Form Fill-in

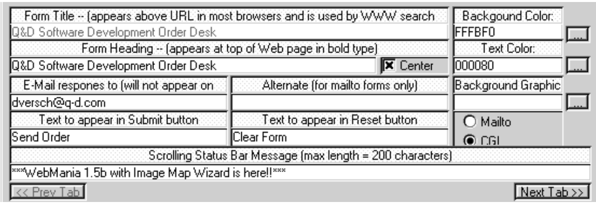
- if data entry is required, menus can be very cumbersome:

- when many fields of data are necessary, the appropriate interaction style is the form fill-in

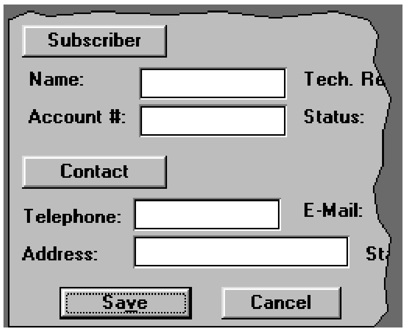
- in terms of locus of control, forms are very attractive, as the users feel they are in control of the dialogue
- on the other hand, users need to be familiar with keyboards (backspace, enter keys etc.), use of some way to move across the form (TAB key, cursor, mouse), field-labels and permissible field contents
- form fill-in is widely used on the World Wide Web

Form Fill-in Specific Guidelines

- meaningful title
- comprehensible instructions
- logical grouping and sequencing of fields (e.g. city field close to the postal code field)
- clear layout
- familiar field labels (Home Address is better than Domicile)
- consistent terminology and abbreviations
- visible space and boundaries for data entry fields
- error prevention, when possible
- error messages for unacceptable values
- marking of optional fields
- explanatory message for fields
- explicit closure: the user should know what to do when the data entry is finished



Labels not aligned to the fields. Difficult to read (eyes zig-zag around the screen)



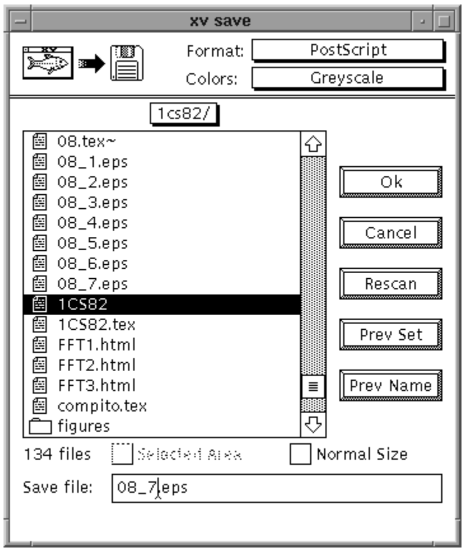
Section titles appear like buttons

Coded Fields

- some data may require a specific format
- a layout can be provided to help the user
- examples:
  - telephone numbers:
    `Telephone: (_ _ _) _ _ _ - _ _ _ _`
  - times:
    `_ _ : _ _  _ _  (09:45 AM or PM)`
  - dates:
    `_ _ /_ _ / _ _ _ _`
    `(03/04/1999 indicate 3rd April 1999)`
  - a currency amount:
    `£ _ _ _ . _ _`
- but: consider international issues!!

Dialogue Boxes

- sometimes users are requested to both make choices and enter data in order to perform a task
- dialogue boxes combine menu selection and form fill-in



Unix *xv* "Save" dialogue box

Dialogue Boxes Specific Guidelines

- Internal layout:
  - meaningful title, consistent style
  - top-left to bottom-right sequencing
  - clustering and emphasis
  - consistent layouts (margins, grid, white space, lines, boxes)
  - consistent terminology, fonts, capitalization, justification
  - standard buttons (OK, Cancel)
  - error prevention by direct manipulation
- External relationship:
  - smooth appearance and disappearance
  - distinguishable but small boundary
  - size small enough to reduce overlap problems
  - display close to appropriate items
  - no overlap of required items
  - easy to make disappear
  - clear how to complete/cancel

Goals of Language Design

- Basic:
  - Precision
  - Compactness
  - Ease in writing and reading
  - Speed in learning
  - Simplicity to reduce errors
  - Ease of retention over time
- Higher-Level:
  - Close correspondence between reality and the notation
  - Convenience in carrying out manipulations relevant to user's tasks
  - Compatibility with existing notations
  - Flexibility to accommodate novice and expert users
  - Expressiveness to encourage creativity
  - Visual appeal

9 Steps to Design a Language

1. determine functionality of the system by studying users' task domain
2. create a list of task actions and objects
3. abstract this list into a set of interface actions and objects
4. represent low-level interface syntax
5. create a table of user communities and tasks, with expected use frequency
6. determine hierarchy of importance of user communities (i.e. prime users)
7. evaluate destructive actions (e.g. deleting objects) to ensure reversibility
8. identify error conditions and prepare error messages
9. allow shortcuts for expert users, such as macros and customizing system parameters

Simple Command Set

- a command is chosen to carry out a single task
- number of commands = number of tasks
- if the tasks are not so many, this can produce a system easy to learn and use
- otherwise, it is necessary to use complex strategies employing letters, shift, CTRL etc., especially if you want to keep the number of keystrokes low
- Example, in the text editor *emacs*:

Home	go to upper left
End	go to last line
`CTRL-A`	go to beginning of line
`CTRL-E`	go to end of line
`CTRL-K`	delete till end of the line
`CTRL-W`	delete selected paragraph
`CTRL-X-S`	save
`CTRL-_`	undo
`ESC-1-ESC-Q`	justify paragraph

Command + Argument

- each command is followed by one or more arguments, indicating the objects to manipulate
- Example, in *Unix*:

`cd dir`	set working directory to dir
`rm file`	delete file
`cp file1 file2`	copy file1 to file2

Command + Options + Argument

- options indicate command special cases
- as the number of options grows, the command can become quite complex
- even with good training, error rates can be very high
- nevertheless, expert users seem to prefer this approach
- example: in *Unix*, the synopsis for the command `lpr` is:

```
lpr [-Pprinter] [-#num] [-C class] [-J job]
    [-T title] [-U user] [-i[numcols]]
    [-1234 font] [-wnum] [-cdfghlnmprstv]
    [name ...]
```

Consistent Argument Ordering

Several studies have shown that people perform much better when the order of the arguments in a command is consistent.

Inconsistent
SEARCH file no, message id
TRIM message id, segment size
REPLACE message id, code no
INVERT group size, message id

$$\Downarrow$$

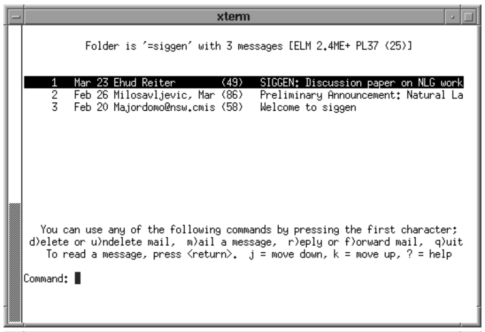
Consistent
SEARCH message id, file no
TRIM message id, segment size
REPLACE message id, code no
INVERT message id, group size

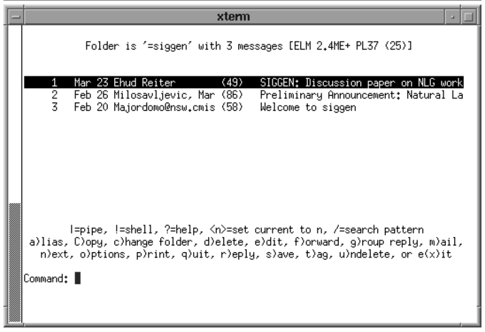
- Coherence and consistence also in the way options are supposed to be entered
- but avoid wasting user's time:

```
[floriana@mersey]$$ ps -a
warning: `-' deprecated; use `ps a', not `ps -a'
  PID TTY STAT TIME COMMAND
  429   1 S    0:00 /sbin/mingetty tty1
  430   2 S    0:00 /sbin/mingetty tty2
  431   3 S    0:00 /sbin/mingetty tty3
  432   4 S    0:00 /sbin/mingetty tty4
  433   5 S    0:00 /sbin/mingetty tty5
  434   6 S    0:00 /sbin/mingetty tty6
10490  p0 S    0:00 -bash
10813  p1 S    0:00 elm
10976  p2 S    0:00 /usr/local/bin/bash
10980  p0 S    0:00 xdvi.bin -name xdvi 102
10987  p0 Z    0:03 (gs <zombie>)
11207  p0 R    0:00 ps -a
```

Command Menu

- to help memory, you may consider a command menu
- it should not occupy too much screen space
- and should be adaptable to user's expertise

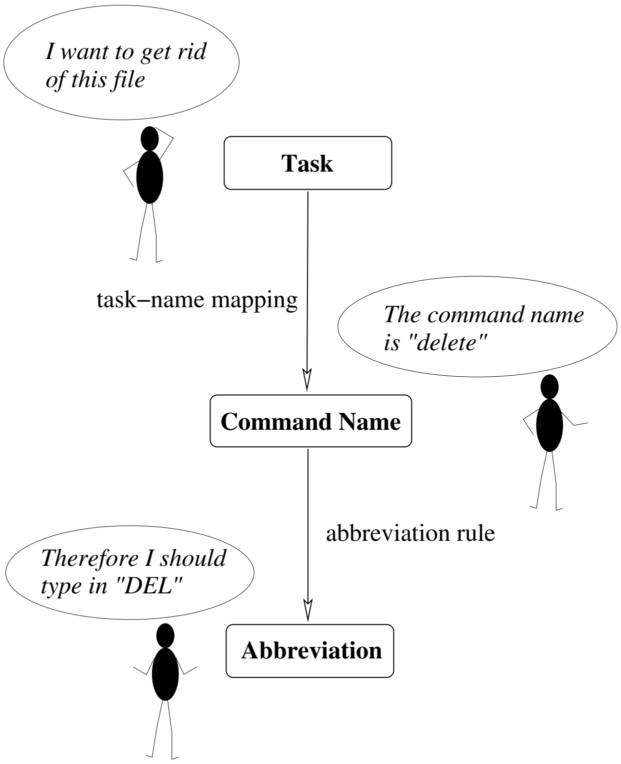




What's in a name?

- name choice is important for learning, problem solving and retention
- it becomes crucial when the number of commands and options is large
- *specific* names (insert/delete) versus *general* names (add/remove):
  - specific names are more descriptive
  - general names can be more familiar and then easier to accept
  - however, experiment shows that specific names perform better
- *concrete* names (show) versus *abstract* names (*grep*)
  - concrete names are of course easier to learn/remember
  - but when concrete names do not discriminate pair of tasks (e.g. alter/correct for the tasks of add/delete), then abstract names perform better
- abbreviations can help achieving the goal of brevity

Abbreviation



Abbreviation Strategies

1. simple truncation: use the first N letters
   - Delete = DEL
   - Insert = INS
   - Cancel = CAN
2. vowel drop with truncation: eliminate vowels and use part of what remains
   - Delete = DLT
   - Insert = NSR
   - Cancel = CNC
3. first and final letter:
   - Delete = DE
   - Insert = IT
   - Cancel = CL
4. first letter of each word:
   - Print Working Directory = PWD
   - Change Directory = CD
5. standard abbreviations (QTY, PRT, BAK)
6. based on phonetic (XQT for execute)

Examples of Abbreviations

To ...	DOS	UNIX
display list of files	dir/w dir	ls
	dir	ls -l
show contents of file	type	cat
show file with pauses	type filename | more	more
copy file	copy	cp
find string in file	find	grep
compare files	comp	diff
rename file	rename OR ren	mv
delete file	erase OR del	rm
delete directory	rmdir OR rd	rmdir
create directory	mkdir OR md	mkdir
change directory	chdir OR cd	cd
get help	help	man
		apropos
display free disk space	chkdsk	df
print file	print	lpr
display print queue	print	lpq