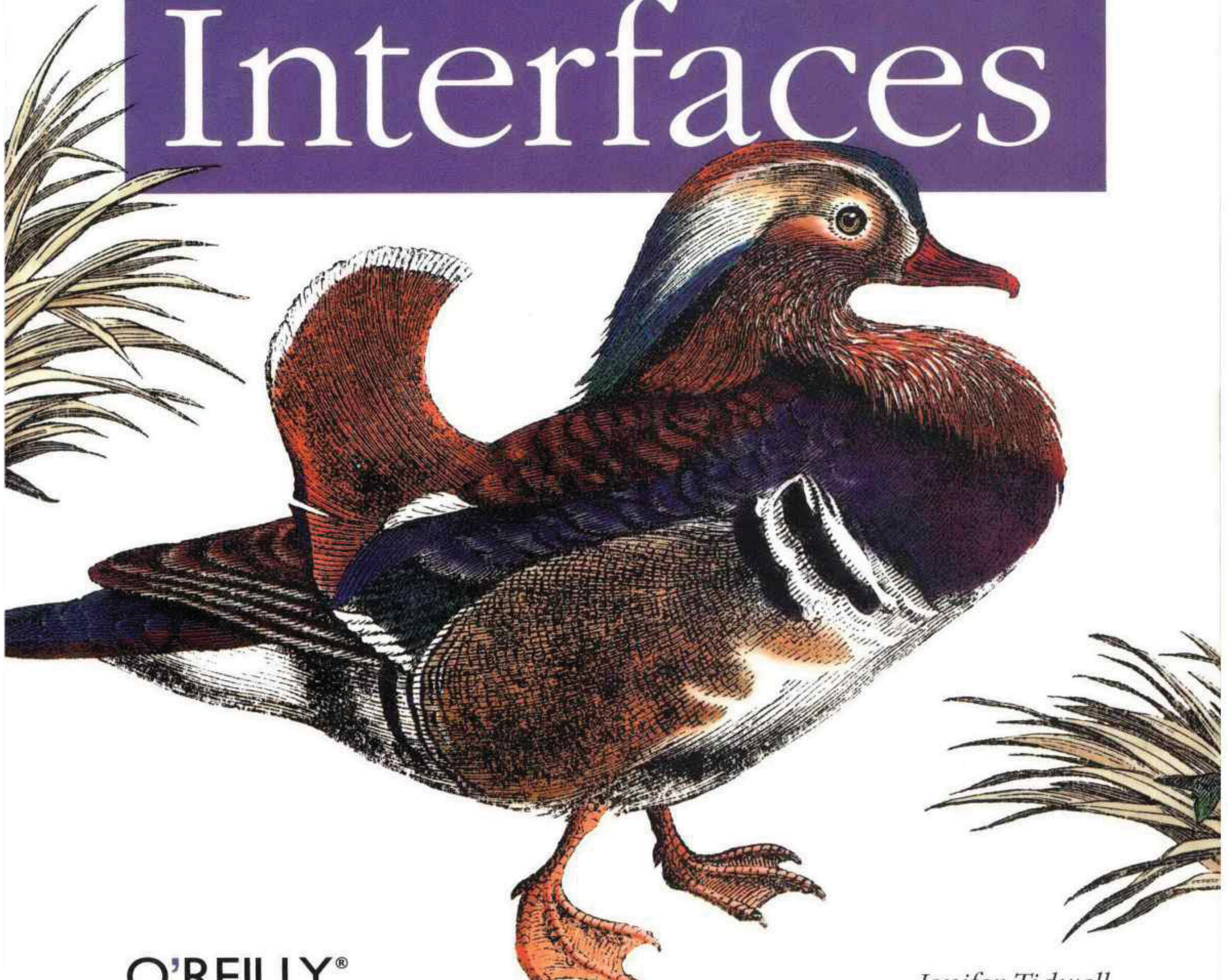
Patterns for Effective Interaction Design

Designing Interfaces



O'REILLY®

Jenifer Tidwell

Designing Interfaces

Jenifer Tidwell

O'REILLY®

BEIJING • CAMBRIDGE • FARNHAM • KÖLN • SEBASTOPOL • TAIPEI • TOKYO

Designing Interfaces
by Jenifer Tidwell

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*safari.oreilly.com*). For more information, contact our corporate/institutional sales department: (800) 998-9938 or *corporate@oreilly.com*.

Editors: Andrew Odewahn and Mary O'Brien
Production Editor: Genevieve d'Entremont
Cover Designer: Mike Kohnke
Interior Designer: NOON

Printing History:
November 2005: First Edition.

This book uses RepKover, a durable and flexible lay-flat binding.

ISBN: 978-0-596-00803-1

[C] [9/08]

ABOUT THE AUTHOR

Jenifer Tidwell is an interaction designer and software developer for The MathWorks, makers of technical computing software. She specializes in the design and construction of data analysis and visualization tools, and has been working on new designs for the data tools in MATLAB, which is used by researchers, students, and engineers worldwide to develop cars, planes, proteins, and theories about the universe. She has been known to design web sites, and was an early enthusiast for rich Internet application (RIA) technology, having helped design and develop Curl in the early 2000s.

Jenifer received her technical education at MIT and her design education at the Massachusetts College of Art, but she's not finished learning yet. She has been researching user interface patterns since 1997. Photography and writing are her creative outlets, and she spends as much time as she can in the New England outdoors—on a bike, on a boat, on foot, on skis, and on belay.

Jenifer's personal web site can be found at *http://jtidwell.net*.

COLOPHON

Our look is the result of reader comments, our own experimentation, and feedback from distribution channels. Distinctive covers complement our distinctive approach to technical topics, breathing personality and life into potentially dry subjects.

The animal on the cover of this book is a Mandarin duck (***Aix galericulata***), one of the most beautiful of the duck species. Originating in China, these colorful birds can be found in southeast Russia, northern China, Japan, southern England, and Siberia.

The males have diverse and colorful plumage, characterized by an iridescent crown, chestnut-colored cheeks, and a white eye stripe that extends from their red bills to the back of their heads. Females are less flamboyant in appearance and tend to be gray, white, brown, and greenish brown, with a white throat and foreneck.

These birds live in woodland areas near streams and lakes. Being omnivorous, they tend to have a seasonal diet, eating acorns and grains in autumn; insects, land snails, and aquatic plants in spring; and dew worms, grasshoppers, frogs, fish, and mollusks during the summer months.

The mating ritual of Mandarin ducks begins with an elaborate and complex courtship dance that involves shaking movements, mimed drinking gestures, and preening. Males fight each other to win a female, but it is ultimately the female who decides her mate. Mandarin ducklings instinctively follow their notoriously protective mothers, who will feign injury to distract predators such as otters, raccoon dogs, mink, polecats, eagle owls, and grass snakes.

Mandarin ducks are not an endangered species, but they are considered to be threatened. Loggers continuously encroach upon their habitats, and hunters and poachers prize the males for their plumage. Their meat is considered unpalatable by humans, and they are generally not hunted for food.

Genevieve d'Entremont was the production editor and proofreader for ***Designing Interface***s. Ann Schirmer was the copyeditor. Susan Honeywell was the page compositor. Phil Dangler and Claire Cloutier provided quality control. Kelly Talbot and Johnna VanHoose Dinse wrote the index.

Mike Kohnke designed the cover of this book, based on a series design by Edie Freedman. The cover image is from *Johnson's Natural History*. Karen Montgomery produced the cover layout in Adobe InDesign CS, using Adobe's ITC Garmond font.

NOON (*www.designatnoon.com*) designed the interior layout. This book was converted by Joe Wizda to Adobe InDesign CS. The text fonts are Gotham Book and Adobe Garamond; the heading fonts are Univers and Gotham Bold. The illustrations that appear in the book were produced by Robert Romano, Jessamyn Read, and Lesley Borash using Macromedia FreeHand MX and Adobe Photoshop CS. This colophon was written by Jansen Fernald.

CONTENTS

PREFACE

Once upon a time, interface designers worked with a woefully small toolbox.

We had a handful of simple controls: text fields, buttons, menus, tiny icons, and modal dialogs. We carefully put them together according to the Windows Style Guide or the Macintosh Human Interface Guidelines, and we hoped that users would understand the resulting interface—and too often, they didn't. We designed for small screens, few colors, slow CPUs, and slow networks (if the user was connected at all). We made them gray.

Things have changed. If you design interfaces today, you work with a much bigger palette of components and ideas. You have a choice of many more user interface toolkits than before, such as Java™ Swing, Qt, HTML and Javascript, Flash, and numerous open-source options. Apple's and Microsoft's native UI toolkits are richer and nicer-looking than they used to be. Display technology is better. Web applications often look as professionally designed as the web sites they're embedded in, and while desktop-UI ideas like drag-and-drop are integrated into web applications slowly, some of those web sensibilities are migrating back into desktop applications in the form of blue underlined links, Back/Next buttons, daring fonts and background images, and nice, non-gray color schemes.

But it's still not easy to design *good* interfaces. Let's say you're not a trained or self-taught interface designer. If you just use the UI toolkits the way they should be used, and if you follow the various style guides or imitate existing applications, you can probably create a mediocre but passable interface.

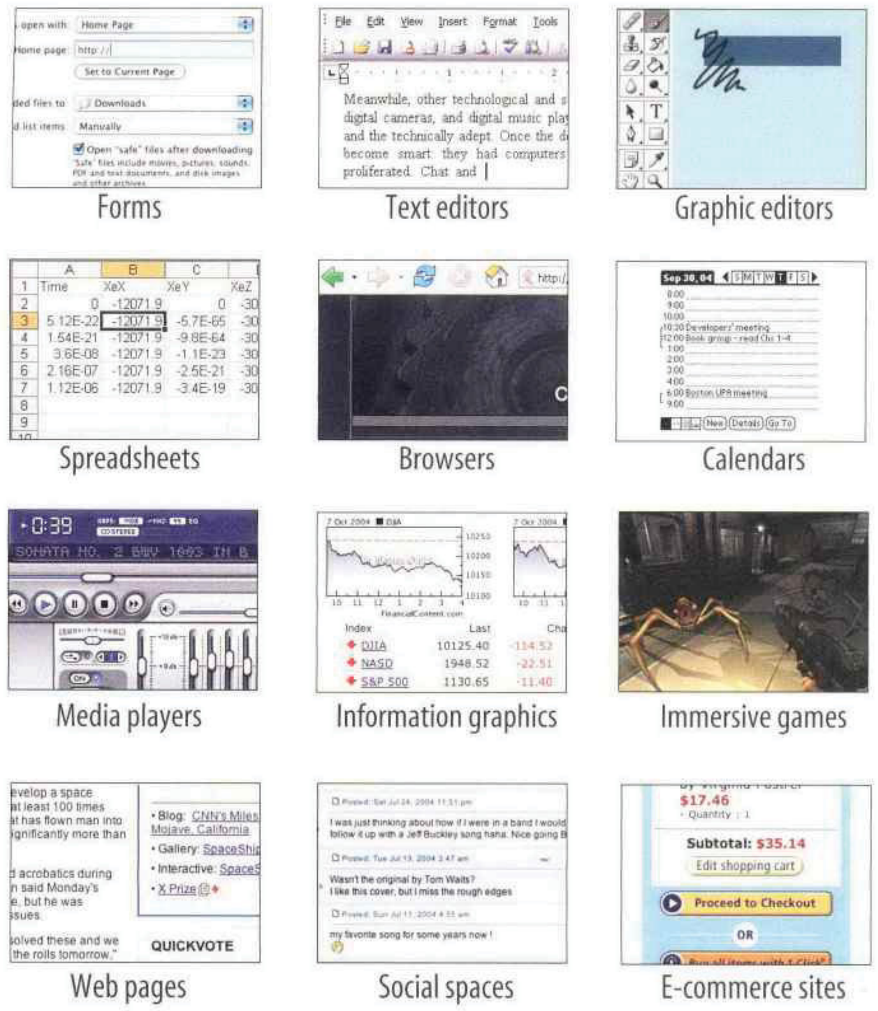
Alas, that may not be enough anymore. Users' expectations are higher than they used to be—if your interface isn't easy to use "out of the box," users will not think well of it. Even if the interface obeys all the standards, you may have misunderstood users' preferred workflow, used the wrong vocabulary, or made it too hard to figure out what the software even does. Impatient users often won't give you the benefit of the doubt. Worse, if you've built an unusable web site or web application, frustrated users can give up and switch to your competitor with just the click of a button. So the cost of building a mediocre interface is higher than it used to be, too.

It's even tougher if you design products outside of the desktop and web worlds, because there's very little good design advice out there. Palmtops, cell phones, car navigation systems, digital TV recorders—designers are still figuring out what works and what doesn't, often from basic principles. (And their users often tolerate difficult interfaces—but that won't last long.)

Devices like phones, TVs, and car dashboards once were the exclusive domain of industrial designers. But now those devices have become smart. Increasingly powerful computers drive them, and software-based features and applications are multiplying in response to market demands. They're here to stay, whether or not they are easy to use. At this rate, good interface and interaction design may be the only hope for our collective sanity in 10 years.

SMALL INTERFACE PIECES, LOOSELY JOINED

As an interface designer trying to make sense of all the technology changes in the last few years, I see two big effects on the craft of interface design. One is the proliferation of *interface idioms*: recognizable types or styles of interfaces, each with its own vocabulary of objects, actions, and visuals. You probably recognize all the ones shown in the figure on the next page, and more are being invented all the time.



A sampler of interface idioms

The second effect is a loosening of the rules for putting together interfaces from these idioms. It no longer surprises anyone to see several of these idioms mixed up in one interface, for instance, or to see parts of some controls mixed up with parts of other controls. Online help pages, which have long been formatted in hypertext anyway, might now have interactive applets in them, animations, or links to a web-based bulletin board. Interfaces themselves might have help texts on them, interleaved with forms or editors; this used to be rare. Combo boxes' dropdown menus might have funky layouts, like color grids or sliders, instead of the standard column of text items. You might see web applications that look like document-centered paint programs, but have no menu bars, and save the finished work only to a database somewhere.

The freeform-ness of web pages seems to have taught users to relax their expectations with respect to graphics and interactivity. It's okay now to break the old Windows style-guide strictures, as long as users can figure out what you're doing.

And that's the hard part. Some applications, devices, and web applications are easy to use. Many aren't. Following style guides never guaranteed usability anyhow, but now designers have even more choices than before (which, paradoxically, can make design a *lot* harder). What characterizes interfaces that are easy to use?

One could say, "The applications that are easy to use are designed to be intuitive." Well, yes. That's almost a tautology.

Except that the word "intuitive" is a little bit deceptive. Jef Raskin once pointed out that when we say "intuitive" in the context of software, we really mean "familiar." Computer mice aren't intuitive to someone who's never seen one (though a growling grizzly bear would be). There's nothing innate or instinctive in the human brain to account for it. But once you've taken 10 seconds to learn to use a mouse, it's familiar, and you'll never forget it. Same for blue underlined text, play/pause buttons, and so on.

Rephrased: "The applications that are easy to use are designed to be *familiar*."

Now we're getting somewhere. "Familiar" doesn't necessarily mean that everything about a given application is identical to some genre-defining product (e.g., Word, Photoshop, Mac OS, or a Walkman). People are smarter than that. As long as the parts are recognizable enough, and the relationships among the parts are clear, then people can apply their previous knowledge to a novel interface and figure it out.

That's where patterns come in. This book catalogs many of those familiar parts, in ways you can reuse in many different contexts. Patterns capture a common structure—usually a very "local" one, like funky layouts on a combo box—without being too concrete on the details, which gives you flexibility to be creative.

If you know what users expect of your application, and if you choose carefully from your toolbox of idioms (large-scale), controls (small-scale), and patterns (covering the range), then you can put together something which "feels familiar" while remaining original.

And that gets you the best of both worlds.

ABOUT PATTERNS IN GENERAL

In essence, patterns are structural and behavioral features that improve the "habitability" of something—a user interface, a web site, an object-oriented program, or even a building. They make things easier to understand or more beautiful; they make tools more useful and usable.

As such, patterns can be a description of best practices within a given design domain. They capture common solutions to design tensions (usually called "forces" in pattern literature) and thus, by definition, are not novel. They aren't off-the-shelf components; each implementation of a pattern differs a little from every other. They aren't simple rules or heuristics either. And they won't walk you through an entire set of design decisions—if you're looking for a complete step-by-step description of how to design an interface, a pattern catalog isn't the place!

This book describes patterns literally as solutions to design problems because part of their value lies in the way they resolve tensions in various design contexts. For instance, an interface designer who needs to pack a lot of stuff into a too-small space can use a Card Stack. What remains for the designer is information architecture—how to split up the content into pieces, what to name them, etc.—and what exactly the Card Stack will look like when it's done. Tabs? A lefthand-side list or tree? That's up to the designer's judgment.

Some very complete sets of patterns make up a "pattern language." These patterns resemble visual languages, in that they cover the entire vocabulary of elements used in a design (though pattern languages are more abstract and behavioral; visual languages talk about shapes, icons, colors, fonts, etc.). This set isn't nearly so complete, and it contains techniques that don't qualify as traditional patterns. But it's concise enough to be manageable and useful.

OTHER PATTERN COLLECTIONS

The text that started it all dealt with physical buildings, not software. Christopher Alexander's *A Pattern Language*, and its companion book, *The Timeless Way of Building* (both Oxford University Press), established the concept of patterns and described a 250-pattern multilayered pattern language. It is often considered the gold standard for a pattern language because of its completeness, its rich interconnectedness, and its grounding in the human response to our built world.

In the mid-1990s, the publication of *Design Patterns,* (Addison-Wesley) by Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides profoundly changed the practice of commercial software architecture. This book is a collection of patterns describing object-oriented "micro-architectures." If you have a background in software engineering, this is the book that probably introduced you to the idea of patterns. Many other authors have written books about software patterns since *Design Patterns*. Software patterns such as these do make software more habitable—for those who write the software, not those who use it!

The first substantial set of user-interface patterns was the predecessor of this patterns collection, "Common Ground."[1] Many other collections and languages followed, notably Martijn van Welie's "Interaction Design Patterns,"[2] and Jan Borchers's book *A Pattern Approach to Interaction Design* (Wiley). More recently, a full-fledged web site pattern language was published, called *The Design of Sites* (Addison-Wesley). I highly recommend it, especially if you're designing traditional web sites. If you're building web or desktop applications, or if you're pushing the boundaries in either domain, look at all of these publications; you might find inspiration in any of them.

ABOUT THE PATTERNS IN THIS BOOK

So there's nothing really new in here. If you've done any web or UI design, or even thought much about it, you should say, "Oh, right, I know what that is" to most of these patterns. But a few of them might be new to you, and some of the familiar ones may not be part of your usual design repertoire.

These patterns work for both desktop and web-based applications. Many patterns also apply to such digital devices as palmtops, cell phones, and TV-based devices like digital recorders. Ordinary web sites might also benefit, but I'll talk more about that topic in the next section.

Though this book won't exhaustively describe all the interface idioms mentioned earlier, they organize part of the book. Three chapters focus on the more common idioms: forms, information graphics, and WYSIWYG editors (like those used for text and graphics). Other chapters address subjects that are useful across many idioms, such as organization, navigation, actions, and visual style.

This book is intended to be read by people who have some knowledge of such interface design concepts and terminology such as dialog boxes, selection, combo boxes, navigation bars, and white space. It does not identify many widely accepted techniques, such as copy-and-paste, since you already know what they are. But, at the risk of belaboring the obvious, this book describes some common techniques to encourage their use in other contexts—for instance, many desktop applications could better use Global Navigation—or to discuss them alongside alternative solutions.

This book does *not* present a complete process for constructing an interface design. When doing design, a sound process is critical. You need to have certain elements in a design process:

- Field research, to find out what the intended users are like and what they already do
- Goal and task analysis, to describe and clarify what users will do with what you're building
- Design models, such as personas (models of users), scenarios (models of common tasks and situations), and prototypes (models of the interface itself)

1. *http://www.mit.edu/~jtidwell/common_ground.html*

2. *http://www.welie.com/patterns*

- Empirical testing of the design at various points during development, like usability testing and *in situ* observations of the design used by real users
- Enough time to iterate over several versions of the design, because you won't get it right the first time

The topic of design process transcends the scope of this book, and plenty of other books and workshops out there cover it well. Read them; they're good.

But there's a deeper reason why this book won't give you a recipe for designing an interface. Good design can't be reduced to a recipe. It's a creative process, and one that changes under you as you work—in any given project, for instance, you won't understand some design issues until you've designed your way into a dead end. I've personally done that many times.

And design isn't linear. Most chapters in this book are arranged more or less by scale, and therefore by their approximate order in the design progression: large decisions about content and scope are made first, followed by navigation, page design, and, eventually, the details of interactions with forms and canvases and such. But you'll often find yourself moving back and forth through this progression. Maybe you'll know very early in a project how a certain screen should look, and that's a "fixed point"; you may have to work backward from there to figure out the right navigational structure. (No, it's not ideal, but things like this do happen in real life.)

That said, here are some ways you can use these patterns:

Learning

If you don't have years of design experience already, a set of patterns may serve as a learning tool. You may want to read over it to get ideas, or refer back to specific patterns as the need arises. Just as expanding your vocabulary helps you express ideas in language, expanding your interface design "vocabulary" helps you create more expressive designs.

Examples

Each pattern in this book has at least one example. Some have many; they might be useful to you as a sourcebook. You may find wisdom in the examples that is missing in the text of the pattern.

Terminology

If you talk to users, engineers, or managers about interface design, or if you write specifications, then you could use the pattern names as a way of communicating and discussing ideas. This is another well-known benefit of pattern languages. (The terms "singleton" and "factory," for instance, were originally pattern names, but they're now in common usage among software engineers.)

Inspiration

Each pattern description tries to capture the reasons why the pattern works to make an interface easier or more fun. If you get it, but want to do something a little different from the examples, you can be creative with your "eyes open."

One more word of caution: a catalog of patterns is not a checklist. You cannot measure the quality of a thing by counting the patterns in it. Each design project has a unique context, and even if you need to solve a common design problem (such as how to fit too much content onto a page), a given pattern might be a poor solution within that context. No reference can substitute for good design judgment. Nor can it substitute for a good design process, which helps you find and recover from design mistakes.

Ultimately, you should be able to leave a reference like this behind. As you become an experienced designer, you will have internalized these ideas to the point at which you don't notice you use them anymore; the patterns become second nature. They're part of your toolbox from then on.

AUDIENCE

If you design user interfaces in any capacity, you might find this book useful. It's intended for people who work on:

- Desktop applications
- Web applications or "rich internet applications" (RIAs)
- Highly interactive web sites
- Software for handhelds, cell phones, or other consumer electronics
- Turnkey systems, such as kiosks
- Operating systems

The list might also include traditional web sites such as corporate home pages, but I deliberately did not focus on web sites. They are amply covered by the existing literature, and talking more about them here seems redundant. Also, most of them don't have the degree of interactivity taken for granted in many patterns; there's a qualitative difference between a "read-only" site and one that actually interacts with its users.

Of course, profound differences exist among all these design platforms. However, I believe they have more in common than we generally think. You'll see examples from many different platforms in these patterns, and that's deliberate—they often use the same patterns to achieve the same ends.

This book isn't Design 101; it's more like Design 225. As mentioned earlier, it's expected that you already know the basics of UI design, such as available toolkits and control sets, concepts like drag-and-drop and focus, and the importance of usability testing and user feedback. If you don't, some excellent books listed in the references can get you started with the essentials.

Specifically, this book targets the following audiences:

- Software developers who need to design the UIs that they build.
- Web page designers who are now asked to design web apps or sites with more interactivity.
- New interface designers and usability specialists.

- More experienced designers who want to see how other designs solve certain problems; the examples can serve as a sourcebook for ideas.
- Professionals in adjacent fields, such as technical writing, product design, and information architecture.
- Managers who want to understand what's involved in good interface design.
- Open-source developers and enthusiasts. This isn't quite "open-source design," but the idea here is to open up interface design best practices for everyone's benefit.

HOW THIS BOOK IS ORGANIZED

These patterns are grouped into thematic chapters, and each chapter has an introduction that briefly covers the concepts those patterns are built upon. I want to emphasize *briefly*. Some of these concepts could have entire books written about them. But the introductions will give you some context; if you already know this stuff, they'll be review material, and if not, they'll tell you what topics you might want to learn more about.

The first set of chapters are applicable to almost any interface you might design, whether it's a desktop application, web application, web site, hardware device, or whatever you can think of:

- Chapter 1, *What Users Do*, talks about common behavior and usage patterns supported well by good interfaces.
- Chapter 2, *Organizing the Content*, discusses information architecture as it applies to highly interactive interfaces. It deals with different organizational models, the amount of content a user sees at one time, and the best way to use windows, panels, and pages.
- Chapter 3, *Getting Around*, discusses navigation. It describes patterns for moving around an interface—between pages, among windows, and within large virtual spaces.
- Chapter 4, *Organizing the Page*, describes patterns for the layout and placement of page elements. It talks about how to communicate meaning simply by putting things in the right places.
- Chapter 5, *Doing Things*, talks about how to present actions and commands; use these patterns to handle the "verbs" of an interface.

Next comes a set of chapters that deal with specific idioms. It's fine to read them all, but real-life projects probably won't use all of them. Chapters 6 and 7 are the most broadly applicable, since most modern interfaces use trees, tables, or forms in some fashion.

- Chapter 6, *Showing Complex Data*, contains patterns for trees, tables, charts, and information graphics in general. It discusses the cognitive aspects of data presentation, and how to use them to communicate knowledge and meaning.
- Chapter 7, *Getting Input from Users*, deals with forms and controls. Along with the patterns, this chapter has a table that maps data types to various controls that can represent them.

- Chapter 8, *Builders and Editors*, discusses techniques and patterns often used in WYSIWYG graphic editors and text editors.

Finally, the last chapter comes at the end of the design progression, but it too applies to almost anything you design.

- Chapter 9, *Making It Look Good*, deals with aesthetics and fit-and-finish. It uses graphic-design principles and patterns to show how (and why) to polish the look-and-feel of an interface, once its behavior is established.

I chose this book's examples based on many factors. The most important factor is how well an example demonstrates a given pattern or concept, of course, but other considerations include general design fitness, printability, platform variety—desktop applications, web sites, devices, etc.—and how well-known and accessible these applications might be to readers. As such, the examples are weighted heavily toward Microsoft and Apple software, certain web sites, and easily-found consumer software and devices. This is not to say that they are always paragons of good design. They're not, and I do not mean to slight the excellent work done by countless designers on less well-known applications. If you know of examples that might meet these criteria, please suggest them to me.

COMMENTS AND QUESTIONS

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
(800) 998-9938 (in the United States or Canada)
(707) 829-0515 (international or local)
(707) 829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at:

http://www.oreilly.com/catalog/designinterfaces

Visit *http://designinginterfaces.com* for more information.

To comment or ask technical questions about this book, send email to:

bookquestions@oreilly.com

For more information about our books, conferences, Resource Centers, and the O'Reilly Network, see our web site at:

http://www.oreilly.com

ACKNOWLEDGMENTS

First, my deepest thanks to the technical reviewers of this book: Eric Freeman, Peter Morville, William Wake, Robert Reimann, Jeff Johnson, Martijn van Welie, and Ron Jeffries. Your suggestions unquestionably made this book better.

Other readers, reviewers, and suggestion-makers include Andrea Midtmoen Fease, Jan Stetson, Helen Rennie, Rhon Porter, Geoff Dutton, Steve Eddins, Lynn Cherny, Tom Lane, Joe Conti, Will Schroeder, Janice Kutz, Tim Wright, Ben Bederson, Robert Nero, and Michael Del Gaudio. You all offered your time and energy to help with this project—thank you. And extra thanks to my other colleagues at The MathWorks, especially Chris Wood, for all your patience while I was so busy writing this.

Thanks to the really early reviewers, Ralph Johnson and the Software Architecture Group at the University of Illinois at Urbana-Champaign: Tankut Baris Aktemur, John Bordan, John Brant, Nicholas Bray, Danny Dig, Christos Evaggelou, Alejandra Garrido, Brian Foote, Munawar Hafiz, Thuc Si Mau Ho, Pablo Montesinos, Jeff Overbey, Weerasak Witthawaskul, Spiros Xanthos, and Joseph Yoder. (I'm sure I missed some of you because your names on the recordings were so faint!)

Doug Hull, Tom Lane, Gerard Torenvliet, Alex Conn, Amy Groden-Morrison, Susan Fowler, and Robert Nero all supplied material and links that I've included as key examples—thank you all for those screenshots and suggestions, and I hope the text does them credit.

At O'Reilly, thanks to Nat Torkington, who "found" the original web site and contacted me; Mike Hendrickson, for guiding me through the early stages of the book and coming up with some terrific ideas for it; and Andrew Odewahn and Mary O'Brien, my helpful and kind editors during the later stages of writing the draft.

The thousands of anonymous visitors to my UI patterns web site are the reason why this book exists. Without your log records, I'd never have known how heavily used the site actually is. And thanks to those of you who took the time to write and tell me you liked it. I hope this book meets your expectations!

Many, many thanks to my family and friends, especially Rich. You all encouraged me to turn the web site into a book, and supported me wholeheartedly throughout the writing process.

Finally, I want to thank our local champions: the 2003 and 2004 New England Patriots and the 2004 Boston Red Sox. You all helped us believe that anything—anything!—can be possible if we try hard enough.

Even writing a book.

01

WHAT USERS DO

This book is almost entirely about the look and behavior of applications, web applications, and interactive devices. But this first chapter will be the exception to the rule. No screenshots here; no layouts, no navigation, no diagrams, and no visuals at all.

Why not? After all, that's why you may have picked up this book in the first place.

It's because good interface design doesn't start with pictures. It starts with an understanding of people: what they're like, why they use a given piece of software, and how they might interact with it. The more you know about them, and the more you empathize with them, the more effectively you can design for them. Software, after all, is merely a means to an end for the people who use it. The better you satisfy those ends, the happier those users will be.

Each time someone uses an application, or any digital product, they carry on a conversation with the machine. It may be literal, as with a command line or phone menu, or tacit, like the "conversation" an artist has with her paints and canvas—the give and take between the craftsperson and the thing being built. With social software, it may even be a conversation by proxy. Whatever the case, the user interface mediates that conversation, helping the user achieve whatever ends he or she had in mind.

As the user interface designer, then, you get to script that conversation, or at least define its terms. And if you're going to script a conversation, you should understand the human's side as well as possible. What are the user's motives and intentions? What "vocabulary" of words, icons, and gestures does the user expect to use? How can the application set expectations appropriately for the user? How do the user and the machine finally end up communicating meaning to each other?

There's a maxim in the field of interface design: "Know thy users, for they are not you!"

So this chapter will talk about people. It covers a few fundamental ideas briefly in this introduction, and then discusses the patterns themselves. These patterns differ from those in the rest of the book. They describe human behaviors—as opposed to system behaviors—that the software you design may need to support. Software that supports these human behaviors help users achieve their goals.

A MEANS TO AN END

Everyone who uses a tool, software or otherwise, has a reason to use it. For instance:

- Finding some fact or object
- Learning something
- Performing a transaction
- Controlling or monitoring something
- Creating something
- Conversing with other people
- Being entertained

Well-known idioms, user behaviors, and design patterns can support each of these abstract goals. Interaction designers have learned, for example, how to help people search through vast amounts of online information for specific facts. They've learned how to present tasks so that it's easy to walk through them. They are learning ways to support the building of documents, illustrations, and code.

The first step in designing an interface is figuring out what its users are really trying to accomplish. Filling out a form, for example, almost never is a goal in and of itself—people only do it because they're trying to buy something online, get their driver's license renewed, or install a networked printer.[1] They're performing some kind of transaction.

Asking the right questions can help you connect user goals to the design process. Users and clients typically speak to you in terms of desired features and solutions, not of needs and problems. When a user or client tells you he wants a certain feature, ask why he wants it—determine his immediate goal. Then, to the answer of this question, ask "why" again. And again. Keep asking until you move well beyond the boundaries of the immediate design problem.[2]

Why should you ask these questions if you have clear requirements? Because if you love designing things, it's easy to get caught up in an interesting interface-design problem. Maybe you're good at building forms that ask for just the right information, with the right controls, all laid out nicely. But the real art of interface design lies in *solving the right problem.*

So don't get too fond of designing that form. If there's any way to finish the transaction without making the user go through that form at all, get rid of it altogether. That gets the user closer to his goal, with less time and effort spent on his part. (And maybe yours, too.)

Let's use the "why" approach to dig a little deeper into some typical design scenarios.

- Why does a mid-level manager use an email client? Yes, of course—"to read email." Why does she read and send email in the first place? To converse with other people. Of course, other means might achieve the same ends: the phone, a hallway conversation, a formal document. But apparently email fills some needs that the other methods don't. What are they, and why are they important to her? Privacy? The ability to archive a conversation? Social convention? What else?
- A father goes to an online travel agent, types in the city where his family will take a summer vacation, and tries to find plane ticket prices on various dates. He's learning from what he finds, but his goal isn't just browsing and exploring different options. Ask why. His goal is actually a transaction: buying plane tickets. Again, he could have done that at many different web sites, or over the phone with a live travel agent. How is this site better than those other options? Is it faster? Friendlier? More likely to find a better deal?

1. See Eric Raymond's essay, "The Luxury of Ignorance: An Open-Source Horror Story," about his travails with a Linux print utility at *http://www.catb.org/-esr/writings/cups-horror.html.*

2. This is the same principle that underlies a well-known technique called "root cause analysis." However, root cause analysis is a tool for fixing organizational failures; here, you use its "five whys" (more or less) to understand everyday user behaviors and feature requests.

- A cell phone user wants a way to search through his phone list more quickly. You, as the designer, can come up with some clever ideas to save keystrokes while searching. But why did he want it? It turns out that he makes a lot of calls while driving, and he doesn't want to take his eyes off the road more than he has to—he wants to make calls while staying safe (to the extent that that's possible). The ideal case is that he doesn't have to look at the phone at all! A better solution is voice dialing: all he has to do is speak the name, and the phone makes the call for him.
- Sometimes goal analysis really isn't straightforward at all. A snowboarding site might provide information (for learning), an online store (transactions), and a set of Flash movies (entertainment). Let's say someone visits the site for a purchase, but she gets sidetracked into the information on snowboarding tricks—she switched goals from accomplishing a transaction to browsing and learning. Maybe she'll go back to purchasing something, maybe not. And does the entertainment part of the site successfully entertain both the twelve-year-old and the thirty-five-year-old? Will the thirty-five-year-old go elsewhere to buy his new board if he doesn't feel at home there, or does he not care?

It's deceptively easy to model users as a single faceless entity—"The User"—walking through a set of simple use cases, with one task-oriented goal in mind. But that won't necessarily reflect your users' reality.

To do design well, you need to take many "softer" factors into account: gut reactions, preferences, social context, beliefs, and values. All of these factors could affect the design of an application or site. Among these "softer" factors, you may find the critical feature or design factor that makes your application more appealing and successful.

So be curious. Specialize in it. Find out what your users are really like, and what they really think and feel.

THE BASICS OF USER RESEARCH

Empirical discovery is the only really good way to obtain this information. To get a design started, you'll need to characterize the kinds of people who will use whatever you design (including the "softer" categories just mentioned), and the best way to do that is to go out and meet them.

Each user group is unique, of course. The target audience for, say, a new cell phone will differ dramatically from that for a piece of scientific software. Even if the same person uses both, his expectations for each are different—a researcher using scientific software might tolerate a less-polished interface in exchange for high functionality, whereas that same person may trade in his new phone if he finds its UI to be too hard to use after a few days.

Every user is unique, too. What one person finds difficult, the next one won't. The trick is to figure out what's *generally* true about your users, which means learning about enough individual users to separate the quirks from the common behavior patterns.

Specifically, you'll want to learn:

- Their goals in using the software you design
- The specific tasks they undertake in pursuit of those goals
- The language and words they use to describe what they're doing
- Their skill at using software similar to what you're designing
- Their attitudes toward the kind of thing you're designing, and how different designs might affect those attitudes

I can't tell you what your particular target audience is like. You need to find out what they might do with the software you design, and how it fits into the broader context of their lives. Difficult though it may be, try to describe your potential audience in terms of how and why they might use your software. You might get several distinct answers, representing distinct user groups; that's okay. You might be tempted to throw up your hands and say, "I don't know who the users are," or, "Everyone is a potential user." That doesn't help you focus your design at all—without a concrete and honest description of those people, your design will proceed with no grounding in reality.

Unfortunately, this user-discovery phase will consume serious time early in the design cycle. It's expensive, but always worth it, because you stand a better chance at solving the right problem—you'll build the right thing in the first place.

Fortunately, lots of books, courses, and methodologies now exist to help you. Although this book does not address user research, here are some methods and topics to consider.

Direct observation

Interviews and on-site user visits put you directly into the user's world. You can ask users about what their goals are and what tasks they typically do. Usually done "on location," where users would actually use the software (e.g., in a workplace or at home), interviews can be structured—with a predefined set of questions—or unstructured, in which you might probe whatever subject comes up. Interviews give you a lot of flexibility; you can do many or a few, long or short, formal or informal, on the phone or in person. These are great opportunities to learn what you don't know. Ask why. Ask it again.

Case studies

Case studies give you deep, detailed views into a few representative users or groups of users. You can sometimes use them to explore "extreme" users that push the boundaries of what the software can do, especially when the goal is a redesign of existing software. You also can use them as longitudinal studies—exploring the context of use over weeks, months, or even years. Finally, if you design custom software for a single user or site, you'll want to learn as much as possible about the actual context of use.

Surveys

Written surveys can collect information from many users. You can actually get statistically significant numbers of respondents with these. Since there's no direct human contact, you will miss a lot of extra information—whatever you don't ask about, you won't learn about—but you can get a very clear picture of certain aspects of your target audience. Careful survey design is essential. If you want reliable numbers instead of a qualitative "feel" for the target audience, you absolutely must write the questions correctly, pick the survey recipients correctly, and analyze the answers correctly—and that's a science.

Personas

Personas aren't a data-gathering method, but they do help you figure out what to do with that data once you've got it. This is a design technique that "models" the target audiences. For each major user group, you create a fictional person that captures the most important aspects of the users in that group: what tasks they're trying to accomplish, their ultimate goals, and their experience levels in the subject domain and with computers in general. They help you stay focused. As your design proceeds, you can ask yourself questions like, "Would this fictional person really do X? What would she do instead?"

And there's more. You might notice that some of these methods and topics, like interviews and surveys, sound suspiciously like marketing activities. That's exactly what they are. Focus groups can be useful too (though not so much as the others), and the concept of market segmentation resembles the definition of target audiences we've used here. In both cases, the whole point is to understand the audience as best you can.

The difference is that as a designer, you're trying to understand the people who use the software. A marketing professional tries to understand those who buy it.

It's not easy to understand the real issues that underlie users' interaction with a system. Users don't always have the language or introspective skill to explain what they really need to accomplish their goals, and it takes a lot of work on your part to ferret out useful design concepts from what they *can* tell you—self-reported observations usually are biased in subtle ways.

Some of these techniques are very formal, and some aren't. Formal and quantitative methods are valuable because they're good science. When applied correctly, they help you see the world as it actually is, not how you think it is. If you do user research haphazardly, without accounting for biases like the self-selection of users, you may end up with data that doesn't reflect your actual target audience—and that can only hurt your design in the long run.

But if you don't have time for formal methods, it's better to just meet a few users informally than to not do any discovery at all. Talking with users is good for the soul. If you're able to empathize with users and imagine those individuals actually using your design, you'll produce something much better.

USERS' MOTIVATION TO LEARN

Before you start the design process, consider your overall approach. Think about how you might design its overall interaction style—its personality, if you will.

When you carry on a conversation with someone about a given subject, you adjust what you say according to your understanding of the other person. You might consider how much he cares about the subject, how much he already knows about it, how receptive he is to learning from you, and whether he's even interested in the conversation in the first place. If you get any of that wrong, then bad things happen—he might feel patronized, uninterested, impatient, or utterly baffled.

This analogy leads to some obvious design advice. The subject-specific vocabulary you use in your interface, for instance, should match your users' level of knowledge; if some users won't know that vocabulary, give them a way to learn the unfamiliar terms. If they don't know computers very well, don't make them use sophisticated widgetry or uncommon interface-design conventions. If their level of interest might be low, respect that, and don't ask for too much effort for too little reward.

Some of these concerns permeate the whole interface design in subtle ways. For example, do your users expect a short, tightly focused exchange about something very specific, or do they look for a conversation that's more of a free-ranging exploration? In other words, how much openness is there in the interface? Too little, and your users feel trapped and unsatisfied; too much, and they stand there paralyzed, not knowing what to do next, unprepared for that level of interaction.

Therefore, you need to choose how much freedom your users have to act arbitrarily. At one end of the scale might be a software installation wizard: the user is carried through it with no opportunity to use anything other than Next, Previous, or Cancel. It's tightly focused and specific, but quite efficient—and satisfying, to the extent that it works and is quick. At the other end might be an application like Excel, an "open floor plan" interface that exposes a huge number of features in one place. At any given time, the user has about 872 things that she can do next, but that's considered good because self-directed, skilled users can do a lot with that interface. Again, it's satisfying, but for entirely different reasons.

Here's an even more fundamental question: how much effort are your users willing to spend to learn your interface?

It's easy to overestimate. Maybe they use it every day on the job—clearly they'd be motivated to learn it well in that case, but that's rare. Maybe they use it sometimes, and learn it only well enough to get by. Maybe they'll only see it once, for 30 seconds. Be honest: can you expect most users to become intermediate-to-expert users, or will most users remain perpetual beginners?

Software designed for intermediate-to-expert users include:

- Photoshop
- Dreamweaver
- Emacs
- Code development environments
- System-administration tools for web servers

In contrast, here are some things designed for occasional users:

- Kiosks in tourist centers or museums
- Windows or Mac OS controls for setting desktop backgrounds
- Purchase pages for online stores

- Installation wizards
- Automated teller machines

The differences between the two groups are dramatic. Assumptions about users' tool knowledge permeate these interfaces, showing up in their screen-space usage, labeling, widget sophistication, and the places where help is (or isn't) offered.

The applications in the first group have lots of complex functionality, but they don't generally walk the user through tasks step-by-step. They assume users already know what to do, and they optimize for efficient operation, not learnability; they tend to be document-centered or list-driven (with a few being command-line applications). They often have entire books and courses written about them. Their learning curves are steep.

The applications in the second group are the opposite: restrained in functionality but helpful about explaining it along the way. They present simplified interfaces, assuming no prior knowledge of document- or list-centered application styles (e.g., menu bars, multiple selection, etc.). Wizards frequently show up, removing attention-focusing responsibility from the user. The key is that users aren't motivated to work hard at learning these interfaces—it's usually just not worth it!

Now that you've seen the extremes, look at the applications in the middle of the continuum:

- Microsoft Office
- Email clients
- Web browsers
- Cell phone applications
- PalmOS

The truth is, most applications fall into this middle ground. They need to serve people on both ends adequately—to help new users learn the tool (and satisfy their need for instant gratification), while enabling frequent-user intermediates to get their work done smoothly. Their designers probably knew that people wouldn't take a three-day course to learn an email client. Yet the interfaces hold up under repeated usage. People quickly learn the basics, reach a proficiency level that satisfies them, and don't bother learning more until they are motivated to do so for specific purposes.

Alan Cooper coined the terms "sovereign posture" and "transient posture" to discuss these approaches. Sovereign-posture applications work with users as partners; users spend time in them, give them their full attention, learn them well, and expand them to full-screen size. Transient-posture programs are brought up briefly, used, and dismissed. These roughly correspond to the two extremes I posited, but not entirely. See the book *About Face 2.0: The Essentials of Interaction Design* for a more nuanced explanation of postures.

Someday you will find yourself in tension between the two ends of this spectrum. Naturally you want people to be able to use your application "out of the box," but you also might want to support frequent or expert users as much as possible. Find a balance that works for your situation. Organizational patterns in Chapter 2 such as **Multi-Level Help**, **Intriguing Branches**, and **Extras on Demand** can help you serve both constituencies.

THE PATTERNS

Even though individuals are unique, people behave predictably. Designers have been doing site visits and user observations for years; cognitive scientists and other researchers have spent many hundreds of hours watching how people do things and how they think about what they do.

So when you observe people using your software, or performing whatever activity you want to support with new software, you can expect them to do certain things. The behavioral patterns listed below often are seen in user observations. Odds are good that you'll see them too, especially if you look for them.

A note for patterns enthusiasts: These patterns aren't like the others in this book. They describe human behaviors, not interface elements, and they're not prescriptive like the patterns in other chapters. Instead of being structured like the other patterns, these are presented as small essays.

Again, an interface that supports these patterns well will help users achieve their goals far more effectively than interfaces that don't support them. And the patterns are not just about the interface, either. Sometimes the entire package—interface, underlying architecture, feature choice, and documentation—needs to be considered in light of these behaviors. But as the interface designer or interaction designer, you should think about these as much as anyone on your team. You may be in the best position to advocate for the users.

1. **Safe Exploration**
2. **Instant Gratification**
3. **Satisficing**
4. **Changes in Midstream**
5. **Deferred Choices**
6. **Incremental Construction**
7. **Habituation**
8. **Spatial Memory**
9. **Prospective Memory**
10. **Streamlined Repetition**
11. **Keyboard Only**
12. **Other People's Advice**

1 safe exploration

"Let me explore without getting lost or getting into trouble."

When someone feels like she can explore an interface and not suffer dire consequences, she's likely to learn more—and feel more positive about it—than someone who doesn't explore. Good software allows people to try something unfamiliar, back out, and try something else, all without stress.

Those "dire consequences" don't even have to be very bad. Mere annoyance can be enough to deter someone from trying things out voluntarily. Clicking away popup windows, re-entering data mistakenly erased, suddenly muting the volume on one's laptop when a web site unexpectedly plays loud music—all can be discouraging. When you design almost any kind of software interface, make many avenues of exploration available for users to experiment with, without costing the user anything.

Here are some examples:

- A photographer tries out a few image filters in an image-processing application. He then decides he doesn't like the results and hits "Undo" a few times to get back to where he was. Then he tries another filter, and another—each time being able to back out of what he did. (The pattern named **Multi-Level Undo**, in Chapter 5, describes how this works.)
- A new visitor to a company's home page clicks various links just to see what's there, trusting that the Back button will always get her back to the main page. No extra windows or popups open, and the Back button keeps working predictably. You can imagine that if a web application does something different in response to the Back button—or if an application offers a button that seems like a Back button, but doesn't behave quite like it—then confusion might ensue. The user can get disoriented while navigating, and may abandon the application altogether.
- A cell phone user wants to try out some intriguing new online functionality, like getting sports scores for the World Series in real time. But he's hesitant to try it because the last time he used an online service, he was charged an exorbitant amount of money just for experimenting with it for a few minutes.

2 instant gratification

"I want to accomplish something now, not later."

People like to see immediate results from the actions they take—it's human nature. If someone starts using an application and gets a "success experience" within the first few seconds, that's gratifying! He'll be more likely to keep using it, even if it gets harder later. He will feel more confident in the application, and more confident in himself, than if it had taken a while to figure things out.

The need to support instant gratification has many design ramifications. For instance, if you can predict the first thing a new user is likely to do, then you should design the UI to make that first thing stunningly easy. If the user's goal is to create something, for instance, then show a new canvas and put a palette next to it. If the user's goal is to accomplish some task, point the way toward a typical starting point.

It also means that you shouldn't hide introductory functionality behind anything that needs to be read or waited for, such as registrations, long sets of instructions, slow-to-load screens, or advertisements. These are discouraging because they block users from finishing that first task quickly.

3 satisficing

"This is good enough. I don't want to spend more time learning to do it better."

When people look at a new interface, they don't read every piece of it methodically and then decide, "Hmmm, I think this button has the best chance of getting me what I want." Instead, a user will rapidly scan the interface, pick whatever he sees first that might get him what he wants, and try it—even if it might be wrong.

The term "satisficing" is a combination of "satisfying" and "sufficing." It was devised in 1957 by the social scientist Herbert Simon, who used it to describe the behavior of people in all kinds of economic and social situations. People are willing to accept "good enough" instead of "best" if learning all the alternatives might cost time or effort.

Satisficing is actually a very rational behavior, once you appreciate the mental work necessary to "parse" a complicated interface. As Steve Krug points out in his book *Don't Make Me Think*, (New Riders) people don't like to think any more than they have to—it's work! But if the interface presents an obvious option or two that the user sees immediately, he'll try it. Chances are good that it will be the right choice, and if not, there's little cost in backing out and trying something else (assuming that the interface supports **Safe Exploration**).

This means several things for designers:

- Make labels short, plainly worded, and quick to read. (This includes menu items, buttons, links, and anything else identified by text.) They'll be scanned and guessed about; write them so that a user's first guess about meaning is correct. If he guesses wrong several times, he'll be frustrated and you're both off to a bad start.
- Use the layout of the interface to communicate meaning. Chapter 4, *Layout*, explains how to do so in detail. Users "parse" color and form on sight, and they follow these cues more efficiently than labels that must be read.
- Make it easy to move around the interface, especially for going back to where a wrong choice might have been made hastily. Provide "escape hatches" (see Chapter 3). On typical web sites, using the Back button is easy, so designing easy forward/backward navigation is especially important for web applications, but it's also important for installed applications and devices.
- Keep in mind that a complicated interface imposes a large cognitive cost on new users. Visual complexity will often tempt nonexperts to satisfice: they look for the first thing that may work.

Satisficing is why many users end up with odd habits after they've been using a system for a while. Long ago, a user may have learned Path A to do something, and even though a later version of the system offers Path B as a better alternative (or was there all along), he sees no benefit in learning it—that takes effort, after all—and he keeps using the less-efficient Path A. It's not necessarily an irrational choice. Breaking old habits and learning something new takes energy, and a small improvement may not be worth the cost to the user.

4 changes in midstream

"I changed my mind about what I was doing."

Occasionally, people change what they're doing in the middle of doing it. Someone may walk into a room with the intent of finding a key she had left there, but while she's there, she finds a newspaper and starts reading it. Or she may visit Amazon to read product reviews, but ends up buying a book instead. Maybe she's just sidetracked; maybe the change is deliberate. Either way, the user's goal changes while she's using the interface you designed.

What this means for designers is that you should provide opportunities for people to do that. Make choices available. Don't lock users into a choice-poor environment with no global navigation, or no connections to other pages or functionality, unless there's a good reason to do so. Those reasons do exist. See the patterns called **Wizard** (Chapter 2), **Hub and Spoke** (Chapter 3), and **Modal Panel** (Chapter 3) for examples.

You also can make it easy for someone to start a process, stop in the middle, and come back to it later to pick up where he left off—a property often called "reentrance." For instance, a lawyer may start entering information into a form on a PDA. Then when a client comes into the room, the lawyer has to turn off the PDA with the intent of coming back to finish the form later. The entered information shouldn't be lost.

To support reentrance, you can make dialog boxes remember values typed previously (see **Good Defaults** in Chapter 7), and they don't usually need to be modal; if they're not modal, a user can drag them aside on the screen for later use. Builder-style applications—text editors, code development environments, and paint programs—can let a user work on multiple projects at one time, thus letting the user put any number of projects aside while she works on another one.

Online surveys hosted by surveymonkey.com sometimes offer a button on each page of a survey that says, "I'll finish it later." This button closes the browser page, records the choices made up to that point, and lets the user come back to finish the survey later.

5 deferred choices

"I don't want to answer that now; just let me finish!"

This follows from people's desire for instant gratification. If you ask a user several seemingly unnecessary questions while he's trying to get something done, he'd often rather skip the questions and come back to them later.

For example, some web-based bulletin boards have long and complicated procedures for registering users. Screen names, email addresses, privacy preferences, avatars, self-descriptions...the list goes on and on. "But I just wanted to post one little thing," says the user plaintively. Why not skip most of the questions, answer the bare minimum, and come back later (if ever) to fill in the rest? Otherwise he might be there for half an hour answering essay questions and finding the perfect avatar image.

Another example is creating a new project in Dreamweaver or other web site editors. There are some things you do have to decide up front, like the name of the project, but you can defer other choices easily—where on the server are you going to put this when you're done? I don't know yet!

Sometimes it's just a matter of not wanting to answer the questions. At other times, the user may not have enough information to answer yet. What if a music-writing software package asked you up front for the title, key, and tempo of a new song, before you've even started writing it? (See Apple's GarageBand for this lovely bit of design.)

The implications for interface design are simple to understand, though not always easy to implement:

- Don't accost the user with too many up-front choices in the first place.
- On the forms that he does have to use, clearly mark the required fields, and don't make too many of them required. Let him move on without answering the optional ones.
- Sometimes you can separate the few important questions or options from others that are less important. Present the short list; hide the long list. See the **Extras on Demand** pattern in Chapter 2.
- Use **Good Defaults** (Chapter 7) wherever possible, to give users some reasonable default answers to start with. But keep in mind that prefilled answers still require the user to look at them, just in case they need to be changed. They have a small cost, too.
- Make it possible for users to return to the deferred fields later, and make them accessible in obvious places. Some dialog boxes show the user a short statement such as "You can always change this later by clicking the Edit Project button." Some web

sites store a user's half-finished form entries or other persistent data, like shopping carts with unpurchased items.

- If registration is required at a web site that provides useful services, users may be far more likely to register if they're first allowed to experience the web site—drawn in and engaged—and then asked later about who they are. In fact, TurboTax 2005 allows a user to work through an entire tax form before creating a username.

6 incremental construction

"Let me change this. That doesn't look right; let me change it again. That's better."

When people create things, they don't usually do it all at once. Even an expert doesn't start at the beginning, work through the creation process methodically, and come out with something perfect and finished at the end.

Quite the opposite. Instead, she starts with some small piece of it, works on it, steps back and looks at it, tests it (if it's code or some other "runnable" thing), fixes what's wrong, and starts to build other parts of it. Or maybe she starts over if she really doesn't like it. The creative process goes in fits and starts. It moves backwards as much as forwards sometimes, and it's often incremental, done in a series of small changes instead of a few big ones. Sometimes it's top-down; sometimes it's bottom-up.

Builder-style interfaces need to support that style of work. Make it easy to build small pieces one at a time. Keep the interface responsive to quick changes and saves. Feedback is critical: constantly show the user what the whole thing looks and behaves like while the user works. If you deal with code, simulations, or other executable things, make the "compile" part of the cycle as short as possible so the operational feedback feels immediate—leave little or no delay between the user making changes and seeing the results.

When good tools support creative activities, the activities can induce a state of "flow" in the user. This is a state of full absorption in the activity, during which time distorts, other distractions fall away, and the person can remain engaged for hours—the enjoyment of the activity is its own reward. Artists, athletes, and programmers all know this state.

Bad tools will keep someone distracted, guaranteed. If the user has to wait even half a minute to see the results of the incremental change she just made, then her concentration is broken; flow is disrupted.

If you want to read more about flow, look at the books by Mihaly Csikszentmihalyi, who studied it for years.

7 habituation

"That gesture works everywhere else; why doesn't it work here, too?"

When one uses an interface repeatedly, some frequently used physical actions become reflexive: typing Control-S to save a document, clicking the Back button to leave a web page, pressing Return to close a modal dialog box, using gestures to show and hide windows, or even pressing a car's brake pedal. The user no longer needs to think consciously about these actions. They've become habitual.

This tendency certainly helps people become expert users of a tool (and it helps create a sense of flow, too). Habituation measurably improves efficiency, as you can imagine. But it can also lay traps for the user. If a gesture becomes a habit and the user tries to use it in a situation when it doesn't work—or, worse, does something destructive—then the user is caught short. He suddenly must think about the tool again (What did I just do? How do I do what I intended?), and he might have to undo any damage done by the gesture.

For instance, Control-X, Control-S is the "save this file" key sequence used by the emacs text editor.

Control-A moves the text-entry cursor to the beginning of a line. These acts become habitual for emacs users. When a user types Control-A, Control-X, Control-S at emacs, it performs a fairly innocuous pair of operations.

Now what happens when he types that same habituated sequence in Microsoft Word?

1. Control-A: select all
2. Control-X: cut the selection (the whole document, in this case)
3. Control-S: save the document (whoops)

This is why consistency across applications is important!

Just as important, though, is consistency within an application. Some applications are evil because they establish an expectation that some gesture will do Action X, except in one special mode, where it suddenly does Action Y. Don't do that. It's a sure bet that users will make mistakes, and the more experienced they are—i.e., the more habituated they are—the more likely they are to make that mistake.

This is also why confirmation dialog boxes often don't work to protect a user against accidental changes. When modal dialog boxes pop up, the user can easily get rid of them just by clicking "OK" or hitting Return (if the OK button is the default button). If dialogs pop up all the time when the user is making intended changes, such as deleting files, it becomes a habituated response. Then when it actually matters, the dialog box doesn't have any effect, because it slips right under the user's consciousness.

I've seen at least one application that sets up the confirmation dialog box's buttons randomly from one invocation to another. One actually has to *read* the buttons to figure out what to click! This isn't necessarily the best way to do a confirmation dialog box—in fact, it's better to not have them at all under most circumstances—but at least this design sidesteps habituation creatively.

8 spatial memory

"I swear that button was here a minute ago. Where did it go?"

When people manipulate objects and documents, they often find them again later by remembering where they are, not what they're named.

Take the Windows, Mac, or Linux desktop. Many people use the desktop background as a place to put documents, frequently used applications, and other such things. It turns out that people tend to use spatial memory to find things on the desktop, and it's very effective. People devise their own groupings, for instance, or recall that "the document was at the top right over by such-and-such." (Naturally, there are real-world equivalents too. Many people's desks are "organized chaos," an apparent mess in which the office owner can find anything instantly. But heaven forbid that someone should clean it up for them.)

Many applications put their dialog buttons—OK, Cancel, etc.—in predictable places, partly because spatial memory for them is so strong. In complex applications, people also may find things by remembering where they are relative to other things: tools on toolbars, objects in hierarchies, and so on. Therefore, you should use patterns like **Responsive Disclosure** (Chapter 4) carefully. Adding something to an interface doesn't usually cause problems, but rearranging existing controls can disrupt spatial memory and make things harder to find. It depends. Try it out on your users if you're not sure.

Along with habituation, which is closely related, spatial memory is another reason why consistency across and within applications is good. People may expect to find similar functionality in similar places.

Spatial memory explains why it's good to provide user-arranged areas for storing documents and objects, like the aforementioned desktops. Such things aren't always practical, especially with large numbers of objects, but it works quite well with small numbers. When people arrange things themselves, they're likely to remember where they put them. (Just don't rearrange things for them unless they ask!) The **Movable Panels** pattern in Chapter 4 describes one way to do this.

Also, this is why changing menus dynamically sometimes backfires. People get used to seeing certain items on the tops and bottoms of menus. Rearranging or compacting menu items "helpfully" can work against habituation and lead to user errors.

Incidentally, the tops and bottoms of lists and menus are special locations, cognitively speaking. People notice and remember them more than stuff in the middle of menus. They are the worst items to change out from under the user.

9 prospective memory

"I'm putting this here to remind myself to deal with it later."

Prospective memory is a well-known phenomenon in psychology that doesn't seem to have gained much traction yet in interface design. But I think it should.

We engage in prospective memory when we plan to do something in the future, and we arrange some way of reminding ourselves to do it. For example, if you need to bring a book to work the next day, you might put it on a table beside the front door the night before. If you need to respond to someone's email later (just not right now!), you might leave that email on your screen as a physical reminder. Or, if you tend to miss meetings, you might arrange for Outlook or your Palm to ring an alarm tone five minutes before each meeting.

Basically, this is something almost everyone does. It's part of how we cope with our complicated, highly scheduled, multitasked lives: we use knowledge "in the world" to aid our own imperfect memories. We need to be able to do it well.

Some software does support prospective remembering. Outlook and PalmOS, as mentioned above, implement it directly and actively; they have calendars (as do many other software systems), and they sound alarms. But what else can you use for prospective memory?

- Notes to oneself, like virtual "sticky notes"
- Windows left onscreen
- Annotations put directly into documents (like "Finish me!")
- Browser bookmarks, for web sites to be viewed later
- Documents stored on the desktop, rather than in the usual places in the filesystem
- Email kept in an inbox (and maybe flagged) instead of filed away

People use all kinds of artifacts to support passive prospective remembering. But notice that almost none of the techniques in the list above were designed with that in mind! What they *were* designed for is flexibility—and a laissez-faire attitude toward how users organize their information. A good email client lets you create folders with any names you want, and it doesn't care what you do with messages in your inbox. Text editors don't care what you type, or what giant bold magenta text means to you; code editors don't care that you have a "Finish this" comment in a method header. Browsers don't know why you keep certain bookmarks around.

In many cases, that kind of hands-off flexibility is all you really need. Give people the tools to create their own reminder systems. Just don't try to design a system that's too smart for its own good. For instance, don't assume that just because a window's been idle for a while, no one's using it and it should be closed. In general, don't "helpfully" clean up files or objects that the system may think are useless; someone may be leaving them around for a reason. Also, don't organize or sort things automatically unless the user asks the system to do so.

As a designer, is there anything positive you can do for prospective memory? If someone leaves a form half-finished and closes it temporarily, you could retain the data in it for the next time—it will help remind the user where she left off. Similarly, many applications recall the last few objects or documents they edited. You could offer bookmarks-like lists of "objects of interest"—both past and future—and make that list easily available for reading and editing.

Here's a bigger challenge: if the user starts tasks and leaves them without finishing them, think about how to leave some artifacts around, other than open windows, that identify the unfinished tasks. Another one: how might a user gather reminders from different sources (email, documents, calendars, etc.) into one place? Be creative!

10 streamlined repetition

"I have to repeat this how many times?"

In many kinds of applications, users sometimes find themselves having to perform the same operation over and over again. The easier it is for them, the better. If you can help reduce that operation down to one keystroke or click per repetition—or, better, just a few keystrokes or clicks for all repetitions—then you will spare users much tedium.

Find and Replace dialog boxes, often found in text editors (Word, email composers, etc.), are one good adaptation to this behavior. In these dialog boxes, the user types the old phrase and the new phrase. Then it takes only one "Replace" button click per occurrence in the whole document. And that's only if the user wanted to see or veto each replacement—if they're confident that they really should replace all occurrences, then they can click the "Replace All" button. One gesture does the whole job.

Here's a more general example. Photoshop lets you record "actions" when you want to perform some arbitrary sequence of operations with a single click. If you want to resize, crop, brighten, and save 20 images, you can record those four steps as they're done to the first image, and then click that action's "Play" button for each of the remaining 19. See the **Macros** pattern in Chapter 5 for more information.

Scripting environments are even more general. Unix and its variants allow you to script anything you can type into a shell. You can recall and execute single commands, even long ones, with a Control-P and return. You can take any set of commands you issue to the command line, put them in a for-loop, and execute them by hitting the Return key once. Or you can put them in a shellscript (or in a for-loop in a shellscript!) and execute them as a single command. Scripting is very powerful, and as it gets more complex, it becomes full-fledged programming.

Other variants include copy-and-paste capability (preventing the need to retype the same thing in a million places), user-defined "shortcuts" to applications on operating-system desktops (preventing the need to find those applications' directories in the filesystem), browser bookmarks (so users don't have to type URLs), and even keyboard shortcuts.

Direct observation of users can help you find out just what kinds of repetitive tasks you need to support. Users won't always tell you outright. They may not even be aware that they're doing repetitive things that they could streamline with the right tools—they may have been doing it so long that they don't even notice anymore. By watching them work, you may see what they don't see.

In any case, the idea is to offer users ways to streamline the repetitive tasks that could otherwise be time consuming, tedious, and error prone.

11 keyboard only

"Please don't make me use the mouse."

Some people have real physical trouble using a mouse. Others prefer not to keep switching between the mouse and keyboard because that takes time and effort—they'd rather keep their hands on the keyboard at all times. Still others can't see the screen, and their assistive technologies often interact with the software using just the keyboard API.

For the sakes of these users, some applications are designed to be "driven" entirely via the keyboard. They're usually mouse-driven too, but there is no operation that must be done with *only* the mouse—keyboard-only users aren't shut out of any functionality.

Several standard techniques exist for keyboard-only usage:

- You can define keyboard shortcuts, accelerators, and mnemonics for operations reachable via application menu bars, like Control-S for Save. See your platform style guide for the standard ones.
- Selection from lists, even multiple selection, usually is possible using arrow keys in combination with modifiers (like the Shift key), though this depends on which platform you use.
- The Tab key typically moves the keyboard focus—the control that gets keyboard entries at the moment—from one control to the next, and Shift-Tab moves backwards. This is sometimes called "tab traversal." Many users expect it to work on form-style interfaces.
- Most standard controls, even radio buttons and combo boxes, let users change their values from the keyboard by using arrow keys, the Return key, or the spacebar.
- Dialog boxes and web pages often have a "default button"—a button representing an action that says "I'm done with this task now." On web pages, it's often Submit or Done; on dialog boxes, OK or Cancel. When users hit the Return key on this page or dialog box, that's the operation that occurs. Then it moves the user to the next page or returns him to the previous window.

There are more techniques. Forms, control panels, and standard web pages are fairly easy to drive from the keyboard. Graphic editors, and anything else that's mostly spatial, are much harder, though not impossible. See **Spring-Loaded Mode**, in Chapter 8, for one way to use keyboards in graphic editors.

Keyboard-only usage is particularly important for data-entry applications. In these, speed of data entry is critical, and users can't afford to move their hands off the keyboard to the mouse every time they want to move from one field to another, or even from one page to another. (In fact, many of these forms don't even require users to hit the Tab key to traverse between controls; it's done automatically.)

12 other people's advice

"What did everyone else say about this?"

People are social. As strong as our opinions may sometimes be, what our peers think tends to influence us.

Witness the spectacular growth of online "user comments": Amazon for books, IMDb.com for movies, photo.net and flickr for photographs, and countless retailers who offer space for user-submitted product reviews. Auction sites like eBay formalize user opinions into actual prices. Blogs offer unlimited soapbox space for people to opine about anything they want, from products to programming to politics.

The advice of peers, whether direct or indirect, influences people's choices when they decide any number of things. Finding things online, performing transactions (should I buy this product?), playing games (what have other players done here?), and even building things—people can be more effective when aided by others. If not, they might at least be happier with the outcome.

Here's a more subtle example. Programmers use the MATLAB application to do scientific and mathematical tasks. Every few months, the company that makes MATLAB holds a public programming contest; for a few days, every contestant writes the best MATLAB code they can to solve a difficult science problem. The fastest, most accurate code wins. The catch is that every player can see everyone else's code—and copying is encouraged! The "advice" in this case is indirect, taking the form of shared code, but it's quite influential. In the end, the winning program is never truly original, but it's undoubtedly better code than any single solo effort would have been. (In many respects, it's a microcosm of open-source software development, which is driven by a powerful set of social dynamics.)

Not all applications and software systems can accommodate a social component, and not all should try. But consider whether it might enhance the user experience to do so. And you could be more creative than just tacking a web-based bulletin board onto an ordinary site. How can you persuade users to take part constructively? How can you integrate it into the typical user's workflow?

If the task is creative, maybe you can encourage people to post their creations for the public to view. If the goal is to find some fact or object, perhaps you can make it easy for users to see what other people found in similar searches.

Of the patterns in this book, **Multi-Level Help** (Chapter 2), most directly addresses this idea; an online support community is a valuable part of a complete help system for some applications.

02

ORGANIZING THE CONTENT:
INFORMATION ARCHITECTURE AND APPLICATION STRUCTURE

At this point, you may know what your users want out of your application. You may know which idiom or interface type to use, such as a graphic editor, a form, web-like hypertext, or a media player—or an idea of how to combine several of them. If you're really on the ball, you've written down some typical scenarios that describe how people might use high-level elements of the application to accomplish their goals. You have a clear idea of what value this application adds to people's lives.

Now what?

You could start making sketches of the interface. Many visual thinkers do that at this stage. If you're the kind of person who likes to think visually, and needs to play with sketches while working out the broad strokes of the design, go for it.

But if you're not a visual thinker by nature (and sometimes even if you are), then hold off on the interface sketches. They might lock your thinking into the first visual designs you manage to put on paper. You need to stay flexible and creative for a little while, until you work out the overall organization of the application.

High-level organization is a wickedly difficult topic. It helps to think about it from several angles, so this introduction takes two aspects that I've found useful and discusses them in some depth.

The first, "Dividing Stuff Up," encourages you to try separating the content of the application entirely from its physical presentation. Rather than thinking in terms of windows, tree views, and links, you might think abstractly about how to organize the actions and objects in your application in the way truest to your subject matter. You can postpone the decisions about using specific windows and widgets. Clearly this separation of concerns is useful when you design multimodal applications (e.g., the same content presented both on the Web and on a palmtop, with very different physical presentations), but it's also good for brand new applications or deep redesigns. This approach forces you to think about the right things first: organization and task flows.

Second, "Physical Structure" gets into the presentation of the material in pages, windows, and panels. In truth, it's very difficult to completely separate presentation from organization of the content; they're interdependent. The physical forms of devices and web pages can place tight constraints on a design, and on the desktop, an application's window structure is a major design choice. So it earns a place in this chapter.

THE BASICS OF INFORMATION ARCHITECTURE: DIVIDING STUFF UP

In the Preface, I talked a bit about interface idioms.[1] These, you might recall, are interface types or styles that have become familiar to some user populations. They include text editors, forms, games, command lines, and spreadsheets. They're useful because they let you start a design with a set of familiar conventions; you don't have to start from first principles. And once a first-time user recognizes the idiom being used, she has a head start on understanding the interface.

Whatever it is you're building, you've probably decided which idioms to use. But what may not be so obvious is how to organize the "stuff" you're presenting via these idioms. If your application is small enough to fit on one page or physical panel, great—you're off and running. But odds are good that you're dealing with a lot of features, tools, or content areas. The nature of the high-tech industry is to keep cramming more stuff into these interfaces, since features usually are what sell.

If you've done any work on web sites, you may know the term "information architecture." That's essentially what you'll be doing first. You need to figure out how to structure all this content and functionality: how to organize it, label it, and guide a user through the interface to get what they came for. Like a real-world architect, you're planning the informational "space" where people will dwell.

But applications are different from traditional web sites. Think about it in terms of "nouns" versus "verbs." In web sites and many other media—books, movies, music, graphic design—you work with nouns. You arrange them, present them, categorize them, and index them. Users know what to do with text and images and such. But applications, by definition, exist so people can get things done: write, draw, perform transactions, interact with others, and keep track of things. You're manipulating verbs now. You need to structure the interface so users always know what to do next (or at least have a good idea where to look).

Most applications (and many web sites) are organized according to one or more of the following approaches. Some use nouns, others use verbs:

- Lists of objects—e.g., an inbox full of email messages
- Lists of actions or tasks—e.g., browse, buy, sell, or register
- Lists of subject categories—e.g., health, science, or technology
- Lists of tools—e.g., calendar, address book, or notepad

You should base your choice on several interrelated factors: the nature and domain (subject matter) of the application, users' domain knowledge, users' comfort level with computers in general, and, most of all, how closely your application needs to match the *mental models* that users already have of the domain. (Mental models represent what users believe to be true about something, based on previous experience or understanding: classifications, vocabulary, processes, cause and effect, and so on.)

1. The term "idiom" comes from Scott McCloud's *Understanding Comics*, where it's used to describe a genre of work that has developed its own vocabulary of styles, gestures, and content. Another term might be "type," as used in Malcolm McCullough's *Digital Ground* to describe architectural forms and conventions.

You can trace many problems in UI design to a poor choice here, or worse, a confusing mixture of more than one type of organization—like tools and subject categories mixed into one navigation bar with ambiguous titles.

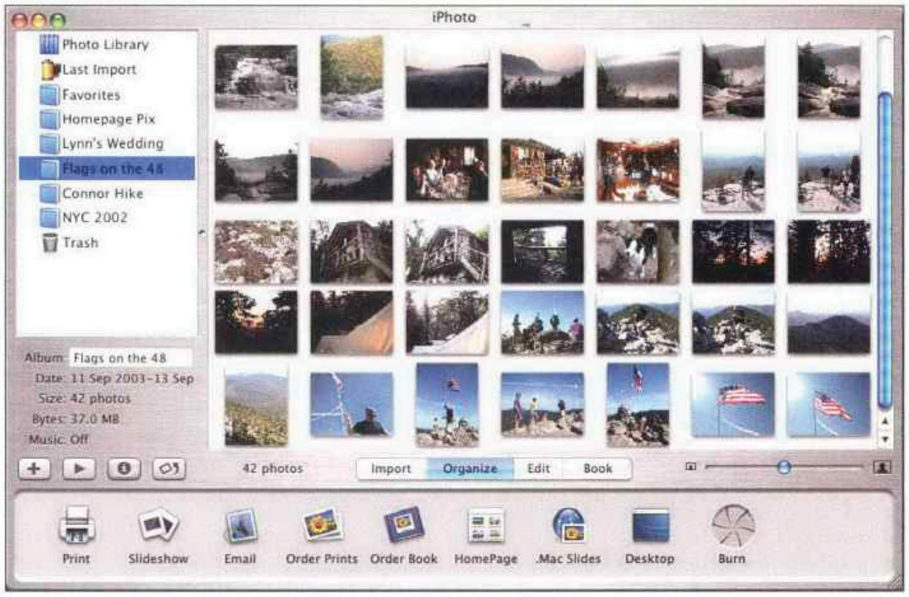
On the other hand, sometimes a mixed organization works fine. Some of the more interesting small-scale UI innovations have come from mixing nouns with verbs on the same menu, for instance; its usability depends on context. Also, you can apply these divisions not only to the top level of the application, but to numerous levels inside them. Different parts of an interface demand different organizational approaches.

Again, this isn't rocket science; you've seen these concepts before. But sometimes it's easy to choose one kind of division by default and not think carefully about which might be best. By calling them out, we make them visible and amenable to discussion. This will be true about many patterns and organizational models described in this book.

Let's take a closer look at these four categorizations and see what they're each best for.

LISTS OF OBJECTS

Most of the time, it will be pretty obvious when to use this categorization. Collections of email messages, songs, books, images (see the iPhoto example in Figure 2-1), search results, and financial transactions—we cope with them in the software we use every day. From these lists, we reach various familiar interface idioms: forms to edit things, media players to play things, and web pages to view things.



Album: Flags on the 48
Date: 11 Sep 2003–13 Sep
Size: 42 photos
Bytes: 37.0 MB
Music: Off

FIGURE 2-1 / Lists of photos in iPhoto, sorted by album and displayed as thumbnails in a table

You will find these objects in selectable lists, tables, trees, or whatever is appropriate; some UIs are very creative. At one extreme, cell phone phonebooks may be short and linear, comprising only a few entries that you can scan quickly on a tiny screen. But TiVos list their recorded TV shows in multilevel hierarchies that you must traverse with several clicks, and the most sophisticated email clients allow all kinds of complex sorting and filtering. When you build these kinds of interfaces, make sure the design scales up appropriately, and take care to match the capabilities and needs of users with the functionality your interface provides.

There's much to be said about organizing and presenting the objects in such an interface. That's your next task as information architect. These models are most common:

- Linear, usually sorted
- 2D tables, also sorted, which often let the user sort via column headers, or filter according to various criteria
- A hierarchy that groups items into categories (and possibly subcategories)
- A hierarchy that reveals relationships: parent/child, containers, etc.
- Spatial organizations, such as maps, charts, or desktop-like areas in which users can place things where they want

In fact, all of these models (except 2D tables) apply to all four approaches to dividing up an interface: objects, tasks, categories, and tools. Your choice should depend upon what people want to do with the application, what best fits their mental models, and what best suits the natural organization—if any—of the objects in question.

If you present timetables for city buses, for instance, the natural organization is by bus or route number. A linear list of routes is a valid way to organize it. But not everyone will know what bus number they want; a spatial organization, like an interactive city map, may be more useful. You also might consider a hierarchy of areas, stations in those areas, and routes leaving those stations.

Chapter 6, *Showing Complex Data*, covers these organizational models for "nouns" in more detail. Of the patterns in this chapter, **Two-Panel Selector** is commonly used to structure this kind of interface, as is **One-Window Drilldown**.

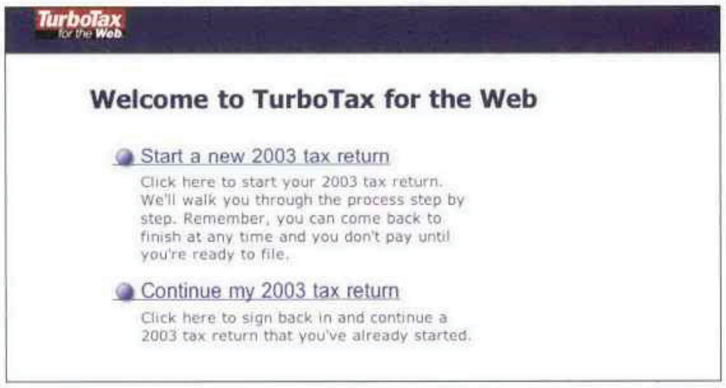
Then, once the user has selected some object, what do they do with it? Read on!

LISTS OF ACTIONS

This approach is verb- instead of noun-centered. Instead of asking the user, "What do you want to work on?", these kinds of interfaces ask, "What do you want to do?" Such interfaces range from TurboTax's high-level decision tree (one screen of which is shown in Figure 2-2) to long menus of actions to be performed on an edited document or selected object.

What's nice about these is that they're often described in plain English. People can take them at face value. When you understand the application's domain well enough to define the correct set of tasks, the interface you design becomes quite usable, even to first-time users.

The hard part is dealing with the proliferation of actions that might be available to the user. Too many actions, more so than too many objects, can make it very hard for users to figure out what to do.



TurboTax for the Web

Welcome to TurboTax for the Web

Start a new 2003 tax return

Click here to start your 2003 tax return. We'll walk you through the process step by step. Remember, you can come back to finish at any time and you don't pay until you're ready to file.

Continue my 2003 tax return

Click here to sign back in and continue a 2003 tax return that you've already started.

FIGURE 2-2 / A friendly task-based organization at *http://turbotax.com*, described in terms of verbs—"Start" and "Continue"—and supplemented by helpful explanations

Desktop applications have menu bars and toolbars available for displaying large numbers of actions at once; most users understand these familiar conventions, at least superficially. Applications that use the **One-Window Drilldown** pattern can present whole-page menus, provided they're not too long. And the **Canvas Plus Palette** pattern talks about one very typical way to organize creational actions for use in many kinds of visual builders. In fact, all of Chapter 5 is devoted to various ways of placing, sorting, and organizing actions on an interface.

But the designers of small device interfaces, such as cell phones and PDAs, have interesting constraints. All they can easily do is present single-click choices of a few functions: three at a time if they're lucky, but usually only one or two. For them, it's critical to prioritize which actions are the most frequently chosen at any given point in the interaction, so they can be assigned to those one or two "softkeys" or buttons (see Figure 2-3). That careful prioritization is good discipline, even for web and desktop applications.

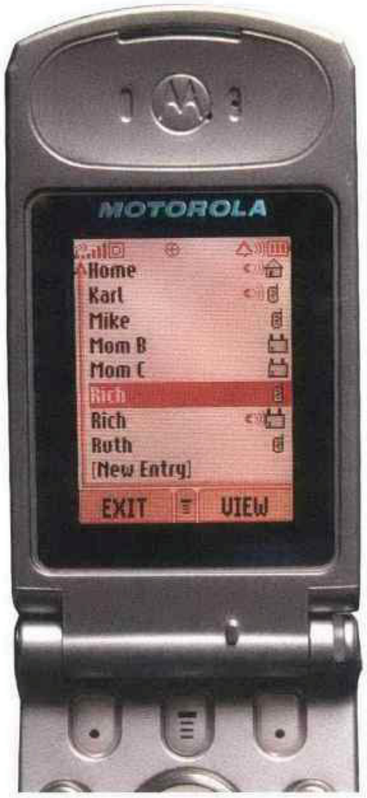


FIGURE 2-3 / This cell phone contains a linear list of entries in a phone book. At the bottom of the screen, you see a pair of softkeys—changeable labels for the hardware buttons underneath them—labeled "Exit" and "View." The lefthand button is almost always Exit for all applications (users thus can become habituated to that button). However, the righthand button changes according to what you're doing—it's always the most common action.

All other possible actions are hidden inside a menu, reachable via the middle softkey with the T-shaped icon on it. This division of common versus not-so-common is an example of **Extras On Demand**, a pattern in this chapter. The designers had to make a difficult choice about which action was most important, since showing all of them at once wasn't an option.

LISTS OF SUBJECT CATEGORIES

Web sites and online references divide up their content by subject category all the time. They have large amounts of stuff for people to browse through, and it often makes the most sense to organize it by subject, or by a similar categorization. But the success of a category-based organization, like that of tasks, depends on how well you've anticipated what users are looking for when they first see your interface. Again, you need to understand the application's domain well enough to match the mental models that your users already have.

Most applications aren't organized this way. Subject categories are better for sorting out nouns than verbs, and action-oriented software usually isn't a good fit for it. That said, help systems—which should be an integral part of an application's design—often do use it, and if an application really does combine knowledge lookup with actions (like medical applications or mapping software), this could come in handy.

Figure 2-4 shows a popular example. The iTunes Music Store organizes its thousands of songs by album, artist, and then genre; that's how its users browse the catalog when they're not using the search facility. What if it were organized by some other means, like the musicians' hair length at the time of the 2004 Grammy Awards? That organization doesn't match most people's mental models of the music universe (hopefully). No one could find anything they were looking for.

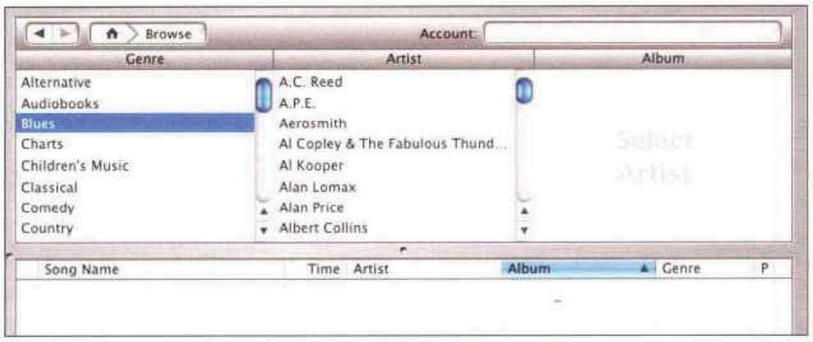


FIGURE 2-4 / The iTunes Music Store categorizes songs by album, artist, and genre. iTunes itself adds playlists as a category. This organizational model, combined with the familiar media-player idiom, really is the heart of the iTunes application.

In any case, organization by subject category might be useful to your application. Related types of organizations include alphabetical, chronological, geographical, and even audience type (like frequent users versus first-time users). See the book *Information Architecture for the World Wide Web* (O'Reilly) for more information.

LISTS OF TOOLS

Operating systems, palmtops, and cell phones all provide access to a range of tools, or sub-applications, used within their physical frameworks. Some applications, such as Microsoft Money, work that way too, and some web sites offer self-contained web applications such as wizards or games.

Once again, this works best when users have a clear, predictable expectation of what the tools should be. Calendars and phone lists are pretty recognizable. So are check-balancing programs and investment interfaces, as one might find in financial applications like MS Money. If your web site offers some novel and strangely named web apps, and they are mixed in with subject categories, users generally are not going to "get it" (unless they're very motivated).

For some reason, a tool-based organizational model fails particularly badly when the names of tools mix with actions, tasks, or objects. Mixing them tends to break people's expectations of what those items do—whether presented as menu choices, list items, buttons, or links—especially for new users who don't know what the names of the tools are yet. Users you observe often do not articulate this confusion well, so beware. You should watch them for subtle signs of confusion, but don't ask them about it directly.

On the other hand, if the user's goal isn't to get something important done, but rather to explore and play, then this strategy might work. Interesting names might attract attention and cause people to click on them just to see what they are. (See the **Intriguing Branches** pattern in this chapter.) But in this case, predictability isn't necessarily a huge benefit. In most software, predictability is quite important.

How can you organize the presentation of a list of tools? Linear organizations are common, since there usually aren't many of them in the first place. PalmOS and many other small devices use a grid of them (see Figure 2-5), which essentially is a linear list. Usually they're sorted alphabetically, for ease of lookup, or by expected frequency of use. When there are a lot of tools—and users might add lots more—then you might group them by category, like the Windows start bar does. Some systems let the users place tools wherever they want.

In the next chapter, on navigation, there's a pattern called **Hub and Spoke**. It's often used to structure an application around a tool-based organizational model.

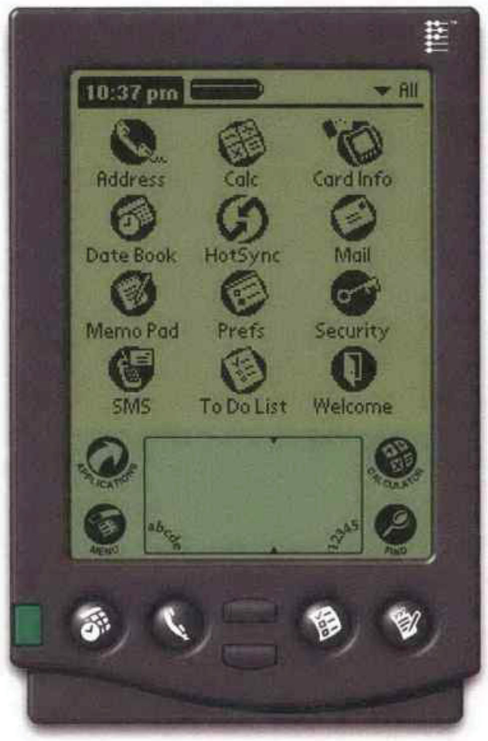


FIGURE 2-5 / The PalmOS applications screen is a simple linear list of tools.

PHYSICAL STRUCTURE

Once you've come up with the beginnings of a design, you have to translate it into a physical structure of windows, pages, and controls. That's one of the first aspects of an application that people perceive, especially on the desktop, which can host all the types of window arrangements described here.

I've heard this debate many times before: should an application use multiple windows, a single window with several tiled panes, or one window whose content "swaps out" like a web page? Should it use some combination thereof? See Figure 2-6.

You may already know by now which to use—the technology you're using often will set your course. Handhelds, cell phones, and most other consumer electronics simply don't give you the option for multiple windows or multiple panes. Even if you could, it's a bad idea, simply because users will find it too hard to navigate without a mouse.

Desktop software and large-screen web applications give you more choices. There aren't any hard-and-fast rules for determining what's best for any given design, but the sections that follow provide some guidelines. Before you decide, analyze the kinds of tasks your users will perform—especially whether they need to work in two or more UI areas at the same time. Do they need to refer to panel A while editing something in panel B? Do they need to compare A and B side-by-side? Do they need to keep panel A in view at all times to monitor something? Let your understanding of users' tasks drive your decisions.

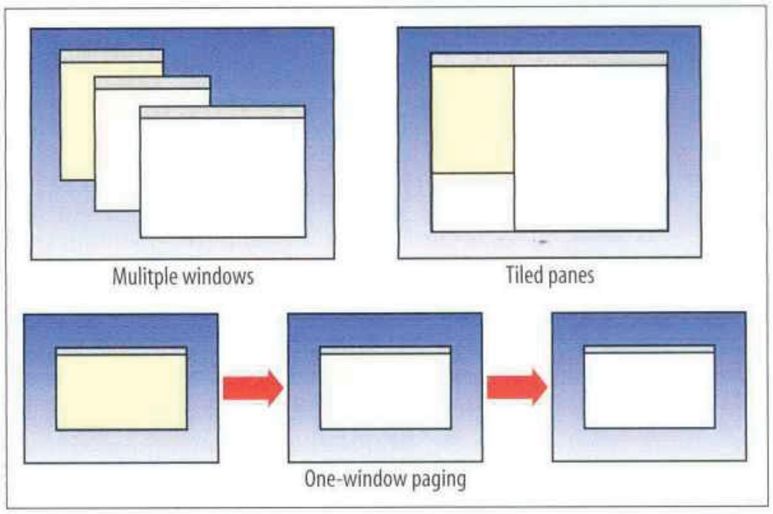


FIGURE 2-6 / Three different physical structures

MULTIPLE WINDOWS

Multiple windows sometimes are the right choice, but not often. They work best in sophisticated applications when users want to customize the layout of their screen. Infrequent users, and sometimes frequent users too, may find multiple windows irritating or confusing.

Sometimes users simply lose them if there are too many of them onscreen at once. On the other hand, if users really need to see two or more windows "in parallel," you need either this or the tiled-pane model.

ONE-WINDOW PAGING

Simple web applications work best with a single-window paging model, which can show one page at a time. It's how the Web has worked from Day One, after all, and people are very familiar with that model. Also, because it conserves space so well—there's nothing else onscreen to compete with the viewed content—it's the best choice for small handhelds and cell phones. (You couldn't fit tiled or multiple windows, anyhow.) See the **One-Window Drill-down** pattern for more; it shows how to fit a hierarchical list-driven or task-centered interface into a one-window model.

TILED PANES

Many applications and web applications use tiled panes on one window. It's great for users who want to see a lot at once while expending little effort on window management. Countless windows and dialog boxes are designed with a two-pane structure, and three is becoming more common, thanks to the prevalence of Outlook and similar applications. People intuitively grasp the idea that you "click in one pane to see something in the other."

Tiled panes can take up a lot of screen space, however. I've sometimes had to switch to a multiple window approach when the number of panes got too high, and users just couldn't fit enough of them in the window at once.

The first pattern in this chapter, **Two-Panel Selector**, describes one situation that depends upon tiled panes for its effectiveness. You can structure **Canvas Plus Palette** with them, too. Some web sites arrange small modules of interactive content onto otherwise ordinary pages; individually, these modules might behave like single-pane windows, but on the page, they're tiled.

The tiled and multiple-windows approaches together constitute the "open floor plan" idea mentioned in Chapter 1, in the discussion of focused versus open interfaces. Layouts that use tiled or multiple windows provide access to several things at once, and users take responsibility for focusing their attention on the various panels or windows at the right times. Sometimes technology prevents you from using tiled or multiple windows; at other times you ought to choose not to indulge in them, but instead use a single window to lead the user through the interface along carefully predesigned paths.

So it's all about tradeoffs. Of course, this is always true. In the end, what matters is whether or not your users—novices, intermediates, and experts—can use what you build and enjoy it. Play with the design. Draw pictures and build prototypes. Try them out on yourself, your colleagues, and, most importantly, the users themselves.

THE PATTERNS

This chapter's patterns cover both of the approaches to application design just discussed. Some of them mix content structure with physical structure. They illustrate combinations that are known to work exceedingly well, such as the first four patterns:

13 Two-Panel Selector

14 Canvas Plus Palette

15 One-Window Drilldown

16 Alternative Views

The next few patterns don't go much into physical presentation, but instead deal with content in the abstract. **Wizard** talks about "linearizing" a path through a task; it can be implemented as any number of physical presentations. **Extras on Demand** and **Intriguing Branches** describe additional ways to divide up content.

17 Wizard

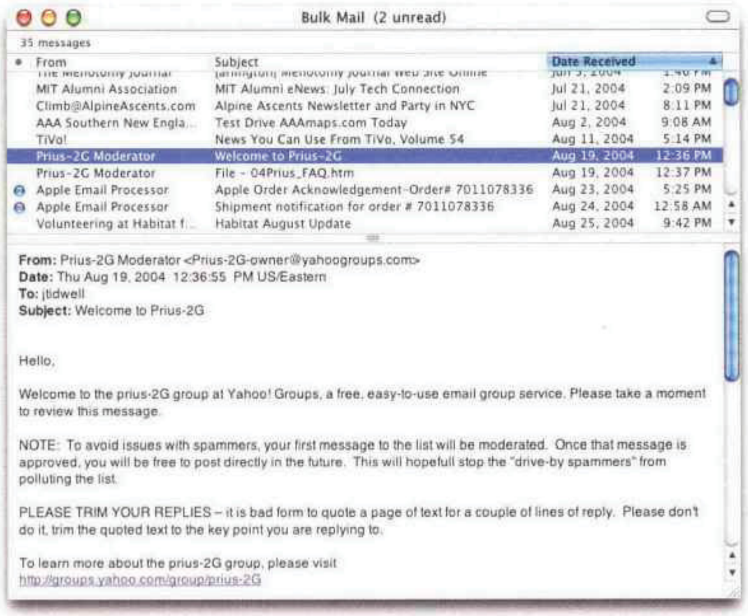
18 Extras on Demand

19 Intriguing Branches

Many patterns, here and elsewhere in the book, contribute in varying degrees to the learnability of an interface. **Multi-Level Help** sets out ways to integrate help directly into the application, thus supporting learnability for a broad number of users and situations.

20 Multi-Level Help

13 two-panel selector



Bulk Mail (2 unread)

35 messages

●	From	Subject	Date Received	
	MIT Alumni Association	MIT Alumni eNews: July Tech Connection	Jul 21, 2004	2:09 PM
	Climb@AlpineAscents.com	Alpine Ascents Newsletter and Party in NYC	Jul 21, 2004	8:11 PM
	AAA Southern New Engla...	Test Drive AAAmaps.com Today	Aug 2, 2004	9:08 AM
	TiVo!	News You Can Use From TiVo, Volume 54	Aug 11, 2004	5:14 PM
	Prius-2G Moderator	Welcome to Prius-2G	Aug 19, 2004	12:36 PM
	Prius-2G Moderator	File - 04Prius_FAQ.htm	Aug 19, 2004	12:37 PM
●	Apple Email Processor	Apple Order Acknowledgement-Order# 7011078336	Aug 23, 2004	5:25 PM
●	Apple Email Processor	Shipment notification for order # 7011078336	Aug 24, 2004	12:58 AM
	Volunteering at Habitat f...	Habitat August Update	Aug 25, 2004	9:42 PM

From: Prius-2G Moderator <Prius-2G-owner@yahoogroups.com>
Date: Thu Aug 19, 2004 12:36:55 PM US/Eastern
To: jtidwell
Subject: Welcome to Prius-2G

Hello,

Welcome to the prius-2G group at Yahoo! Groups, a free, easy-to-use email group service. Please take a moment to review this message.

NOTE: To avoid issues with spammers, your first message to the list will be moderated. Once that message is approved, you will be free to post directly in the future. This will hopefull stop the "drive-by spammers" from polluting the list.

PLEASE TRIM YOUR REPLIES – it is bad form to quote a page of text for a couple of lines of reply. Please don't do it, trim the quoted text to the key point you are replying to.

To learn more about the prius-2G group, please visit
http://groups.yahoo.com/group/prius-2G

FIGURE 2-7 / Mac Mail

what

Put two side-by-side panels on the interface. In the first, show a set of items that the user can select at will; in the other, show the content of the selected item.

use when

You're presenting a list of objects, categories, or even actions. Messages in a mailbox, sections of a web site, songs or images in a library, database records, files—all are good candidates. Each item has interesting content associated with it, such as the text of an email message or details about a file's size or date. You want the user to see the overall structure of the list, but you also want the user to walk through the items at his own pace, in an order of his choosing.

Physically, the display you work with is large enough to show two separate panels at once. Very small cell phone displays cannot cope with this pattern, but a screen such as the Blackberry's can.

why

The Two-Panel Selector is a learned convention, but an extremely common and powerful one. People quickly learn that they're supposed to select an item in one panel to see its contents in the other. They might learn it from their email clients, from Windows Explorer, or from web sites; whatever the case, they apply the concept to other applications that look similar.

When both panels are visible side-by-side, users can quickly shift their attention back and forth, looking now at the overall structure of the list ("How many more unread email messages do I have?"), and now at an object's details ("What does this email say?"). This tight integration has several advantages over other physical structures, such as two separate windows or **One-Window Drilldown**:

- It reduces physical effort. The user's eyes don't have to travel a long distance between the panels, and he can change the selection with a single mouse click or key press, rather than first navigating between windows or screens (which can take an extra mouse click).

- It reduces visual cognitive load. When a window pops to the top, or when a page's contents are completely changed (as happens with **One-Window Drilldown**), the user suddenly has to pay more attention to what he's now looking at; when the window stays mostly stable, as in a Two-Panel Selector, the user can focus on the smaller area that did change.
- It reduces the user's memory burden. Think about the email example again: when the user looks at just the text of an email message, there's nothing onscreen to remind him where that message is in the context of his inbox. If he wants to know, he has to remember or navigate back to the list. But if the list is already onscreen, he merely has to look, not remember. The list thus serves as a "You are here" signpost (see Chapter 3 for an explanation of signposts).

how

Place the selectable list on the top or left panel, and the details panel below it or to its right. This takes advantage of the visual flow that most users who read left-to-right languages expect. (Try reversing it for right-to-left language speakers.)

When the user selects an item, immediately show its contents or details in the second panel. Selection should be done with a single click. But while you're at it, give the user a way to change selection from the keyboard, particularly with the arrow keys—this reduces both the physical effort and the time required for browsing, and contributes to keyboard-only usability (see **Keyboard Only**, in Chapter 1).

Make the selected item visually obvious. Most GUI toolkits have a particular way of showing selection, e.g., reversing the foreground and background of the selected list item. If that doesn't look good, or if you're not using a GUI toolkit with this feature, try to make the selected item a different color and brightness than the unselected ones—that helps it stand out.

What should the selectable list look like? It depends—on the inherent structure of the content, or perhaps on the task to be done. For instance, most filesystem viewers show the directory hierarchy, since that's how filesystems are structured. Animation and video-editing software use interactive timelines. A GUI builder simply may use the layout canvas itself; selected objects on it then show their properties in a **Property Sheet** (Chapter 4) next to the canvas.

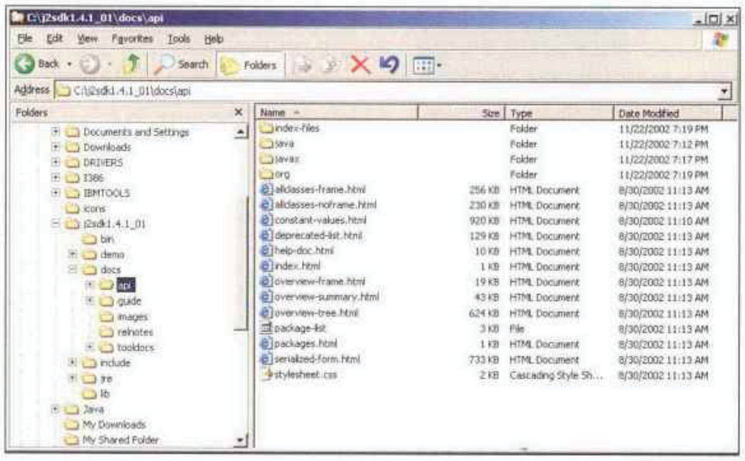
Consider using one of the models described in the "List of objects" section of this chapter's introduction:

- Linear, usually sorted
- 2D tables, also sorted, which often let the user sort via column headers or filter according to various criteria
- A hierarchy that groups items into categories (and possibly subcategories)
- A hierarchy that reveals relationships: parent/child, containers, etc.
- Spatial organizations, such as maps, charts, or desktop-like areas in which users can place things where they want

You also can use information-presentation patterns such as **Sortable Table** and **Tree-Table** (both found in Chapter 6) in Two-Panel Selectors, along with simpler components such as lists and trees. **Card Stack** (Chapter 4) closely relates to Two-Panel Selector; so does **Overview Plus Detail** (Chapter 6).

When the select-and-show concept extends through multiple panels, to facilitate navigation through a hierarchical information architecture, you get the **Cascading Lists** pattern (also Chapter 6).

examples

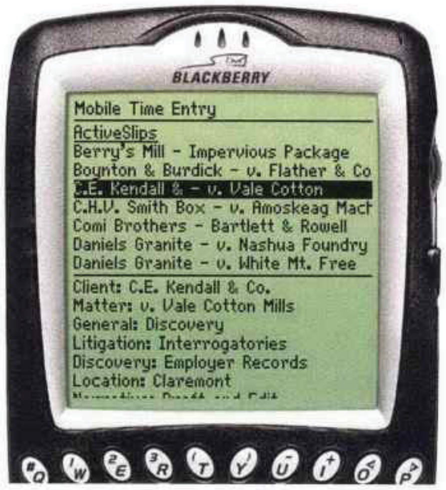

C:\j2sdk1.4.1_01\docs\api

Address C:\j2sdk1.4.1_01\docs\api

Folders: Documents and Settings, Downloads, DRIVERS, I386, IBMTOOLS, icons, j2sdk1.4.1_01, bin, demo, docs, api, guide, images, relnotes, tooldocs, include, jre, lib, Java, My Downloads, My Shared Folder

Name	Size	Type	Date Modified
index-files		Folder	11/22/2002 7:19 PM
java		Folder	11/22/2002 7:12 PM
javax		Folder	11/22/2002 7:17 PM
org		Folder	11/22/2002 7:19 PM
allclasses-frame.html	256 KB	HTML Document	8/30/2002 11:13 AM
allclasses-noframe.html	230 KB	HTML Document	8/30/2002 11:13 AM
constant-values.html	920 KB	HTML Document	8/30/2002 11:10 AM
deprecated-list.html	129 KB	HTML Document	8/30/2002 11:13 AM
help-doc.html	10 KB	HTML Document	8/30/2002 11:13 AM
index.html	1 KB	HTML Document	8/30/2002 11:13 AM
overview-frame.html	19 KB	HTML Document	8/30/2002 11:13 AM
overview-summary.html	43 KB	HTML Document	8/30/2002 11:13 AM
overview-tree.html	624 KB	HTML Document	8/30/2002 11:13 AM
package-list	3 KB	File	8/30/2002 11:13 AM
packages.html	1 KB	HTML Document	8/30/2002 11:13 AM
serialized-form.html	733 KB	HTML Document	8/30/2002 11:13 AM
stylesheet.css	2 KB	Cascading Style Sh...	8/30/2002 11:13 AM

FIGURE 2-8 / The Windows Explorer is probably one of the most familiar uses of Two-Panel Selector. Its content is organized hierarchically, using a selectable tree; in contrast, the Mac Mail example (Figure 2-7) uses a selectable table, which has a strictly linear organization. In both UIs, the dark backgrounds indicate the selected item.


Mobile Time Entry

ActiveSlips
Berry's Mill - Impervious Package
Boynton & Burdick - v. Flather & Co
C.E. Kendall & - v. Vale Cotton
C.H.V. Smith Box - v. Amoskeag Mach
Comi Brothers - Bartlett & Rowell
Daniels Granite - v. Nashua Foundry
Daniels Granite - v. White Mt. Free

Client: C.E. Kendall & Co.
Matter: v. Vale Cotton Mills
General: Discovery
Litigation: Interrogatories
Discovery: Employer Records
Location: Claremont

FIGURE 2-9 / Nortel's Mobile Time Entry application is a rare example of Two-Panel Selector use in a handheld device. The Blackberry screen offers just enough space for two usefully sized panes; when you select an item in the top pane, its contents appear on the bottom pane. (Both are scrollable with the Blackberry's scroll wheel, barely visible on the right side.)

In practice, this interface was quite effective. The lawyers who used this time-billing application could easily find items that they wanted—the two views together give enough context, but also enough details, to identify items quickly and accurately.

14 canvas plus palette

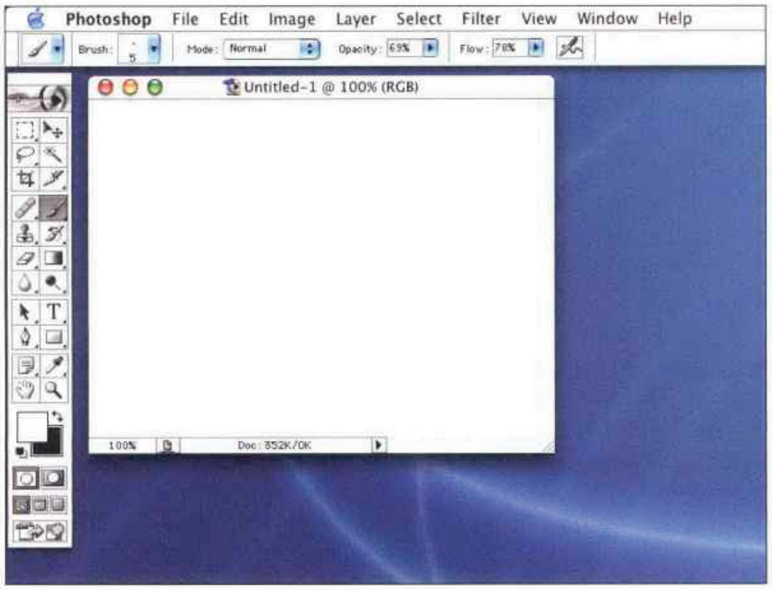


FIGURE 2-10 / Photoshop

what

Place an iconic palette next to a blank canvas; the user clicks on the palette buttons to create objects on the canvas.

use when

You're designing any kind of graphical editor. A typical use case involves creating new objects and arranging them on some virtual space.

why

This pair of panels—a palette with which to create things, and a canvas on which to put them—is so common that almost every user of desktop software has seen it. It's a natural mapping from familiar physical objects to the virtual onscreen world. And the palette takes advantage of visual recognition: the most common icons (paintbrush, hand, magnifying glass, etc.) are reused over and over again in different applications, with the same meaning each time.

how

Present a large, empty area to the user as a canvas. It might be in its own window, as in Photoshop (in Figure 2-10), in a tiled panel, or embedded in a single page with other tools. It works no matter what physical structure you've chosen, as long as you can see the canvas side-by-side with the palette.

The palette itself should be a grid of iconic buttons or button-like areas. They can have text in them if the icons are too cryptic; some GUI-builder palettes list the names of GUI components alongside their icons. So does Visio, with its palettes of complex visual constructs tailored for specific domains. But the presence of icons appears necessary for users to recognize the palette for what it is.

Place the palette to the left or top of the canvas. (It's likely that speakers of right-to-left languages might prefer it to the right of the canvas, not the left; usability-test it if this is an issue for you.) It can be divided into subgroups, and you may want to use a **Card Stack**, such as tabs, to present those subgroups.

Most palette buttons should create the pictured object on the canvas. But some builders have successfully integrated other things, like zoom mode and lassoing, into the palette. This started early; MacPaint mixed its modes into its palette (see Figure 2-12), and people have learned what the arrow, hand, and other icons do. But be careful. I recommend not mixing other actions into a creational palette—it can be very confusing for users.

The gestures used to create items on a palette vary from one application to another. Some use drag-and-drop only; some use a single click on the palette and single click on the canvas; and some use **One-off Modes** (see Chapter 8), **Spring-Loaded Modes**, and other carefully designed gestures. I have always found that usability testing in this area is particularly important, since users' expectations vary greatly.

examples

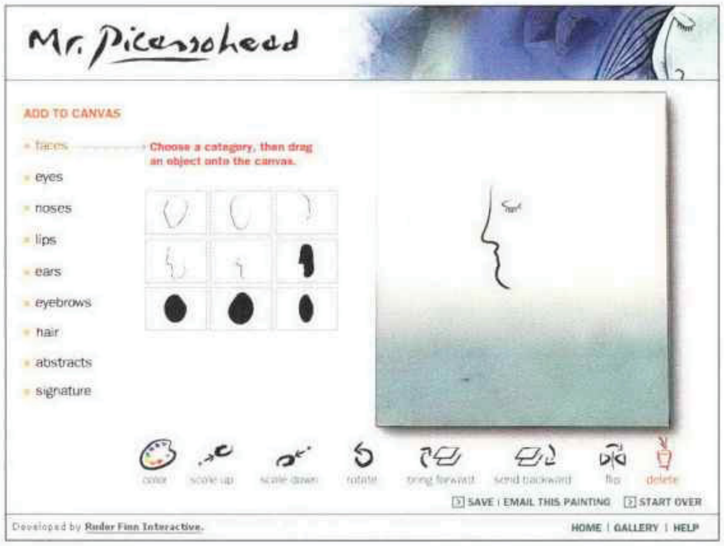


FIGURE 2-11 / You don't need all the trappings of a document-centered desktop application to make Canvas Plus Palette work. This web application, Mr. Picassohead, is a whimsical twist on this familiar pattern. The palette itself is merely a grid of icons, and it doesn't look like a set of buttons at all; the palette is subdivided and "tabbed" by category (a use of Two-Panel Selector). When you click on the words "eyes," "noses," or "lips," the palette changes to show those objects. The canvas itself is neutral, but not white. Its purpose is clear to the first-time user simply because it's a big open space, framed by a border. See *http://mrpicassohead.com*.

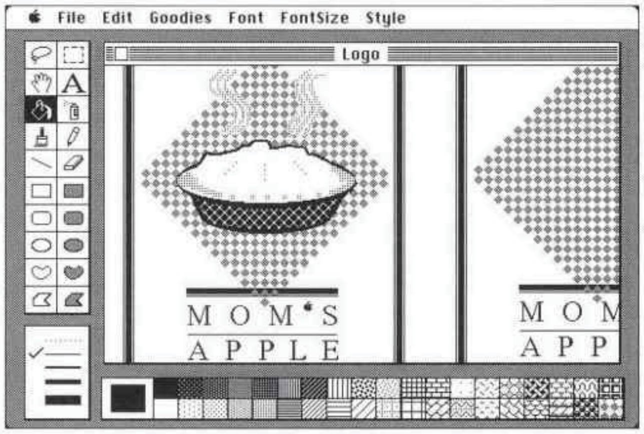
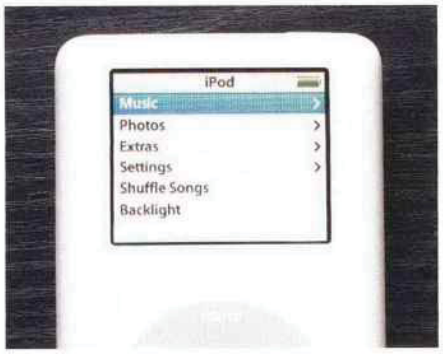


FIGURE 2-12 / Taking a trip back in time, let's look at one of the interfaces that popularized this pattern: MacPaint. The pattern hasn't changed much since 1984—the basic elements are all there, in the same spatial configuration used by contemporary software such as Mr. Picassohead and Photoshop. Photoshop and other visual builders, in fact, still use many of MacPaint's icons over 20 years later. The screenshot is from *http://mac512.com*.

15 one-window drilldown



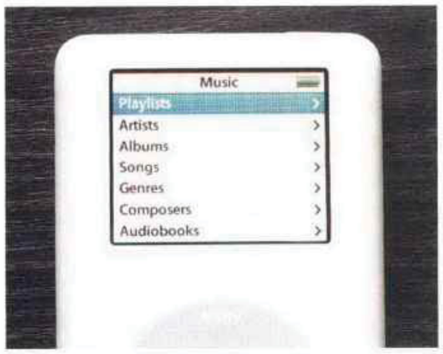


FIGURE 2-13 / Two iPod menus

what

Show each of the application's pages within a single window. As a user drills down through a menu of options, or into an object's details, replace the window contents completely with the new page.

use when

Your application consists of many pages or panels of content for the user to navigate through. They might be arranged linearly, in an arbitrary hyperlinked network, or—most commonly—in a menu hierarchy. Address books, calendars, web-based email readers, and other familiar applications often use this pattern.

One or both of these constraints might apply to you:

- You are building for a device with tight space restrictions, such as a handheld (see Figure 2-13), a cell phone, or a TV. On these miniature screens, **Two-Panel Selector**—and tiled panes in general—are impractical because there just isn't enough room to use them well. Traversing from one panel to another on a TV screen also is difficult, since TVs have no mice.
- Even if you build for a desktop or laptop screen, you may have a complexity limit. Your users may not be habitual computer users—having many application windows open at once may confuse them, or they may not deal well with complex screens or fiddly input devices. Users of information kiosks fall into this category, as do novice PC users.

why

Keep it simple. When everything's on one screen or window, the options at each stage are clear, and users know they don't need to focus their attention anywhere else.

Besides, everyone these days knows how to use a web browser, with its single window and simple back/forward page model. People expect that when they click on a link or button, the page they're looking at will be replaced, and when they click "Back," they'll go back to where they were before.

You could use multiple windows to show the different spaces that a user navigates through—a click in a main window may bring up a second window, a click in that window brings up a third, etc. But that can be confusing. Even sophisticated users can easily lose track of where their windows are (though they can see several windows side-by-side and place them where they want).

One-Window Drilldown is an alternative to several of the higher-density patterns and techniques discussed here. As pointed out earlier, **Two-Panel Selector** may not fit, or it may make the screen layout or interactions too complex for the typical user. Tiled windows, **Closable Panels**, **Movable Pieces**, and **Cascading Lists** (the last three are patterns in Chapter 4) also have space and complexity issues. They don't work on miniature screens, and they complicate interfaces intended for novice computer users.

how

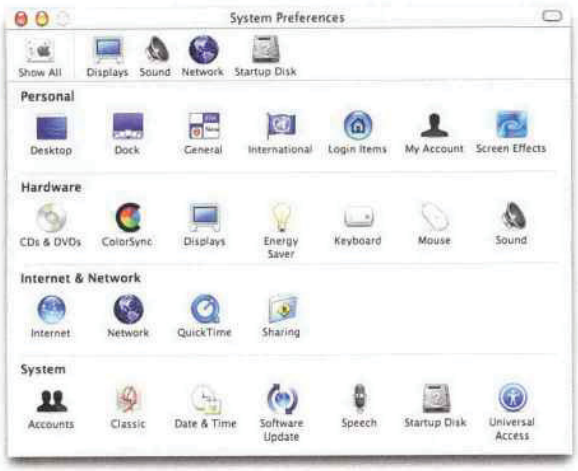
You are given one window to work with—a miniature screen, a full-sized screen, a browser window, or an application window that lives on the desktop alongside other applications. Structure your content into panels that fit gracefully into that window: not too large and not too small.

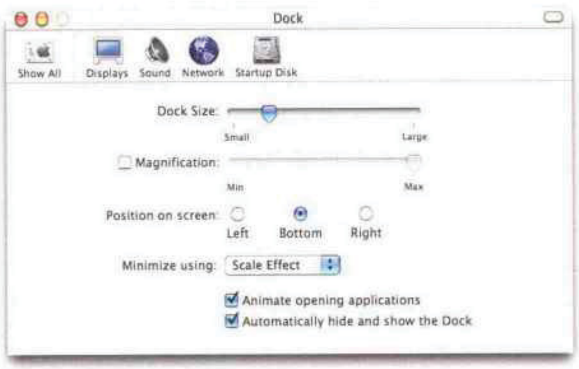
On these panels, design obvious "doors" into other UI spaces, such as underlined links, buttons, or clickable table rows. When the user clicks on one of these, replace the current panel with the new one. Thus the user "drills down" deeper into the content of the application (see Figure 2-14).

How does the user go back? If you're designing for a device with back/forward buttons, that's one solution. If not, create those buttons and put them in one permanent place on the window—usually the upper left, where browsers put them. You should put "Done" or "Cancel" controls on panels where the user completes a task or selection; these controls give the user a sense of closure, of "being done."

Remember that with no graphic depiction of the application's structure, nor of where they are in that structure, a one-window application forces the user to rely on a mental picture of how all these spaces fit together. Simple linear or hierarchical structures work best (see Figure 2-15). In usability tests, make sure people can use the thing without getting lost! **Breadcrumbs** and **Sequence Maps** can help if they do; see Chapter 3, *Navigation*.

Implementations of **Hub and Spoke** often use One-Window Drilldown, especially on the Web and miniature screens. Again, see Chapter 3.

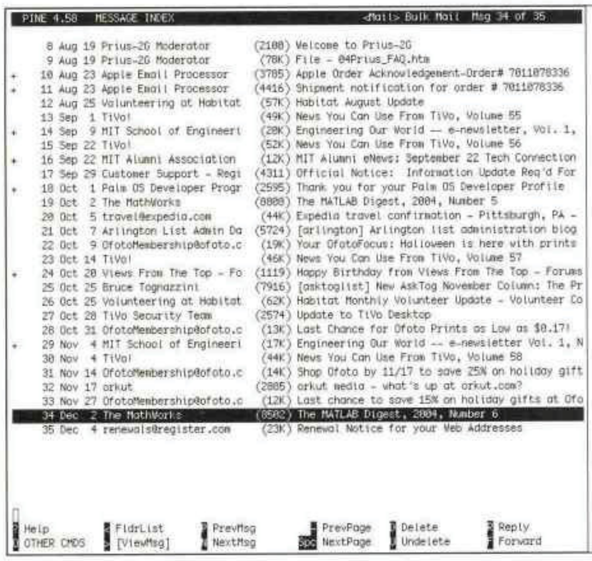




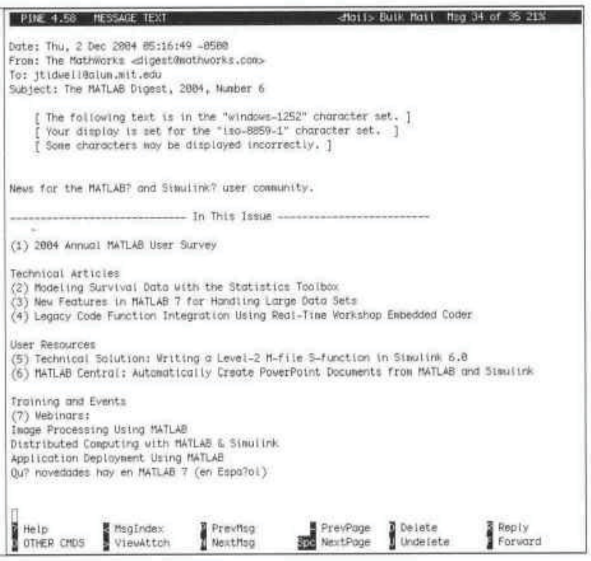
Dock Size:
Magnification:
Position on screen: Left [✓ Bottom] Right
Minimize using: Scale Effect
[✓] Animate opening applications
[✓] Automatically hide and show the Dock

FIGURE 2-14 / The Mac OS X System Preferences utility keeps everything within one window. The main panel is on the left; a drilldown panel (for the Dock) is shown on the right. There's only one level of drilldown panels. The user goes back to the main panel via the "Show All" button in the upper left.

Mac screens often are large, and Mac OS X users are varied in their levels of experience. The System Preferences designers may have chosen a One-Window Drilldown approach not because of small screens, but because of the sheer number and diversity of subpanels. That main panel takes up a lot of space, and it probably didn't work well in a tiled **Two-Panel Selector** window.



```
 PINE 4.58   MESSAGE INDEX                    <Mail> Bulk Mail  Msg 34 of 35

     8 Aug 19 Prius-2G Moderator       (2108) Welcome to Prius-2G
     9 Aug 19 Prius-2G Moderator        (78K) File - 04Prius_FAQ.htm
 +  10 Aug 23 Apple Email Processor    (3785) Apple Order Acknowledgement-Order# 7011078336
 +  11 Aug 23 Apple Email Processor    (4416) Shipment notification for order # 7011078336
    12 Aug 25 Volunteering at Habitat   (57K) Habitat August Update
    13 Sep  1 TiVo!                     (49K) News You Can Use From TiVo, Volume 55
 +  14 Sep  9 MIT School of Engineeri   (20K) Engineering Our World — e-newsletter, Vol. 1,
    15 Sep 22 TiVo!                     (52K) News You Can Use From TiVo, Volume 56
 +  16 Sep 22 MIT Alumni Association    (12K) MIT Alumni eNews: September 22 Tech Connection
    17 Sep 29 Customer Support - Regi  (4311) Official Notice:  Information Update Req'd For
 +  18 Oct  1 Palm OS Developer Progr  (2595) Thank you for your Palm OS Developer Profile
    19 Oct  2 The MathWorks            (8008) The MATLAB Digest, 2004, Number 5
    20 Oct  5 travel@expedia.com        (44K) Expedia travel confirmation - Pittsburgh, PA -
    21 Oct  7 Arlington List Admin Da  (5724) [arlington] Arlington list administration blog
    22 Oct  9 OfotoMembership@ofoto.c   (19K) Your OfotoFocus: Halloween is here with prints
    23 Oct 14 TiVo!                     (46K) News You Can Use From TiVo, Volume 57
 +  24 Oct 20 Views From The Top - Fo  (1119) Happy Birthday from Views From The Top - Forums
    25 Oct 25 Bruce Tognazzini         (7916) [asktoglist] New AskTog November Column: The Pr
    26 Oct 25 Volunteering at Habitat   (62K) Habitat Monthly Volunteer Update - Volunteer Co
    27 Oct 28 TiVo Security Team       (2574) Update to TiVo Desktop
    28 Oct 31 OfotoMembership@ofoto.c   (13K) Last Chance for Ofoto Prints as Low as $0.17!
 +  29 Nov  4 MIT School of Engineeri   (17K) Engineering Our World -- e-newsletter Vol. 1, N
    30 Nov  4 TiVo!                     (44K) News You Can Use From TiVo, Volume 58
    31 Nov 14 OfotoMembership@ofoto.c   (14K) Shop Ofoto by 11/17 to save 25% on holiday gift
    32 Nov 17 orkut                    (2005) orkut media - what's up at orkut.com?
    33 Nov 27 OfotoMembership@ofoto.c   (12K) Last chance to save 15% on holiday gifts at Ofo
    34 Dec  2 The MathWorks            (8502) The MATLAB Digest, 2004, Number 6
    35 Dec  4 renewals@register.com     (23K) Renewal Notice for your Web Addresses



 Help        FldrList     PrevMsg      PrevPage  Delete     Reply
 OTHER CMDS  [ViewMsg]    NextMsg  Spc NextPage  Undelete   Forward
```

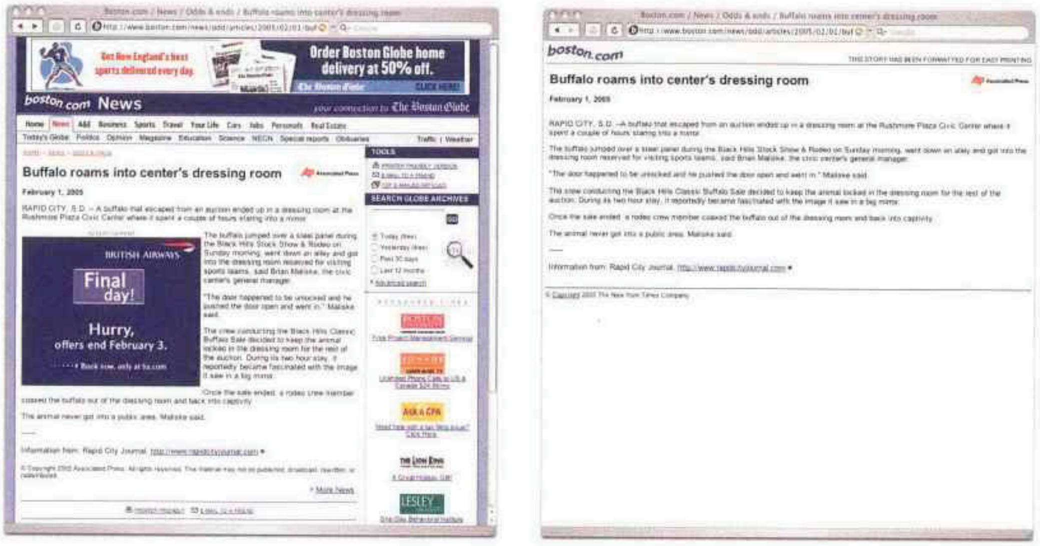


```
 PINE 4.58   MESSAGE TEXT                   <Mail> Bulk Mail  Msg 34 of 35 21%

Date: Thu, 2 Dec 2004 05:16:49 -0500
From: The MathWorks <digest@mathworks.com>
To: jtidwell@alum.mit.edu
Subject: The MATLAB Digest, 2004, Number 6

    [ The following text is in the "windows-1252" character set. ]
    [ Your display is set for the "iso-8859-1" character set.  ]
    [ Some characters may be displayed incorrectly. ]


News for the MATLAB? and Simulink? user community.

----------------------------- In This Issue -------------------------

(1) 2004 Annual MATLAB User Survey

Technical Articles
(2) Modeling Survival Data with the Statistics Toolbox
(3) New Features in MATLAB 7 for Handling Large Data Sets
(4) Legacy Code Function Integration Using Real-Time Workshop Embedded Coder

User Resources
(5) Technical Solution: Writing a Level-2 M-file S-function in Simulink 6.0
(6) MATLAB Central: Automatically Create PowerPoint Documents from MATLAB and Simulink

Training and Events
(7) Webinars:
Image Processing Using MATLAB
Distributed Computing with MATLAB & Simulink
Application Deployment Using MATLAB
Qu? novedades hay en MATLAB 7 (en Espa?ol)


 Help        MsgIndex     PrevMsg      PrevPage  Delete     Reply
 OTHER CMDS  ViewAttch    NextMsg  Spc NextPage  Undelete   Forward
```

FIGURE 2-15 / The Pine email client is a lightweight, text-only user interface driven entirely from the keyboard. The greater-than and less-than keys navigate the application's hierarchy: the main screen, the list of mailboxes, the list of messages within the selected mailbox (left), the text of the selected message (right), and the attachments to the message.

Pine thus contains a deep hierarchy of UI "spaces," but it works well within one window. Compare this screenshot to that of Mac Mail in the **Two-Panel Selector** pattern—both are good interfaces, but they have very different designs. We might guess that two stringent requirements drove Pine's One-Window Drilldown design: to run in a text-based terminal window, and to be operated with only the keyboard.

16 alternative views



Boston.com / News / Odds & ends / Buffalo roams into center's dressing room

http://www.boston.com/news/odd/articles/2005/02/01/buf

Get New England's best sports delivered every day.

Order Boston Globe home delivery at 50% off.

boston.com News

Home News A&E Business Sports Travel Your Life Cars Jobs Personals Real Estate

Today's Globe Politics Opinion Magazine Education Science NECN Special reports Obituaries Traffic | Weather

TOOLS

SEARCH GLOBE ARCHIVES

BRITISH AIRWAYS Final day! Hurry, offers end February 3.

boston.com

THIS STORY HAS BEEN FORMATTED FOR EASY PRINTING

Buffalo roams into center's dressing room

February 1, 2005

RAPID CITY, S.D. —A buffalo that escaped from an auction ended up in a dressing room at the Rushmore Plaza Civic Center where it spent a couple of hours staring into a mirror.

The buffalo jumped over a steel panel during the Black Hills Stock Show & Rodeo on Sunday morning, went down an alley and got into the dressing room reserved for visiting sports teams, said Brian Maliske, the civic center's general manager.

"The door happened to be unlocked and he pushed the door open and went in," Maliske said.

The crew conducting the Black Hills Classic Buffalo Sale decided to keep the animal locked in the dressing room for the rest of the auction. During its two hour stay, it reportedly became fascinated with the image it saw in a big mirror.

Once the sale ended, a rodeo crew member coaxed the buffalo out of the dressing room and back into captivity.

The animal never got into a public area, Maliske said.

—

Information from: Rapid City Journal, http://www.rapidcityjournal.com

© Copyright 2005 The New York Times Company

FIGURE 2-16 / A boston.com article in standard format and in printer-friendly format

what

Let the user choose among alternative views that are structurally different, not just cosmetically different, from the default view.

use when

You're building a web page, an editor, a map application, or anything that displays formatted content of any kind. People will use it under many conditions. Maybe you already provide some customizability—font sizes, languages, sort order, zoom level, etc.—but those lightweight changes don't go far enough to accommodate all the things people typically do with it.

why

Try as you might, you can't always accommodate all possible usage scenarios in a single design. For instance, printing is typically problematic for applications because the information display requirements differ—navigation and interactive gizmos should be removed, for instance, and the remaining content reformatted to fit the printer paper. See the news article example above (Figure 2-16) for what that can look like.

Beyond different usage scenarios, you should use Alternative Views for several other reasons:

- Different technologies at the user's end—one person might view the application on a desktop, but another person would view it on a PDA, and a third would use a screen reader such as JAWS.
- Users' preferences with regard to speed, visual style, and other factors.
- A need to temporarily view something differently, in order to gain insight.

how

Choose a few usage scenarios that the application's (or page's) normal mode of operation cannot easily serve. Design specialized views for them, and present those views as alternatives within the same window or screen.

In these alternative views, some information might be added, and some might be taken away, but the primary content should remain more or less the same. If you need to strip down the interface—for use by a printer or screen reader, for instance—then consider removing secondary content, shrinking or eliminating images, and cutting out all navigation but the most basic.

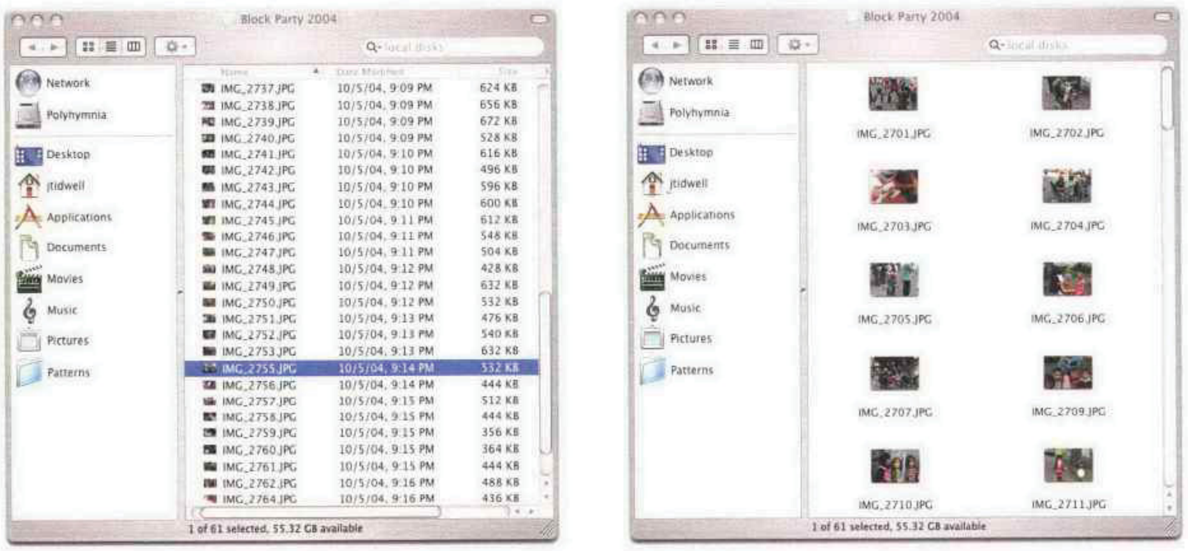
Stuffing content into a PDA or cell phone screen is trickier; it could force you to redesign the navigation itself. Rather than showing a lot of content on one screen or page, as you could with a desktop computer or TV, you might split it up into multiple pages, each of which fits gracefully onto a smaller screen.

Put a "switch" for the mode somewhere on the main interface. It doesn't have to be prominent; Powerpoint and Word (as you'll see in Figure 2-19) put their mode buttons in the lower-left corner, which is an easily overlooked spot on any interface. Most applications represent the alternative views with iconic buttons. Make sure it's easy to switch back to the default view, too. As the user switches back and forth, preserve all of the application's current state—selections, the user's location in the document, uncommitted changes, Undo/Redo operations, etc. Losing them will surprise the user.

Applications that "remember" their users often retain the user's alternative-view choice from one use to the next. In other words, if a user decides to switch to an alternative view, the application will just use that view by default next time. Web sites can do this with cookies; desktop applications can keep track of preferences per user; an app on a digital device like a cell phone can simply remember what view it used the last time it was invoked.

Web pages may have the option of implementing Alternative Views as alternative CSS pages. This is how some sites cope with print-suitable views, for example.

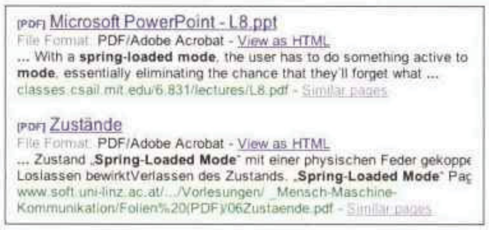
examples



Name	Date Modified	Size
IMG_2737.JPG	10/5/04, 9:09 PM	624 KB
IMG_2738.JPG	10/5/04, 9:09 PM	656 KB
IMG_2739.JPG	10/5/04, 9:09 PM	672 KB
IMG_2740.JPG	10/5/04, 9:09 PM	528 KB
IMG_2741.JPG	10/5/04, 9:10 PM	616 KB
IMG_2742.JPG	10/5/04, 9:10 PM	496 KB
IMG_2743.JPG	10/5/04, 9:10 PM	596 KB
IMG_2744.JPG	10/5/04, 9:10 PM	600 KB
IMG_2745.JPG	10/5/04, 9:11 PM	612 KB
IMG_2746.JPG	10/5/04, 9:11 PM	548 KB
IMG_2747.JPG	10/5/04, 9:11 PM	504 KB
IMG_2748.JPG	10/5/04, 9:12 PM	428 KB
IMG_2749.JPG	10/5/04, 9:12 PM	632 KB
IMG_2750.JPG	10/5/04, 9:12 PM	532 KB
IMG_2751.JPG	10/5/04, 9:13 PM	476 KB
IMG_2752.JPG	10/5/04, 9:13 PM	540 KB
IMG_2753.JPG	10/5/04, 9:13 PM	632 KB
IMG_2755.JPG	10/5/04, 9:14 PM	532 KB
IMG_2756.JPG	10/5/04, 9:14 PM	444 KB
IMG_2757.JPG	10/5/04, 9:15 PM	512 KB
IMG_2758.JPG	10/5/04, 9:15 PM	444 KB
IMG_2759.JPG	10/5/04, 9:15 PM	356 KB
IMG_2760.JPG	10/5/04, 9:15 PM	364 KB
IMG_2761.JPG	10/5/04, 9:15 PM	444 KB
IMG_2762.JPG	10/5/04, 9:16 PM	488 KB
IMG_2764.JPG	10/5/04, 9:16 PM	436 KB

FIGURE 2-17 / Both the Windows Explorer and the Mac Finder permit several alternative views of the files in a filesystem. This example shows two views: a sortable multicolumn list (see the **Sortable Table** pattern in Chapter 6) and a grid of icons.

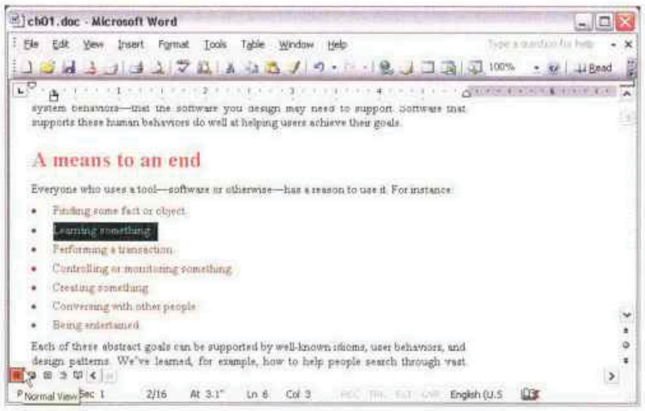
Each view has pros and cons. The table is terrific for managing lots of information—the user can find things by sorting on the columns, for instance. But the icons work better if she is looking for a particular image that she can recognize on sight. These views address different use contexts, and a user might go back and forth among them at one sitting.



[PDF] Microsoft PowerPoint - L8.ppt
File Format: PDF/Adobe Acrobat - View as HTML
... With a **spring-loaded mode**, the user has to do something active to **mode**, essentially eliminating the chance that they'll forget what ...
classes.csail.mit.edu/6.831/lectures/L8.pdf - Similar pages

[PDF] Zustände
File Format: PDF/Adobe Acrobat - View as HTML
... Zustand „**Spring-Loaded Mode**" mit einer physischen Feder gekoppe
Loslassen bewirktVerlassen des Zustands. „**Spring-Loaded Mode**" Pag
www.soft.uni-linz.ac.at/.../Vorlesungen/_Mensch-Maschine-Kommunikation/Folien%20(PDF)/06Zustaende.pdf - Similar pages

FIGURE 2-18 / Google's search results can return not just ordinary HTML Web pages, but PDF, Word, and Powerpoint documents as well. What if you don't have Word or Powerpoint on your client machine? That technology problem dictates the use of an alternative view: the HTML "translation." What if you really don't want to download a large Powerpoint slideshow and just want to skim the HTML translation in a hurry? That's a user preference.



ch01.doc - Microsoft Word

system behaviors—that the software you design may need to support. Software that supports these human behaviors do well at helping users achieve their goals.

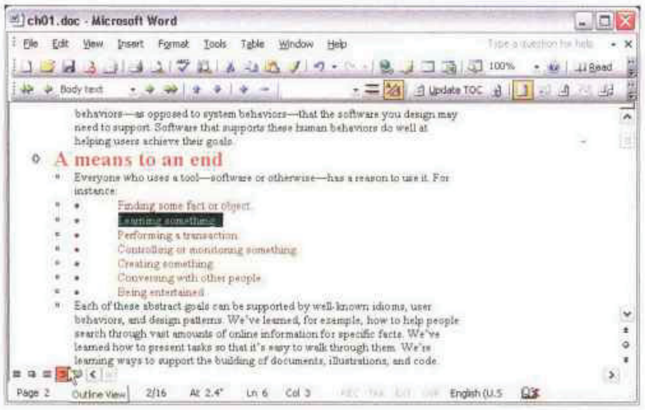
A means to an end

Everyone who uses a tool—software or otherwise—has a reason to use it. For instance:

- Finding some fact or object
- Learning something
- Performing a transaction
- Controlling or monitoring something
- Creating something
- Conversing with other people
- Being entertained

Each of these abstract goals can be supported by well-known idioms, user behaviors, and design patterns. We've learned, for example, how to help people search through vast

Normal View Sec 1 2/16 At 3.1" Ln 6 Col 3 English (U.S



ch01.doc - Microsoft Word

behaviors—as opposed to system behaviors—that the software you design may need to support. Software that supports these human behaviors do well at helping users achieve their goals.

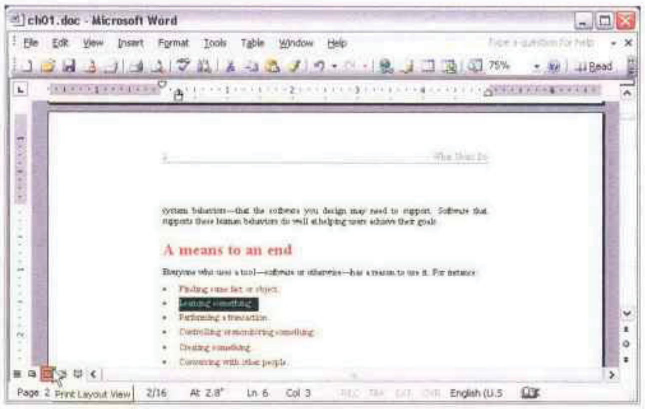
A means to an end

Everyone who uses a tool—software or otherwise—has a reason to use it. For instance:

- Finding some fact or object
- Learning something
- Performing a transaction
- Controlling or monitoring something
- Creating something
- Conversing with other people
- Being entertained

Each of these abstract goals can be supported by well-known idioms, user behaviors, and design patterns. We've learned, for example, how to help people search through vast amounts of online information for specific facts. We've learned how to present tasks so that it's easy to walk through them. We're learning ways to support the building of documents, illustrations, and code.

Page 2 Outline View 2/16 At 2.4" Ln 6 Col 3 English (U.S



ch01.doc - Microsoft Word

system behaviors—that the software you design may need to support. Software that supports these human behaviors do well at helping users achieve their goals.

A means to an end

Everyone who uses a tool—software or otherwise—has a reason to use it. For instance:

- Finding some fact or object
- Learning something
- Performing a transaction
- Controlling or monitoring something
- Creating something
- Conversing with other people

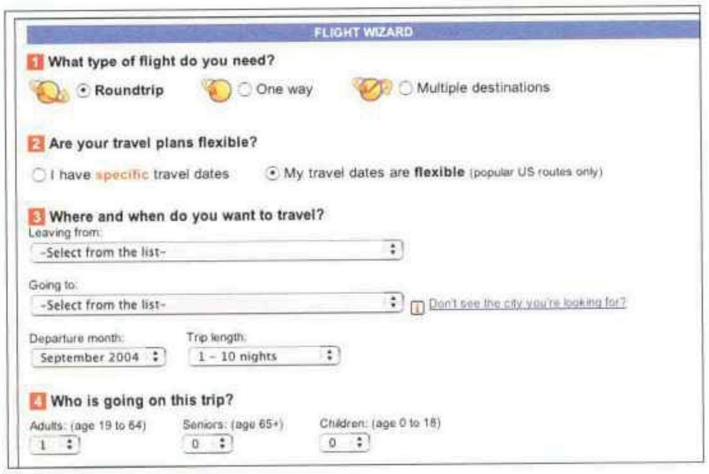
Page 2 Print Layout View 2/16 At 2.8" Ln 6 Col 3 English (U.S

FIGURE 2-19 / Of course, we have Word and Powerpoint themselves. Both full-fledged WYSIWYG editors can construct fairly complex documents—a Powerpoint presentation has a sequence, a template, perhaps notes for some slides, fancy transitions from one slide to another, and even code to support interactivity. A user who's authoring the slideshow may need to see all that. A user who's just running the slideshow doesn't. Again, these are different use contexts.

Word's views include the normal view, intended for most editing tasks; a "print layout" view, which shows how the document might appear on printed pages; a "reading" mode for uncluttered viewing; and an "outline" view, which shows the structure of the document. Someone might use the outline view to gain insight—if you loaded a large, unfamiliar document, for instance, you might switch to outline mode just to see a "table of contents" overview of it. Both applications put the Alternative Views buttons in the lower lefthand corner of the document's window.

Note that in these Word examples, the selected text remains the same as the user switches from one view to another. The position in the document also stays the same through most transitions. However, different toolbars come and go, the zoom level changes, and some kinds of content—notably footnotes and annotations—are visible only in the print layout view.

17 wizard



FLIGHT WIZARD

1 What type of flight do you need?

[✓ Roundtrip] One way Multiple destinations

2 Are your travel plans flexible?

I have specific travel dates [✓ My travel dates are **flexible**] (popular US routes only)

3 Where and when do you want to travel?

Leaving from: -Select from the list-

Going to: -Select from the list- Don't see the city you're looking for?

Departure month: September 2004 Trip length: 1 - 10 nights

4 Who is going on this trip?

Adults: (age 19 to 64) 1 Seniors: (age 65+) 0 Children: (age 0 to 18) 0

FIGURE 2-20 / Flight Wizard from *http://expedia.com*

what

Lead the user through the interface step by step, doing tasks in a prescribed order.

use when

You are designing a UI for a task that is long or complicated, and that will be novel for the user—it's not something that they do often or want much fine-grained control over. You're reasonably certain that those of you who design the UI will know more than the user does about how best to get the task done.

Tasks that seem well-suited for this approach tend to be either branched or very long and tedious—they consist of a series of user-made decisions that affect downstream choices.

The catch is that the user *must* be willing to surrender control over what happens when. In many contexts, that works out fine, since making decisions is an unwelcome burden for people doing certain things: "Don't make me think, just tell me what to do next." Think about moving through an unfamiliar airport—it's often easier to follow a series of signs than it is to figure out the airport's overall structure. You don't get to learn much about how the airport is designed, but you don't care about that.

But in other contexts, it backfires. Expert users often find Wizards frustratingly rigid and limiting. This is particularly true for software that supports creative processes—writing, art, or coding. It's also true for users who actually *do* want to learn the software; wizards don't show users what their actions really do, nor what application state is changed as choices are made. That can be infuriating to some people. Know your users well!

why

Divide and conquer. By splitting up the task into a sequence of chunks, each of which the user can deal with in a discrete "mental space," you effectively simplify the task. You have put together a preplanned road map through the task, thus sparing the user the effort of figuring out the task's structure—all they need to do is address each step in turn, trusting that if they follow the instructions, things will turn out OK.

"CHUNKING" THE TASK

Break up the operations constituting the task into a series of chunks, or groups of operations. You may need to present these groups in a strict sequence, or not; there is value in breaking up a task into Steps 1, 2, 3, and 4 just for convenience.

A thematic breakdown for an online purchase may include screens for product selection, payment information, a billing address, and a shipping address. The presentation order doesn't much matter, because later choices don't depend on earlier choices. Putting related choices together just simplifies things for people filling out those forms.

You may decide to split up the task at decision points so that choices made by the user can change the downstream steps dynamically. In a software installation wizard, for example, the user may choose to install optional packages that require yet more choices; if they choose not to do a custom installation, those steps are skipped. Dynamic UIs are good at presenting branched tasks such as this because the user never has to see what is irrelevant to the choices she made.

In either case, the hard part of designing this kind of UI is striking a balance between the sizes of the chunks and the number of them. It's silly to have a two-step wizard, and a fifteen-step wizard is tedious. On the other hand, each chunk shouldn't be overwhelmingly large, or you've lost some benefits of this pattern.

PHYSICAL STRUCTURE

Wizards that present each step in a separate page, navigated with Back and Next buttons, are the most obvious and well-known implementation of this pattern. They're not always the right choice, though, because now each step is an isolated UI space that shows no context—the user can't see what went before or what comes next. But an advantage of such wizards is that they can devote an entire page to each step, including illustrations and explanations.

If you do this, allow the user to move back and forth at will through the task sequence. There's nothing more frustrating than having to start a task over just because the software won't let you change your mind about a previous decision. Back buttons are, of course, standard equipment on separate-page wizards; use them, and make sure the underlying software supports stepping backwards. Additionally, many UIs show a selectable map or overview of all the steps, getting some of the benefits of **Two-Panel Selector**. (In contrast to that pattern, Wizard implies a prescribed order—even if it's merely suggested—as opposed to completely random access.)

If you choose to keep all the steps on one page, you could use one of several patterns from Chapter 4:

- **Titled Sections**, with prominent numbers in the titles. This is most useful for tasks that aren't heavily branched, since all steps can be visible at once.
- **Responsive Enabling**, in which all the steps are present on the page, but each remains disabled until the user has finished the previous step.
- **Responsive Disclosure**, in which you wait to show a step on the UI until the user finishes the previous one. Personally, I think this is the most elegant way to implement a short wizard. It's dynamic, compact, and easy to use.

Good Defaults (from Chapter 7) are useful no matter how you arrange the steps. If the user is willing to turn over control of the process to you, then odds are good he's also willing to let you pick reasonable defaults for choices he may not care much about, such as the location of a software installation.

(*The Design of Sites* discusses this concept under the pattern name "Process Funnel." Their pattern aims more at web sites, for tasks such as web shopping, but the concepts generalize well.)

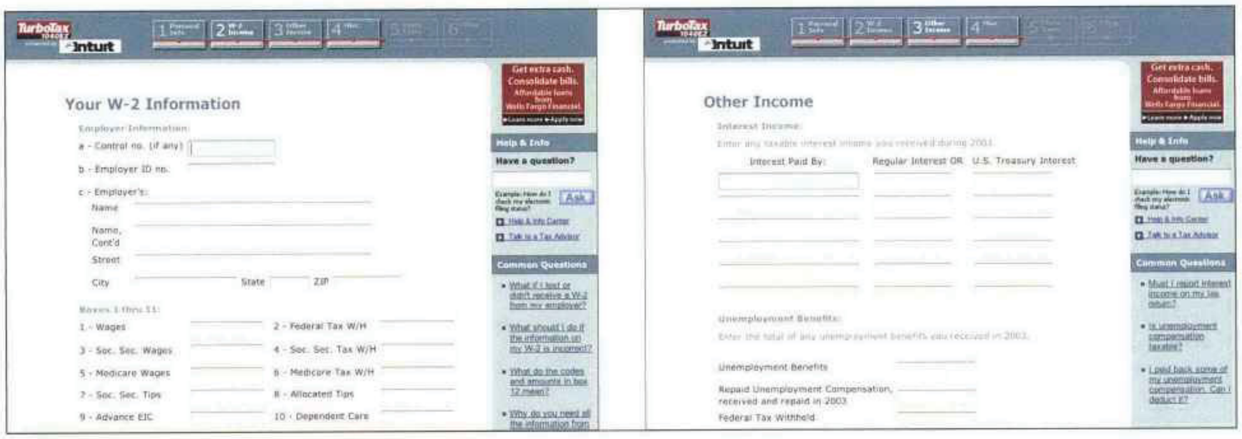
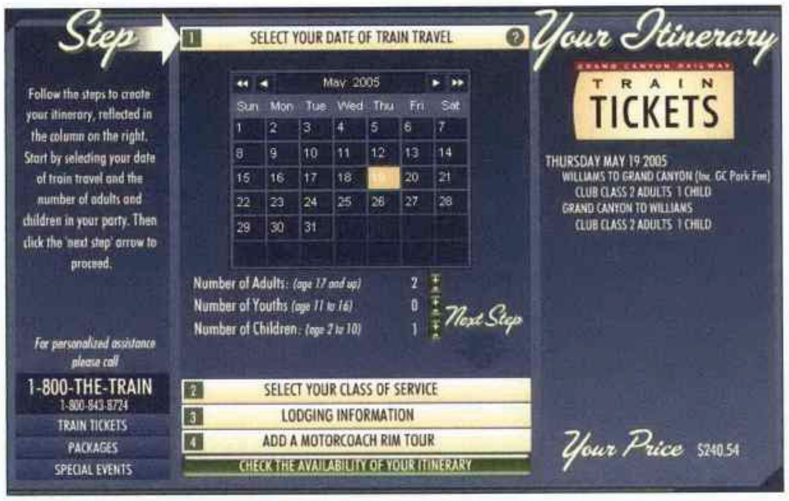


FIGURE 2-21 / TurboTax is a web application that presents several steps in a wizard-like fashion. Each major step is on a different page, and the pages have "Back" and "Continue" (or "Done") links at the ends of the pages, as traditional wizards do. (They're not visible on these screenshots, but trust me, they're there.) A **Sequence Map** at the top shows where you are in the steps at all times. **Good Defaults** generally aren't used here. That may be because sensitive information—personal financial data—is involved, but many users will find themselves entering "0" for a lot of fields.


Step
1 SELECT YOUR DATE OF TRAIN TRAVEL
Your Itinerary
May 2005
Sun Mon Tue Wed Thu Fri Sat
Number of Adults: (age 17 and up) 2
Number of Youths (age 11 to 16) 0
Number of Children: (age 2 to 10) 1
Next Step
2 SELECT YOUR CLASS OF SERVICE
3 LODGING INFORMATION
4 ADD A MOTORCOACH RIM TOUR
CHECK THE AVAILABILITY OF YOUR ITINERARY
1-800-THE-TRAIN
1-800-843-8724
TRAIN TICKETS
PACKAGES
SPECIAL EVENTS
GRAND CANYON RAILWAY
TRAIN TICKETS
Your Price $240.54

Follow the steps to create your itinerary, reflected in the column on the right. Start by selecting your date of train travel and the number of adults and children in your party. Then click the 'next step' arrow to proceed.

For personalized assistance please call

THURSDAY MAY 19 2005
WILLIAMS TO GRAND CANYON (Inc. GC Park Fee)
CLUB CLASS 2 ADULTS 1 CHILD
GRAND CANYON TO WILLIAMS
CLUB CLASS 2 ADULTS 1 CHILD

FIGURE 2-22 / The Expedia example showed a wizard structured with **Titled Sections** (Chapter 4); the TurboTax example uses a paged model, replacing the contents of the browser window with each successive step. This example uses a **Card Stack** (also Chapter 4) to fit all steps into a very limited space. When the user selects a date from this calendar and clicks "Next Step," panel 2 opens up. When that step is done, the user moves on to Panel 3. The user can go back to previous steps anytime by clicking on the yellow titlebars.

Also note the use of a **Preview** (Chapter 5) on the righthand pane of the window, entitled "Your Itinerary." This tracks the user's choices (no pun intended) and shows a summary of those choices. This is convenient because only one wizard page is visible at a time; the user can't see all their choices at once. See *http://thetrain.com*.

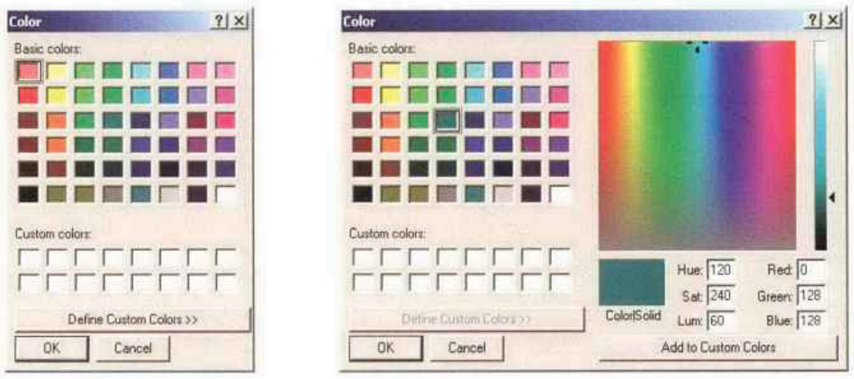


FIGURE 2-23 / The color dialog box in Windows 2000

what

Show the most important content up front, but hide the rest. Let the user reach it via a single, simple gesture.

use when

There's too much stuff to be shown on the page, but some of it isn't very important. You'd rather have a simpler UI, but you have to put all this content somewhere.

why

A simple UI is often better than a complex one, especially for new users, or users who don't need all the functionality you can provide. Let the user choose when to see the entire UI in its full glory—they're a better judge of that than you are.

If your design makes 80 percent of the use cases easy, and the remaining 20 percent are at least possible (with a little work on the user's part), your design is doing as well as can be expected!

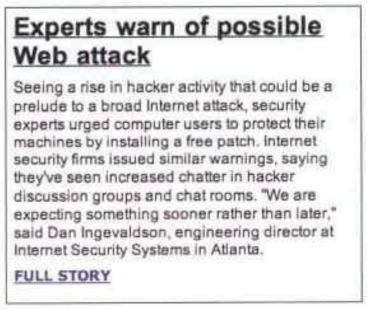
When done correctly, Extras On Demand can save a lot of space on your interface.

how

Ruthlessly prune the UI down to its most commonly used, most important items. Put the remainder into their own page or section. Hide that section by default; on the newly simplified UI, put a clearly marked button or link to the remainder, such as "More Options." Many UIs use arrows or chevrons, ">>", as part of the link or button label. Others use "...", especially if the button launches a new window or page.

That section should have another button or other affordance to let the user close it again. Remember, most users won't need it most of the time. Just make sure the entrance to and exit from this "extras" page are obvious.

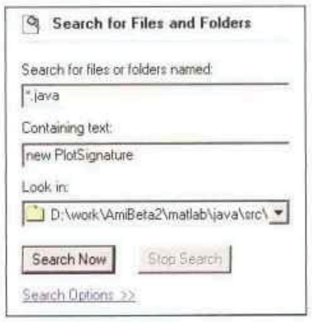
In some interfaces, the window literally expands to accommodate the details section, and then shrinks down again when the user puts it away. See the **Closable Panels** pattern (Chapter 4) for one way to do this. Various desktop UIs provide another mechanism: a dropdown for fill color, for instance, contains a "More Fill Colors..." item that brings up a separate dialog box.


Experts warn of possible Web attack

Seeing a rise in hacker activity that could be a prelude to a broad Internet attack, security experts urged computer users to protect their machines by installing a free patch. Internet security firms issued similar warnings, saying they've seen increased chatter in hacker discussion groups and chat rooms. "We are expecting something sooner rather than later," said Dan Ingevaldson, engineering director at Internet Security Systems in Atlanta.

FULL STORY

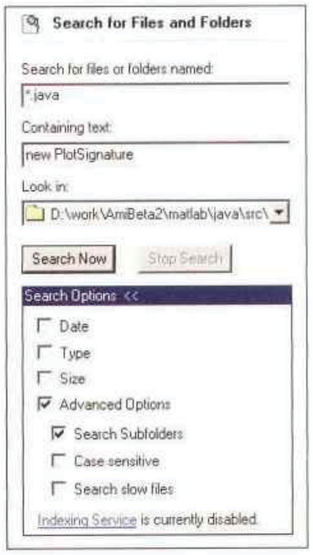
FIGURE 2-24 / Narratives frequently use Extras On Demand to separate the gist of an article from its full text. A reader can scan the leader, such as this one from CNN, and decide whether or not to read the rest of the article (by clicking "Full Story," or the headline itself). If they don't go to the jump page, that's fine—they've already read the most important part.


Search for Files and Folders

Search for files or folders named: *.java

Containing text: new PlotSignature

Look in: D:\work\AmiBeta2\matlab\java\src\


Search for Files and Folders

Search for files or folders named: *.java

Containing text: new PlotSignature

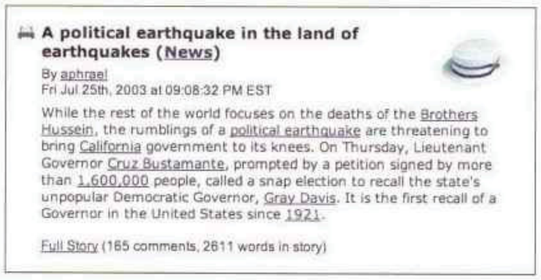
Look in: D:\work\AmiBeta2\matlab\java\src\

- [ ] Date
- [ ] Type
- [ ] Size
- [✓] Advanced Options
  - [✓] Search Subfolders
  - [ ] Case sensitive
  - [ ] Search slow files

Indexing Service is currently disabled.

FIGURE 2-25 / This is the file search facility in Windows 2000. Clicking "Search Options" opens a box of extra options. Likewise, clicking the titlebar of the Search Options box, with its "<<" chevron, closes the box. Not shown is another level of Extras on Demand: when the user unchecks "Advanced Options," the indented checkboxes below it disappear. This makes it similar to Responsive Disclosure (Chapter 4), which talks about content that comes and goes as a *side effect* of the choices the user makes, as opposed to **Extras on Demand**, which requires an *intentional* act to open or close content.

19 intriguing branches



A political earthquake in the land of earthquakes (News)

By aphrael
Fri Jul 25th, 2003 at 09:08:32 PM EST

While the rest of the world focuses on the deaths of the Brothers Hussein, the rumblings of a political earthquake are threatening to bring California government to its knees. On Thursday, Lieutenant Governor Cruz Bustamante, prompted by a petition signed by more than 1,600,000 people, called a snap election to recall the state's unpopular Democratic Governor, Gray Davis. It is the first recall of a Governor in the United States since 1921.

Full Story (165 comments, 2611 words in story)

FIGURE 2-26 / From *http://kuro5hin.org*

what

Place links to interesting content in unexpected places, and label them in a way that attracts the curious user.

use when

The user moves along a linear path—a text narrative, a well-defined task, a slideshow, a Flash movie, etc. You want to present additional content that's not the main focus of attention, however. It might be information tangential to a story, as in Figure 2-26. It might be supporting text—examples, explanations of concepts, definitions of terms—or full-fledged help text. Or it could be hidden functionality, like an "Easter egg."

In any case, you want a graceful way of presenting the content so it's ignorable by users trying to get something done quickly, but still available to users for whom it's appropriate.

why

People are curious. If they see something that looks interesting, and they have the time and initiative to check it out, they will. Web surfing would never have become popular without this natural curiosity and willingness to follow links into the unknown. Skillful and playful use of Intriguing Branches can make your interface more fun.

A tradition of creating Intriguing Branches as inline links (also known as "embedded links") already is well-established on the Web. But functional applications might provide a more interesting use of it. It's well-known that users tend to ignore what is labeled specifically as "Help." But what if you put help-like content behind links (or buttons, or icons) that were labeled in some other way, like "Learn more..."? You can exploit users' natural curiosity to get them into a place where they can learn what they need.

how

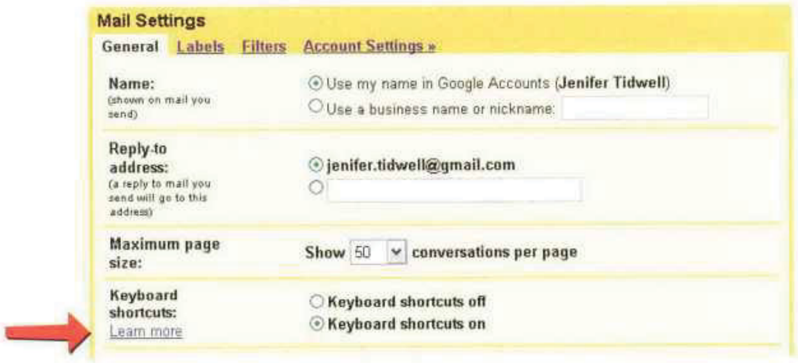
Start with a deep understanding of your users. What might interest them? Where in the interface are they likely to take time to explore something further, and where do they just need to get something done?

Create "doors" into the supplemental content that would appeal to users. These doors might be underlined links (even in desktop applications), headlines, buttons, menu items, icons, or clickable image regions—it's up to you to figure out how to label them in a way that inspires curiosity. There's an art to it. When in doubt, usability-test it with a representative sample of your user base.

With particularly obscure affordances, like icons or images, you might want to add tooltips or some other kind of short description to inform the user where they might go when they click on it. (With an Easter egg, though, its very non-obviousness is part of the fun.)

Also, provide an obvious way for the user to get back to their original workflow. The idea is to persuade users to read the branch content, and then go back to what they were originally doing; don't get them stranded in a backwater! Pop-up windows should provide "Close" buttons, and new pages in a browser-like UI should provide "Back" links or buttons.

examples



Mail Settings

General | Labels | Filters | Account Settings »

Name: (shown on mail you send)	[✓ Use my name in Google Accounts (**Jenifer Tidwell**)] ○ Use a business name or nickname:
Reply-to address: (a reply to mail you send will go to this address)	[✓ **jenifer.tidwell@gmail.com**] ○
Maximum page size:	**Show** 50 **conversations per page**
Keyboard shortcuts: Learn more	○ **Keyboard shortcuts off** [✓ **Keyboard shortcuts on**]

FIGURE 2-27 / Gmail's settings page offers links that are clearly help-related, but are phrased as suggestions, not as "help." Here, a "Learn more" link is under the Keyboard Shortcuts caption. This is akin to other forms of context-sensitive help, like pop-up menus, help buttons, and function keys. "Learn more" is an active phrase, unlike "Help," though, and it's clearly visible on the page, unlike menus and function keys. One can assume that it opens yet another web page, so its operation is entirely predictable.

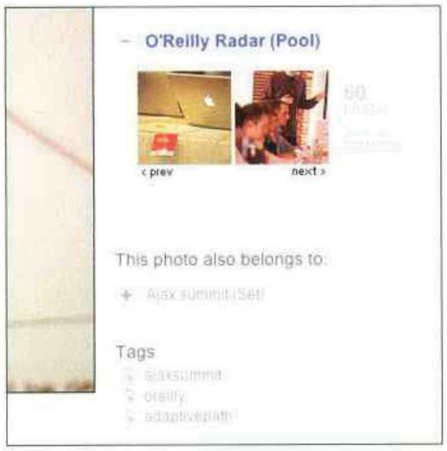


FIGURE 2-28 / Browsing photos in Flickr is often an exercise in following various intriguing branches—it offers "side trips" into other photostreams, image sets, and groups of images with common tags. The result is a thoroughly engaging (and very popular) user experience.

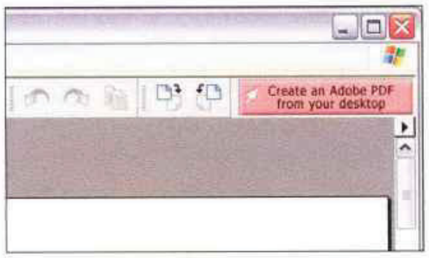
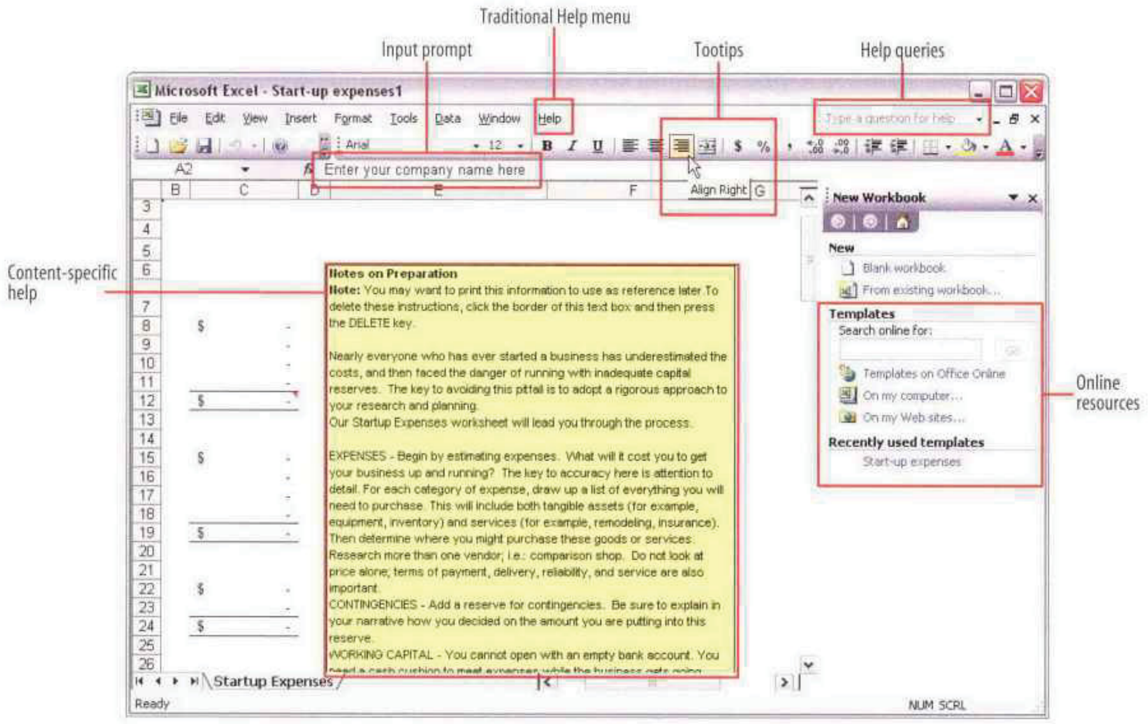


FIGURE 2-29 / A not-so-great example lies in Adobe's PDF reader for Windows. The pink button, "Create an Adobe PDF from your desktop," takes you to a page on Adobe's web site that explains how to use a different product to achieve that task. The button's color and content actually changes every few minutes; it shows a different teaser each time. This is an unusual case of such a link being present in an application—most don't take up valuable toolbar space with something like this.

But these buttons are somewhat self-serving on Adobe's part, since they encourage you to look at other Adobe products and services. Unfortunately, it looks like an advertising device. It would be interesting to know how effective they are, both at cross-selling and at helping users figure out how to do things they need to do anyway.

20 multi-level help



Notes on Preparation
Note: You may want to print this information to use as reference later To delete these instructions, click the border of this text box and then press the DELETE key.

Nearly everyone who has ever started a business has underestimated the costs, and then faced the danger of running with inadequate capital reserves. The key to avoiding this pitfall is to adopt a rigorous approach to your research and planning.
Our Startup Expenses worksheet will lead you through the process.

EXPENSES - Begin by estimating expenses. What will it cost you to get your business up and running? The key to accuracy here is attention to detail. For each category of expense, draw up a list of everything you will need to purchase. This will include both tangible assets (for example, equipment, inventory) and services (for example, remodeling, insurance). Then determine where you might purchase these goods or services. Research more than one vendor; i.e.: comparison shop. Do not look at price alone; terms of payment, delivery, reliability, and service are also important.
CONTINGENCIES - Add a reserve for contingencies. Be sure to explain in your narrative how you decided on the amount you are putting into this reserve.
WORKING CAPITAL - You cannot open with an empty bank account. You

FIGURE 2-30 / Excel's various help techniques, all integrated into the UI

what

Use a mixture of lightweight and heavyweight help techniques to support users with varying needs.

use when

Your application is complex. Some users are likely to need a full-fledged help system, but you know most users won't take the time to use it. You want to support the impatient and/or occasional users too, to the extent you can. In particular, your software may be intended for intermediate-to-expert users—how will you help beginners become experts?

why

Users of almost any software artifact need varying levels of support for the tasks they're trying to accomplish. Someone approaching it for the first time ever (or the first time in a while) needs different support than someone who uses it frequently. Even among first-time users, enormous differences exist in commitment level and learning styles. Some people will want to read a tutorial, some won't; most find tooltips helpful, but a few find them irritating.

Help texts that are provided on many levels at once—even when they don't look like traditional "help systems"— reach everyone who needs them. Many good help techniques put the help texts within easy reach, but not directly in the user's face all the time, so users don't get irritated. However, the techniques need to be familiar to your users. If they don't notice or open a **Closable Panel**, for instance, they'll never see what's inside it.

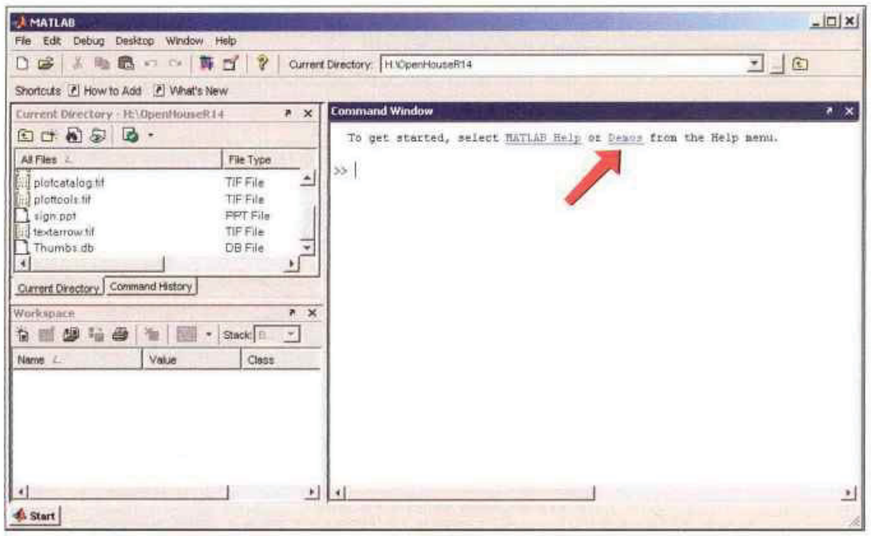
how

Create help on several levels, including some techniques (but not necessarily all) from the following list. Think of it as a continuum: each requires more effort from the user than the previous one, but can supply more detailed and nuanced information.

- Captions and instructions directly on the page, including patterns like **Input Hints** and **Input Prompt** (both Chapter 7). Be careful not to go overboard with them. If done with brevity, frequent users won't mind them, but don't use entire paragraphs of text—few users will read them.
- Tooltips. Use them to show brief, one- or two-line descriptions of interface features that aren't self-evident. For icon-only features, tooltips are critical; users can take even nonsensical icons in stride if a rollover says what the icon does! (Not that I'd recommend poor icon design, of course.) Tooltips' disadvantages are that they hide whatever's under them, and that some users don't like them popping up all the time. A short time delay for the mouse hover—e.g., one or two seconds—removes the irritation factor for most people.
- Slightly longer descriptions that are shown dynamically as the user selects or rolls over certain interface elements. Set aside an area on the page itself for this, rather than using a tiny tooltip.
- Longer help texts contained inside **Closable Panels** (see Chapter 4).
- Help shown in a separate window, often in HTML via browsers, but sometimes in WinHelp or Mac Help. Help is often an online manual—an entire book—reached via menu items on a Help menu, or from "Help" buttons on dialog boxes and HTML pages.
- "Live" technical support, usually by email, the Web, or telephone.
- Informal community support, almost always on the Web. This applies only to the most heavily used and invested software—the likes of Photoshop, Linux, Mac OS X, or MATLAB—but users may consider it a highly valuable resource.

examples

MATLAB is a complex, multilayered software application that offers help on all these levels. Each of the following examples comes from it.

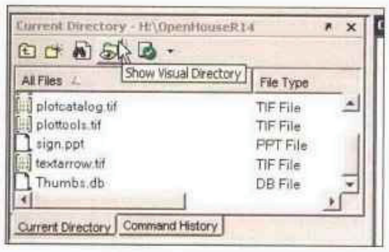

MATLAB
File Edit Debug Desktop Window Help
Current Directory: H:\OpenHouseR14
Shortcuts How to Add What's New
Current Directory - H:\OpenHouseR14
Command Window
Current Directory
Command History
Workspace
Stack: Base
Start

```
To get started, select MATLAB Help or Demos from the Help menu.

>> |
```

All Files	File Type
plotcatalog.tif	TIF File
plottools.tif	TIF File
sign.ppt	PPT File
textarrow.tif	TIF File
Thumbs.db	DB File

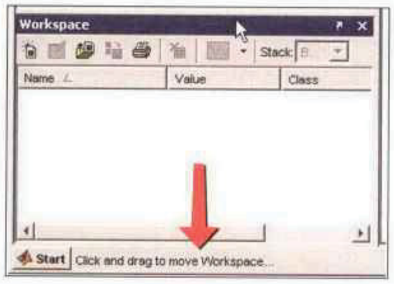
Name	Value	Class

FIGURE 2-31 / When you start MATLAB, the command line immediately directs you to help documents: "To get started, select MATLAB Help or Demos from the Help menu." You also can see captioned items on the Shortcuts toolbar, on which the "How to Add" and "What's New" buttons bring up help texts in separate windows.



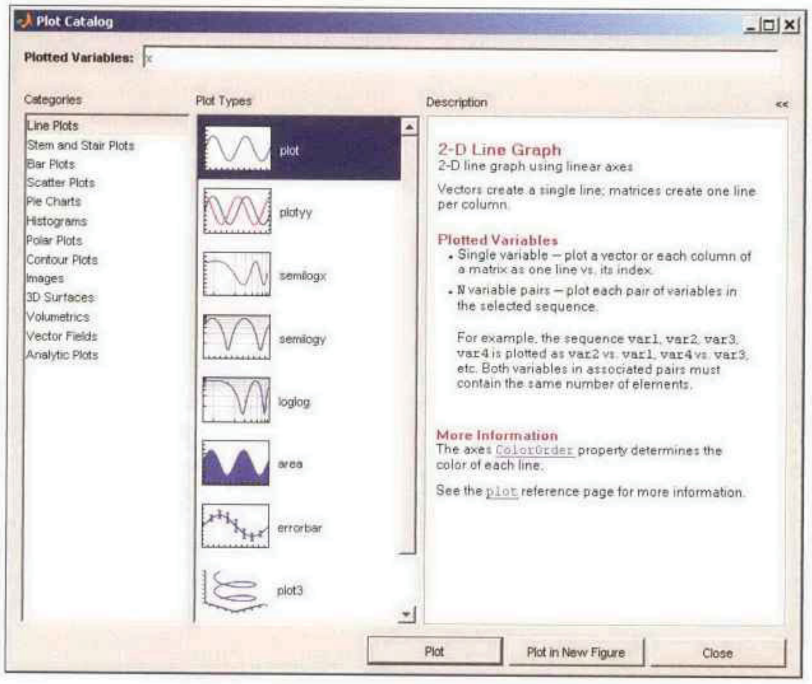
All Files	File Type
plotcatalog.tif	TIF File
plottools.tif	TIF File
sign.ppt	PPT File
textarrow.tif	TIF File
Thumbs.db	DB File

FIGURE 2-32 / Each toolbar button on the main window has a tooltip. Since MATLAB is an application designed for intermediate and expert users, it packs a lot of unusual functionality into small spaces, and tooltips are necessary to learn—or remember—what some of these buttons do.



Click and drag to move Workspace...

FIGURE 2-33 / Rollover help is used in a few areas, such as the status bars of movable panels. The sentence shown in this status bar, "Click and drag to move Workspace...," is a little too long to view comfortably in a tooltip. More importantly, the status bar is less obtrusive—a user can ignore it more easily than he can a tooltip.



2-D Line Graph
2-D line graph using linear axes

Vectors create a single line; matrices create one line per column.

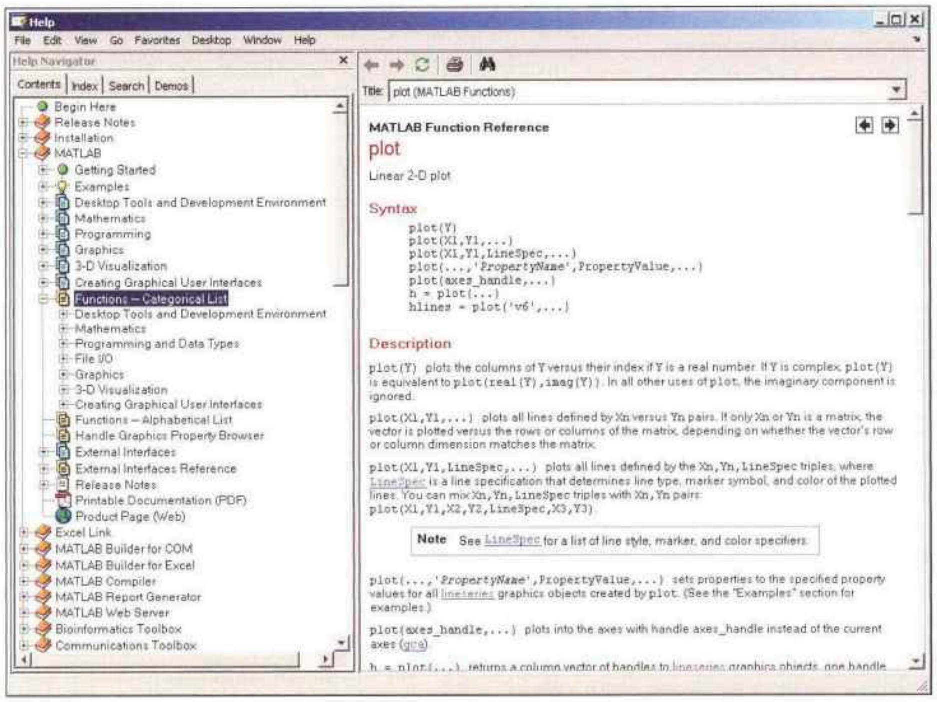
Plotted Variables
- Single variable — plot a vector or each column of a matrix as one line vs. its index.
- N variable pairs — plot each pair of variables in the selected sequence.

For example, the sequence var1, var2, var3, var4 is plotted as var2 vs. var1, var4 vs. var3, etc. Both variables in associated pairs must contain the same number of elements.

More Information
The axes ColorOrder property determines the color of each line.

See the plot reference page for more information.

FIGURE 2-34 / Elsewhere in the MATLAB interface, the selection of an object might cause a short description to appear in a closable panel. In this window, selecting a plot type causes the "Description" panel to show a short formatted help page. The help text is longer than often needed (the user can close that panel if he wants), but it's more immediate—and thus more likely to be used—than help in a separate window.


Help
File Edit View Go Favorites Desktop Window Help
Help Navigator
Contents Index Search Demos
Begin Here
Release Notes
Installation
MATLAB
Getting Started
Examples
Desktop Tools and Development Environment
Mathematics
Programming
Graphics
3-D Visualization
Creating Graphical User Interfaces
Functions — Categorical List
Desktop Tools and Development Environment
Mathematics
Programming and Data Types
File I/O
Graphics
3-D Visualization
Creating Graphical User Interfaces
Functions — Alphabetical List
Handle Graphics Property Browser
External Interfaces
External Interfaces Reference
Release Notes
Printable Documentation (PDF)
Product Page (Web)
Excel Link
MATLAB Builder for COM
MATLAB Builder for Excel
MATLAB Compiler
MATLAB Report Generator
MATLAB Web Server
Bioinformatics Toolbox
Communications Toolbox
Title: plot (MATLAB Functions)

MATLAB Function Reference

plot

Linear 2-D plot

Syntax

```
plot(Y)
plot(X1,Y1,...)
plot(X1,Y1,LineSpec,...)
plot(...,'PropertyName',PropertyValue,...)
plot(axes_handle,...)
h = plot(...)
hlines = plot('v6',...)
```

Description

`plot(Y)` plots the columns of Y versus their index if Y is a real number. If Y is complex, `plot(Y)` is equivalent to `plot(real(Y),imag(Y))`. In all other uses of `plot`, the imaginary component is ignored.

`plot(X1,Y1,...)` plots all lines defined by Xn versus Yn pairs. If only Xn or Yn is a matrix, the vector is plotted versus the rows or columns of the matrix, depending on whether the vector's row or column dimension matches the matrix.

`plot(X1,Y1,LineSpec,...)` plots all lines defined by the `Xn,Yn,LineSpec` triples, where `LineSpec` is a line specification that determines line type, marker symbol, and color of the plotted lines. You can mix `Xn,Yn,LineSpec` triples with `Xn,Yn` pairs: `plot(X1,Y1,X2,Y2,LineSpec,X3,Y3)`.

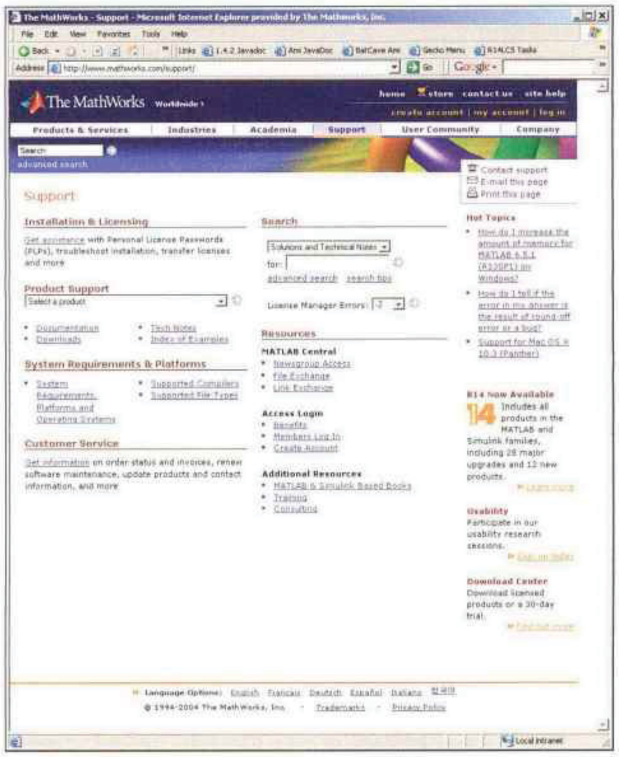
Note See `LineSpec` for a list of line style, marker, and color specifiers.

`plot(...,'PropertyName',PropertyValue,...)` sets properties to the specified property values for all lineseries graphics objects created by `plot`. (See the "Examples" section for examples.)

`plot(axes_handle,...)` plots into the axes with handle axes_handle instead of the current axes (gca).

`h = plot(...)` returns a column vector of handles to lineseries graphics objects, one handle

FIGURE 2-35 / In the previous window, clicking the underlined word "plot" in "See the plot reference page for more information" brings up a full-fledged help window. This tool gives the user access to the entire MATLAB manual.


The MathWorks - Support - Microsoft Internet Explorer provided by The Mathworks, Inc.
File Edit View Favorites Tools Help
Back Links 1.4.2 Javadoc Ant JavaDoc BatCave Ant Gecko Menu R14LCS Tasks
Address http://www.mathworks.com/support/ Go
The MathWorks Worldwide
home store contact us site help
create account | my account | log in
Products & Services Industries Academia Support User Community Company
Search
advanced search
Contact support
E-mail this page
Print this page

Support

Installation & Licensing

Get assistance with Personal License Passwords (PLPs), troubleshoot installation, transfer licenses and more

Product Support

Select a product

- Documentation
- Downloads
- Tech Notes
- Index of Examples

System Requirements & Platforms

- System Requirements, Platforms and Operating Systems
- Supported Compilers
- Supported File Types

Customer Service

Get information on order status and invoices, renew software maintenance, update products and contact information, and more

Search

Solutions and Technical Notes

for:

advanced search search tips

License Manager Errors: -2

Resources

MATLAB Central

- Newsgroup Access
- File Exchange
- Link Exchange

Access Login

- Benefits
- Members Log In
- Create Account

Additional Resources

- MATLAB & Simulink Based Books
- Training
- Consulting

Hot Topics

- How do I increase the amount of memory for MATLAB 6.5.1 (R13SP1) on Windows?
- How do I tell if the error in my answer is the result of round-off error or a bug?
- Support for Mac OS X 10.3 (Panther)

R14 Now Available

14 Includes all products in the MATLAB and Simulink families, including 28 major upgrades and 12 new products.

Learn more

Usability

Participate in our usability research sessions.

Sign up today

Download Center

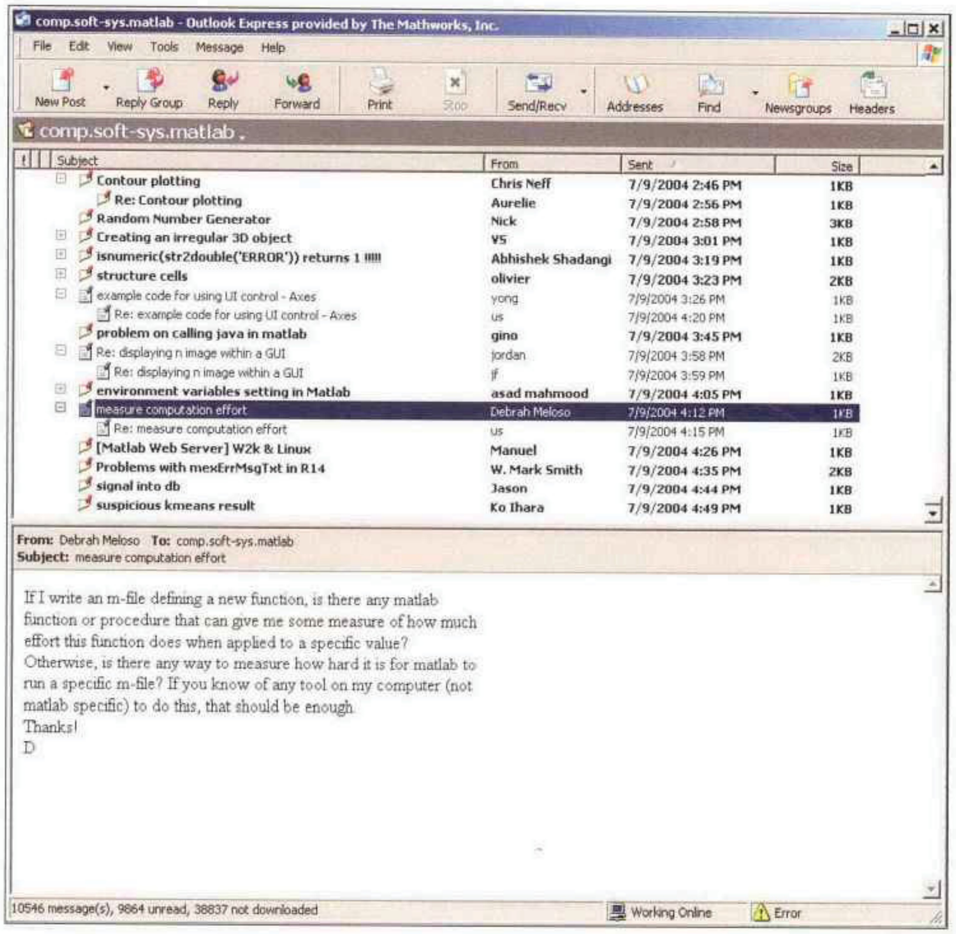
Download licensed products or a 30-day trial.

Find out more

Language Options: English Français Deutsch Español Italiano

© 1994-2004 The MathWorks, Inc. · Trademarks · Privacy Policy

FIGURE 2-36 / MATLAB users also get technical support over the phone and the Web. We've now moved beyond the realm of software design per se, but this is still product design—user experience extends beyond the bits installed on their computers. It includes the interactions they have with the company and its web site.


comp.soft-sys.matlab - Outlook Express provided by The Mathworks, Inc.
File Edit View Tools Message Help
New Post
Reply Group
Reply
Forward
Print
Stop
Send/Recv
Addresses
Find
Newsgroups
Headers
comp.soft-sys.matlab .

Subject	From	Sent	Size
Contour plotting	Chris Neff	7/9/2004 2:46 PM	1KB
Re: Contour plotting	Aurelie	7/9/2004 2:56 PM	1KB
Random Number Generator	Nick	7/9/2004 2:58 PM	3KB
Creating an irregular 3D object	VS	7/9/2004 3:01 PM	1KB
isnumeric(str2double('ERROR')) returns 1 !!!!!	Abhishek Shadangi	7/9/2004 3:19 PM	1KB
structure cells	olivier	7/9/2004 3:23 PM	2KB
example code for using UI control - Axes	yong	7/9/2004 3:26 PM	1KB
Re: example code for using UI control - Axes	us	7/9/2004 4:20 PM	1KB
problem on calling java in matlab	gino	7/9/2004 3:45 PM	1KB
Re: displaying n image within a GUI	jordan	7/9/2004 3:58 PM	2KB
Re: displaying n image within a GUI	jf	7/9/2004 3:59 PM	1KB
environment variables setting in Matlab	asad mahmood	7/9/2004 4:05 PM	1KB
measure computation effort	Debrah Meloso	7/9/2004 4:12 PM	1KB
Re: measure computation effort	us	7/9/2004 4:15 PM	1KB
[Matlab Web Server] W2k & Linux	Manuel	7/9/2004 4:26 PM	1KB
Problems with mexErrMsgTxt in R14	W. Mark Smith	7/9/2004 4:35 PM	2KB
signal into db	Jason	7/9/2004 4:44 PM	1KB
suspicious kmeans result	Ko Ihara	7/9/2004 4:49 PM	1KB

From: Debrah Meloso **To:** comp.soft-sys.matlab
Subject: measure computation effort

If I write an m-file defining a new function, is there any matlab
function or procedure that can give me some measure of how much
effort this function does when applied to a specific value?
Otherwise, is there any way to measure how hard it is for matlab to
run a specific m-file? If you know of any tool on my computer (not
matlab specific) to do this, that should be enough.
Thanks!
D

10546 message(s), 9864 unread, 38837 not downloaded | Working Online | Error

FIGURE 2-37 / Finally, if all other sources of help are exhausted, a user can turn to the wider user community for advice. In the case of MATLAB, users ask and answer one another's questions on a specialized newsgroup, *comp.soft-sys.matlab*. (Web-based discussions can serve the same purpose.) Community-building like this happens only for products in which users become deeply invested, perhaps because they use it every day at work or home, or because they have some emotional attachment to it, as many people do with games, Macs, or open source software.