window itself is selected from the window list.

Figure 3.24. An example of a dialog

Icon: Same as application

Title Format: `Name of command that opened the dialog` (without any trailing ellipsis)

Window Commands: Minimize, Roll-up/Unroll

Buttons: Follow the guidelines for Alert buttons, see Section 4.2, “Alert Buttons”.

Your dialog may specify a default button, that is activated when the user presses the **Return** key. See Section 3.3, “Default Buttons” for guidance on choosing an appropriate default button.

6.1. Additional Buttons

You can include other buttons in a dialog's main button area in addition to the affirmative button and Cancel, but any more than one or two such buttons will make the dialog appear complicated and difficult to use. As with any other button, keep the labels as concise as possible to minimize this effect.

Guidelines

- Place buttons that apply to the dialog as a whole in the main button area row at the bottom of the dialog, to the left of the Cancel button.
- Place buttons that apply to one or a few controls next to their associated controls. For instance, place a Browse... button at the trailing edge of the text field it fills in.

6.2. Layout

A clean, logical dialog layout helps the user to quickly understand what information is required from them.

Guidelines

- Arrange controls in your dialog in the direction that people read. In western locales, this is generally left-to-right, top-to-bottom. Position the main controls with which the user will interact as close to the upper left corner as possible. Follow similar guidelines for arranging controls within groups in the dialog, and for specifying the order in which controls are traversed using the **Tab** key.

- When opening a dialog, provide initial keyboard focus to the component that you expect users to operate first. This focus is especially important for users who must use a keyboard to navigate your application.
- Provide and show sensible default values for as many of the controls in your dialog as possible when it is opened, so the user does not have to generate the information from scratch. These defaults may come from system settings (for example, hostname or IP address), or from information that the user has previously entered in this or another application (for example, email address or network proxy).

See Chapter 8, *Visual Design* for more detailed information on arranging controls in dialogs.

See Section 16, “Tabbed Notebooks” for information on using tabbed notebook controls in dialogs.

6.3. Common Dialogs

The gtk and GNOME libraries provide standard dialogs for many common tasks, including opening and saving files, choosing fonts and colors, and printing. Always use these when the user is performing one of these tasks. You may modify the dialogs to reflect the needs of your particular application (for example, adding preview Play and Stop buttons to the Open File dialog in an audio application), but do not change or remove features so much as to make them unrecognizable.

7. Assistants

An assistant is a secondary window that guides the user through an operation by breaking it into sequential steps. Assistants are useful for making complex operations less intimidating, as they restrict the information visible to the user at any given moment.

Because assistants provide a relatively small number of controls on the screen at any given time, they have sufficient space for inline documentation. Therefore, do not include a Help button in an assistant window. If you cannot make an operation sufficiently clear in an assistant without resorting to a Help button, you need to simplify it further.

Assistants do have major downsides. After using an assistant it is often hard to figure out where the individual settings aggregated into the assistant are stored. Often people will resort to re-running the assistant, re-entering many settings that they don't want to change.

Warning

Assistants are often used in situations where a better solution would be to simplify, or even better automate, the process. Before using an assistant to step people through a complex operation, consider if the operation can be fundamentally simplified so an assistant is unnecessary.

Window Commands: Close, Minimize/Unminimize, Roll-up/Unroll

7.1. Introductory Page

The first page provides the user with the "big picture". Place the title of the assistant in the window's title bar and the assistant's title area, along with an optional picture. Beneath this, state the goal of the assistant, and, if it is not obvious, where the user can find the information the assistant will be asking for.

Icon: Same as application

Title Format: *`Assistant Title`*

Buttons: Cancel, Forward

Figure 3.25. Example of the first page of an assistant

7.2. Content Pages

Content pages contain the actual settings of the assistant. Summarize the type of setting present on each content page in its title area. For example, Mail Server.

Title Format: *`Assistant Title`* - (*`Current Page`* of *`Total Pages`*)

Buttons: Cancel, Back, Forward

7.3. Last Page

The last page should summarize the settings that will be changed by the assistant, and how the user can modify them later.

Title Format: Finish *`Assistant Title`*

Buttons: Cancel, Back, Finish

Chapter 4. Menus

Menus present the whole range of an application's commands to the user, and often a subset of its preferences. When designing a new application, place common menu items in the same locations as they appear in other applications, as this makes it much easier for the user to learn.

In most applications, only primary windows should have a menubar. Utility windows and dialogs should be simple enough that their functions can be provided by controls such as buttons placed within the window.

Occasionally, however, a utility window or dialog is so complex that there would be too many such controls. In this case, you may use a menubar provided that:

- the menus follow the same standard layout as described in Section 4, “Standard Menus”
- the window does not include a dialog button area or any buttons that dismiss it, such as OK, Close or Cancel. Place these commands on the File menu or equivalent instead.

Guidelines

- Label menu items with verbs for commands and adjectives for settings, according to the rules in Section 3.2, “Capitalization”.
- Make a menu item insensitive when its command is unavailable. For example, the Edit → Copy item, which issues the command to copy selected data to the clipboard, should not be active when there is no data selected.
- Provide an access key for every menu item. You may use the same access key on different menus in your application, but avoid duplicating access keys on the same menu. Note that unlike other controls, once a menu is displayed, its access keys may be used by just typing the letter; it is not necessary to press the Alt key at the same time.
- Design your menu structure to avoid more than one level of submenus. Deep menu hierarchies are harder to memorize and physically difficult to navigate.
- Do not have menus with less than three items on them (except the standard Help menu, which has only two items by default). If you have a submenu with fewer than three items on it, move them into their parent menu. If you have a top-level menu with fewer than three items on it, find another suitable menu to add them to, or find suitable items from other menus to add to it.

1. The Menubar

Figure 4.1. A typical menubar

The menubar provides a number of drop-down menus. Only the menu titles are displayed, until the user clicks on one of them.

The menubar is normally visible at all times and is always accessible from the keyboard, so make all the commands available in your application available on the menubar.

Full screen mode

When your application is running in full screen mode, hide the menubar by default. However, make its menus and items accessible from the keyboard as usual. Pressing **ESC** should cause the application to leave full screen mode. A Leave Fullscreen button should be placed in the upper right hand corner of the window. The button should disappear after the mouse is unused for 5 seconds, and should appear again when the moused is moved. Alternately, in applications where the mouse is used frequently in full screen mode, all but a two pixel row of the button may be slid off the top of the screen. The button should slide back on the screen when the mouse moves near it.

Guidelines

- Provide a menubar in each primary application window, containing at least a File and a Help menu.
- Organize menu titles in the standard order--; see Section 4, “Standard Menus”
- Do not disable menu titles. Allow the user to explore the menu, even though there might be no available items on it at that time.
- Menu titles on a menubar are single words with their first letter capitalized. Do not use spaces in menu titles, as this makes them easily-mistaken for two separate menu titles. Do not use compound words (such as WindowOptions) or hyphens (such as Window-Options) to circumvent this guideline.
- Do not provide a mechanism for hiding the menubar, as this may be activated accidentally. Some users will not be able to figure out how to get the menu bar back in this case.

2. Types of Menu

2.1. Drop-down Menus

Figure 4.2. A typical drop-down menu

A drop-down menu appears when the user clicks on its title in a menubar, or focuses the title and presses **Return**.

Guidelines

- Only place items on a menu that relate to that menu's title.
- Organize menu items in the standard order--; see Section 4, “Standard Menus”. For application-specific items where there is no standard order, arrange in numerical or other logical order (for example, 50%, 100%, 200%), task order (for example, Compile followed by Debug) or by expected frequency of use.

- Limit top-level menus to a maximum of about 15 items. If you have any more items than this, consider moving a functionally-related subset of the items into a submenu or a new top-level menu.
- Do not add or remove individual menu items while the application is running, make them insensitive instead. Entire menus may be added or removed from the menubar at runtime, however, for example in component-based applications.
- Immediately update menu items that are edited directly or indirectly by the user, such as those on the Open Recent submenu and the Bookmarks menu.

2.2. Submenus

Figure 4.3. A drop-down menu with a submenu

A submenu appears when the user clicks its title, which is indicated by a small arrow symbol beside its label. You can save space on long menus by grouping related commands onto a single submenu.

Guidelines

- Use submenus sparingly, as they are physically difficult to navigate and make it harder to find and reach the items they contain.
- Do not create submenus with fewer than three items, unless the items are added dynamically (for example the File → New Tab submenu in gnome-terminal).
- Do not nest submenus within submenus. More than two levels of hierarchy are difficult to memorize and navigate.

2.3. Popup Menus

Figure 4.4. A popup menu for a mail folder

Popup menus provide shortcuts to those menu items that are applicable only to the currently selected object. As such, they are sometimes known as "context menus" or "shortcut menus". A popup menu is shown when the user right-clicks on an object, or selects the object and presses **Shift-F10**.

Popup menus are used primarily by intermediate and advanced users. Even some users who have used graphical desktops for many years do not know about popup menus until somebody shows them.

Guidelines

- Provide a popup menu for every object, selectable part, and text input target such as entry

fields.

- Provide the same popup menu for all objects of the same type. If the properties of the selected instance make any of an object's popup menu items temporarily inappropriate, make those items insensitive rather than hiding them.
- Provide an access key for each item. Do not show keyboard shortcuts in popup menus however; doing so decreases their readability and spatial efficiency.
- Since the user may not be aware of their presence, do not provide functions that are only accessible from popup menus unless you are confident that your target users will know how to use popup menus.
- Order items on a popup menu as follows:
  - the double-click action for object, when it exists
  - other commands and settings in expected frequency-of-use order
  - transfer commands such as **Cut**, **Copy**, and **Paste**
  - Input Methods, where applicable. Input Methods is provided by GTK+ for supporting alternatives to the keyboard for input (such as used for Japanese, Chinese, and some accessibility technologies).
- Popup menus need to be as simple as possible to maximize their efficiency. Do not place more than about ten items on a popup menu, and do avoid submenus.

3. Designing a Menu

3.1. Grouping Menu Items

Figure 4.5. Items grouped on a menu with separators

Menu separators are the horizontal dividing lines that visually separate groups of related items on a drop-down menu, submenu, or popup menu. For example, the separators in Figure 4.5, “Items grouped on a menu with separators” divide the menu into five functionally-related groups. Good use of separators helps to "chunk" the information on a menu and make it easier to scan and memorize.

Guidelines

- The best size for a group is around 2-5 items. Single-item groups are best placed at the top or bottom of a menu, otherwise try to group them with other single items of the same type on the same menu.
- Order items within a group logically, numerically, in task order or by expected frequency of use, as appropriate.
- Only place one type of menu item in each group--; command, mutable, check box or radio button. For example, do not place commands (such as View → Reload) and settings (such as

View → Toolbar) in the same group.

3.2. Types of menu item

3.2.1. Command Items

Figure 4.6. A group of command items on a menu

Command items are menu items that initiate a command or perform an action, such as Save, Print or Quit. They may act on the currently active document in a document based application, or on the application itself.

Guidelines

- Provide a keyboard shortcut for standard or frequently used command items. See Section 2.3, “Choosing Shortcut Keys” for more information on choosing shortcut keys.
- Do not remove command items from the menu when they are unavailable, make them insensitive instead. This allows the user to infer what functionality the application provides even if it is not currently available, and keeping the menu structure static makes it easier to memorize.
- Label the menu item with a trailing ellipsis ("...") only if the command requires further input from the user before it can be performed. Do not add an ellipsis to items that only present a confirmation dialog (such as Delete), or that do not *require* further input (such as Properties, Preferences or About).

3.2.2. Mutable Command Items

A mutable command item changes its label when selected. For example, View → Reload in a browser may change to Stop to allow the user to interrupt the operation if it is taking a long time.

Note that mutable menu items can be problematic because the user never sees the menu item changing, so it is not obvious that a different function has become available.

Guidelines

- If your mutable menu items are command items, and you have sufficient space on your menu, consider providing two adjacent menu items for the commands instead. Then make the items sensitive or insensitive as the situation demands. This also makes it easier for the user to tell when different shortcuts are available for each of the commands, for example **Ctrl**-**R** for Reload, and **Esc** for Stop.
- Do not use mutable menu items to toggle a two-state setting (for example, Show Toolbar and Hide Toolbar). Present such items as a single check box item instead.

3.2.3. Checkbox Items

Figure 4.7. A group of check box items on a menu

A check box menu item shows the current state of a two-state setting, and allows the user to toggle it by selecting the menu item.

Guidelines

- Use a check box menu item only when it is obvious from the label what the set and unset states mean. This usually means that the two states are logical or natural opposites, such as "on" and "off". If this is not the case, use two radio button items instead.
- Never change the label of a check box menu item in response to the user selecting the item.

3.2.4. Radio Button Items

Figure 4.8. A group of radiobutton items on a menu

Radio button menu items show which of two or more mutually-exclusive settings are currently selected, and allow the user to choose a different setting by selecting its menu item.

- If you need to offer a choice of two mutually-exclusive settings to the user, use a group of two radio button items instead of a single check box menu item if the settings are not clearly opposites. For example, represent View as Icons and View as List as two radio button items.
- Never change the label of a radio button menu item in response to the user selecting or deselecting the item.

4. Standard Menus

Most applications have many functions in common, such as Cut, Copy, Paste and Quit. To aid learning and memorability, these menu items, and the menus on which they appear, must appear with the same labels and in the same order in every application. The same commands must also behave the same way in different applications, to avoid surprising the user.

This section details the most common menus, menu items and their behaviors. You will not need all of these menus or menu items in every application you write, but do preserve the order of the menu titles and of the menu items that you do use.

Guidelines

- Place application-specific menus after the Format menu and before the Go menu
- Place application-specific menu items towards the middle of a standard menu, unless they logically fit with one of the standard groups already on the menu.

Figure 4.9. A menubar showing all the standard menu titles in their correct order

4.1. File

The File menu contains commands that operate on the current document. It is the left-most item in the menubar because of its importance and frequency of use, and because it is a relevant menu in many applications. Historically, because most applications already had this menu, and because the distinction between closing documents and closing windows became blurred over time, the File menu has also become the standard location for Quit.

The items on the File menu are generally ordered by locality, closest first. That is, items to save or load from file, followed by printing, followed by sending to a remote user. Try to maintain this ordering if you have to add new items to the menu.

If your application does not operate on documents, name this item for the type of object it displays. For example, many games should have a Game instead of a File menu. However, place the Quit menu item last on this menu nonetheless.

Figure 4.10. A generic File menu

4.1.1. Creation and Opening Operations

Table 4.1. Creation and Opening operation menu items

Label	Shortcut	Description
New	**Ctrl-N**	Creates a new document. Open a new primary window, with the title *`Document name`*, containing a blank document. How this window is displayed, e.g. as a tab or a separate window, is up to the window manager. If your application can create a number of different types of document, you can make the New item a submenu, containing a menu item for each type. Label these items New *`document type`*, make the first entry in the submenu the most commonly used document type, and give it the **Ctrl-N** shortcut. Note: A blank document will not necessarily be completely blank. For example, a

Label	Shortcut	Description
		document created from a template may already contain some data.
Open...	**Ctrl-O**	Opens an existing document in a new window. Present the user with a standard Open File dialog from which they can choose an existing file. If the chosen file is already open in the application, raise that window instead of opening a new one.

4.1.2. Saved State Operations

Table 4.2. Saved State Operation menu items

Label	Shortcut	Description
Save	**Ctrl-S**	Saves the document with its current filename. If the document already has a filename associated with it, save the document immediately without any further interaction from the user. If there are any additional options involved in saving a file (eg. DOS or UNIX-style line endings in a text file), prompt for these first time the document is saved, but subsequently use the same values each time until the user changes them. If the document has no current filename or is read-only, selecting this item should be the same as selecting Save As.
Save As...	**Shift-Ctrl-S**	Saves the document with a new filename. Present the user with the standard Save As dialog, and save the file with the chosen file name.
Save a Copy...	None	Prompts the user to enter a filename, with which a copy of the document is then saved. Do not alter either the view or the filename of the original document. All subsequent changes are still made to the original document until the user specifies otherwise, for

Label	Shortcut	Description
		example by choosing the Save As command. Like the Save As dialog, the Save a Copy dialog may present different ways to save the data. For example, an image may be saved in a native format or as a PNG.
Revert	None	Reverts the document to the last saved state. Present the user with a warning that all changes will be lost, and offer the option of canceling before reloading the file.
Save Version...	None	An alternative to the Save a Copy command. Only use this item in conjunction with the Restore Version. command.
Restore Version...	None	Prompts the user for a version of the current document to be restored. Present the user with with a warning that all changes will be lost, and offer the option of cancelling before restoring the version. Only use this item in conjunction with the Save Version command.
Versions...	None	An alternative to the Save Version and Restore Version commands. Use this when more utilities, such as a diff, are available.

4.1.3. Export Operations

Table 4.3. Export Operation menu items

Label	Shortcut	Description
Page Setup...	None	Allows the user to control print-related settings. Present the user with a dialog allowing the user to set such options as portrait or landscape format, margins, and so on.
Print Preview	**Shift-Ctrl-P**	Shows the user what the printed document will look like. Present a new window containing an accurate representation of the appearance of the document as it would be printed. The

Label	Shortcut	Description
		libgnomeprintui library provides a standard Print Preview window that you should use if possible.
Print...	**Ctrl-P**	Prints the current document. Present the user with a dialog allowing them to set options like the page range to be printed, the printer to be used, and so on. The dialog must contain a button labeled Print that starts printing and closes the dialog. The `libgnomeprintui` library provides a standard Print dialog that you should use if possible.
Send To...	**Ctrl-M**	Provides the user a means to attach or send the current document as an email or email attachment, depending on its format. You may provide more than one Send item depending on which options are available. If there are more than two such items, move them into a submenu. For example, if only Send by Email and Send by Fax are available, leave them on the top-level menu If there is a third option, such as Send by FTP, place all the options in a Send submenu.

4.1.4. File Properties

Table 4.4. Properties menu items

Label	Shortcut	Description
Properties	**Alt-Return**	Opens the document's Properties window. This may contain editable information, such as the document author's name, or read-only information, such as the number of words in the document, or a combination of both. The **Alt-Return** shortcut should not be provided where **Return** is most frequently used to insert a new line.

4.1.5. Closing Operations

Table 4.5. Closing Operation menu items

Label	Shortcut	Description
Close	**Ctrl-W**	Closes the current document. If it has unsaved changes, present the user with a confirmation alert giving the option to save changes, discard them, or cancel the action without closing or saving the document. If the window you are closing is the last open document in the application, the correct action depends on your application type: • Single document interface: close the application • Controlled single document interface: leave only the control window open • Multiple document interface: close the current document and create a new blank document
Quit	**Ctrl-Q**	Closes the application. If there are unsaved changes in any open documents, present the user with a confirmation alert for each affected document, giving the option to save the changes, discard them, or cancel. If there are no unsaved changes, close the application immediately without presenting any further messages or dialogs. In particular, non-document based applications, for example a game or a calculator, should save their state and exit immediately. This state should be restored the next time the application is started.

4.2. Edit

The Edit menu contains items relating to editing both the document (clipboard handling, search and replace, and inserting special objects) and the user's preferences. Preferences are edited here rather than on a Settings menu, because:

- most applications' preferences windows are accessed via a single menu item, and single-item menus offer poor usability
- most applications already contain a suitable Edit menu.

Figure 4.11. A generic Edit menu

4.2.1. Modification History

Document-based applications should maintain a history of modifications to a document and the state of the document between each action. The Undo and Redo commands move backwards and forwards through this history.

Table 4.6. Modification History menu items

Label	Shortcut	Description
Undo *`action`*	**Ctrl-Z**	Undoes the effect of the previous action in the undo history list. Revert the document to its state before the previous action was performed. If your application supports undo, and the user undoes all changes since it was last saved, treat the document as unmodified. Note: provide a separate Undo and Redo menu item even if your application only supports one level of undo.
Redo *`action`*	**Shift-Ctrl-Z**	Performs the next action in the undo history list, after the user has moved backwards through the list with the Undo command. Move the user one step forwards again, restoring the document to the state it was in after that action was originally performed. Note: provide a separate Undo and Redo menu item even if your application only supports one level of undo.

4.2.2. Manipulating Selected Data

Table 4.7. Selected Data Manipulation menu items

Label	Shortcut	Description
Cut	**Ctrl-X**	Removes the selected content and places it onto the clipboard. Visually, remove the content from the document in the same manner as Delete.
Copy	**Ctrl-C**	Copies the selected content onto the clipboard.
Paste	**Ctrl-V**	Inserts the contents of the clipboard into the document. If there is no current selection, use the caret as the insertion point. If there is a current selection, replace it with the clipboard contents.
Paste Special...	**Shift-Ctrl-V**	Inserts a non-default representation of the clipboard contents into the document. Open a dialog presenting a list of the available formats from which the user can select. For example, if the clipboard contains a PNG file copied from a file manager, the image may be embedded in the document, or a link to the file inserted so that changes to the image on disk are always reflected in the document.
Duplicate	**Ctrl-U**	Creates a duplicate copy of the selected object. Do not prompt for a name for the duplicate object, but give it a sensible default (for example, `Copy of ShoppingList.abw`) and allow the user to change it later. Place the duplicate copy as near the original as possible without overlapping it, even if this means breaking the current sort order within the container, so the user sees it immediately.
Delete	**Delete**	Removes the selected content without placing it on the clipboard.
Select All	**Ctrl-A**	Selects all content in the current document.
Deselect All	**Shift-Ctrl-A**	Deselects all content in the current document. Only

Label	Shortcut	Description
		provide this item in situations when no other method of undoing selection is possible or apparent to the user. For example, in complex graphics applications where selection and deselection is not usually possible simply by using the cursor keys. Note: Do not provide Deselect All in text entry fields, as **Shift-Ctrl-**hex digit is used to enter Unicode characters so its shortcut will not work.

4.2.3. Searching and Replacing

Table 4.8. Search and Replace menu items

Label	Shortcut	Description
Find...	**Ctrl-F**	Opens a window or dialog allowing the user to search for specific content in the current document. Highlight each match in-place. If the command allows the user to search for content in places other than the current document, for example other open documents, other documents on disk, or a remote network location, label this item Search instead of Find.
Find Next	**Ctrl-G**	Selects the next instance of the last Find term in the current document.
Find Previous	**Shift-Ctrl-G**	Selects the previous instance of the last Find term in the current document.
Replace...	**Ctrl-H**	Opens a window or dialog allowing the user to search for specific content in the current document, and replace each occurrence with new content.

4.2.4. Inserting Special Objects

Where applicable, provide items on the Edit menu that insert special objects such as images, links, GUI controls or the current date and time.

If you have two or less types of object that can be inserted, add them as individual items to this menu, for example Insert Image, or Insert External Link. If you have between three and six types, place them on an Edit → Insert submenu. If you have more than six, add a separate Insert menu to the menubar.

4.2.5. User Preferences

Table 4.9. User Preferences menu items

Label	Shortcut	Description
Preferences	None	Opens a preferences window allowing the user to change preferences for the whole application. Changes will apply to all running and subsequent instances of the application.

4.3. View

The View menu contains only items that affect the user's view of the current document. Do not place any items on the View menu that affect the content of the current document. (Exception: View → Reload may change the current contents if, for example, the document is a webpage that has been recently updated on the server).

Figure 4.12. A generic View menu

4.3.1. Toolbar and Statusbar

Table 4.10. Toolbar and Statusbar menu items

Label	Shortcut	Description
Toolbar	None	Shows or hides the application's toolbar. This is a check box menu item. Include this item in every application that has a single toolbar. See Section 2, “Controlling Display and Appearance” for information on how to deal with multiple toolbars.

4.3.2. Content Presentation

Table 4.11. Content Presentation menu items

Label	Shortcut	Description
Icons	None	Shows the contents of the

Label	Shortcut	Description
		selected container as rows and columns of large icons, each with its name underneath. This is a radio button menu item.
List	None	Shows the contents of the selected container as a list of small icons, possibly in multiple columns, each with its name on its right-hand side. This is a radio button menu item.
Details	None	Shows the contents of the selected container as single column of small icons, each with its name on its right-hand side. Additional columns give extra information about the object each icon represents, for example the size and modification date of files in a file manager. This is a radio button menu item. If your application has no need for both List and Details modes, use the List item for whichever of the two modes you support.
Sort By...	None	Sorts the contents of an container by user-specified criteria. Open a dialog allowing the user to choose from predefined sort keys (for example, Name, Size, or Modification Date in a file manager), or to specify their own if applicable.
Filter...	None	Hides objects that are not of interest to the user. Open a dialog allowing the user to choose from a list of types of object they want to display, or to enter their own criteria (for example, a regular expression matched against a particular property of the objects).
Zoom In	**Ctrl-+**	Zooms into the document. Make the center of the new view the same as the center of the previous view.
Zoom Out	**Ctrl--**	Zooms out of the document. Make the center of the new view the same as the center of the previous view.
Normal Size	**Ctrl-0**	Resets the zoom level back to

Label	Shortcut	Description
		the default value, normally 100%. Make the center of the new view the same as the center of the previous view.
Best Fit	None	Makes the document fill the window. Show the document, or the current page of the document, at as high a zoom level as will fit in the window whilst allowing the whole document or page to be visible without scrolling.
Refresh	**Ctrl-R**	Redraws the current view of the document from local storage. For example, in a web browser application, this would redraw the page from the browser page cache.
Reload	**Ctrl-R**	Redraws the current view of the document, checking the data source for changes first. For example, checks the web server for updates to the page before redrawing it. If your application requires both Reload and Refresh, use **Shift-Ctrl-R** as the shortcut for Reload.

4.4. Insert

The Insert menu lists the type of special objects that can be inserted into the document at the current caret position, for example images, links, page breaks or GUI objects. Only provide this menu if you have more than about six types of object that can be inserted, otherwise place individual items for each type on the Edit menu.

Figure 4.13. A generic Insert menu

The types of object will vary between applications, but the table below shows some common types that may be applicable.

Table 4.12. Insert menu items

Label	Shortcut	Description
Page Break	None	Inserts a page break at the caret position. Show the page break visually, for example as a dotted line across the page, unless the user has specifically

Label	Shortcut	Description
		requested not to see them.
Date and Time...	None	Inserts the current date and/or time at the caret position. Open a dialog giving a choice of date and time formats. If applicable, also offer the choice to insert either as plain text, so the specified date and time will always appear in the document, or as a special field that will updated every time the document is opened, refreshed or printed.
Symbol...	None	Inserts a special symbol, such as a mathematical symbol or foreign character, at the caret position. Open a dialog showing all the available symbols as a table, from which the user can choose. The user must be able to add multiple symbols to the document at one time without having to close and re-open the dialog.
Sheet...	None	Adds a new sheet to the current workbook. Do not prompt for a name, but choose a sensible default (such as Sheet-2) and allow the user to change it later.
Rows...	None	Adds new rows to a table in which one or more rows or cells are currently selected. Open a dialog asking whether to insert rows above or below the current selection, and for any other required information. Copy the row format from the last or first row of the current selection respectively, unless the user specifies otherwise.
Columns...	None	Adds new columns to a table in which one or more columns or cells are currently selected. Open a dialog asking whether to insert columns to the left or right of the current selection, and for any other required information. Copy the column format from the right- or left-most column of the current selection respectively, unless the user specifies otherwise.
Image...	None	Inserts an image into the document from a file. Present a

Label	Shortcut	Description
		standard Open File dialog filtered on acceptable file types, from which the user can choose an image file to insert.
Graph...	None	Inserts a graph into the document. Open a dialog or assistant that allows the user to build (or open from a file) a graph of their choice, using the current selection as an indication of which values, axis labels and data labels to use.
From FIle...	None	Inserts an object from any acceptable file type, for example plain text, formatted text, or an image. Present a standard Open File dialog filtered on acceptable file types, from which the user can choose a file to insert.
External Link...	None	Inserts a link to an object stored in a different file, or on a remote system. The object is not embedded in or saved with the document, only a link to it. Open a dialog in which the user can type or choose the name of the object, for example a filename or a webpage URL. Show the link in the document in as informative way as possible. For example, show a link to an image as a thumbnail of that image, unless the user specifies otherwise.

4.5. Format

A Format menu contains commands to change the visual appearance of the document. For example, changing the font, color, or line spacing of a text selection.

The difference between these commands and those on the View menu is that changes made with Format commands are persistent and saved as part of the document, for example as HTML or RTF tags.

Figure 4.14. A generic Format menu

Items found on the Format will be very application-specific, but some common items are listed in the table below.

Table 4.13. Format menu items

Label	Shortcut	Description
Style...	None	Sets the style attributes of the selected text or objects either individually or to a named, predefined style. Open a dialog allowing the user to set attributes such as bold, italic, size and spacing individually, and to create their own named styles where applicable.
Font...	None	Sets the font properties of the selected text or objects. Open a dialog allowing the user to choose font, size, style, color, or whatever other attributes are applicable.
Paragraph...	None	Sets the properties of the selected paragraph. Open a dialog allowing the user to choose style, line spacing, tabulation, or whatever other attributes are applicable.
Bold	**Ctrl-B**	Toggles the boldness of the current text selection on or off. If some of the selection is currently bold and some is not, this command should bolden the selected text.
Italic	**Ctrl-I**	Toggles the italicization of the current text selection on or off. If some of the selection is currently italicized and some is not, this command should italicize the selected text.
Underline	**Ctrl-U**	Toggles underlining of the current text selection. If some of the selection is currently underlined and some is not, this command should underline the selected text.
Cells...	None	Sets the properties of the selected table cells. Open a dialog allowing the user to choose alignment, borders, shading, text style, number format, or whatever other attributes are applicable.
List...	None	Sets the properties of the selected list, or turns the selected paragraphs into a list if they are not already formatted as such. Open a dialog allowing

Label	Shortcut	Description
		the user to choose number or bullet style, spacing, tabulation, or whatever other attributes are applicable.
Layer...	None	Sets the properties of all or selected layers of a multi-layered document. Open a dialog allowing the user to choose name, size, visibility, opacity, z-ordering, or whatever other attributes are applicable.
Page...	None	Sets the properties of all or selected pages of the document. Open a dialog allowing the user to choose paper size, orientation, columns, margins, or whatever other attributes are applicable.

4.6. Bookmarks

Provide a Bookmarks menu in any application that allows the user to browse files and folders, help documents, web pages or any other large information space.

Icons

Show icons for bookmark entries on the Bookmarks menu that indicate the type of the bookmark, even if the user has globally turned off icons for other menu items on the desktop.

Figure 4.15. A generic Bookmarks menu

Table 4.14. Bookmark menu items

Label	Shortcut	Description
Add Bookmark	**Ctrl-D**	Adds a bookmark for the current document to the default bookmark list. Do not pop up a dialog asking for a title or location for the bookmark, instead choose sensible defaults (such as the document's title or filename as the bookmark name) and allow the user to change them later using the Edit Bookmarks feature.
Edit Bookmarks	**Ctrl-B**	Allows the user to edit the application's bookmark list.

Label	Shortcut	Description
		Open a window in which the user can arrange bookmarks into a hierarchy, move, copy, and delete bookmarks, and change their properties.
Bookmark List	None	The user's current list of bookmarks for the application.

4.7. Go

A Go menu provides commands for quickly navigating around a document or collection of documents, or an information space such as a directory structure or the web.

The contents of the menu will vary depending on the type of application. Different standard menus are presented here for browser-based and document-based applications , but your application may require a combination of both.

Figure 4.16. A generic Go menu for a browser application

Table 4.15. Go menu items for a browser application

Label	Shortcut	Description
Back	**Alt-Left**	Navigates to the previous document in the browser's history list.
Forward	**Alt-Right**	Navigates to the next document in the browser's history list.
Up	**Alt-Up**	Navigates to the current document's (or folder's) parent document (or folder). For a document browser, such as an on-line help viewer, this usually means navigating to the enclosing sub-section, section, chapter or contents page.
Home	**Alt-Home**	Navigates to a starting page defined by the user or the application.
Location...	**Ctrl-L**	Navigates to a user-specified URI. Open a dialog into which the user can type a suitable URI, or select one from a list where applicable (for example, a file selection dialog for applications that can handle file:// URIs).

Figure 4.17. A generic Go menu for document-based applications

Table 4.16. Go menu items for a document-based application

Label	Shortcut	Description
Previous Page	**PageUp**	Navigates to the previous page in the document.
Next Page	**PageDown**	Navigates to the next page in the document.
Go to Page...	None	Navigates to a user-specified page number. Open a dialog into which the user can type a page number. Text-based applications may also include a Go to Line... menu item, which allows the user to jump to a specified line number.
First Page	**Ctrl-Home**	Navigates to the first page in the document.
Last Page	**Ctrl-End**	Navigates to the last page in the document.

4.8. Windows

The Windows menu contains commands that apply to all of the application's open windows. Only use a Windows menu in multiple document interface (MDI) applications.

MDI Applications

The use of MDI is discouraged, as they have a number of inherent usability problems.

You may also label this menu Documents, Buffers, or similar according to the type of document handled by your application.

The last items on this menu are a numbered list of the application's primary windows, for example 1shoppinglist.abw. Selecting one of these items raises the corresponding window.

Figure 4.18. A generic Windows menu

Table 4.17. Windows menu items

Label	Shortcut	Description
Save All	None	Saves all open documents. If any documents have no current filename, prompt for a filename for each one in turn using the standard Save dialog.
Close All	None	Closes all open documents. If there are any unsaved changes in any documents, post a confirmation alert for each one in turn.
1. *`first open window title`* 2. *`second open window title`* etc.	None	Raises the corresponding window to the top of the window stack.

4.9. Help

The Help menu provides access to all on-line documentation for your application. This includes both the user guide, and the About window which includes a brief description of your application's functionality.

Figure 4.19. A generic Help menu

Table 4.18. Help menu items

Label	Shortcut	Description
Contents	**F1**	Opens the default help browser on the contents page for the application.
About	None	Opens the About dialog for the application. Use the standard dialog provided by the GNOME libraries, which contains the name and version number of the application, a short description of the application's functionality, author contact details, copyright message and a pointer to the license under which the application is made available.

Chapter 5. Toolbars

A toolbar is a strip of controls that allows convenient access to commonly-used functions. Most toolbars only contain graphical buttons, but in more complex applications, other types of controls such as dropdown lists, can also be useful.

Figure 5.1. Example toolbar from a web browser window

Careful and consistent toolbar design speeds up the user's task by giving direct access to functions that would otherwise be hidden on a menu. Use them only for the most important functions, however. Having too many toolbar controls reduces their efficiency by making them harder to find, and too many rows of toolbars reduces the amount of screen space available to the rest of the application.

1. Appearance and Content

The effectiveness of toolbars is increased by maintaining a level of consistency between different applications. The toolbar is one of the first parts of your application that a user will see the first time they run it, so by providing a toolbar that looks familiar to them, you can immediately make them feel comfortable about using your application.

As well as following the recommendations and examples given in this section, look at the toolbars in other well-designed GNOME 2.0 applications for guidance when deciding what— and what not— to put on your application's toolbar.

However many toolbars or toolbox windows your application offers, provide one main toolbar by default that contains a representative subset of the application's overall functionality. Many of the buttons on this toolbar will be the same regardless of the type of application.

For example, the main toolbar in an office application will nearly always have New, Open and Save as its first three toolbar buttons. Similarly, the first few buttons in a browser application should always include Back, Forward, Stop and Reload, in that order.

Guidelines

- Place only the most commonly-used application functions on your toolbars. Don't just add buttons for every menu item.
- By default, have your toolbars appear directly below the main menu bar.
- Allow toolbars to be turned on and off in your application's Preferences dialog and by using the View → Toolbar menu item. If there is more than one toolbar, they are turned on and off by individual entries in the View → Toolbar submenu.
- All functions that appear on your toolbars must also accessible via the main menu bar, either directly (i.e. an equivalent menu item) or indirectly (e.g. in the Options → Settings dialog).
- Arrange toolbar buttons in the same order and groupings as their equivalents on the main menu bar. In particular, always group sets of mutually-exclusive toolbar buttons.
- Don't add buttons for Help, Close or Quit to your toolbar by default, as these are rarely used

and the space is better used for more useful controls. Similarly, only provide buttons for Undo, Redo and the standard clipboard functions if there is space on the toolbar to do so without sacrificing more useful, application-specific controls.

- Provide options to show toolbar buttons as text, graphics or both— see Figure 5.2, “Example View menu fragments for applications with one toolbar (left), two or three toolbars (middle), or four or more toolbars (right)” for the menus to use for controlling toolbar display. Also provide an option to return all toolbars in your application to the control center default for this setting.
- Allow users to configure toolbars to contain their own selection of commands, in whatever order they choose. Provide an option in the configuration dialog to return the toolbars to their default configuration.
- Save your application's toolbar position and contents as part of the application configuration, and restore them when the application is restarted.

1.1. Vertical Toolbars

In general, don't use vertical toolbars. The eye does not scan vertically as well as it does horizontally, groups of mutually exclusive buttons are less obvious when arranged vertically, and showing button labels is more awkard and less space-efficient. Also, some toolbar controls just cannot be used vertically, such as dropdown lists.

Only consider using a vertical toolbar if:

- the configuration of the application window means there would be a lot of wasted space if a horizontal toolbar was used instead, or
- your application would otherwise require three or more rows of toolbars to appear below the main menu bar by default. Note however that in this situation, the better alternative is usually to display fewer toolbars by default.

If you must use a vertical toolbar, ensure the user can configure it to appear horizontally if they prefer.

1.2. Media Player Toolbars

Many applications are able to play sound or video clips. For consistency, always present the buttons that control playback in the same order and with the same stock icons.

Guidelines

- Show separate Stop and Pause buttons. Do not change Play to Pause while the clip is playing.

2. Controlling Display and Appearance

For each toolbar in your application, the user should be able to choose whether or not to show that

toolbar, and whether to show its contents as icons only, text only or both.

Guidelines

- Allow the user to override the control center toolbar defaults for your particular application in the application's Preferences dialog. In particular, ensure that the user can:
  - separately choose to show each toolbar in your application as icons only, text only, or both
  - return the icon/text/both status for all toolbars in your applicaton to the system default
  - choose to show text labels either to the side of some or below all toolbar icons, and to return this setting to the system default
  - return the layout and ordering of all toolbars in your application to the application default
- If your application has a single toolbar, allow the user to turn it on or off with a View → Toolbar check box menu item.
- If your application has two or three toolbars, allow the user to turn them on or off individually by placing a menu item for each one on the application's View menu. For example, Main Toolbar, Drawing Toolbar, Formatting Toolbar. Place the items together in a single group on the menu, with Main Toolbar first (if your application has one), followed by the others in alphabetical order.
- If your application has more than three toolbars, allow the user to turn them on or off individually by placing a menu item for each one in a Toolbars sub-menu on the application's View menu. Place the Main Toolbar item first (if your application has one), followed by the others in alphabetical order.

Figure 5.2. Example View menu fragments for applications with one toolbar (left), two or three toolbars (middle), or four or more toolbars (right)

3. Labels and Tooltips

Most controls that appear on your toolbar will require a text label that appears on, below or beside it. Keep this description as short as possible, preferably a single verb. For example, Open or Undo.

Every control that appears on your toolbar should have a tooltip, whether or not that control has an associated text label. The tooltip should be a concise description of the control, but should provide more information than its text label where possible. For example, Open an existing document, or Undo last operation.

Guidelines

- For buttons that correspond directly to menu items, make the text label the same as the menu item, but without any trailing ellipsis. For example, Open, Save.
- Do not provide access keys for toolbar buttons. Since toolbars are in the same keyboard focus

context as the menubar, it would be too difficult to provide unique access keys for every menu title and toolbar control. Toolbars are primarily intended as a shortcut for mouse users, although they are keyboard-navigable for accessibility reasons.

- If your toolbar is configured to show labels below button icons, show a label for every control on the toolbar. For example:

Figure 5.3. Toolbar with labels under all buttons

- If your toolbar is configured to show labels beside button icons rather than below them (using the "priority text" setting), do not show labels for every button. Show labels only for the buttons that will be most-frequently used. Choose no more than four such icons on any one toolbar, otherwise the effect will be diluted and the toolbar will become very wide. For example:

Figure 5.4. Toolbar with "priority text" labels beside the first few buttons only

If you are unsure which buttons will be most frequently used, choose the first few buttons on your toolbar and provide labels for those only.

- Ensure all toolbar controls have tooltips. The tooltip should be more descriptive than the corresponding menu item, if it has one, but still concise. For example, Create new document for the Open toolbar button. Use sentence capitalization for tooltips—see Section 3.2, “Capitalization” for more information.

Chapter 6. Controls

1. Using Controls Effectively

GNOME provides a set of controls, also known as widgets, which allow users to interact with your applications. Using these controls appropriately and not altering their standard behavior is important. This allows users to predict the effects of their actions, and thus learn to use your application more quickly and efficiently. Controls that behave in non-standard ways break the user's mental model of how your application works, and dilute the meaning of the interface's visual language.

2. Terminology

Although they are known as "widgets" in the GNOME APIs and developer documentation, do not use this term in your user interface or user documentation. Refer to them by their specific names (for example, "buttons" or "menus"), or by the generic name "controls".

3. Sensitivity

Sometimes it does not make sense to allow the user to interact with a control in the current context, for example, to press a Paste button when the clipboard is empty. At these times, make the control insensitive to minimize the risk of user error. While a control is insensitive, it will appear dimmed and will not be able to receive the focus, although assistive technologies like screenreaders will still be able to detect and report it.

It is usually better to make a control insensitive than to hide it altogether. This way, the user can learn about functionality they may be able to use later, even if it is not available right now.

Figure 6.1. Two check boxes: sensitive (top) and insensitive (bottom)

3.1. Locked Controls

In a network-managed environment, like a computer lab, system administrators usually want to "lock down" the values of certain settings, or remove them from the user interface altogether. This makes it easier for them to troubleshoot any problems that their users may encounter. In GNOME, the correct way for the system administrator to do this is by restricting write access to the GConf keys corresponding to those settings.

When you are designing your application, consider which settings a system administrator might want to make unavailable to users. These may typically include:

- Settings that, if set wrongly, could prevent the application from functioning at all. For example, proxy settings in a network application.
- Settings that could refer to networked resources. For example, the Templates directory in an office application, where shared stationery such as fax cover sheets might be stored.
- Settings that customize the user interface, other than those required for accessibility. For example, certain menu, keyboard or toolbar customization options.

Your application needs to decide every time these controls are displayed whether or not they are available for editing, depending on the writable state of the GConf key that holds its value. In the simplest case, your code for each control could look like that in the example below.

Example 6.1. Sample code fragment showing how to make a GConf-locked control insensitive

```
if (!gconf_key_is_writable (http_proxy))
        gtk_widget_set_sensitive (http_proxy_field, FALSE);
```

Include a section for system administrators in your user guide, explaining which settings they can lock, and their corresponding GConf keys.

Explain to the user why these controls cannot be edited at this time. You can do this with static text, tooltips or online help, depending on the situation. For example:

Figure 6.2. Example of a dialog with locked controls

Note that although they cannot be edited, the settings are still visible and selectable, and may be copied to the clipboard.

4. Text Entry Fields

Text entry fields are used for entering one or more lines of plain text. In GTK 2, the GtkEntry [http://developer.gnome.org/doc/API/2.0/gtk/gtkentry.html] control is used for single-line text entry, and GtkTextView [http://developer.gnome.org/doc/API/2.0/gtk/gtktextview.html] for multiple-line text entry.

Figure 6.3. Single and multi-line entry fields

Guidelines

- Label the entry field with a text label above it or to its left, using sentence capitalization. Provide an access key in the label that allows the user to give focus directly to the entry field.
- Right-justify the contents of entry fields that are used only for numeric entry, unless the convention in the user's locale demands otherwise. This is useful in windows where the user might want to compare two numerical values in the same column of controls. In this case, ensure the right edges of the relevant controls are also aligned.
- When the user gives focus to an entry field using the keyboard, place the text cursor at the end of the existing text and highlight its contents (but don't overwrite the existing PRIMARY clipboard selection). This makes it easy to immediately overwrite or append new text, the two most common operations performed on entry fields.
- Size text entry fields according to the likely size of the input. This gives a useful visual cue to

the amount of input expected, and breaks up the dialog making it easier to scan. Don't make all the fields in the dialog the same width just to make everything line up nicely.

- In an instant-apply property or preference window, validate and apply the contents of the entry field when it loses focus or when the window is closed, but not after each keypress. Exception: if the field accepts only a fixed number of characters, such as a hexadecimal color code, validate and apply the change as soon as that number of characters have been entered.

- Provide a static text prompt for text boxes that require input in a particular format or in a particular unit of measurement. For example:

Figure 6.4. Text entry field with static text prompt

- Where possible, provide an additional or alternative control that limits the required input to the valid range. For example, provide a spinbox or slider if the required input is one of a fixed range of integers, or provide access to a GtkCalendar [http://developer.gnome.org/doc/API/2.0/gtk/gtkcalendar.html] control if the user has to enter a valid date:

Figure 6.5. Text entry field requiring a date as input, with a button beside it to pop up a GtkCalendar control to simplify the task

This is less error-prone than expecting the user to format their text input in some arbitrary format. You may still want to provide the entry field control as well, however, for expert users who are familiar with the required format.

- If you implement an entry field that accepts only keystrokes valid in the task context, such as digits, play the system warning beep when the user tries to type an invalid character. If the user types three invalid characters in a row, display an alert that explains the valid inputs for that textfield.

- The cursor blink rate is globally defined by the XSettings "gtk-cursor-blink" and "gtk-cursor-blink-time". Standard toolkit controls use these and they must not be altered in applications by any means. New controls with text cursors must respect these global values.

4.1. Behavior of Return key

Normally, pressing **Return** in a dialog should activate the dialog's default button, unless the focused control uses **Return** for its own purposes. You should therefore set the `activates-default` [http://developer.gnome.org/doc/API/2.0/gtk/gtkentry.html#GTK-ENTRY-SET-ACTIVATES-DEFAULT] property of most entry fields to TRUE. (Note that GtkTextView does not have such a setting— pressing **Return** always inserts a new line.).

However, if your dialog contains several entry fields that are usually filled out in order, for example Name, Address and Telephone Number, consider setting the `activates-default` property on those entry fields to FALSE. Pressing **Return** should then move focus on to the next control instead. Doing this will help prevent the user from accidentally closing the window before they have entered all the information they wanted to.

As a further safeguard, remember not to set the default button in a dialog until the minimum

amount of required information has been entered, for example, both a username and a password in a login dialog. Again, in this case you should move focus to the next control when the user presses **Return**, rather than just ignoring the keypress.

If you need to provide a keyboard shortcut that activates the default button while a GtkTextView control has focus, use **Ctrl-Return**.

Note

Gtk does not currently move focus to the next control when **Return** is pressed and either `activates-default=FALSE`, or there is no default button in the window. For now, **Return** does nothing in these situations, so remember to implement the focus change behavior yourself.

4.2. Behavior of Tab key

Normally, pressing **Tab** in a single-line entry field should move focus to the next control, and in a multi-line entry field it should insert a tab character. Pressing **Ctrl-Tab** in a multi-line entry field should move focus to the next control.

If you need to provide a keyboard shortcut that inserts a tab character into a single line entry field, use **Ctrl-Tab**. You are unlikely to find many situations where this is useful, however.

5. Spin Boxes

A spin box is a text box that accepts a range of values. It incorporates two arrow buttons that allow the user to increase or decrease the current value by a fixed amount.

Figure 6.6. Example of a spin box

Guidelines

- Use spin boxes for numerical input only. Use a list or option menu when you need the user to select from fixed data sets of other types.
- Use a spin box if the numerical value is meaningful or useful for the user to know, and the valid input range is unlimited or fixed at one end only. For example, a control for specifying the number of iterations of some action, or a timeout value. If the range is fixed at both ends, or the numerical values are arbitrary (for example, a volume control), use a slider control instead.
- Label the spin box with a text label above it or to its left, using sentence capitalization. Provide an access key in the label that allows the user to give focus directly to the spin box.
- Right-justify the contents of spin boxes, unless the convention in the user's locale demands otherwise. This is useful in windows where the user might want to compare two numerical values in the same column of controls. In this case, ensure the right edges of the relevant controls are also aligned.
- In an instant-apply property or preference window, validate and apply the contents of a spin box when a spin button is clicked, when it loses focus or when the window is closed, but not

after each keypress. Exception: if the field accepts only a fixed number of characters, such as a hexadecimal color code, validate and apply the change as soon as that number of characters have been entered.

6. Sliders

A slider allows the user to quickly select a value from a fixed, ordered range, or to increase or decrease the current value. The control looks like the type of slider that you might find on an audio mixing desk or a hi-fi's graphic equalizer. In gtk, you implement a slider using the GtkHScale or GtkVScale controls, for horizontal or vertical sliders respectively.

Figure 6.7. A simple slider control

Guidelines

- Use a slider when:
  - adjusting the value relative to its current value is more important than choosing an absolute value. For example, a volume control: the average user will usually think about turning the volume up or down to make a sound louder or quieter, rather than setting the peak output to a specific decibel value.
  - it is useful for the user to control the rate of change of the value in real time. For example, to monitor the effects of a color change in a live preview window as they drag the RGB sliders.
- Label the slider with a text label above it or to its left, using sentence capitalization. Provide an access key in the label that allows the user to give focus directly to the slider.
- Mark significant values along the length of the slider with text or tick marks. For example the left, right and center points on an audio balance control in Figure 6.7, “A simple slider control”.
- For large ranges of integers (more than about 20), and for ranges of floating point numbers, consider providing a text box or spin box that is linked to the slider's value. This allows the user to quickly set or fine-tune the setting more easily than they could with the slider control alone.

Figure 6.8. Slider controls with linked spin boxes

7. Buttons

A button initiates an action when the user clicks it.

Figure 6.9. Typical buttons in a modal dialog

Guidelines

- Label all buttons with imperative verbs, using header capitalization. For example, Save, Sort or Update Now. Provide an access key in the label that allows the user to directly activate the button from the keyboard.
- After pressing a button, the user should expect to see the result of their action within 1 second. For example, closing the window or opening another. See Chapter 7, *Feedback* for guidance on what to do if your application cannot respond this quickly.
- Use an ellipsis (...) at the end of the label if the action requires further input from the user before it can be carried out. For example, Save As... or Find.... Do not add an ellipsis to commands like Properties, Preferences, or Settings, as these open windows that do not *require* further input.
- Once a dialog is displayed, do not change its default button from one button to another. You may add or remove default status from the same button if it helps prevent user error, however. Changing the default from one button to another can be confusing and inefficient, especially for users relying on assistive technologies.
- If your button can display text, an icon, or both, choose which label to display at runtime according to the user's preference in the GNOME Menus and Toolbars Preferences dialog. However, you may over-ride this preference when there is no suitable icon to describe the button's action graphically, for example.
- Do not use more than one or two different widths of button in the same window, and make all of them the same height. This will help give a pleasing uniform visual appearance to your window that makes it easier to use.
- Do not assign actions to double-clicking or right-clicking a button. Users are unlikely to discover these actions, and if they do, it will distort their expectations of other buttons on the desktop.
- Make invalid buttons insensitive, rather than popping up an error message when the user clicks them.

In a dialog, one button may be made the default button, which is shown with a different border and is activated by pressing **Return**. Often this will be the OK or equivalent button. However, if pressing this button by mistake could cause a loss of data, do not set a default button for the window. Do not make Cancel the default button instead. See Section 3.3, “Default Buttons” for more information.

If it does not make sense to have a default button until several fields in the dialog have been correctly completed—for example, both the Username and Password fields in a login dialog—do not set the default button until they have both been completed.

8. Check Boxes

Check boxes are used to show or change a setting. Its two states, set and unset, are shown by the presence or absence of a checkmark in the labeled box.

Figure 6.10. A typical group of check boxes

Guidelines

- Do not initiate an action when the user clicks a check box. However, if used in an instant-apply property or preference window, update the setting represented by the check box immediately.
- Clicking a check box should not affect the values of any other controls. It may sensitize, insensitize, hide or show other controls, however.
- If toggling a check box affects the sensitivity of other controls, place the check box immediately above or to the left of the controls that it affects. This helps to indicate that the controls are dependent on the state of the check box.
- Use sentence capitalization for check box labels, for example Use custom font.
- Label check boxes to clearly indicate the effects of both their checked and unchecked states, for example, Show icons in menus. Where this proves difficult, consider using two radio buttons instead so both states can be given labels. For example:

Figure 6.11. Ambiguous check box (top), radio buttons work better in this case (bottom)

The single check box in this example is ambiguous, as it is not clear where the "progress indicator" will go if the box is unchecked. Two radio buttons are better in this case, as they make the options clear.

- Provide an access key in all check box labels that allows the user to set or unset the check box directly from the keyboard.
- If the check box represents a setting in a multiple selection that is set for some objects in the selection and unset for others, show the check box in its mixed state. For example:

Figure 6.12. Check boxes (right) showing properties for a multiple selection of files in Nautilus (left)

In this example, both selected files are hidden (since their filenames start with "."), and the emblems on their icons show that neither file is writable, but one is readable. The Readable check box is therefore shown in its mixed state.

When a check box is in its mixed state:

  - clicking the box once should check the box, applying that setting (when confirmed) to all the selected objects
  - clicking the box a second time should uncheck the box, removing that setting (when confirmed) to all the selected objects
  - clicking the box a third time should return the box to its mixed state, restoring each selected object's original value for that setting (when confirmed)

- Label a group of check boxes with a descriptive heading above or to the left of the group.

- Use a frame around the group if necessary, but remember that blank space often works just as well and results in a less visually-cluttered dialog.
- Do not place more than about eight check boxes under the same group heading. If you need more than eight, try to use blank space, heading labels or frames to divide them into smaller groups. Otherwise, consider using a check box list instead— but you probably also need to think about how to simplify your user interface.
- Try to align groups of check boxes vertically rather than horizontally, as this makes them easier to scan visually. Use horizontal or rectangular alignments only if they greatly improve the layout of the window.

9. Radio Buttons

Radio buttons are used in groups to select from a mutually exclusive set of options. Only one radio button within a group may be set at any one time. As with check boxes, do not use radio buttons to initiate actions.

Figure 6.13. A typical group of radio buttons

Guidelines

- Only use radio buttons in groups of at least two, never use a single radio button on its own. To represent a single setting, use a check box or two radio buttons, one for each state.
- Exactly one radio button should be set in the group at all times. The only exception is when the group is showing the properties of a multiple selection, when one or more of the buttons may be in their mixed state.
- Do not initiate an action when the user clicks a radio button. However, if used in an instant-apply property or preference window, update the setting represented by the radio button immediately.
- Clicking a radio button should not affect the values of any other controls. It may sensitize, insensitize, hide or show other controls, however.
- If toggling a radio button affects the sensitivity of other controls, place the radio button immediately to the left of the controls that it affects. This helps to indicate that the controls are dependent on the state of the radio button.
- Use sentence capitalization for radio button labels, for example Switched movement. Provide an access key in the label that allows the user to set the radio button directly from the keyboard.
- If the radio button represents a setting in a multiple selection that is set for some objects in the selection and unset for others, show the radio button in its mixed state. For example:

Figure 6.14. Radio buttons (right) showing properties for a multiple selection of shapes in a drawing application (left)

. In this situation, clicking any radio button in the group should set the clicked button, and unset all the others. Thereafter, the group should behave like a normal radio button group— there is no way to reset a radio button to its mixed state by clicking on it. Provide a Reset button or equivalent in the window that allows the previous mixed settings to be restored without closing the window or canceling the dialog.

- Label a group of radio buttons with a descriptive heading above or to the left of the group.
- Use a frame around the group if necessary, but remember that blank space often works just as well and results in a less visually-cluttered dialog.
- Do not place more than about eight radio buttons under the same group heading. If you need more than eight, consider using a single-selection list instead— but you probably also need to think about how to simplify your user interface.
- Try to align groups of radio buttons vertically rather than horizontally, as this makes them easier to scan visually. Use horizontal or rectangular alignments only if they greatly improve the layout of the window.

10. Toggle Buttons

Toggle buttons look similar to regular Buttons, but are used to show or change a state rather than initiate an action. A toggle button's two states, set and unset, are shown by its appearing "pushed in" or "popped out" respectively.

Figure 6.15. A typical group of toggle buttons

Guidelines

- Do not use groups of toggle buttons in dialogs unless space constraints force you to do so, or you need to provide consistency with a toolbar in your application. Check boxes or radio buttons are usually preferable, as they allow more descriptive labels and are less easily-confused with other types of control.
- Only use toggle buttons in groups, so they are not mistaken for regular buttons. Make the group behave like either a group of check boxes or a group of radio buttons, as required.
- Provide an access key in the label of all toggle buttons that allows the user to set or unset the button directly from the keyboard.
- Label a group of toggle buttons with a descriptive heading above or to the left of the group, as you would with a group of check boxes or radio buttons.
- Use a frame around the group of buttons if necessary, but remember that blank space often works just as well and results in a less visually-cluttered dialog.
- Try to align groups of toggle buttons horizontally rather than vertically. This is how toggle buttons normally appear on a toolbar, so the user will be more familiar with this arrangement.
- Do not leave any space between toggle buttons in a group, otherwise they may look unrelated or may be mistaken for regular buttons.

- Use header capitalization for toggle button labels, for example No Wallpaper, Embossed Logo.
- If your toggle button can display text, an icon, or both, choose which to display at runtime according to the user's setting in the GNOME Menus and Toolbars preference dialog.
- Use the same text or graphical label for a toggle button whether it is set or unset.
- If the toggle button represents a setting in a multiple selection that is set for some objects in the selection and unset for others, show the button in its mixed state. For example:

Figure 6.16. Toggle buttons (right) showing properties for a multiple selection of shapes in a drawing application (left)

11. Drop-down Lists

Drop-down lists (unfortunately called "combo boxes" in gtk+) are used to select from a mutually exclusive set of options. They can be useful when there is insufficient space in a window to use a group of radio buttons or a single-selection list, with which they are functionally equivalent.

Figure 6.17. A drop-down list showing current selection (left) and the list of available choices when clicked on (right)

Recommendations:

- Do not use drop-down lists with fewer than three items, or more than about ten. To offer a choice of two options, use radio buttons or toggle buttons. To offer a choice of more than ten options, use a list.
- Do not initiate an action when the user selects an item from an drop-down list. However, if used in an instant-apply property or preference window, update the setting that the menu represents immediately.
- Selecting an item from a drop-down list should not affect the values of any other controls. It may sensitize, insensitize, hide or show other controls, however.
- Label the drop-down list with a text label above it or to its left, using sentence capitalization. Provide an access key in the label that allows the user to give focus directly to the drop-down list.
- Use sentence capitalization for drop-down list items, for example Switched movement
- Do not use a drop-down list in a situation where it may have to show a property of a multiple selection, as drop-down lists have no concept of mixed state. Use a group of radio or toggle buttons instead, as these can show set, unset or mixed states.
- Do not use submenus on a drop-down list.

You should normally use radio buttons or a list instead of drop-down lists, as those controls present all the available options at once without any further interaction. However, drop-down lists

may be preferable in a window where:

- there is little available space
- the list of options may change over time
- the contents of the hidden part of the menu are obvious from its label and the one selected item. For example, if you have an option menu labeled "Month:" with the item "January" selected, the user might reasonably infer that the menu contains the 12 months of the year without having to look.

Drop-down lists can also be useful on toolbars, to replace a group of several mutually-exclusive toggle buttons.

12. Drop-down Combination Boxes

Drop-down combination boxes (called "combo box entries" in gtk+) combine a text entry field and a dropdown list of predefined values. Selecting one of the predefined values sets the entry field to that value.

Figure 6.18. A drop-down combination box before and after its dropdown list is displayed

Guidelines

- Only use a drop-down combination box instead of a list, drop-down list or radio button group when it is important that the user be able to enter a new value that is not already amongst the list of predefined choices.
- Do not initiate an action when the user selects an item from the list in a drop-down combination box. If used in an instant-apply property or preference window, update the setting represented by the drop-down combination box immediately if possible. If this isn't possible due to the contents of the entry field being invalid while the user is still typing into it, update the related setting when the drop-down combination box loses focus instead.
- If the user types a value into the drop-down combination box that is not already in the drop-down list, add it to the list when the drop-down combination box loses focus so they can select it next time.
- Interpret user input into a drop-down combination box in a case-insensitive way. For example, if the user types **`blue`**, **`Blue`** and **`BLUE`** into the same drop-down combination box on different occasions, only store one of these in the combo's dropdown list, unless your application makes a distinction between the different forms (which is usually a bad idea).
- Label the drop-down combination box with a text label above it or to its left, using sentence capitalization. Provide an access key in the label that allows the user to give focus directly to the drop-down combination box.
- Use sentence capitalization for the dropdown list items, for example Switched movement.

13. Scrollbars

Often the visual representation of an object, such as a document or a list, will not fit within the space allocated to its viewer control. In these cases, scrollbars can be attached to the viewer control. These allow the user to alter the horizontal or vertical position of the viewport onto the object, and thus the portion of the object that is visible.

Guidelines

- Only display scrollbars when they are required. If an object fits entirely inside the viewer control, don't draw scrollbars. If you are using a GtkScrolledWindow, use `gtk_scrolled_window_set_policy` to set the scrollbar display policy to GTK_POLICY_AUTOMATIC.
- Affix scrollbars to the right or bottom edges only of a viewer control. Make the scrollbar the full height or width of the control.
- Update the view in real time as the user drags a scrollbar. Any delay will negatively impact a user's ability to navigate the content.
- Don't use a scrollbar as a replacement for a slider.

14. Lists

A list control allows the user to inspect, manipulate or select from a list of items. Lists may have one or more columns, and contain text, graphics, simple controls, or a combination of all three.

Figure 6.19. A simple two column list

Guidelines

- Always give list controls a label, positioned above or to the left of the list, in sentence capitalization. Provide an access key in the label that allows the user to give focus directly to the list.
- Make the list control large enough that it can show at least four items at a time without scrolling. For lists of ten or more items, increase this minimum size as appropriate.
- If the list appears in a dialog or utility window, consider making the window and the list within it resizable so that the user can choose how many list items are visible at a time without scrolling. Each time the user opens this dialog, set its dimensions to those that the user last resized it to.
- Do not use lists with less than about five items, unless the number of items may increase over time. Use check boxes, radio buttons or an drop-down list if there are fewer items.
- Only use column headers when:
  - the list has more than one column, or

- the list has only one column, but the user may wish to re-order the list. (This is rarely useful with single column lists).

In most other situations, column headers take up unnecessary space, and the extra label adds visual clutter.

- Always label column headers when used. If the column is too narrow for a sensible label, provide a tooltip for the column instead. Apart from its obvious use, this will help ensure that assistive technologies can describe the use of the column to visually impaired users.
- Right align columns that contain only numbers, as this makes them easier to compare.
- Consider using a check box list for multiple-selection lists, as these make it more obvious that multiple selection is possible:

Figure 6.20. A simple check box list

If you do this, you should normally set the list control itself to be single-selection, but this depends on the particular task for which it will be used.

- For multiple selection lists, show the number of items currently selected in a static text label below the list, for example, Names selected: 3. Such a label also makes it more obvious that multiple selection is possible.
- Consider providing Select All and Deselect All buttons beside multiple selection lists, if appropriate.

14.1. Sortable Lists

Users often prefer to sort long lists, either alphabetically or numerically, to make it easier to find items. Allow users to sort long or multi-column lists by clicking on the column header they want to sort.

Guidelines

- Indicate which column is currently sorted by showing an upward or downward facing arrow in its header:

Sort Order	Arrow Direction
Natural	Down
Reverse	Up

- Clicking an unsorted column header sorts the column in natural order, indicated by showing a down arrow in its header.
- Clicking a column header sorted in natural order re-sorts it in reverse order, indicated by showing an up arrow in its header.

Un-sorting lists

Occasionally, an unsorted state may be useful, for example to show items in the order in which the user added them to the list. In such cases, clicking a column sorted in reverse order should un-sort it, indicated by removing the arrow from the column header.

Usually, however, this is better achieved by adding an extra column that the user can sort in the usual way, such as a sequence number column in this example.

15. Trees

A tree control allows the user to inspect, manipulate or select from a hierarchical list of items. Trees may have one or more columns, and contain text, graphics, simple controls, or a combination of all three.

Use trees with care!

Because of their complexity compared to other controls, novice and some intermediate users often have problems using and understanding tree controls. If your application is designed for that type of user, you might want to consider alternative ways of presenting the information, such as the Nautilus list or icon view, or the hierarchical browser lists found in GNUstep's File Viewer [http://www.gnustep.it/enrico/gworkspace/viewer.html].

Figure 6.21. A simple tree control with one level of hierarchy

Guidelines

- Always give tree controls a label, positioned above or to the left of the tree, in sentence capitalization. Provide an access key in the label that allows the user to give focus directly to the tree.
- Use column headers when:
  - the tree has more than one column
  - the tree has only one column, but the user may wish to re-order the tree. This should rarely be true of single column trees.

  In most other situations, column headers take up unnecessary space, and the extra label adds visual clutter.
- Always label column headers when used. If the column is too narrow for a sensible label, provide a tooltip for the column instead. Apart from its obvious use, this will help ensure that assistive technologies can describe the use of the column to visually impaired users.
- Right align columns that contain only numbers, as this makes them easier to compare.
- Consider using a check box tree for multiple-selection trees, as these make it more obvious

that multiple selection is possible:

Figure 6.22. A simple check box tree

If you do this, you should normally set the tree control itself to be single-selection, but this depends on the particular task for which it will be used.

- For multiple selection trees, show the number of items currently selected in a static text label below the tree, for example, Names selected: 3. Such a label also makes it more obvious that multiple selection is possible.
- Consider providing Select All and Deselect All buttons beside multiple selection trees, if appropriate to the task.

15.1. Sortable Trees

As with lists, the user may find it useful to sort long or multi-column trees. See the guidelines in Section 14.1, “Sortable Lists” for more information.

16. Tabbed Notebooks

A tabbed notebook control is a convenient way of presenting related information in the same window, without having to display it all at the same time. It is analogous to the divider tabs in a ring binder or a file cabinet.

Figure 6.23. A typical notebook control with four tabs

Guidelines

- Do not put too many pages in the same notebook. If you cannot see all the tabs without scrolling or splitting them into multiple rows, you are probably using too many and should use a list control instead. See the example below.
- Label tabs with header capitalization, and use nouns rather than verbs, for example Font or Alignment. Try to keep all labels in a notebook the same general length.
- Do not assign access keys to tab labels, as this means you cannot use those access keys for any other control on *any* of the notebook pages without conflict. Even if you are able to assign access keys that would not conflict, it is better not to as it may be impossible to avoid the conflict when your application is translated to other languages. Assign an access key to every other control on each page, however.
- Do not design a notebook such that changing controls on one page affects the controls on any other page. Users are unlikely to discover such dependencies.
- If a control affects only one notebook page, place it on that notebook page. If it affects every page in the notebook, place it outside the notebook control, for example beside the window's OK and Cancel buttons.

- Use tabs that are proportional to the width of their labels. Don't just set all the tabs to the same width, as this makes them harder to scan visually, and limits the number of tabs you can fit into the notebook without scrolling. For example:

Figure 6.24. Fixed- and proportional-width tabs (preferred)

- Although the contents of each page in a notebook will take up a different amount of space, do not use larger than normal spacing around the controls in the "emptier" pages, and do not center the controls on the page.
- If your tab labels include icons, choose whether or not to show the icons at runtime based on the user's preference in the GNOME Menus and Toolbars desktop preferences dialog. Always show the text part of the label, however.

If you have more than about six tabs in a notebook, use a list control instead of tabs to switch between the pages of controls. For example:

Figure 6.25. Use of list control where there would be too many tabs to fit comfortably in a notebook

As in this example, place the list control on the left-hand side of the window, with the dynamic portion of the window immediately to its right.

16.1. Status Indicators

In some tabbed windows, such as preference windows, it might be desirable to indicate the status of a particular tab. This can be used to notify the user that a web page that is still loading or has been loaded, a new message is waiting in a particular instant messaging conversation, or that a document has not been saved. Such a status indicator should be an icon that is placed directly to the left of the tab label. Additionally, the tab label's color might be changed to indicate a certain status. Do not simply rely on a different coloring scheme for status indication.

17. Progress Bars

Progress bars are visual indicators of the progress of a task being carried out by the application, and provide important feedback. For information on using a progress bar within a progress window, see Section 5, “Progress Windows”.

You can use two main types of progress bars in your application— measured-progress bars and indeterminate-progress bars (the kind that bounce back and forth). In addition there are are three types of measured progress bars.

Guidelines

- Always use a measured progress bar when the length of a task can be precisely or approximately predicted. Otherwise, use an indeterminate progress indicator or a checklist window.

- Ensure that a measured-progress bar measures an operation's total time or total work, not just that of a single step. An exception is a progress bar that measures the total time or work of the current step in a progress checklist.

17.1. Time-remaining Progress Indicator

An animation consisting of a bar whose changing length indicates how much time remains in an operation, and text stating how much time remains before the operation will be complete. Time-remaining bars are the most useful type of progress bar.

Figure 6.26. A simple 'time remaining' progress bar

Use a time-remaining bar if your application will display an initial estimate of an operation's remaining time and then periodically display updated estimates. Each updated estimate should be based on changes that have occurred and that will cause the operation to finish more quickly or more slowly. If the operation will finish more slowly, your application can display an updated estimate that is greater than the estimate previously displayed.

17.2. Typical-time Progress Indicator

A bar whose changing length indicates how much time remains if an operation takes as long as it typically does. Typical-time bars are the least precise type of measured-progress bar, but they are more useful than indeterminate-progress bars.

Figure 6.27. A simple 'typical time remaining' progress bar

For some operations, you cannot estimate the time remaining or the proportion of work completed. However, if you can estimate the typical time for that operation, you can provide feedback with a typical-time bar.

If your application overestimates the completed amount of work, the length of the bar can indicate "almost complete" until the operation is complete. If your application underestimates how much work is complete, the application can fill the remaining portion of the bar when the operation is complete.

17.3. Indeterminate-progress indicator

An animated bar indicating only that an operation is ongoing, not how long it will take. One example is the "throbber" in a web browser. Indeterminate-progress bars are the least precise type of progress bar.

Figure 6.28. A simple 'indeterminate time' progress bar; the slider moves from left-to-right and back again until the operation is complete

Use an indeterminate-progress bar to provide feedback only for operations whose duration you cannot estimate at all.

18. Statusbars

A statusbar is an area at the bottom of a window that can be used to display brief information about the status of the application.

Figure 6.29. A simple statusbar

Guidelines

- Use statusbars only in application or document windows. Do not use them in dialogs, alerts or other secondary windows.
- Only place a statusbar along the bottom of a window.
- Only use statusbars to display non-critical information. This might include:
  - general information about the document or application. For example, current connection status in a network application, or the size of the current document in a text editor.
  - information about the task the user is currently performing. For example, while using the selection tool in a drawing application, "Hold Shift to extend the selection"
  - progress of a background operation. For example, "Sending to printer", "Printing page 10 of 20", "Printing Complete".
  - a description of the control or area of the window under the mouse pointer. For example, "Drop files here to upload them"

  Remember that statusbars are normally in the user's peripheral vision, and can even be turned off altogether using the application's View → Status Bar menu item. The user may therefore never see anything you display there, unless they know when and where to look for it.
- When there is no interesting status to report, leave a status bar panel blank rather than displaying something uninformative like "Ready". This way, when something interesting does appear in the statusbar, the user is more likely to notice it.
- If you want to make all or part of your statusbar interactive, use the following conventions:
  - Inlaid appearance for areas that respond to a double click
  - Flat appearance for areas that are not interactive

  In Figure 6.30, “An interactive statusbar”, the appearance indicates that the left area would respond to a double click (perhaps by saving the document), and the progress indicator on the right is non-interactive.

Figure 6.30. An interactive statusbar

Ensure that double-clicking in the status area does not provide any functionality that is not also available in the main application menu bar, or by some other accessible means.

- Provide a drag handle in the bottom right corner of the status bar of resizable windows. Subclasses of GtkStatusbar should use the drag handle provided by that class. A reimplementation of a status bar, which is discouraged, should also reimplement the GtkStatusbar drag handle in both appearance and function.

19. Frames and Separators

A frame is a box with a title that you can draw around controls to organize them into functional groups. A separator is a single horizontal or vertical line that you can use to divide windows into functional groups.

Frames with a border around their perimeter have traditionally been used for denoting groups of related controls. This is advantageous because it physically separates dissimilar controls, and also avoids repetition of the frame's label in individual member control labels. Unfortunately, they add visual noise that can both make a window appear more complex than it really is, and reduce the ability to quickly scan window elements.

Rather than using bordered frames, use frames without borders, bold labels to make the categories stand out, and indented contents. This, combined with good layout and spacing, is usually a better alternative to bordered frames.

Figure 6.31. Preferred frame style, using bold labels, spacing and indentation

Figure 6.32. Traditional frame style, using borders (deprecated)

Guidelines

- Before you add a frame with a visible border or separator to any window, consider carefully if you really need it. It is usually better to do without, if the groups can be separated by space alone. Do not use frames and separators to compensate for poor control layout or alignment.
- Do not mix framed and unframed groups in the same window.
- Do not nest one frame inside another. This results in visual clutter.
- If all the items in a group are disabled, disable the group title too.

20. Disclosure Triangles

TODO: Write some stuff about disclosure widgets.

Chapter 7. Feedback

1. Characteristics of Responsive Applications

Although highly responsive applications can differ widely from one another, they share the following characteristics:

- They give immediate feedback to users, even when they cannot fulfill their requests immediately.
- They handle queued requests as users would expect, discarding requests that are no longer relevant and reordering requests according to users' probable priorities.
- They let users do other work while long operations proceed to completion— especially operations not requested by users— such as reclaiming unused memory or other "housekeeping" operations.
- They provide enough feedback for users to understand what they are doing, and organize feedback according to users' abilities to comprehend and react to it.
- They let users know when processing is in progress.
- They let users know or estimate how long lengthy operations will take.
- They let users set the pace of work, when possible, and they let users stop requested tasks that have started but not finished.

Highly responsive applications put users in control by quickly acknowledging each user request, by providing continuous feedback about progress toward fulfilling each request, and by letting users complete tasks without unacceptable delays.

Even applications with attractive, intuitive user interfaces can lack responsiveness. Typically, unresponsive applications have at least one of the following problems:

- They provide late feedback— or no feedback— for users' requests, leaving users wondering what the application has done or is doing.
- When performing extended operations, they prevent users from doing other work or canceling the extended operation.
- They fail to display estimates of how long extended operations will last, forcing users to wait for unpredictable periods.
- They ignore users' requests while doing unrequested "housekeeping", forcing users to wait at unpredictable times— often without feedback.

Sometimes you can improve the responsiveness of an application without speeding up its code. For tips on how to make such improvements, see Section 3, “Responding to User Requests”.

2. Acceptable Response Times

Some user interface events require shorter response delays than others. For example, an application's response to a user's mouse click or key press needs to be much faster than its response to a request to save a file. The table below shows the maximum acceptable response delay for typical interface events.

Table 7.1. Maximum acceptable response times for typical events

UI Event	Maximum Acceptable Response Time
Mouse click, pointer movement, window movement or resizing, keypress, button press, drawing gesture, other UI input event involving hand-eye co-ordination	0.1 second
Displaying progress indicators, completing ordinary user commands (e.g. closing a window), completing background tasks (e.g. reformatting a table)	1.0 second
Displaying a graph or anything else a typical user would expect to take time (e.g. displaying a new list of all a company's financial transactions for an accounting period)	10.0 seconds
Accepting and processing all user input to any task	10.0 seconds

Make each response delay in your application as short as possible, unless users need time to see the displayed information before it is erased. The acceptable response delay for each event is based on a typical user's sense that the event is a logical point at which to stop or pause. The greater that sense is, the more willingly the user will wait for a response. Verify that your application responds to users' requests within the limits listed in the table above. If your application cannot respond within those limits, it probably has one or more general problems caused by a particular algorithm or module.

Guidelines

- Verify that your application provides feedback within 100 milliseconds (0.1 second) after each key press, movement of the mouse, or other physical input from the user.
- Verify that your application provides feedback within 100 milliseconds (0.1 second) after each change in the state of controls that react to input from the user— for example, displaying menus or indicating drop targets.
- Verify that your application takes no longer than 1 second to display each progress indicator, complete each ordinary user command, or complete each background task.
- Verify that your application takes no longer than 10 seconds to accept and process all user input to any task—including user input to each step of a multi-step task, such as a wizard.

3. Responding to User Requests

If your application takes too long to respond, users will become frustrated. Use these techniques to

improve the responsiveness of your application.

Guidelines

- Display feedback as soon as possible.
- If you cannot display all the information that a user has requested, display the most important information first.
- Save time by displaying approximate results while calculating finished results.
- If users are likely to repeat a time-consuming command in rapid succession, save time by faking the command's effects instead of repeatedly processing the command. For example, if a user adds several rows to a table stored in a database, you might display each new row immediately but delay actually creating each new row in the database until the user finished adding all the rows.
- Work ahead. Prepare to perform the command that is most likely to follow the current command. That is, use idle time to anticipate users' probable next requests. For example, as the user of an email application reads the currently displayed new message, the application might prepare to display the next new message.
- Use background processing. Perform less important tasks —such as housekeeping— in the background, enabling users to continue working.
- Delay work that is not urgent. Perform it later, when more time is available.
- Discard unnecessary operations. For example, to move back several pages in a web browser, a user might click the browser's Back button several times in rapid succession. To display the final requested page more quickly, the browser might not display the pages visited between the current page and that final page.
- Use dynamic time management. At run time, change how your application prioritizes user input and other processing, based on the application's current state. For example, if a user is typing text in one word-processing document while printing another, the word-processing application might delay the printing task if the user shifts to an editing task (such as cutting and pasting text) that requires greater resources.
- In your application, display an estimate of how long each lengthy operation will take.
    - If a command might take longer than 5 seconds to complete its work on an object, allow users to interact with any parts of the object and parts of the application that are not directly affected by the command.
    - If a command provides lengthy output, show partial results as they become available. Scroll the results (if necessary) until the user moves input focus to a component (e.g. a scrollbar or text area) involved in the scrolling.

4. Types of Visual Feedback

You can use two types of visual feedback for operations in your application— pointer feedback and progress animations.

4.1. Pointer Feedback

Pointer feedback changes the shape of the pointer. For example, a busy pointer indicates that an operation is in progress and that the user cannot do other tasks. A busy-interactive pointer indicates that an operation is in progress but the window is still interactive.

Figure 7.1. Busy pointer (left) and Busy-Interactive pointer (right)

4.2. Progress Animations

Progress animations show either how much of an operation is complete, or only that an operation is ongoing. Normally, these take the form of either a progress bar or a progress checklist.

Guidelines

- When displaying a progress animation, open it as soon as possible after you know it is required, and close it automatically as soon as the associated operation is complete.
- Use a measured-progress bar if your application can estimate either how long the operation will take, or what proportion of the operation is complete.
- If your application can make neither estimate, and the operation only has one step, use an indeterminate-progress bar. For operations with two or more steps, use a progress checklist that dynamically displays a check mark for each completed step.

4.2.1. Progress Bars

For information on different types of progress bars and when to use them see Section 17, “Progress Bars”.

4.2.1.1. Progress Windows vs. the Statusbar

In an application where the primary windows contain a status bar (which in turn contains a progress bar), it will often be the case that an operation's feedback could be presented in either the statusbar or a progress window. A rule of thumb is to use the statusbar when an operation is expected to take fewer than ten seconds, otherwise use a progress window. However, do consider the following when choosing between the two:

- Opening a new window, particularly when an operation is short, can needlessly disrupt the user's work-flow.
- Progress windows can convey more information.
- Multiple progress windows can be open at once, whereas only a single operation can be presented in a statusbar.
- Progress windows provide a Cancel button.

4.2.2. Checklist Windows

A checklist window shows the sequence of stages in an operation. See Section 5.1, “Checklist Windows”.

Figure 7.2. A Checklist Window

5. Choosing Appropriate Feedback

To determine which type of visual feedback to provide for a particular operation, consider these factors:

- Whether your application can provide an estimate of the operation's progress.
- Whether the operation blocks the user from issuing further commands in your application.
- Whether your application has a dedicated space, such as a status bar, for indicating the status of operations.

The table below shows which type of feedback to provide for operations that usually take at least 1 second to finish. In the "Appropriate Feedback" column, "Internal progress animations" means progress animations displayed in an application's dedicated status area, and "External progress animations" means progress animations displayed somewhere other than in a dedicated status area— typically, in an alert box.

Table 7.2. Visual feedback types for operations that take at least 1 second

Typical Duration > 5 seconds?	User blocked from issuing further commands?	Application has dedicated status area?	Appropriate feedback
Yes	Yes	Yes	Internal animation plus pointer feedback
Yes	Yes	No	Pointer feedback
Yes	No	Yes	Internal animation
No	Yes	Yes	Internal animation plus pointer feedback
No	Yes	No	External animation plus pointer feedback
No	No	Yes	Internal animation
No	No	No	External animation

Guidelines

- Use a busy pointer whenever users are blocked from interaction with your application for 1 second or longer. Display the busy pointer less than 1 second after the operation begins.
- If a command will likely take 10 seconds or longer to finish, provide a Stop or Cancel button, which can also be activated by pressing **Esc**, that lets users terminate the command's processing even if your application cannot undo the command's effects. See Section 6, “Allowing Interruptions”.

- When using an external animation, leave the window containing the animation on-screen for at least 1 second after the operation has completed, with a successful completion message. Change the Stop or Cancel button to an OK button during this period— pressing this button should close the window immediately.

6. Allowing Interruptions

Users sometimes need to stop a command— for example, because it is taking too long. Your application should let users stop commands in progress, even if stopping a command cannot undo or "roll back" all the command's effects.

Guidelines

- Place a Stop or Cancel button, which can also be activated by pressing **Esc**, near the progress animation for the interruptible command.
- Label the button Cancel if the whole operation can be cleanly abandoned with no side effects, leaving the system in the state it was in prior to the operation beginning. Terminate the command immediately when the user presses this button.
- Label the button Stop if the command can be interrupted, but its effects up to that point cannot (or should not) be reversed. When the user presses this button, open an alert box that warns of the potential side effects of stopping the command. The alert box should have only two buttons: one for continuing the command's processing, and one for immediately terminating it.

Alternatively, you can place the Stop or Cancel button near the control with which the user issued the command that needs to be stopped. Place the button here only if:

- There is no progress animation for the command, or
- The progress animation is in a window's status area or in another location that lacks space for a Stop or Cancel button.

In the alert box that appears after pressing a Stop button, ensure the message and button labels in the alert box are specific and precise. Ambiguous button labels can cause users to terminate or continue a command unintentionally. For example, use:

```
Continue deleting files? [Continue Deleting]
    [Stop Deleting]
```

rather than

```
Operation interrupted, continue? [Yes]
    [No]
```

since in the latter example, it is not clear whether pressing Yes would continue the operation or continue the interruption (i.e. cancel the operation).

Chapter 8. Visual Design

Visual design is not just about making your application look pretty. Good visual design is about communication. A well-designed application will make it easy for the user to understand the information that is being presented, and show them clearly how they can interact with that information. If you can achieve all that, your application *will* look good to the user, even if it doesn't have any fancy graphics or spinning logos!

1. Color

Color is a good tool for communicating information in a user interface. For example, it can be used to:

- strengthen a desktop's look and feel by enhancing a theme
- accent a dynamic alert in a system management application
- emphasize an element in a long list to expedite scanning
- add aesthetically pleasing details to an icon

However, color should always be regarded as a useful addition to your design, not as a necessity. Never depend upon colors alone to display important information, and keep in mind that if colors cannot be perceived correctly (for example, the user has an 8-bit system, or is color-blind), your application should still be usable.

1.1. Palette

A 32-color palette has been developed for the GNOME desktop. The palette may be downloaded from http://developer.gnome.org/projects/gup/images/ximian-palette. To use it in The GIMP, save it to your `~/.gimp_1.2/palettes` folder, and restart The GIMP. A single, consistently-used palette helps give a unified look and feel to the desktop while minimizing visual distractions. If you need a color that is darker or lighter than the colors in this basic palette (e.g., for anti-aliasing), choose a color that is closest to the hue you need, then darken or lighten as required.

Figure 8.1. The basic GNOME 32-color palette

Table 8.1. RGB and hexadecimal values for the basic palette

Color	Description	RGB	Hex	Color	Description	RGB	Hex
	Basic 3D Highlight	234 232 227	#EAE8E3		Basic 3D Medium	186 181 171	#BAB5AB
	Basic 3D Dark	128 125 116	#807D74		3D Shadow	86 82 72	#565248
	Green Highlight	197 210 200	#C5D2C8		Green Medium	131 166 127	#83A67F
	Green	93 117 85	#5D7555		Green	68 86 50	#445632

Color	Description	RGB	Hex	Color	Description	RGB	Hex
	Dark				Shadow		
	Red Highlight	224 182 175	#E0B6AF		Red Medium	193 102 90	#C1665A
	Red Dark	136 70 49	#884631		Red Shadow	102 56 34	#663822
	Purple Highlight	173 167 200	#ADA7C8		Purple Medium	136 127 163	#887FA3
	Purple Dark	98 91 129	#625B81		Purple Shadow	73 64 102	#494066
	Blue Highlight	157 184 210	#9DB8D2		Blue Medium	117 144 174	#7590AE
	Blue Dark	75 105 131	#4B6983		Blue Shadow	49 78 108	#314E6C
	Face Skin Highlight	239 224 205	#EFE0CD		Face Skin Medium	224 195 158	#E0C39E
	Face Skin Dark	179 145 105	#B39169		Face Skin Shadow	130 102 71	#826647
	Accent Red	223 66 30	#DF421E		Accent Red Dark	153 0 0	#990000
	Accent Yellow	238 214 128	#EED680		Accent Yellow Dark	209 148 12	#D1940C
	Accent Green	70 160 70	#46A046		Accent Green Dark	38 199 38	#267726
	White	255 255 255	#ffffff		Black	0 0 0	#000000

1.2. Hue, Brightness, Contrast

Users with vision disorders, such as color-blindness or low vision, require alternatives to default settings. A good user interface anticipates these needs by providing customizable preferences and support for accessible themes. Even better is an application that is already configured with carefully-chosen color and contrast defaults.

An estimated 11% of the world population has some sort of color-blindness. Those affected typically have trouble distinguishing between certain hues such as red and green (deuteranopia or protanopia), or blue and yellow (tritanopia). Therefore it is necessary to allow the user to customize colors in any part of your application that conveys important information. This means that your application must effectively convey information using just the colors from any theme that the user chooses.

A useful tool for reviewing information about color-blindness and checking legibility of images for color-blind users is Vischeck [http://www.vischeck.com/], an on-line tool that simulates the way an image or a website might appear to a user who has deuteranopia, protanopia, or tritanopia.

Figure 8.2. How the earth looks to a user with normal color vision (left), deuteranopia (middle), and tritanopia (right). (Images from

http://www.vischeck.com).

Other users have more problems with contrast levels rather than hue on their screen. Some users require a high level of contrast between background and foreground colors, such as black on white, white on black, or some other high-contrast combination. Others can experience discomfort unless they use low-contrast settings, such as gray text on a lighter gray background.

You can meet these needs by ensuring your application supports the accessible GNOME themes (found in the gnome-themes module in cvs), which include high and low contrast themes, and large print themes. This means you must supply default and large sizes of high-, low- and regular-contrast icon sets with your application.

Guidelines

- Use the GNOME color palette. If you need a darker or lighter shade, start from one of the colors from the palette and darken or lighten as needed.
- Do not use color as the only means to distinguish items of information. All such information should be provided by at least one other method, such as shape, position or textual description.
- Ensure your application is not dependent on a particular theme. Test it with different themes, especially high and low contrast accessibility themes, which use fewer colors, to ensure your application respects the settings. For example, all text should appear in the foreground color against the background color specified in the chosen theme.
- Select colors carefully. When they need to be recognizably different, select the light colors from orange, yellow, green or blue-green, and darker colors from blue, violet, purple or red, as most people affected by color-blindness already see blue, violet, purple and red as darker than normal.

2. Window Layout

2.1. General

Placement of visual components in an application is important because relationships between elements are indicated by their positions. This is called "layout" in interface design.

A clean layout is crucial to creating a smooth visual flow of information for the user. This section describes the proper component placement and spacing to use in GNOME applications. The major components discussed will be labels, icons, radio buttons and check boxes, text fields, command buttons, and drop-down menus.

2.2. Dialogs

When a user is scanning a complex preferences dialog consisting of many labels and corresponding check boxes, text fields, and drop-down combination boxes, it is easy to see how she can quickly become hindered by poor layout in the visual design. For information on laying out Alerts, see Section 4.3, “Spacing and Positioning Inside Alerts”

Figure 8.3. Improved window layout

In Figure 8.3, “Improved window layout”, the dialog on the left presents labels which are not left-aligned. The user's eye is not given a proper anchor to scan the dialog quickly.

As the labels are all similar in length, they should be left-aligned. Now the user has a firm left margin to anchor the eye and scan the list of items vertically more easily. If most of the labels in a group greatly differ in length, right-align them instead, so that the controls do not end up too far away from their corresponding labels.

Using frames with visible borders to separate groups within a window is deprecated. Use spacing and bold headers instead. This is more effective because there are fewer gratuitous lines to distract the user from the main content in the window. See Section 19, “Frames and Separators” for more details.

Try to keep components consonant with each other in terms of size and alignment. This is particularly important within a group of controls, so that the user's ability to quickly scan information is not sacrificed. Minimize as much as possible the need for the user's eye to jump around when scanning a layout.

Figure 8.4. Layout specifications

Guidelines

- Leave a 12-pixel border between the edge of the window and the nearest controls.
- Leave a 12-pixel horizontal gap between a control and its label. (The gap may be bigger for other controls in the same group, due to differences in the lengths of the labels.)
- Labels must be concise and make sense when taken out of context. Otherwise, users relying on screenreaders or similar assistive technologies will not always be able to immediately understand the relationship between a control and those surrounding it.
- Assign access keys to all editable controls. Ensure that using the access key focuses its associated control.

2.3. Size

GNOME is used on by many different people on a wide variety of different hardware. This includes people with older hardware and small displays. Laptop users and users of other portable devices also need to be able to work on smaller screens. Users who require large fonts, or assistive technologies that take over part of the screen, also have a correspondingly smaller area in which to display application windows and dialogs.

While these space restrictions are intended primarily to allow people with smaller displays to use Gnome it is also good discipline to keep windows and dialogs clear with not to many concepts all at once.

Guidelines

- Try to ensure that the size of a window is no larger than 480 pixels wide and 640 pixels hight in the default theme (including window decorations). Ideally your application should be usable at 600 by 400 pixels which leaves enough room for both top and bottom Panels.
- The Ratio of the screen width to height is 2:3 and things will look aesthetically pleasing if we can keep (near to) this proportion. . Dialogs [like a properties dialog] should [probably] be 300 by 400. Message Dialogs should be 600 by 200.

2.4. Spacing and Alignment

Provide adequate space between controls and groups of controls. This white space will make it easier for the user to find the information they need.

Guidelines

- As a basic rule of thumb, leave space between user interface components in increments of 6 pixels, going up as the relationship between related elements becomes more distant. For example, between icon labels and associated graphics within an icon, 6 pixels are adequate. Between labels and associated components, leave 12 horizontal pixels. For vertical spacing between groups of components, 18 pixels is adequate. A general padding of 12 pixels is recommended between the contents of a dialog window and the window borders.
- Break long lists of choices into smaller groups. For lists of less than about eight items, use radio buttons or check boxes. For longer lists, use a list control or drop-down list.
- Try to keep elements of the same type left-aligned with each other. For instance, in Figure 8.4, “Layout specifications”, the group titles (General and Actions) are left-aligned and justified with each other.
- Indent group members 12 pixels to denote hierarchy and association.
- Minimize the number of alignment points in your window. An alignment point is an imaginary vertical or horizontal line through your window that touches the edge of one or more labels or controls in the window.
- Right-justification within groups or the overall window (as indicated by the line labeled "justification" in Figure 8.4, “Layout specifications” is pleasing to the eye, but not crucial.
- Lay out components left-to-right, top-to-bottom. Generally, the first element the user is meant to encounter should be in the top-left, and the last in the bottom right. Keep in mind that when localized for non-western locales, interfaces may be reversed so that they read from right to left.
- Using "white" or blank spacing and indentation to delineate groups is cleaner and preferable to using graphical separators such as frames.
- Align controls in your layout *exactly*. The eye is very sensitive to aligned and unaligned objects. If nothing lines up with anything else in a window, it will be very hard for the user to scan the contents and find the information he wants. Two things that almost line up, but not quite, are equally distracting.

- Be consistent. Use the same spacing, alignment, and component sizes in all dialogs appearing in your application. The OK and Cancel buttons, for example, should all appear exactly 12 vertical and horizontal pixels from the lower right corner of every dialog window.
- Ensure that light and dark areas as well as spacing are equally distributed around the window. Keep in mind that every control or group of controls in your window has a visual "weight," depending on its overall size, color, and how much white space it includes. Darker, larger areas are "heavier," while paler, smaller areas are "lighter."
- Do not design windows that are more than 50% longer in one dimension than in the other. People are more comfortable looking at windows and dialogs whose dimensions stay within the golden ratio (about 1.6 to 1), a ratio that artists and architects have used to create aesthetically-pleasing paintings and buildings for thousands of years.

3. Text Labels

To a user with normal vision, textual output provides the majority of the information and feedback in most applications. To a visually-impaired user who may not be able to see or understand any additional graphical output, clear textual output is critical. You must therefore choose and position text carefully on the screen, and leave the choice of fonts and sizes to the user, to ensure that all users are able to use your application effectively.

3.1. Spacing and Alignment

Use spacing and alignment of text uniformly throughout your application. A basic rule of thumb is to put space between user interface components in increments of 6 pixels, going up as the relationship between related elements becomes more distant.

Table 8.2. Alignment and spacing for different Text elements

Element	Placement	Example
Large Icons (file browser)	Horizontally centered and (6 pixels, if specification necessary) below large icons	
Small icons (toolbar)	Vertically centered and (6 pixels, if specification necessary) to the right of small icons	
List control label	6 pixels above and horizontally left aligned with the list control or 12 pixels to the left of and horizontally top aligned with list control	
Radio button and check box labels	6 pixels to the right of and vertically center aligned with the radio button	
Textfield labels	6 pixels to the left of and vertically center aligned with the textfield control	
Button labels	12 pixels of padding to either side of centered text (and any accompanying graphic). If	

Element	Placement	Example
	appearing in a group of buttons, longest button label sets button size, center all other button labels and accompanying graphics in same-sized buttons	
Other component labels (e.g., spin boxes, text fields	12 pixels between the longest text label and its associated component, all other text labels in component grouping left aligned with the longest label. All labels vertically center aligned with associated components	

Guidelines

- If the label precedes the control it is labelling, end the label with a colon. For example, Email: to label a text field into which the user should type their email address. This helps identify it as a control's label rather than an independent item of text. Some assistive technology screen review utilities may also use the presence of a colon to identify text as a control label.
- Ensure that a label with a mnemonic is associated with the control it labels.
- Left-align components and labels, unless all the labels in a group have very different lengths. If they do, right-align the labels instead, to ensure that no controls end up too far away from their corresponding labels.
- Choose label names carefully. Label objects with names that make sense when taken out of context. Users relying on screenreaders or similar assistive technologies will not always be able to immediately understand the relationship between a control and those surrounding it.
- Be consistent with label usage and semantics. For example, if you use the same label in different windows, it will help if it means the same thing in both windows. Equally, don't use labels that are spelled differently but sound the same, e.g., "Read" and "Red", as this could be confusing for users relying on screenreaders.
- Don't use the same label more than once in the same window. This makes life difficult for users relying on tools like magnifiers or screen readers, which cannot always convey surrounding context to the user.
- Do not hard-code font styles and sizes. The user should be able to adjust all sizes and typefaces.
- Do not use more than two or three different fonts and sizes in your application, and choose visually distinct rather than similar-looking fonts in one window. Too many font sizes and styles will make the interface look cluttered and unprofessional, and be harder to read. In general, always use fonts from the current theme, and specify relative rather than absolute sizes.
- Do not use graphical backdrops or "watermarks" behind text, other than those specified by the user's chosen theme. These interfere with the contrast between the text and its background. This can cause difficulty for users with visual impairments, who will therefore normally choose themes that always use plain backdrops.

3.2. Capitalization

Two styles of capitalization are used in GNOME user interface elements:

Header capitalization	Header capitalization Capitalize all words in the element, with the following exceptions: • Articles: *a*, *an*, *the*. • Conjunctions: *and*, *but*, *for*, *not*, *so*, *yet* ... • Prepositions of three or fewer letters: *at*, *for*, *by*, *in*, *to* ...
Sentence capitalization	Sentence capitalization Capitalize the first letter of the first word, and any other words normally capitalized in sentences, such as application names.

The following table indicates the capitalization style to use for each type of user interface element.

Table 8.3. Capitalization Style Guidelines for User Interface Elements

Element	Style
Check box labels	Sentence
Command button labels	Header
Column heading labels	Header
Desktop background object labels	Header
Dialog messages	Sentence
Drop-down combination box labels	Sentence
Drop-down list box labels	Sentence
Field labels	Sentence
Filenames	Sentence
Graphic equivalent text: for example, Alt text on web pages	Sentence
Group box or frame labels	Header
Items in drop-down combination boxes, drop-down list boxes, and list boxes	Sentence
List box labels	Sentence
Menu items	Header
Menu items in applications	Header
Menu titles in applications	Header
Radio button labels	Sentence
Slider labels	Sentence
Spin box labels	Sentence

Element	Style
Tabbed section titles	Header
Text box labels	Sentence
Titlebar labels	Header
Toolbar button labels	Header
Tooltips	Sentence
Webpage titles and navigational elements	Header

Capitalization guidelines for other languages

Languages other than English may have different rules about capitalization. For example, Swedish has no concept of Header capitalization. Contact the GNOME Translation Project [http://developer.gnome.org/projects/gtp/contact.html] if you are in doubt about how to capitalize labels in a particular language.

4. Fonts

Only use the fonts that the user has specified in their theme, and in sizes relative to the default size specified in their theme. This will ensure maximum legibility and accessibility for all users.

Do not mix more than two or three font sizes and styles (underlined, bold, italicized) in one window, as this will look unprofessional and distract the user from the information being conveyed.

Provide alternatives to WYSIWYG where applicable. Some users may need to print text in a small font but edit in a larger screen font, for example. Possible alternatives include displaying all text in the same font and size (both of which are chosen by the user); a "wrap-to-window" option that allows you to read all the text in a window without scrolling horizontally; a single column view that shows the window's contents in a single column even if they will be printed in multiple columns; and a text-only view, where graphics are shown as placeholders or text descriptions.

Chapter 9. Icons

Icons are a graphical metaphor presenting a visual image that the user associates with a particular object, state or operation. When a user sees a good icon they are immediately reminded of the item it represents, whether that be an application in the panel menu or the "right aligned" state in a word processor toolbar.

- Icons can assist the user in rapidly scanning a large number of objects to select the desired item. Particularly after a user is accustomed to an icon's appearance, they can identify it more rapidly than a text label.
- Icons can augment text by providing visual suggestions to accompany the descriptive text. Some things are easier to communicate with a picture, even a very small one.
- Icons can compactly represent a large number of objects when there is insufficient space to display textual descriptions (such as in a toolbar).

1. Style

GNOME uses a soft, three-dimensional look. This style is achieved by using antialiasing, shading and highlighting techniques. The *Gnome Icons* [http://developer.ximian.com/articles/tutorials/icons/] tutorial details how one of GNOME's leading artists creates some of these effects.

Components of an icon style can be broken down into several categories such as perspective, dimensionality, lighting effects and palette. These components play an important part in giving a group of icons a collectively distinctive look. For instance, the Java Look and Feel is recognizable by its use of a primary eight-color palette, interior highlighting and diagonal gradients. The Macintosh Aqua style is recognizable by its use of a cool palette based on blue, lighting effects mimicking reflectivity and antialiasing. The GNOME style exhibits a subdued thirty-two color palette, soft drop shadows and a mix between cartoonish and photorealistic graphics.

Table 9.1. A globe in different icon styles

Java Metal	MacOS/X Aqua	GNOME

1.1. Perspective

Table perspective. Presents objects as if they were sitting on a table or desk in front of the user.

Figure 9.1. Illustration of the table perspective

Shelf perspective. Presents objects as if they were propped up on a shelf at eye level. Make it look like a police line-up.

Figure 9.2. Illustration of the shelf perspective

1.2. Lighting

Upper left. Design as if there is lighting coming from the upper left corner, with a soft drop-shadow cast within the icon's 48x48 (original design size) borders (120 degrees, 4 pixel distance, 4 pixel blur).

Overhead. Design as if there is a light source placed above the "camera", casting a shadow down.

1.3. Palette

Icons should use colors based on the basic thirty-two color palette, darkening or lightening the colors to achieve the desired look. See Section 1.1, “Palette”

2. Kinds of Icons

Table 9.2. Specifications for different kinds of icons used within GNOME

Icon Type	Sizes (pixels)	Perspective	Light Source
Object / Document Icons	24x24, 48x48*, 96x96	Table	Upper Left
Application Icons	24x24, 48x48*	Table	Upper Left
Toolbar Icons	24x24*, 48x48	Shelf	Overhead
Menu Icons	16x16	Shelf	Overhead

(* denotes the primary size for this kind of icon)

2.1. Document Icons

If possible, document icons should convey the type of the file using a physical object. For example a good icon for MPEG video would be a movie reel. Failing the existence of an appropriate object, when a document type corresponds to a specific application, another option is to use a piece of paper with the corresponding application's icon overlayed it as the document icon. This may be appropriate for a document type such as an application's settings files.

- Do *not* display a piece of paper behind a document icon unless the document type has a use correspondence with physical paper (or a suitable object was not found and you are using the application icon). For example, the final state of most word processing documents is a piece of paper, so it is appropriate to use a piece of paper in the icon. On the other hand, a movie on the computer has little association with a piece of paper, so a piece of paper behind the movie reel primarily introduces visual noise. The use of a piece of paper in most or all document types creates an additional problem: it is harder to scan large numbers of icons because they do not possess distinct outlines. A useful technique for creating a subtle difference between document types with similar roles (for example, between "JPEG", "PNG", "GIF", etc) is to use different colours. Expert users who need to make this distinction frequently will become accustomed to these differences.

- Do *not* include a file extension in the icon. The document icon's job is not to convey such precise information but to allow for rapid visual distinction between documents. Additionally, this text will not be displayed in the user's preferred font and size. Because many document types are associated with multiple file extensions, a file extension embedded in the icon will also frequently be wrong. In a context where the file extension is actually useful, the

application should composite the information onto the icon at runtime (thereby using the correct font and size, as well as getting the extension right).

- Do *not* customize document icons to a particular Nautilus theme. Document icons are likely to be used in conjunction with a variety of different icon themes, and should work well with all of them.

2.2. Application Icons

Application's that handle documents should reflect the kind of document they handle in the icon. If an application's predominant purpose is to edit a particular kind of document, it should use this document's icon as its icon.

2.3. Toolbar Icons

The idea of a toolbar as a shelf filled with tools should be reflected in toolbar icons. Toolbar icons should have the perspective of being viewed head on, as if they were actually sitting on a shelf at eye-level. Some design guides refer to this perspective as "flush".

- Ensure that toolbar icons which will be used together are easy to visually distinguish. Try to make the icons' silhouettes distinct from one another.
- While most user's will view toolbar icons at 24x24 pixels, it is important to include a "large print" set of icons at 48x48 pixels for accessibility reasons.
- Often, you will not have to design any toolbar icons yourself as GTK provides a wide variety of stock icons. You should use these whenever representing one of their intended items. This establishes consistent language across applications, and makes it easier for users to search for items on the toolbar. Do not use stock toolbar icons for anything other than their intended purpose, however, as this will make your application inconsistent with others, and could easily confuse your users.

  To browse the available stock icons, install the development packages for GTK version 2.x and run **gtk-demo**. Double click on Stock Item and Icon Browser to activate the stock icon browser. Note that icons vary in available resolution, so the images presented in the icon browser should not be taken as indicative of the maximum quality of an image. To view the images in PNG format, look in the GTK 2 source code under `gtk/stock-icons`.

2.4. Menu Icons

Principles of toolbar icon design should be followed with menu icons, just at a smaller size. Where a corresponding toolbar icon exists, a menu icon should mirror its design.

3. Designing Effective Icons

Rule of Thumb for Icon Metaphors

"If you have to think about an icon to 'get it', the metaphor is too complex"

- **Design Functionally Suggestive Icons.** Icons should be suggestive of the functionality with

which they are associated. The best icon will suggest to the user the primary purpose of the program or operation without having to read accompanying text. Users recognize functionally suggestive icons more rapidly than other forms because they directly associate with a physical object or action.

Figure 9.3. A functionally suggestive icon for a word processor

Figure 9.4. A functionally suggestive icon for underline

- **Make Icon Silhouettes Distinct.** It is important to make it easy to visually distinguish icons that will be used together, for example toolbar icons and document icons. The human visual system is excellent at making rapid distinctions between items based on shape, thus a good way to help your users sort through a large number of icons is to use different shapes. You can see the shape of an icon most clearly by turning it into a silhouette: blacken all areas of the icon which are not transparent.

Example 9.1. Distinct silhouettes from the Nautilus Crux theme

3.1. Suggested Design Process For Toolbar and Menu Icons

For accessibility reasons, you should create high and low contrast and large print versions of all icons, in addition to the regular size and contrast icon. A suggested process for conveniently integrating this into your icon design is as follows:

1. Draw the basic outline as close to 48x48 pixels as possible:
2. Fill in with black and white to create detail. Do not add gratuities such as drop shadows or anti-aliasing:
3. Use the finished image as the large print high contrast icon:
4. GNOME will automatically scale it down to create the 24x24 high contrast icon:
5. Or you may hand-create a 24x24 version, which will be superior in contrast and sharpness:
6. Add color and anti-aliasing to the large print high contrast icon:
7. Add gradients for a smooth, realistic effect:
8. Add a drop shadow (120 degree global angle, 4 pixel distance, 4 pixel blur, 40% opacity), and use the finished image as the large print regular contrast icon:
9. Now you should hand-create create a version of this icon at 24x24. Do *not* simply scale the larger icon, as this icon will be seen by the majority of users and the result of scaling would be less distinct:

10. Create a layer with the large print regular contrast icon's same outline and size then overlay that on the color icon. Give the overlay layer 40% opacity, and use the finished image as the large print low contrast icon:

11. GNOME will automatically scale it down to create the 24x24 low contrast icon:

12. Or you may hand-create a 24x24 version, which will be superior in contrast and sharpness:

3.2. Problems to Avoid

- **Avoid name suggestive icons.** Some icons, such as the Nautilus icon, do not suggest the program's purpose, but instead suggest the program's name. This is less desirable than a functionally suggestive icon, because an extra layer of abstraction is added (rather than associating file management with an icon representing files, they have to associate file management with nautilus with an image of a nautilus shell). Additionally it makes it difficult for new users who may not know what "Nautilus" is, and hence will not recognize a shell icon as the file manager.

Figure 9.5. A name suggestive icon for Nautilus

- **Do *not* include meaningful text in icons.** Icons which contain the text of the program name in the icon. They effectively contain no metaphor or picture for the user to identify with, and are probably harder to read than the accompanying caption. Since icons draw the eyes, an icon that is harder to identify than text is potentially worse than no icon at all. Hence "text icons" should not be used. Moreover, text should be avoided in icons because it makes the icons difficult to translate. If there is text in icons it should not form words in your native language, a good metric for ensuring that the particular text is not lending to the meaning of the icon.

Figure 9.6. Text in the old GEdit icon

- **Do *not* rely on information your users will not have.** Random icons appear to have no association with the program (except perhaps some odd connection in the mind of the developer). These icons should *never* be used and will likely serve to confuse the user more than help them. The icon's purpose should not be to "look pretty"; this is merely a very desirable side effect. The sodipodi logo is a squirrel, which they show as their icon. However, because the logo has no obvious connection *to a user*, it is a poor icon. Make sure that you are not relying on information that users won't necessarily posses.

Figure 9.7. A seemingly random icon for SodiPodi

- **Do *not* include extraneous information.** Remember that icons will often be viewed in a smaller form. Too much information may render the icon unintelligible when it is shrunk in size (e.g. to be placed on a panel, or in the tasklist). Too much information also makes it easier for users confuse the purpose of the application. For example, in user testing many users thought the Evolution icon would launch a word processor. They were misled by the pencil and the paper, which could be seen as extraneous information: it is implicit that the mail program will allow you to write messages as well as receive them. A better icon might have

been a simple envelope. Foremost in the icon designer's mind should be a consideration of the minimal visual elements necessary to express the purpose of the program.

Figure 9.8. Extraneous information - the Evolution icon

The Gnumeric icon is a great icon except for the introduction of extra visual noise. The extra sheet of paper with the 'g' on it behind the spreadsheet and chart adds no significant value to the icon and provides extra visual distraction. In this case the contribution of the extraneous element to the appearance of the icon is negative. Simple well balanced icons look more attractive than cluttered icons. An improved icon might contain only the spreadsheet and chart; larger because they can use all of the space in the icon, and hence more visually distinct.

Figure 9.9. Extraneous information - the old Gnumeric icon

- **Do *not* include body parts in the icon.** Because GNOME aims to be an international desktop, it needs to avoid imagery that is potentially offensive or crass to other cultures. A prime source of offensive imagery is various body parts in a number of different configurations. Aside from offensive gestures with the hands, arms or fingers; body parts that are considered "clean" in one culture (such as eyes), will be considered tasteless or gross to another (such as a nose). Based on a survey of icons in GNOME, body parts frequently appear in the least communicative icons (often "pointing" at some element in the icon); they are being used as an ineffective crutch for poor metaphor. In these situations body parts should *not* be used. Even in situations where the metaphor is appropriate (for example an eye representing the sawfish appearance capplet) it is better to avoid using a body part. Often body parts have been used in GNOME to suggest a human "choosing" or "using" something. This is normally an unnecessary point for the icon designer to make. People naturally attempt to understand objects in reference to themselves (show someone a bat and they will think of hitting something with the bat, show someone a tool and they will think of using it, etc). For example, the font selector shows a finger pointing to an "F" suggesting the user choosing between a series of fonts. A better icon would be the text "Aa" presented in an ornate font (calling attention to the font rather than the text). The user doesn't need to be told that they are "choosing" the font, they can infer that easily.

Figure 9.10. Using body parts - the font selector icon

Figure 9.11. A better icon for the Font Selector

- **Do *not* base icons off word puns.** This should be avoided for a couple reasons, the most obvious of which is that puns do not translate well. For example, representing the "system log monitor" as a log will likely be uncommunicative in languages other than English. Additionally, most users do not comprehend the word play until it is too late for the icon to assist them. Even after being familiar with the "system log monitor" being represented as a log, users do not form the association fast enough for the icon to assist through in scanning through menu entries. A popular instance of this problem was the proliferation of icons representing the "World Wide Web" as a spider web in the mid 1990s. Part of the value of icons is that they bypass linguistic comprehension and hence are complementary to captions, allowing users to utilize more areas of the mind than linguistic recognition (already used in

scanning for captions) when they hunt for items.

Figure 9.12. Word play - System Log Monitor icon

- **Do *not* employ violent imagery.** Just as words like "kill" and "slay" are inappropriate in interfaces, violent or destructive icons should be avoided. The "shut down" icon uses the image of an explosive detonation switch, presumably trying to convey the idea of ending something abruptly. However, this icon is likely to intimidate some users of the computer who will not want to click on the icon for fear of breaking something.

Figure 9.13. Destructive-looking Shutdown icon

4. Designing Accessible Icons

The GNOME desktop includes accessible themes that make the desktop and the applications running on it accessible to users with a range of visual impairments. By default, these are:

- a high contrast theme
- an inverse high contrast theme
- a large print theme

The following accessible themes are also available:

- a high contrast large print theme
- an inverse high contrast large print theme
- a low contrast theme
- a low contrast, large print theme

To be considered fully accessible, all icons in your application must be replaced by a suitable alternative when one of these themes is used.

Adding icons to accessible themes

When you have designed high and low contrast versions of all your icons, submit them to the gnome-themes module maintainers for inclusion in the appropriate themes.

4.1. High Contrast Icons

High contrast icons, in conjunction with a high contrast theme, benefit users with reduced visual acuity. A high contrast icon has these characteristics:

- Drawn only in black and white, with an external white border (black for inverse high contrast icons). One additional color may be used, but only if it is essential to the meaning of the icon.
- Simple visual style, typically comprising only a single symbol
- All lines and borders at least 4 pixels wide (at 48x48 pixels)

Figure 9.14. Part of application window rendered in high contrast, large print theme

This style allows high contrast icons to be distinguishable when viewed by a user with reduced visual acuity. Below is an approximation of what well-designed high contrast icons look like when viewed by such a user.

Table 9.3. Simulation of low vision user viewing high contrast icons

Description	High Contrast Icon	Simulated Appearance
Book		
CD-ROM		
Copy		

If a regular icon uses a simple, straightforward metaphor the corresponding high contrast icon can often use the same metaphor. In many cases the same metaphor will need to be drawn differently to create a simplified high contrast icon.

Figure 9.15. Simplified representation of metaphors for high contrast icons

High contrast icons are most easily created in a vector drawing application. Black and white shapes are layered to create a simplified icon. The process feels like layering black and white pieces of construction paper, as if you were assembling a collage.

Figure 9.16. Layered technique for high contrast icons

Reuse existing shapes

Often shapes from existing high contrast icons can be resized and reused to more quickly build up a new icon.

Don't forget the border!

Design high contrast icons over a temporary background color so you don't forget to draw the external white border.

4.1.1. High contrast inverse icons

High contrast inverse icons serve the same purpose as high contrast icons, but are easier to see for some users. Create a high contrast inverse icon simply by inverting the pixel values of the equivalent high contrast icon, for example in GIMP.

Figure 9.17. Part of application window rendered in high contrast inverse, large print theme

4.2. Low Contrast Icons

Low contrast icons and themes are useful to users who are particularly sensitive to light, to the extent that they find even a high contrast inverse theme uncomfortable to use. In low contrast icons, the colors are compressed towards the middle value range, so they do not contain any large luminance values (the 'V' in HSV). That is, dark colors are lightened, and light colors are darkened.

Figure 9.18. Part of application window rendered in low contrast theme

Low contrast icons are generated from the existing regular icons by adjusting the levels. In GIMP, use these values in the Levels dialog:

- Input: 100, 1.25, 200
- Output: 100, 160

Figure 9.19. Levels dialog in GIMP showing correct levels for generating low contrast icons

Batch conversion

Large numbers of regular icons can be quickly converted to low contrast by using GIMP's scripting facilities. A script [http://cvs.gnome.org/viewcvs/gnome-themes/low-contrast-preset.scm?view=markup] to do this is provided in the gnome-themes module.

Chapter 10. User Input

1. Mouse Interaction

1.1. Buttons

Figure 10.1. A plethora of pointing devices: mouse, trackball, foot-operated mouse, joystick, trackpad, and a finger-mounted pointing device.

For most users, the mouse provides the main way of interacting with graphical user interfaces. The term "mouse" is used in this chapter to include other pointing devices that can be used to move the pointer around the screen, such as trackballs, trackpads, spaceballs, graphics tablets, or assistive technology devices that emulate a mouse.

For right-handed users, the left button on a conventional mouse is used for the majority of mouse actions. We therefore call it the left button here, even though that may not physically be the case. For this reason, you may sometimes see this button referred to in code or documentation as "Button 1" or the "Selection Button".

Similarly for right-handed users, the right button on a conventional mouse is used for operations involving pop-up menus. We therefore call it the right button in this chapter. You may sometimes see this button referred to in code or documentation as "Button 3" or the "Menu Button".

A conventional mouse with three buttons normally has its third button (or a scrollwheel that acts as a button when pushed) between the left and right buttons. We therefore call it the middle button, but you may sometimes see this referred to in code or documentation as "Button 2" or the "Transfer Button".

Guidelines

- Your application uses left button gestures for selecting, activating components, dragging, and the display of drop-down menus.
- Your application uses right button gestures to display and select actions from a popup menu.
- Your application uses the middle button to paste the current PRIMARY (usually the last-highlighted) selection at the pointer position, as follows:

Table 10.1. Effect of modifier keys on a middle button transfer operation

Modifier	Function
Unmodified	Copy selection
Ctrl	Copy selection
Shift	Move selection
Shift-Ctrl	Create link, shortcut or alias to selection

Do not over-ride this functionality in any part of your user interface where the transfer action

is likely to be useful. If you do intend to use the middle button for a different purpose somewhere, only do so as a shortcut for experienced users, and only for operations that can also be performed without using the right button or middle button.

- If present on the mouse, the scrollwheel should scroll the window or control under the pointer, if it supports scrolling. Initiating scrolling in this way should not move keyboard focus to the window or control being scrolled.
- **Ctrl**-scrollwheel-up should zoom into the window or control under the mouse pointer, and **Ctrl**-scrollwheel-down should zoom out. Zooming in this way should not move keyboard focus to the window or control being zoomed.
- Do not depend on input from the middle or right mouse buttons. As well as being physically more difficult to click, some pointing devices and many assistive technology devices only support or emulate the left mouse button. Some assistive technologies may not even emulate the mouse at all, but generate keyboard events instead.
- Ensure that every operation in your application that can be done with the mouse can also be done with the keyboard. The only exceptions to this are actions where fine motor control is an essential part of the task. For example, controlling movement in some types of action games, or freehand painting in an image-editing application.
- Do not warp the mouse pointer, or restrict mouse movement to part of the screen. This can interfere with assistive technologies, and is usually confusing even for users who do not rely on assistive technologies.
- Do not require the use of chording (pressing multiple mouse buttons simultaneously) for any operations.
- Do not require the use of multiple (triple- or quadruple-) clicking actions for any operations, unless you also provide an accessible alternative method of performing the same action.
- Allow all mouse operations to be canceled before their completion. Pressing the Esc key should cancel any mouse operation in progress, such as dragging and dropping a file in a file manager, or drawing a shape in a drawing application.
- Do not assign any actions exclusively to the middle button of a three-button mouse, as not all mice have one.
- Do not hard-code mouse target sizes, or make them too small. Define any mouse targets to be at least as large as the arrow button in a GtkSpinBox in the current gtk theme. Bear in mind that a user with impaired dexterity or vision may be using a theme that results in considerably larger widgets than the default theme.
- Do not refer to particular mouse buttons in your interface unless absolutely necessary. Not everybody will be using a conventional mouse with left, middle and right buttons, so any text or diagrams that refer to those may be confusing.

1.2. Selecting Objects

1.2.1. Mouse and keyboard equivalents

For controls or windows that contain a number of objects that the user can select, either singly or multiply, ensure the following mechanisms are in place to allow selections to be made using either the mouse or the keyboard.

Table 10.2. Standard mouse and keyboard selection mechanisms

	Mouse	Keyboard
Select item, deselect all others	Click	Space
Add/remove item from selection	**Ctrl** click (toggles item's selected state)	**Ctrl**-Space (toggles focused item's selected state)
Extend selection	**Shift** click	**Shift**-Space, **Shift-Home**, **Shift-End**, **Shift-PageUp**, or **Shift-PageDown**
Move focus	Click appropriate item to select it	Cursor keys, **Home**, **End**, **PageUp**, and **PageDown** move focus and selection simultaneously. **Ctrl**-cursor keys, **Ctrl-Home**,**Ctrl-End**, **Ctrl-PageUp**, and **Ctrl-PageDown** move focus without affecting current selection.
Select All	Click first item, then **Shift** click last item	**Ctrl-A**
Deselect All	Click container background	**Shift-Ctrl-A**
Activate selection	Double-click to activate a single selection. **Shift** or **Ctrl** double-clicking extends or adds item to selection first before activating the entire selection.	Return activates entire selection. If nothing is currently selected, selects currently-focused item first.
Invert Selection	No mouse equivalent	**Ctrl-I**

1.2.2. Bounding Box Selection

For a container whose objects may be arranged in two dimensions, for example the icon view in a file manager, allow multiple selection by dragging a bounding box (sometimes called a "rubber band") around one or more objects. **Shift** left button drag should add all the objects within the bounding box to the existing selection. **Ctrl** left button drag should toggle the selected state of all the objects within the bounding box.

Guidelines

- Allow a bounding box selection to begin only if the initial mouse button press is made:
  - Within the bounds of the container's background, and
  - outside the bounds of any another object in the same container that can be dragged.

  In a drawing application, for example, this means that a bounding box click and drag could start on a blank area of the canvas, or within a shape that had been locked down to prevent accidental editing, but not in an active shape which would itself be dragged instead.

- Select any objects that lie wholly or partly within the bounding box when the mouse button is released.
- Use dynamic highlighting during the drag to show which objects will be selected. Do not wait until the mouse button is released. This avoids any uncertainty about which objects will be selected by the bounding box.
- When a bounding box is being dragged out within a scrollable window, support automatic scrolling of that window when the box is dragged near the window's edges.

Figure 10.2. Examples illustrating dynamic selection highlighting during bounding box selection. In the first example, the folder color and label highlighting changes to indicate selection. In the second, selection is indicated by the addition of resizing handles to selected objects.

1.3. Drag and Drop

Drag and drop is a direct manipulation technique, where you perform actions on selected objects by moving them around the screen with the mouse. You "drag" an object by clicking it, then holding the button while you move the pointer to the object's target location. The object is "dropped" at that location by releasing the mouse button.

Guidelines

- Use drag and drop only where the user can reasonably guess what the effect might be. The most common uses are:
  - to move or copy objects from one place to another
  - to link one object to another
  - to perform an action on the objects by dropping them onto an icon representing that action, such as a trash can or printer icon.
- Provide visual feedback throughout a drag and drop operation. Highlight valid targets and change the mouse pointer as it passes over them. Use the "no drop" mouse pointer when passing over invalid drop targets. See also Section 1.3.2, “Mouse Pointers to Use for Drag and Drop”.
- Augment the mouse pointer with a representation of the objects being dragged. Keep this representation small or make it translucent, so as not to obscure possible drop targets underneath it. See also Section 1.3.2, “Mouse Pointers to Use for Drag and Drop”.

  Figure 10.3. Example of copy pointer augmented by an icon representing the file being copied

- Only allow objects to be copied between applications, not moved. This avoids any confusion about which application's Undo function reverses the operation.

- Allow the user to cancel a drag and drop operation by all of these methods:
  - pressing **Esc** before releasing the mouse button
  - dropping the object back on its original location
  - performing a query drag and selecting Cancel on the pop-up menu (see Section 1.3.1.2, “Query Drag”)
  - dropping the object on an invalid drop target.
- Allow the user to undo the effects a drag and drop operation by selecting Edit → Undo.
- Allow multiple objects to be dragged by **Shift** or **Ctrl** selecting them, then dragging any one of the selected objects.
- Ensure that keyboard users can replicate all drag and drop actions using only menu items or keyboard shortcuts, such as Copy (**Ctrl-C**) and Paste (**Ctrl-V**).
- When an item is being dragged within or into a scrollable window, support automatic scrolling of that window when the mouse is moved near its edges.
- Pop up a menu when the user attempts to drop multiple objects on a target that only accepts single objects. On the menu, list all the objects being dragged, and a Cancel item.

1.3.1. Overriding drag and drop behavior

1.3.1.1. Keyboard Modifiers

Allow the user to force the behavior of a drag and drop operation by holding the **Ctrl**, **Shift** or both keys throughout. If the user changes modifier keys after they have started the drag, change the mouse pointer immediately and perform the new action when the mouse button is released.

Table 10.3. Effect of modifier keys during a drag and drop operation

Modifier	Function
Ctrl	Copy
Shift	Move
Shift-Ctrl	Create link, shortcut or alias

1.3.1.2. Query Drag

Allow the user to drag objects with the middle button , or with **Alt** left button. Pop up a menu when the mouse button is released, offering the choice of Copy, Move and Link (or whichever subset of those actions is available), and Cancel. Dragging in this way is known as query drag because it prompts the user before changing anything.

1.3.2. Mouse Pointers to Use for Drag and Drop

Use the default GTK drag and drop pointers for the standard transfer operations listed below. This consistency helps ensure the user will know exactly what to expect when they release the mouse button. If you have to design a pointer for a non-standard transfer action not listed here, follow the style of the standard pointers.

Table 10.4. Mouse Pointers for Drag and Drop

Pointer Shape	Meaning
	Move selection. The dragged selection will be moved to the drop location, removing it from its previous location.
	Copy selection. The dragged selection will be copied to the drop location, leaving the original intact.
	Link selection. A link to the selection will be inserted at the drop location. How the link appears will be application-dependent, it may be a hyperlink, an icon, or a duplicate of the original selection, for example.
	Middle button or Alt-left button drag. A pop-up menu will be posted at the drop location to ask whether the user wants to Move, Copy, or Link the selection, or Cancel the operation.
	Can't drop here. Show this pointer while the mouse is over an area where the selection cannot be dropped.

1.4. Mouse Interaction with Panel Applications (Applets)

All objects on the desktop must behave consistently. Despite their specialized nature, applets are no exception.

Guidelines

- The unmodified left mouse button must be sufficient to operate all your applet's controls. Applets are meant to be simple enough that modified clicking, or clicking with other mouse buttons (except to pop up the applet's menu) is never required.
- Clicking the right button *anywhere* within the applet's enclosing window must display either the popup menu for the whole applet, or the popup menu for the control under the mouse pointer. Do not have "dead areas" in your applet that do not respond to a right click.
- Do not use the middle button for anything except dragging the applet to a new location. Middle-clicking and dragging anywhere within the applet window must move the applet, do not require a drag bar or similar device.

  Ctrlleft button drag should copy the applet, if moving to another panel; unmodified drag or **Shift**left button drag should move the applet, if moving to another panel. If moving within same panel, **Ctrl**=switched movement, **Shift**=push movement, **Alt**=free movement.

2. Keyboard Interaction

2.1. Keyboard Navigation

A well-designed keyboard user interface plays a key role when you are designing applications. Many power-users prefer to perform most operations with the keyboard rather than the mouse. Visually-impaired users can navigate software more effectively using the keyboard, because using the mouse depends on visual feedback of the mouse pointer location. And mobility impairments can prevent a user from successfully navigating using the mouse, because of the fine motor control skills required.

Make all mouse actions available from the keyboard, and include keyboard access to all toolbars, menus, links and buttons. Every function your application provides must be available using the keyboard alone. Hiding your mouse while you test your application is a great way to test this!

Figure 10.4. Dialog and menu, with some of their access and shortcut keys indicated

Most functionality is easy to make available from the keyboard, by using access keys and shortcut keys, and the toolkit's built-in keyboard navigation features. All controls with labels should have access keys, and frequently-used menu items should be assigned shortcut keys. However, operations that rely on drag-and-drop, for example, may require more thought to make them keyboard accessible.

Guidelines

- Provide efficient keyboard access to all application features. In particular, ensure every control on menus and in dialogs are directly focusable using access keys or shortcut keys.
- Use a logical keyboard navigation order. When navigating around a window with the Tab key, keyboard focus should move between controls in a predictable order. In Western locales, this is normally left to right and top to bottom.
- Ensure correct tab order for controls whose enabled state is dependent on check box, radio button or toggle button state. When such a button is selected, all its dependent controls should be enabled, and all the dependent controls of any other button in the group should be disabled. When the user selects a check box, radio button or toggle button that has dependent controls, do not automatically give focus to the first dependent control, but instead leave the focus on the button.
- Do not over-ride existing system-level accessibility features. For example, the MouseKeys feature in the GNOME Keyboard Accessibility preferences dialog allows mouse movement and button clicks to be simulated using the keypad. Therefore you cannot add features to your application that can only be accessed by pressing keys on the keypad, as users relying on the MouseKeys feature will not be able to use them.
- Ensure that any text that can be selected with the mouse can also be selected with the keyboard. This is a convenience for all users, but especially for those for whom fine control of the mouse is difficult.
- Ensure that objects that can be resized or moved by drag and drop can also be resized or moved with the keyboard. For example, icons and windows on the desktop. Where precision sizing and placement is potentially important, e.g. shapes in a diagram, also consider providing a dialog into which you can type co-ordinates, or a means of snapping objects to a user-definable grid.

- Do not use general navigation functions to trigger operations. For example, do not use basic **Tab** keyboard navigation in a dialog to activate any actions associated with a control.
- Show keyboard-invoked menus, windows and tooltips near the object they relate to, but without hiding or obscuring the object to which the menu or tooltip refers,. In GNOME, popup menus are activated with **Shift-F10**, and tooltips with **Ctrl-F1**.
- Provide more than one method to perform keyboard tasks where possible. Users may find some keys and key combinations easier to use than others.
- Do not assign awkward reaches to frequently performed keyboard operations. Some people may only be able to use one hand on the keyboard, so shortcuts that can be easily used with one hand are preferable for common operations. In any case, having to frequently perform long or difficult reaches on the keyboard can increase muscle strain for all users, increasing the risk of pain or injury.
- Do not require repetitive use of simultaneous keypresses. Some users are only able to press and hold one key at a time. Assistive technologies such as the GNOME Keyboard Accessibility preferences dialog do allow users to press the keys sequentially rather than simultaneously, but this of course means the operation will take longer for them.

2.2. Choosing Access Keys

Give all labeled components an access key (underlined letter), with the exception of toolbar controls which would use up too many access key combinations.

Choose access keys to be as easy to remember as possible. Normally, this means using the first letter of the label. However, in complex windows, the choice can become more difficult. Here are some simple rules:

1. Assign access keys to the most frequently-used controls first. If it's not clear which controls will be the most frequently used, assign access keys from left to right, top to bottom (for Western locales).
2. Use the first letter of the label, or of one of its other words if it has more than one. If another letter provides a better association (e.g. "x" in Extra Large) however, consider using that letter instead.
3. If the first letter is not available, choose an easy to remember consonant from the label, for example, "p" in Replace.
4. If no such consonants are available, choose any available vowel from the label.

If duplication of access keys in a window is unavoidable, you should still refrain from duplicating the access keys for any of these buttons that appear in the same window: OK, Cancel, Close, Apply or Help.

Also, it is better not to assign access keys to "thin" letters (such as lowercase i or l), or letters with descenders (such as lowercase g or y) unless it is unavoidable. The underline does not show up very well on those characters in some fonts.

Applications using a non-Roman writing system in conjunction with a standard keyboard can have control labels prefixed with Roman characters as access keys.

2.3. Choosing Shortcut Keys

The tables in Section 2.4, “Standard Application Shortcut Keys” summarize the standard shortcut keys to use when your application supports those functions. Your application will not necessarily support all of these functions, see Section 4, “Standard Menus” for more information. However, use the recommended shortcut keys for those functions you do support.

You will probably want to add your own shortcut keys for functions specific to your application. If so, as well as following the guidelines below, look at any other existing similar applications to see which shortcut keys they have defined. Your users may already be using those or similar applications, so being consistent where it is possible and sensible to do so will provide a better user experience for them when they begin to use yours.

Guidelines

- Use **Ctrl**-letter in preference to other combinations when choosing new shortcut keys.
- **Insert**, **Delete**, **Home**, **End**, **Page Up** and **Page Down** are acceptable shortcut keys for functions that are closely related to those keys' normal system-defined uses. Do not assign them to unrelated functions just because you've run out of other shortcut key combinations, however.
- Only assign shortcut keys to the most commonly-used actions in your application. Do not try to assign a shortcut key to everything.
- Choose new shortcut keys to be as mnemonic as possible, as these will be easier to learn and remember. For example, **Ctrl-E** would be a good shortcut for a menu item called Edit Page.
- Use **Shift-Ctrl-letter** for functions that reverse or extend another function. For example, **Ctrl-Z** and **Shift-Ctrl-Z** for Undo and Redo.

Unicode entry shortcuts

Note that you cannot use **Shift-Ctrl-A-thru-F** or **Shift-Ctrl-0-thru-9** for your own purposes, as these combinations are used to enter unicode characters in text fields.

- Do not use **Ctrl-number** or numbered function keys as shortcut keys, unless the number has some obvious relevance to the action. For example, **Ctrl-2** and **Ctrl-3** may be acceptable shortcut keys for View → 2D View and View → 3D View in a 3D modeling application.
- Do not use **Alt-key** combinations for shortcut keys, as these may conflict with window manager or menu access keys.
- Do not use symbols that require **Shift** or other modifiers as part of a shortcut, for example **Ctrl-%**. Remember that symbols that can be accessed without a modifier key on your keyboard may be more difficult to access on different international keyboards.
- Do not assign shortcut keys to menu items that change over time, for example a list of open windows on the Window menu, or a recently-used file list on the File menu. Do assign access keys to these items, however.
- Do not use any of the standard shortcut keys listed in Section 2.4, “Standard Application Shortcut Keys” for your own purposes, even if your application doesn't support those functions. This helps reinforce consistency between all GNOME applications.

2.4. Standard Application Shortcut Keys

If your application uses any of the standard functions listed in the following tables, use the recommended standard keyboard shortcut for that function.

Table 10.5. Standard GNOME application shortcut keys and access keys - File menu

Function	Shortcut	Description
New	**Ctrl-N**	Create a new document
Open	**Ctrl-O**	Open a document
Save	**Ctrl-S**	Save the current document
Print	**Ctrl-P**	Print the current document
Close	**Ctrl-W**	Close the current document
Quit	**Ctrl-Q**	Quit the application

Table 10.6. Standard GNOME application shortcut keys and access keys - Edit menu

Function	Shortcut	Description
Undo	**Ctrl-Z**	Undo the last operation
Redo	**Shift-Ctrl-Z**	Redo the last operation
Cut	**Ctrl-X**	Cut the selected area and store it in the clipboard
Copy	**Ctrl-C**	Copy the selected area into the clipboard
Paste	**Ctrl-V**	Paste contents of clipboard at mouse/cursor position
Duplicate	**Ctrl-U**	Duplicate the currently-selected items and add them to the same window, without affecting the clipboard
Select All	**Ctrl-A**	Select everything in focused control or window
Invert Selection	**Ctrl-I**	Select everything in focused control or window that was previously unselected, and deselect everything that was previously selected
Delete	**Del**	Delete selection
Find...	**Ctrl-F**	Find matches in the current document, highlighting them in-place
Search...	**Ctrl-F** (see note below)	Search for matches in multiple documents, files or other external sources
Find Next	**Ctrl-G**	Find the next match

Function	Shortcut	Description
Replace...	**Ctrl-H**	Find and replace matches
Rename	**F2**	Switch the selected item's label into edit mode, allowing user to type in a new name.

Find and Search

If your application requires both Edit → Find and Edit → Search menu items, use **Shift-Ctrl-F** as the shortcut for Search.

Table 10.7. Standard GNOME application shortcut keys and access keys - View menu

Function	Shortcut	Description
Zoom In	**Ctrl**-Plus	Zoom in on the document
Zoom Out	**Ctrl**-Minus	Zoom out of the document
Normal Size	**Ctrl**-0	Restore to zoom level to normal size (generally 100%)
Refresh	**Ctrl-R**	Redraw current view of document, without checking if content has changed
Reload	**Ctrl-R** (see note below)	Reload the current document, updating content from source if necessary
Properties	**Alt-Enter**	Display the selected object's Properties window. May alternatively appear on the File menu if the document itself is the only object in the application whose properties can be inspected.

Reload and Refresh

If your application requires both View → Reload and View → Refresh menu items, use **Shift-Ctrl-R** as the shortcut for Reload.

Table 10.8. Standard GNOME application shortcut keys and access keys - Bookmarks menu

Function	Shortcut	Description
Add Bookmark	**Ctrl-D**	Add a bookmark for the current location
Edit Bookmarks...	**Ctrl-B** (see note below)	Open a window in which the user can edit and organise saved bookmarks

Bold and Edit Bookmarks

If your application requires both Format → Bold and Bookmarks → Edit Bookmarks... menu items, use **Shift-Ctrl-D** as the shortcut for Edit Bookmarks.

Table 10.9. Standard GNOME application shortcut keys and access keys - Go menu

Function	Shortcut	Description
Back	**Alt**-Left	Go to the previous location in the navigation chain
Next	**Alt**-Right	Go to the next location in the navigation chain
Up	**Alt**-Up	Go up one level in the navigation hierarchy
Home	**Alt**-Home	Go to the starting page defined by the user or application
Location...	**Ctrl**-L	Present or focus an entry field into which the user can type a new address or location to view

Table 10.10. Standard GNOME application shortcut keys and access keys - Format menu

Function	Shortcut	Description
Bold	**Ctrl-B**	Make selected text bold/regular
Underline	**Ctrl-U**	Underline/remove underline from selected text
Italic	**Ctrl-I**	Make selected text italic/ regular

Table 10.11. Standard GNOME application shortcut keys and access keys - Help menu

Function	Shortcut	Description
Contents	**F1**	Show help contents page for the current application

2.4.1. Standard Window Manager Shortcut Keys

The following shortcut keys are used by many window managers, and should not normally be over-ridden by your application.

Table 10.12. Standard window manager shortcut keys and access keys

Function	Shortcut	Description
Switch primary windows	**Alt-Tab**, **Shift-Alt-Tab**	Switch focus to the next or previous top level window on the desktop
Switch panels	**Ctrl-Alt-Tab**, **Shift-Ctrl-Alt-Tab**	Switch focus to the next or previous panel on the desktop
Log out	**Ctrl-Alt-Del**	Open the session logout confirmation dialog
Window menu	**Alt-Space**	Open the window menu
Close	**Alt-F4**	Close the focused window
Restore	**Alt-F5**	Restore the focused to its previous size
Switch secondary windows	**Alt-F6**, **Shift-Alt-F6**	Switch focus to the next or previous secondary window associated with the application ()
Move	**Alt-F7**	Move the focused window
Resize	**Alt-F8**	Resize the focused window
Minimize	**Alt-F9**	Minimze the focused window
Maximize	**Alt-F10**	Maximize the focused window
Full Screen	**Ctrl-F11**	Show the window in full screen mode, with no border, menubar, toolbar or statusbar

2.4.2. Standard Widget Navigation Shortcut Keys

The following shortcut keys are reserved for keyboard navigation use by the various widgets used in GNOME, and should not normally be over-ridden by your application.

Table 10.13. Standard GNOME keyboard navigation keys for widgets

Key	Function
Tab, **Shift-Tab**	Moves keyboard focus to next/previous control
Ctrl-Tab, **Shift-Ctrl-Tab**	Moves keyboard focus out of enclosing widget to next/previous control, in those situations where Tab alone has another function (e.g. GtkTextView)
Ctrl-F1	Pop up tooltip for currently-focused control
Shift-F1	Show context-sensitive help for currently-focused window or control
F6, **Shift-F6**	Give focus to next/previous pane in a GtkPaned window
F8	Give focus to splitter bar in paned window
F10	Give focus to window's menu bar
Shift-F10	Pop up contextual menu for currently-selected objects
Space	Toggle selected state of focused check box, radio button, or toggle button

Key	Function
Return	Activate focused button, menu item etc.
Home, **End**	Select/move to first item in selected widget
PageUp, **PageDown**	Scroll selected view by one page up/down
Ctrl-PageUp, **Ctrl-PageDown**	Scroll selected view by one page left/right

2.4.3. Additional Widget Navigation Shortcut Keys

The following emacs-style navigation shortcut keys are still available in GNOME 2.0 text entry fields (by selecting the "emacs" scheme in the GNOME Keyboard Shortcuts preferences dialog), but are disabled by default. Since some users will still want to use them, do not over-ride them for your own purposes in any situations where a text entry control has focus.

Table 10.14. Emacs-style navigation keys for widgets

Key	Function
Ctrl-A	Move cursor to beginning of line
Ctrl-D	Delete character following/under cursor
Ctrl-E	Move cursor to end of line
Ctrl-K	Delete from cursor to end of line
Ctrl-U	Delete current line
Ctrl-W	Cut to clipboard
Ctrl-Y	Paste from clipboard
Ctrl-Space	Set mark
Ctrl-Del, **Alt-D**	Delete from cursor to end of word
Ctrl-Backspace	Delete from cursor to start of word
Alt-Space	Delete all whitespace around cursor, reinsert single space
Alt-	Delete all whitespace around cursor

2.5. Keyboard Interaction with Panel Applications (Applets)

Panels have been fully keyboard navigable since GNOME 2.0. Since your panel application can gain keyboard focus, you must ensure that it is also keyboard navigable.

The rules for panel application keyboard navigation are mostly the same as those for any other window. However, there is one important difference:

- Do *not* use the the **Tab** key as the means of moving focus between controls in a panel application. Use the arrow keys for this purpose instead.

When an object on a panel has focus, the **Tab** key normally moves focus to the next object on the panel. If your panel application also used **Tab** for its own internal navigation, the user would have to press **Ctrl-Tab** to move focus out of your panel application instead. This inconsistency would be detrimental to the user experience.

Chapter 11. Language

Consistent labeling creates a familiar environment that the user can navigate comfortably. The more familiar the environment, the easier task of finding information.

1. Labels

1.1. Controls

Clear, consistent and concise labeling of controls helps users to work out the purpose of a window or dialog they have never seen before. To a visually-impaired user, clear labels are even more important. A user who relies on a screenreader has no assistance from icons, layout, or spacing to work out what the controls do, so clear labeling is essential.

Guidelines

- Keep labels short. This:
  - Reduces the expansion of text when translated, and thus minimizes the effort required to localize the UI. Translated English text can expand up to 30% in some languages.
  - Facilitates the use of translation engines.
  - Improves speed of comprehension for the user.

  Do not shorten your labels to the point of losing meaning, however. A three-word label that provides clear information is better than a one-word label that is ambiguous or vague. Try to find the fewest possible words to satisfactorily convey the meaning of your label.
- Do not include text in windows that describes *how* to use the interface, for example You can install a new theme by dropping it here. As well as adding visual clutter, descriptive labels can also conflict with information provided in documentation.
- Use standard terms. You can find a list of standard user interface terms in the GNOME Documentation Style Guide, Recommended Terminology [http://developer.gnome.org/documents/style-guide/wordlist.html].
- Apply standard capitalization rules. See Section 3.2, “Capitalization” for guidelines about capitalization of user interface labels

1.2. Tooltips

1.2.1. Toolbar Tooltips

A toolbar tooltip is the short description of a toolbar control's functionality that the user sees when they mouse over it.

Guidelines

- Concisely state the purpose of the control. The tooltip should be more descriptive than the

corresponding menu item name, if there is one, but not verbose. For example, Undo last action for the Undo button.

- Use sentence capitalization rules. See Section 3.2, “Capitalization”.

1.2.2. Application Tooltips

An application tooltip is the short description of your application that the user sees when they mouse over the launcher or menu item for your application. It is stored in the `comment` field of your application's `desktop` file. See Section 1.2, “Menu Item Tooltips”

Guidelines

- Create short tooltips. Aim to accurately communicate the functionality of an element with the fewest words possible.
- Use sentence capitalization rules. See Section 3.2, “Capitalization”.
- Use standard punctuation rules, with the exception that you do not use a period to end the tooltip.

1.3. Menus

Guidelines

- Use the recommended standard labels for menu items and titles, where they exist. Do not use synonyms such as Exit instead of Quit. See Section 4, “Standard Menus” for a list and descriptions of standard menu items and titles.
- Use header capitalization rules for all menu items and titles. See Section 3.2, “Capitalization” for more information.

2. Warning and Error Messages

A good warning or error message contains two elements:

1. A brief description of the problem.
2. A list of ways the user can remedy the problem.

Both of these elements should be presented in non-technical, jargon-free language, unless your target audience is particularly technically-minded.

If your application knows enough about the problem to be able to give all this information to the user, it will often be capable of rectifying the problem itself when the user has decided which course of action they want to take. For example, if the problem is insufficient memory, tell the user which currently-running application is taking up the most memory, and provide a button to close it for them. (Do not offer to launch a graphical process manager, however, which is

something most users should never see!)

See Section 4, “Alerts” for more detailed information on writing and presenting errors, warnings and information alerts.

3. On-line Help

Writing on-line help is a specialized task, and is therefore not covered in any depth here. Refer to the GNOME Documentation Styleguide [http://developer.gnome.org/documents/style-guide/index.html] for guidance on writing clear, consistent and helpful documentation for your application.

Chapter 12. Checklists

1. Things You Can Do Yourself

1.1. Before You Start

Write down the type of people you expect to use your application. Then write some "scenarios" for each type of user— a little story that describes the typical tasks those users will use your application for. These tasks should be along the lines of:

> Fred needs to find an email about widgets that he received last week

rather than

> Fred clicks on the Find button and types **`widgets`** into the dialog.

This way, you can use the same scenarios to test and compare different interface designs, and to spot any missing functionality.

Include these user descriptions and scenarios with the documentation you commit to CVS. This way, other contributors will get to understand your users too, can help to develop the application with that knowledge, and can provide more scenarios of their own.

1.2. Keyboard Access and Focus

When you have started implementing your interface, hide your mouse, and make sure you can still use it to do everything using only the keyboard. Implement keyboard functionality at the same time as mouse functionality— don't leave it until the end.

Using only keyboard commands, move the focus through all menu bars and toolbars in the application. Also confirm that:

- Context sensitive menus display correctly (**Shift**-**F10**).
- Tooltips can be popped up and down for all controls that have them (**Ctrl**-**F1**, **Esc**).
- All functions listed on the toolbar can be performed using the keyboard.
- You can fully operate every control in the client area of the application and dialogs.
- Text and objects within the client area can be selected.
- Any keyboard enhancements or shortcut keys are working as designed.
- Verify that when moving among objects, the visual focus indicator is easy to identify at all times.

1.3. Theming, Colors and Contrast

Test various GNOME themes to ensure that your application respects all the available settings.

Test your application with black and white, high contrast themes and confirm that all information is still conveyed correctly. If you don't have a suitable high contrast GNOME theme available to test, print off some screenshots in black and white (not gray-scale) and make sure all the important information is still visible— this will approximate what a high contrast theme user will see.

1.4. Animation

Ensure you have implemented an option to turn off any animation in your application (for accessibility reasons), and that it is working as designed. Turn the animation off. Confirm that all information is still conveyed correctly.

2. Things You Can Do With Other People

2.1. Get Early Feedback

It's always tempting, but don't start coding your interface straight away. Sketch out some ideas on paper first, or in Glade or HTML if you prefer. (But don't be tempted add any functionality at this point if you do it this way!)

Show these prototypes to other people— the GNOME mailing lists and IRC are ideal for finding likely candidates. Ask them to use these prototype interfaces to run through some of the scenarios you came up with earlier. You'll probably get questions like "how would I do X", "which menu is Y on"... these questions will help you think about the interface from the user's viewpoint. You'll probably also get a few suggestions about how to do things differently— these ideas may or may not turn out to better than yours, but any idea from a potential user is worth considering!

You should also consider seeking opinions from the GNOME Usability team [http://developer.gnome.org/projects/gup/]. They have designed many user interfaces before and may be able to spot potential problems at this early stage, before you take your design too far to change easily.

Once you've decided on the basic interface design and have started coding parts of it, find somebody to try it out again— it doesn't have to be the same person. You'll probably find some more problems that were hard to see on your static paper prototype. By finding these now, it's usually not too late to fix them without too much trouble.

2.2. Internationalization and Localization

If you intend your application to be translated into different languages, show draft designs of your application to the GNOME Translation Team [http://developer.gnome.org/projects/gtp/contact.html]. They'll help you find potential translation problems, such as not leaving enough space for translated labels, shortcut keys that cause problems on a different keyboard layout, or using new terms in your app that are hard to translate.

If possible, try out your application with users from the locales you are targeting. This will help you determine whether users understand how to use the application, if they perceive the graphics and colors the way you intended, and if there are words or images in the application that may cause offense to users of that locale.

Chapter 13. Credit

(lists in alphabetical order, if you were accidentally omitted please mail <hig@gnome.org>)

1. Active Authors

- Calum Benson, <calum.benson@sun.com>
- Bryan Clark, <bclark@redhat.com>
- Seth Nickell, <snickell@redhat.com>

2. Retired/Inactive Authors

- Coleen Baik
- Adam Elman
- Colin Z. Robertson
- Maciej Stachowiak

3. Reviewers and Contributors

- Chip Alexander
- Kathy Fernandes
- John Fleck
- Andrea Mankoski
- Nils Pederson
- Sebastian Rittau
- Christian Rose
- Sharon Snider
- Suzanna Smith
- Matthew Thomas

Bibliography

This bibliography lists books and other resources for software engineers, user interface designers, and human factors specialists, arranged by topic and without a particular ordering. The final section of the bibliography contains information about useful online resources and organizations.

General Design

[Dreyfus1967] Dreyfuss, Henry *Designing for People* New York, NY: allworth press, 2003 A reprint of the 1960s design classic by the designer of everything from the modern airplane cabin to the Bell telephone. Perhaps the best book for introducing the general concerns and thought patterns of design (industrial, product, or interaction), as well as an entertaining read.

[Mandel1997] Mandel, Theo. *The Elements of User Interface Design*. New York, NY: Wiley Computer Publishing, 1997. A useful book covering all the basics and a wide scope of environments and new developments like interface agents, wizards, voice interaction, social user interfaces and web design.

[Norman1990] Norman, Donald A. *The Design of Everyday Things*. New York, NY: Doubleday, 1990. An exceptional and entertaining book about the design behind simple daily things.

Graphical Design

[Horton1994] Horton, William *The Icon Book: Visual Symbols for Computer Systems and Documentation*. New York, NY: John Wiley & Sons, 1994.

[Misjksenaar1999] Misjksenaar, Paul and Piet Westendorpp. *Open Here: The Art of Instructional Design*. London, UK: Thames & Hudson, 1999

[Mullet1995] Mullet, Kevin and Darrell Sano. *Designing Visual Interfaces: Communication Oriented Techniques*. Englewood Cliffs, NJ: Prentice Hall, 1995.

[Rubin1994] Rubin, Jeffrey. *Handbook of Usability Testing: How to Plan, Design and Conduct Effective Tests*. New York, NY: John Wiley & Sons, 1994.

[Tufte1990] Tufte, Edward R. *Envisioning Information*. Cheshire, CT: Graphics Press, 1990 The second classic work on envisioning information by Tufte.

[Tufte1992] Tufte, Edward R. *The Visual Display of Quantitative Information*. Reprint ed., Cheshire, CT: Graphics Press, 1992 The first classic work on envisioning information by Tufte.

[Williams1994] Williams, Robin. *The Non-Designer's Design Book: Design and Typographic Principles for the Visual Novice*. Berkeley, CA: Peachpit Press, 1994.

Usability

[Arlov1997] Arlov, Laura. *GUI Design for Dummies*. Foster City, CA: IDG Books Worldwide, 1997.

[Cooper2003] Cooper, Alan and Reimann, Robert. *About Face 2.0 : The Essentials of User Interface Design*. John Wiley & Sons, 2003.

[Cooper1999] Cooper, Alan. *The Inmates are Running the Asylum : Why High Tech Products Drive us Crazy and How to Restore the Sanity*. SAMS, 1999.

[Isaacs2001] Isaacs, Ellen and Alan Walendowski. *Designing from Both Sides of the Screen*. Indianapolis, IN: New Riders Publishing, 2001.

[Nielsen1993] Nielsen, Jakob. *Usability Engineering*. San Francisco, CA: Morgan Kauffman, 1993.

[Tog1992] Tognazzini, Bruce. *Tog on Interface*. Reading, MA: Addison-Wesley, 1992.