

Language Usage in Design

Cynthia Shehata

Design for Clarity	1
Geekspeak.	1
Error Messages	2
Design for Consistency	2
Design for Readability	4
Design for Global Compatibility	5
Universal Access	6
Style And Grammar	7
References	7

Language Usage in Design

Cynthia Shehata

Design for Clarity

GUI applications often are not clear to end users. One effective way to increase the clarity of applications is to develop and use a list of reserved words (see Table 1 below). A common complaint among users is that certain terms are not clear or consistent.

Developers often try to make textual feedback clear by adding a lot of words. However, they ultimately make the message less clear. Concise wording of text labels, user error messages, and one-line help messages is challenging. Textual feedback can be handled most effectively by assigning these tasks to technical writers.

Use terms that users are familiar with and that are consistent across the product line and developer products. When you write labels for screen elements, try to speak in the user's language.

- Don't use technical jargon or computer science terminology (Geekspeak).
- Don't use file type names to refer to Finder documents that users see.

For example, *File* refers to any entity stored on a disk, regardless of whether the user can open, edit, or print it. This use of the term has its origins in computer science and is best avoided when possible. In developer documentation, it's permissible to use this term as long as it's well defined.

Example:

<u>Previously-used Term</u>	<u>Suggested Terminology</u>	<u>Example</u>
adev	Network extension	EtherTalk network extension
cdev	Control panel	Mouse control panel
DA	Desk accessory	Calculator desk accessory
ddev	Database extension	Data Access Language database ext.
FKEY	Function key	F1 function key
INIT	System extension	File Sharing system extension

Geekspeak

Ex: Linux

When the user attempts to exit XFM, the "X-windows File Manager," three ambiguous options appear Continue, Cancel, Abort. This leads to such questions as:

- Does "Continue" mean "continue using XFM" or "continue to exit"?
- Is "Cancel" somehow different from "Abort"?

“Continue” means “continue to exit”; if you would rather not exit XFM, then you are expected to select either “Cancel” or “Abort”. Apparently there is no difference.

Ex: Program “designed for woodworkers and cabinet makers.”

It is supposed to assist in the process of calculating price quotes for their projects. The program uses esoteric programming terminology like “Databases”, “Records”, and, if the user attempts to enter a duplicate part name, “Key Validation Error.” The typical woodworker should not have to have any practical understanding of such terms to use the product.

Don’t use commands that represent the actual programmatic statements used to write the program if they are unknown to non-programmers. Ex: “Fetch.”

Error Messages

Dialog box and alert box messages should be descriptive rather than evaluative - Say what went wrong, why it went wrong, and what the user can do about it.

- Brief, but informative.
- Tell what is wrong, where the error is, and what corrective actions can be taken.
- Provide information specific to the task rather than generic information.
- Non-judgmental.
- Go away as soon as the error is corrected or the user dismisses the error dialog box.
- When successive user actions result in the repetition of the same error message, rephrase the error message for the third and all subsequent times.
- If the set of valid inputs is small, present the set of alternatives with the error message.
- When the error involves a logical unit of input, the offending entry should be highlighted when the dialog box is removed.
- If there are multiple errors, tell the user there are multiple errors. All fields containing errors should be simultaneously identified to the user.
- State the consequences of an action before describing the action. For ex., say “To delete file, press Return” instead of “Press Return to delete file.”
- If a task has sequential steps, present the task in the order required to perform the task.

Design for Consistency

A lack of consistency ultimately leads to confusion and frustration for users. Good GUIs apply consistent behavior throughout the application and build upon a user's prior knowledge of other successful applications. Consistency in the use of language helps users easily learn to use the application.

Each new and exciting experience you provide in the software can become an anxiety-inducing experience or an expensive call to your help desk.

Microsoft has reported that the term “Find” was most readily understood by new users. If using either Search or Find, be consistent within the application.

Table 1: List of Reserved Words

Text	Meaning And Behavior	Appears On Button	Appears On Menu	Mnemonic Keystrokes	Shortcut Keystrokes
OK	Accept data entered or acknowledge information presented and remove the window	Yes	No	None	<Return> or <Enter>
Cancel	Do not accept data entered and remove the window	Yes	No	None	Esc
Close	Close the current task and continue working with the application; close view of data	Yes	Yes	Alt+C	None
Exit	Quit the application	No	Yes	Alt+X	Alt+F4
Help	Invoke the application's Help facility	Yes	Yes	Alt+H	F1
Save	Save data entered and stay in current window	Yes	Yes	Alt+S	Shift+F12
Save As	Save the data with a new name	No	Yes	Alt+A	F12
Undo	Undo the latest action	No	Yes	Alt+U	Ctrl+Z
Cut	Cut the highlighted characters	No	Yes	Alt+T	Ctrl+X
Copy	Copy highlighted text	No	Yes	Alt+C	Ctrl+C
Paste	Paste the copied or cut text at the insertion point	No	Yes	Alt+P	Ctrl+V

Use singular for menu titles (unless it doesn't make sense in the singular). Apple recommends a number of standard menu titles such as File, Edit, and Font.

Be consistent in your use of menu titles. Try not to create a menu bar that contains both plural and singular menu titles.

Try to be as specific as possible in your labels or names for radio buttons, push buttons, and checkboxes.

Don't sacrifice clarity for space.

An ex. dialog contains an OK and a Cancel button, but the instructions refer to the ENTER and ESCAPE keys. New users are unlikely to know that the ENTER key is the equivalent of selecting the default command button, nor that the Escape key is the keyboard equivalent of selecting the Cancel command button.

If you have to write something like "To exit SQL*Net Easy Configuration at any time, choose CANCEL," label the Cancel button Exit instead.

Example: Paint Shop Pro

After a Browse operation has been performed, the Browse menu item disappears; to browse a different folder, you must select "New Folder" from the File menu. Despite the fact that we've been heavy users of Paint Shop Pro for several years, this "feature" still throws us; when we want to perform a browse, we look for an item labeled... "Browse." While we can somewhat recognize the rationale for renaming the menu item, PSP's designers failed to consider the problems it would cause. In addition to the fact that the function seems to have disappeared, the phrase "New Folder" has a different meaning in the Windows operating system (i.e., "to create a New Folder").

Say what you mean. I buttons means "Yes" and "No," label them "Yes" and "No."

Don't make a simple confirmation dialog appear to be a serious error message.

If the users cannot utilize the design without referring to the help file, then the design probably needs work.

Design for Readability

Employ scannable text, using

- highlighted keywords.
- meaningful sub-headings (not "clever" ones).
- bulleted lists.
- one idea per paragraph (users will skip over any additional ideas if they are not caught by the first few words in the paragraph).
- the inverted pyramid style, starting with the conclusion.
- half the word count (or less) than conventional writing.

Using improvements in writing style -- concise, scannable, and objective -- resulted in an increase in readability of 124% (Usability Improvement).

Avoid promotional (marketese) writing because it imposes a cognitive burden on the user.

Design for Global Compatibility

Provide support for multiple script systems and multicultural sensitivity to target audience.

Design applications to be compatible with regional, linguistic, and writing system differences around the globe.

Remember that cultures assign varying values and characteristics to living creatures, plants, and inanimate objects.

Because it's easier to include worldwide compatibility from the beginning of development process than to try to incorporate support for script systems after product is complete, create application so that it is easy to localize, or adapt for use in a specific area. Localizing software involves translating an application's menus, dialog boxes, alert boxes, and content areas into a language or regional dialect.

Differences exist in the use of calendars, text, and the representation of time in various regions around the world. Make your application flexible in handling differences.

Provide the user with a way to change the representation of time. Use the text utilities to handle numbers, dates, and sorting.

Ex: Automate Pro

While many European users are comfortable with a 24-hour time format, most American users will resent the fact that they have to perform some mental calculations to convert times from the familiar 12-hour format. Automate Pro should have respected the user's wishes and used the format specified in the user preferences section of the operating system.

Store region-dependent information in resources so that text the user sees can be translated during localization without modification of application's code. When creating resources, consider text size, location, and direction.

- Text size varies in different languages.
- Direction of text may change (Left to right, right to left).
- Alignment of controls in the dialog box may vary with localization.

When the alignment of items is reversed, it's important that the elements appear vertically aligned. When creating dialog box items, make sure that their display rectangles are the same size.

When dialog box items are longer than the boundaries of the dialog box and the text direction is reversed, the text appears outside of the dialog box and isn't visible on the screen. Make dialog box items shorter than the width of the dialog box.

When translated, text can become up to 50 percent larger than U.S. English text. Text needs room to grow up, down and sideways.

Leave room for space between lines of text and between the top and bottom lines of any enclosing rectangle (some characters may extend farther up or down than Latin characters).

Don't use colloquial phrases or nonstandard usage and syntax. Carefully choose words for command names in menus and for messages in dialog boxes, alert boxes, and help balloons.

Use complete sentences whenever possible. Don't use phrases that you then concatenate to create sentences. The word order of messages may become completely different in translation, rendering such a message nonsensical when translated.

Keep in Mind:

- Characters aren't necessarily 1 byte; they can be 2 bytes.
- Text isn't always left-aligned and read from left to right.
- Text isn't always read by a person; it may be spoken through a text-to-speech converter.
- System and application fonts aren't always Chicago and Geneva.

Translate both the application's user interface and the user interface of the installation program.

Localizing an application is not simply a matter of translating the text of the interface to the target language. An example of a particularly problematic aspect of many applications:

Most U.S. designed applications assume that every country in the world has the same address conventions and ask us to select U.S. states from drop down boxes, put the zip code in behind the state name etc. Using a U.S. designed form generally ensures that the mail will not be delivered. In Norway, we don't use state names in our address, and we put our zip codes in front of the address. Also the "City:" field is not correct for Norway. Because we are so few, we do not have many cities and do not need to address by city. Kolsås is one of several districts of the city where I live, but by no means a city in itself.

A more usable alternative would be to simply provide a free-form text field and trust that the user has entered a valid address.

Universal Access

Provide for people with disabilities by means of alternative input devices or output devices

People who have a speech or language disability may use computers for augmentative and assistive communication. For example, they can use a computer to generate speech. Many people with speech or language disabilities are not able to use the standard keyboard and mouse because of some associated physical disability. To address the needs of these people, augmentative and assistive communications software is designed to be used with a variety of alternative input devices. Such software communicates with applications, windows, dialog boxes, and the desktop by emulating keyboard and mouse input. Don't create any barriers in your application to this type of communication, such as designing your application to communicate directly with the hardware (especially the keyboard and mouse). Make sure that the augmentative and assistive software will work with the product.

Style And Grammar

- Do not use anthropomorphisms -- do not personify the computer.
- Words used in user guidance should be common, meaningful, unambiguous, and complete (i.e., no contractions or abbreviations).
- Avoid computer terminology, unfamiliar, or esoteric terms.
- Use positively rather than negatively worded statements. Negative statements, however, are appropriate for conveying exceptions to rules.
- Use the active not the passive voice.
- Use consistent grammatical construction.
- State messages in short, simple sentences.
- A fact that must be remembered should be at the beginning of the message.
- Place a period at the end of each sentence.
- Use drawings or pictures to illustrate text whenever possible.
- Don't use exclamation points on your command buttons! They make it seem as if the application is shouting at the user!
- In labels or names for menu items, etc., use book title capitalization style. This style is referred to as caps/ lowercase. Capitalize every word except articles (a, an, the), coordinating conjunctions (for example, and, or), and prepositions of three or fewer letters (except when a preposition is part of a verb phrase).

References

- *Ameritech Graphical User Interface. Standards and Design Guidelines: Output to User.* <http://www.ameritech.com:1080/corporate/testtown/library/standard/guix5.html#5.3.1>.
- Interface Hall of Shame. "Globalization." <http://www.iarchitect.com/global.htm#MOR-EINFO>
- Interface Hall of Shame. <http://www.iarchitect.com/clarity.htm>
- *Macintosh Human Interface Guidelines.* Addison-Wesley Publishing Company, 1995.
- Nielsen, Jakob. "How Users Read on the Web." <http://www.useit.com/alertbox/9710a.html>
- "Principles of good GUI Design." http://axp16.iie.org.mx/Monitor/v01n03/ar_ihc2.htm