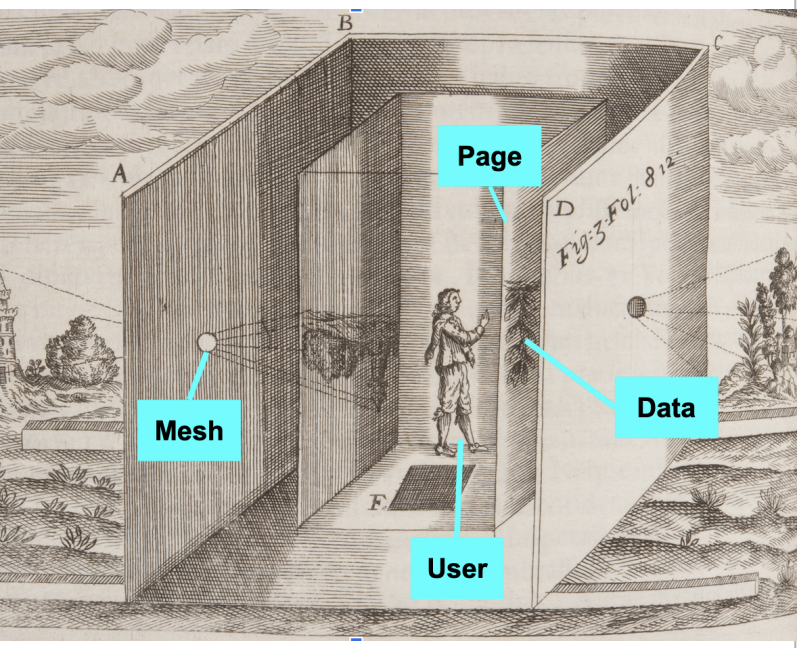
working draft!

Glossary of some Ui terms

Working definitions I find useful.

Between User — Page — Data — Mesh



 ← *double-click this to edit the diagram.*

- cursor
- control
  - control size
- styles: typographic, chrome, and object
- typographic style
  - inline controls
- chrome style
  - Chiselation
- object style
- page permutations
  - Ui modes
- input device mode
  - pointer
- short-term mode (hotkey-persistent)
  - modifier keys
  - drag&drop
  - gestures
- spring-loaded mode (mode-persistent)
  - recepticle
  - target mode
- tool mode (cursor-persistent)
  - toolbar
  - toggle button
  - multitool
  - palette
- stickaround mode (volatile mode)
  - pop-up selection
  - overflow
  - stickaround dialog
  - scrolling list selection
  - skrim
- collapsible mode (must close)
  - tool expansion
  - property sheet
  - info window
- config mode (tab-persistent)
  - representational choice, or viewmode
  - multiconfig
  - iconification
  - radio
  - checkbox
  - token
- sequential navigation (per Url-fragment)
  - stepped sheets or dialogs
  - TOC
- multi-dataset navigation (tabbed, Url-parameter persistent)
  - tabbed sheets or dialogs
  - ribbon (tabbed dropdown)
- site navigation (Url-path persistent)
  - landing page
  - preferences
  - global tool mode
  - edit mode
- setting mode (user-persistent)
  - switch
- object-contextuals
  - non-dismissable, contextual toolbar
  - non-dismissable, contextual palette window
- mode controls
- focus
  - focus indicator
    - focus ring
    - tab ring
  - overlay
    - linear selection
    - marching ants selection
    - handles
    - HUD
- Generic commands
  - Cut, Delete
  - Copy, Paste, Paste-in-Place, Duplicate
    - clipboard
  - (De/Re)activate
  - Undo, Redo
  - Find, Replace
  - Cancel, Submit
  - Bookmark
  - Comment on
  - Close
  - Alt Shortcuts

User

what, or who, is a user?

This concept developed in the mainframe era when a single machine would be timesliced to give a whole university access to it. In this sense, a user is an OS-related pattern. In the web, the user is app-related, i.e. each app has a hermetic model of their users. How does the federation federate users? If a user is a human (wetware), they may have many personae, but most users will just have one. Some users are AIs. A user may operate several clients st a time, but if they are not working in a clickfarm, they may be happy with operating only one at a time. It is funny how user is king in software development (User Xperience anyone?) Somewhat antithetical to that sucrose-rich user-centric neoliberalsim, consider Pouzin. Or Lialina [pdf 14MB]. Obviously, the underlying question is whether the user is *consuming commodities* (thus becoming commodity themself) or, literally, *using* the beast that is the computer.

avatar

a virtual creature that may have certain privileges on the collage. Prefixed with @, an avatar name can be mentioned. People may share an avatar, or for example organize a temporal takeover. Access/authentification to embody an @vatar should not be too much coupled to one civic identity. Normally, one user has exactly one avatar but the browser or client (or the app) can store several users for quick switching.

persona

a combination of 1 credential × 1 account, managed by the browser. it's just for convenience, but it shouldn't persist in a public client such as in a library.

authentification

exact credentials give a client the privileges of 1 user. Traditionally, credentials are 1 eMail × 1 password. More recently, × 1 phone number/SMS code for 2FA. Authentification happens on an auth server or service which stores salted and then hashed credentials.

account

user data, some of which may be very private? The account stores the avatar(s). A user may want to migrate their account to different credentials. Accounts are stored on servers but some account data might also be offline?

anonymous

is a user who doesn't log in, or who logs out. Contemporary services such as web shops create a temporary avatar, and once you log in or sign up, the data generated in that temporary embodiment will be merged into the logged-in account. Thus, 'anonymous' may have the very same tools and preferences at their disposal as any other user.

client

a machine that may have several tabs, usually a browser. different clients have different capabilities (Web platform). for example, a client nasty serve as a p2p server for nearby fellow clients. A client can be situated in a private place or in a public one, so personas may change or persist on one client.

app

Since the dawn of personal computing, two paradigms have been in a sort of competition: document-centric vs. application-centric. The document-centric approach may have died in the 90s with OpenDoc. Still, some file formats are app-agnostic.

session

temporary interval of app use. Reload will start a new session from the present Url, empty all form controls, and re-download any resource that is cache-expired (including the latest rolling 'app release', if that is still worth mentioning these days), Shift reload will re-initiate all the caches.

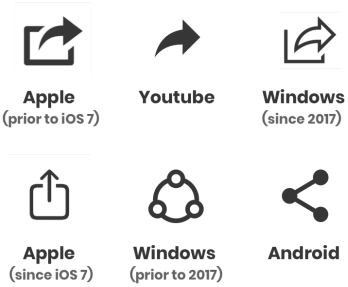
Sessions may be isolated per tab or shared across a single browser or across a single user. It might be worth looking up what the best (or common) practice is in that regard. Closing a tab, or the last tab of our app, will initiate an 'end session' procedure. An avatar may disappear on the shared composition once a user has ended all their sessions. For clarity, we may distinguish:

(a) tab session
(b) client session
(c) user session

What does it mean to 'resume' a session?

sharing

In an app-centric world, this becomes a hard problem. So we get lossy, unidirectional interchange formats. It's funny how Microsoft designed itself into a dead end a few years ago:



In contrast, *copy and paste* never get old:

Perhaps, a Share-button should really only pop up when a thing has already been copied, as part of the confirmation toast.

Here are things I may want to share:

- my avatar location
- some previous state of the composition, as recorded in my local history
- an object (text, media, group-selection...):
  - as a social media post or e-mail
  - into the federation?
  - into my device clipboard (copy)

tab

an app instance. On a client, many tabs with the same app may be open, but in most cases, only one has the focus of the operating system, the others are either visible but idle, or so-called background tabs. Each tab has exactly one URL, which can be manually edited by the user. In more general approach, the W3C calls this the viewport, but I think for the sake of aesthetics, we can omit for example iframes and the like from this glossary. Unfortunately, tabs are often confounded with screens (and the word viewport is a scopic metaphor), but a tab may live purely through a loudspeaker!

A single tab may hold its own session.

URL

what's in the the location bar in a tab. Can be bookmarked, pinned, edited, copied&pasted, shared, understood by users. The lingua franca of the web? Not in instagram, but everywhere else. A nice contemporary thing is a deep link, i.e. a link that smartly encodes userjourney-state so that any multistep operation can be continued on a different tab or client later. Urls encode those modes (settings) that are specific per-tab.

As URLs are expected to go in tandem with the history API, a user may expect the URL to change only when they click a hypertext or hypergraphic link.

user interface

The totality of an app's user interface is its set of implemented use patterns, living in its tabs.

use patterns

A pattern describes a discernible unit in how a user uses an app. Of course, patterns are never exhaustive in any nontrivial app. In this text, I wrote more on the subject. We can name some facets that structure each pattern, and positioning a feature on each of these dimensions helps anticipate its potential uses:

- **Aesthetic context**: *Why* do I want to (inter)act?
- **Social context**: *With whom* to (re)connect / *who* informs me / *whom* to position myself towards?
- **Machine context**: *Where* am I in the web/app/menu?
- **Objective**: *What* can I want to achieve?
- **Texture**: *How* does it feel/taste/resonnate?
- **Path** through a series of locations: *Which direction/option* do I have to navigate/click/browse?

use constraints

Orthogonal to the patterns are the constraints, i.e. invariants that limit the design freedom:

- **discoverability** (through familiarity, explicit hints, analogy)
- **accessibility** (through a minimum-set of inputs and outputs per device)
- **feedback** (as direct as possible)
- **malleable grammar**, including macro, meme, jargon, dada, and constructor.

use tenets

Besides patterns and constraints, an app defines tenets, which guide the implementation of features. These tenets are highly time- and

space-specific, in other words: fads. Yet, some of them have become standard. Some examples:

- Edit-in-place
- Direct Manipulation
- Object permanence
- Progressive Disclosure
- WYSIWYG

metrics

can be implemented in order to measure the extent, or set the threshold, for many of the patterns, tenets and especially constraints. Most a11y metrics including techniques are presented by the WCAG21. A test page would make some metrics explicit and variable, and evaluate them, including but not limited to:

- target size level (44 by 44 pixels)
- icon size (24 pixels)
- minimum font x-height (5px)
- median font-size/X-/x-height (16/14/8)
- body leading ratio (11:8)
- chrome leading ratio (2:1)
- caption leading ratio (1:2)
- maximum measure (line-length)
- natural proportions up to :11:
- vertical rhythm (per leading)
- dark mode
- border/bevel/shadow style for chrome
- accent colors
- focus ring style
- contrast
- speed
- non-vision accessibility

vertical rhythm

in its simplest form is the vertical quantization of significant breaks. The quantum is the body text line height. As many horizontal lines as possible should snap to this rhythm. In more advanced cases, you can add guides at lines that are especially emphasized, or define sections for sub-rhythms. Then, you'd aim for natural ratios between these additional rhythms. Advanced vertical rhythm is too hard to achieve on the web.

density

applies to devices where a keyboard can be expected. A dense layout emulates the metrics prevalent in the pre-mobile era. For example, a button in OS/2 was just 24 pixels high (6-7mm), and the Mac systems of the 80s and 90s had a similar optical size (at a quarter lower nominal pixel density when measured in point). Google Docs has the same target size for toolbar buttons as of 2021, which translate to a mere 5.5mm on a 2020 Macbook Air! While the WCAG21 recommends 44px in both dimensions (11-12mm), we can safely assume that 33px (8-9mm) are fine with most desktop use and 22px (5-6mm) are close to a default on 'productivity' apps. After all, a user can easily increase the rem or zoom into a portion of the screen.

	Chrome	C. (dense)
Line height	**×2** (44px)	**×1½** (33px)
min. width	**×2** (44px)	**×1** (22px)
Icon size	24dp	18dp

typographic scale

The default **Body size** is provided by the user agent and would be 16px in most cases. The other invariant we get from a usability angle is the minimum target size, which is 44 pixels. This is the minimum useful line-height for chrome controls. So let's say we want a relatively dense layout and stick with 44px. Then we can decide whether 2 or 3 lines of body text fit within the single chrome leading. In the latter case, the chrome leading should be 48 and the body leading ratio 1:1 (for a 16px leading). Of course, this only works for fonts with a very dense kerning or small x-height. In the first case, we get a body leading ratio of 11:8, which is generous but not unusual, and makes for an airy look. Then we design the denser sections. A dense chrome section can half the leading at the

expense of the target size, so this option should be opt-in, and is only useful in the case of heavy media manipulation such as sound or 3D UIs. The typographic scale would appear such:

typographic	chrome style	x-h.	size (px)	line-h. /v.R. *
H^{1}		**34**	68	66/88
Hx^{2} 36/*Display*		**24**	48	44/66
	icon	**24**	24	44
Intro 18pt		**12**	24	33/44
Hn^{-2}		**12**	24	33/44
Hn^{-1}		**12**	24	33/44
	Rubric	**12**	18	22/44
	Control (dense)	**8 (5)**	12 (9)	11/44 (/33)
Hn^{0} (12pt)		**8**	16	22
Body 12pt/*Book*		**8**	16	22
Caption (9pt)		**8**	12	11/22
	Hint (dense)	**5**	9	11/22 (/11)
Small 8pt		**5**	11	11/22

* *The second line height is the raster quant needed for vertical rhythm and achieved through padding.*

Optical size: *The point numbers in the specimen can be consulted to select font variants that alter weight and kerning to balance perceived thickness. Serif fonts will also draw serifs thicker and wider at low sizes:*

ITC Bodoni Six
ITC Bodoni Twelve
ITC Bodoni Seventy-Two

Page

The conventional unit of *Shell*. One page can be a screenful, a printed page, or a Homepage. In any case, a page has its proper organization and can undergo a limited number of permutations. A page implicitly frames its contents.Since there is no other interface, all user experience (UX) happens through pages. Thus, while a humanist/cybernetic understanding suggests that the user experience is "mediated" by the interface, erasing the materiality of the screen, the *Shell* figure instead suggests that this experience is a function of the page, and vice-versa. A page is thus the unit and instance of *Shell*, and all Shell Properties apply. Interestingly, there is no Shell per se, only its instance as a Page, which is why I don't wiki a strict hierarchy here.

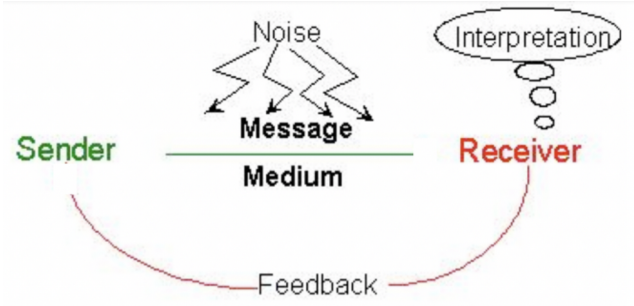


Fig. 1: Cybernetic communication model. Source: Google Image Search ;-)

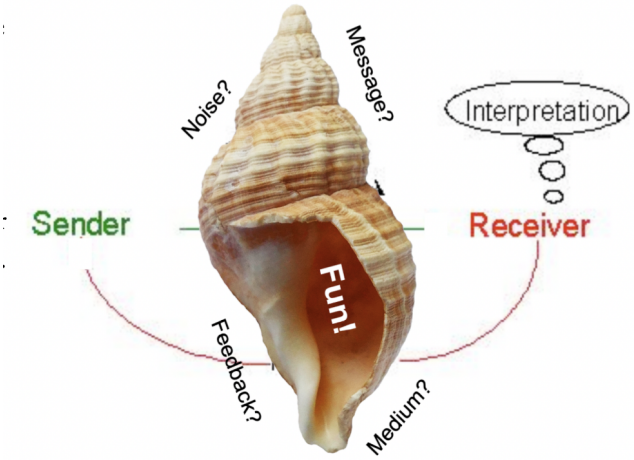
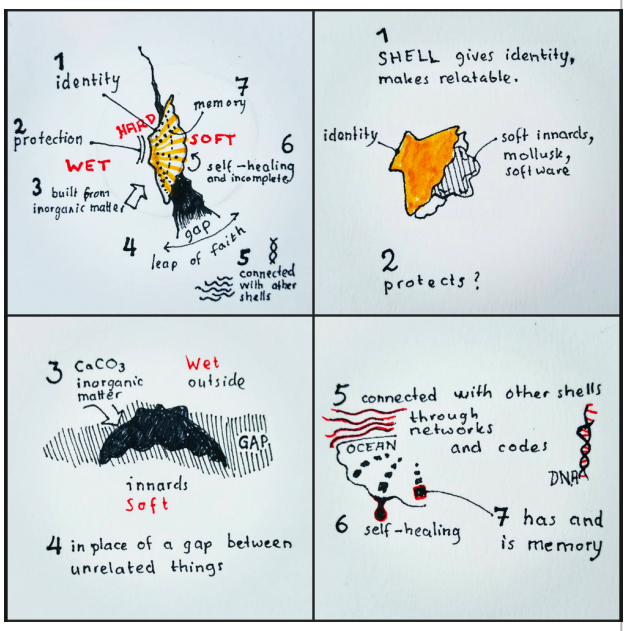


Fig. 2: "I put a Shell on you Descartes!"

Shell Properties ≙ Page Properties



(1) Shell gives both identity and unity to an otherwise amorphous mollusc. A page is what we know about its content, and what we can refer to; every Page is bookmarkable. Don't mess with the RAM.

2) A Shell protects the mollusc from the ocean currents , from dissolution, and from gourmands, but it also protects us from its unsightly slimyness. In the same way, a Page is protective in both ways. Pouzin highlights this *Shell* feature beautifully.
3) Shells inhabit a hybrid space between soft and wet. Their code is organic but their matter is inorganic. A page is neither software nor wetware (human) although the cyberspace has been imagined as an extension of the human brain by evil transhumanists. It's not. It's so much more beautiful than that!
4) Shells exist at a gap and bridge it. Just imagine how you would approach a computer without a shell. Archaic! The Shell is unlike anything; Pouzin did a leap without precedent.
5) Shells span not islands but nodes. Pages are never alone. They are implicitly connected by their code (like DNA), and explicitly linked by references. Thus emerge meshes.
6) Shells secrete smart liquids to heal themselves. A page is never whole but always completing.
7) A page is the trace of its own history, while being a memory device itself.

Page Organization

context

Each control in a page pertains to exactly one context. This is a central concept for accessibility and the W3C outlines it in its necessary complexity and graduality here. Besides user agent, viewport ("windows, frames, loudspeakers, and virtual magnifying glasses") and focus, "opening a new window, moving focus to a different component, going to a new page (including anything that would look to a user as if they had moved to a new page) or significantly re-arranging the content of a page are examples of changes of context." The gist is that a user would form an expectation as to which region of a page, or, more abstractly, which facet of a model of data that is

being represented on the page, is potentially being affected by a concrete control. This leads to several considerations:

- Keep controls "near" their context, either through proximity (see overlay) or through convention (see toolbar). When contexts are nested, it needs to be clear which level a control affects. Consider for example local vs. global undo. Often a global control is preferable.

chunk

the largest unit perceptible as one. A chunk may be any formalized group (singleton, word, sentence, list...) or an accidental group (page at a given scroll position, accidentally alliterated verses, lines in a paragraph of the same length, a span that is compressed by a floating image). Chunks were used extensively in medieval book layout as a memory device, based on much older traditions, but fell out of common use at the advent of accessible books with indexical structure).

item

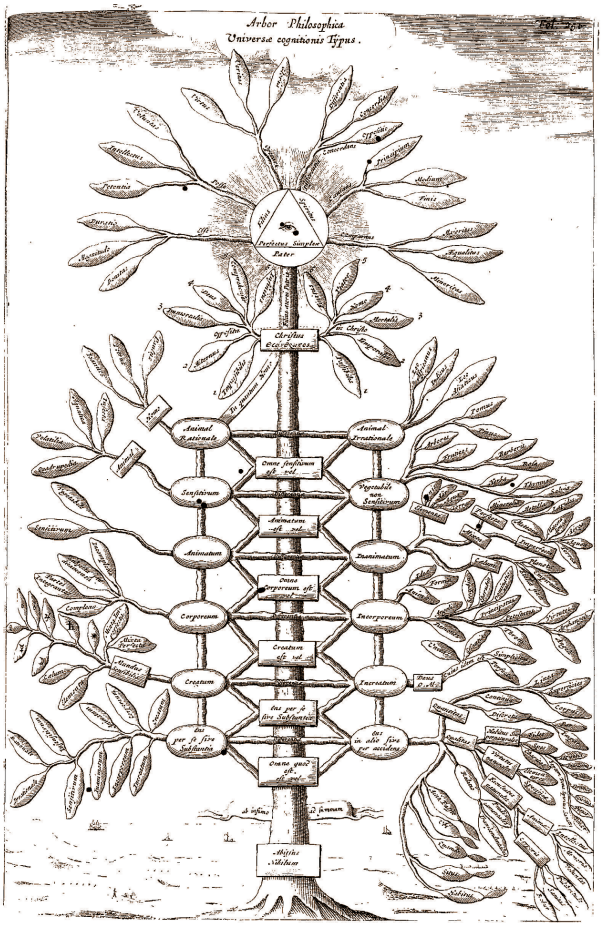
a single segment as part of a grouping, along with fellow items. If an item is also a chunk, the magic number seven plusminus two applies to the grouping. An item can be a grouping itself. The grouping is formalized, as in a list or a set or an aside or a reference. If it's accidental, the item is a chunk.

¶

A paragraph (¶) is perhaps the most common chunk. The DOM model imposes some restriction on the nesting of elements within a paragraph: while it may appear within any context, its content is limited to flow content.

index

Perhaps the largest project of humanism was the hierarchical categorization of what they call knowledge. But knowledge is not universal. So indexing remains a very personal affair. Or, as pragmatic Berners-Lee quipped in 1990: "The usual problem with keywords, however, is that two people never chose the same keywords." And here is the tree of all knowledge, from Athanasius Kirchner, for your sublime viewing pleasure:



grouping

a perceptible coherence across a set of items or chunks. If it's formalized, i.e. part of the semantic model, then it's a group of items, otherwise it's a chunking.

section

A semantic subdivision unit. In Html, a section is a node in the DOM tree. A heading is strongly recommended. Apart from that, a header/main/footer trifecta is often found in sections. While **`section`** marks up a generic section, more concrete subdivision tags are for example **`nav`** and **`article`**. In contrast to lists, a group of sections is often polymorphic but won't grow, which reminds me of the difference between tuples `(a:A, b:B, c:C)` and lists `[a1:A, a2:A]` in Elm.

rubric

This is a concept from medieval illumination. Rubrics are mnemonic indices: red (rubio) marks that highlight a heading or an aside or a marginal. Rubrics may be synchronized with an index, guide, or tabula.

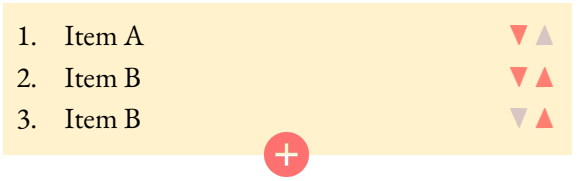
emphasis

In european typography, we distinguish three types of emphasis which serve different purposes and can be orthogonally combined. **(1)** Normal emphasis, typically expresses by *italic* typeface or a change in the reading voice, lifts a word out of the phrase without altering the ink density. Therefore, it is not glance-able and doesn't make the paragraph uneven. Uses include foreign-language words or important names. If italic fonts are not available, such as in blackletter, a printer uses increased letter spacing. In handwriting, unterlining is common. **(2)** So-called **strong** emphasis, usually in a bold face, creates a visible subdivision in the paragraph, it pops out, and is thus used for indexical markup, where a reader can scan a page or paragraph for boldprinted keywords before diving into the copy. Example: the numbers 1, 2, 3 in this paragraph. **(3)** Underlining is a technique mostly found in handwriting and annotating, with various purposes. Notably, in the www, hyperlinks are underlined and blue.

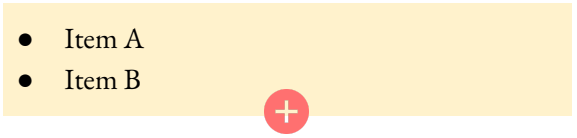
Styles other than typographic may employ more emphasis modes such as overlay or accent color.

lists

ordered list



unordered list



marginalia

notes on the margins. Typically, you can find asides or collapsed comment threads there.

aside

contains marginal content. Implemented in Html5 as **`aside`**.

nav

a section with links. Html5 **`nav`**.

article

a section with metadata that is fit for exchange (rss, blogging, sharing...)

toast

confirms 'invisible' actions such as undo, delete or copy, and notifies about incoming stuff. It's called a toast because it's a tiny message that pops up from the bottom of a screen or the silence of your speaker like a toast pops out of the toaster. Once it gains appropriate buttons such as share or undo or redo, or links to relevant user preferences, and an ☒ for closing, it is called a snackbar.

snackbar

more engaging than a naked toast.

a(nchor)

add paths orthogonal to the scrolling (block) direction. It can appear in any surroundings, and should be styled blue and underlined if it's text, and with a blue outline if it's a picture. In the typographic style, a link may retain spatial properties of the surrounding body so that an **a** markup would not affect the layout. (make sure that the underlining does not alter the width of the link text!). In other style surroundings, use the default chrome style plus blue/underline decoration.

Importantly, anchors add URL waypoints to the browsing history, which can be re-traversed through the browser's back and forward controls. These controls can be reproduced within the Ui in a familiar way: arrows or hand-icons. Keep in mind that 'forward' and 'backward' are inline-directions and thus depend on the writing system in use (see flow layout on the MDN).

Hypertext and Hypergraphic Links

As this museumized document eloquently says, a link *"gives the user the ability to connect to different units of text and graphics. The connections that join these units are known as hypertext or hypergraphic links. For example, a user can select a particular link to obtain related information or perhaps to see a graphical description of the topic.*

An advantage of using hypertext or hypergraphic links is that the author can present information in a ***nonlinear*** *way. Users can then access information both sequentially and randomly. This lets them explore or branch into subject matter that may be unclear or that needs to be reviewed."*

A link may or may not alter the whole page, but it will always alter the context, as described here. The single-page app may render a link target within a floating, dismissible window such as on ShellCongress.com. In this case, dismissing the window is an operation on the history where all path nodes after the link origin are popped.

cursor

It's that little fairy cursing the things on the page!

Check out input, caret and drag&drop for cases where the cursor shape changes.

control

anything that doesn't *represent* data but gives the user *something to modify*. Buttons, inputs, tabs, toolbars, dialogs, lists, and overlays are examples of controls, and the list goes on. A control is either a mode control or a data control.

control size

Any control that is not inline needs a target size of 44 by 44 CSS pixels, as per WCAG21. As such a large control would be very heavy in a dense UI, we can add invisible padding. The easiest path would probably be extending the label under the control and making it min. 44px in each dimension (with fully rounded corners). See metrics for a general discussion on quantifying tenets, constraints and patterns.

styles: *typographic, chrome, and object*

There are three 'Styles' that dictate the expected rendering of elements on the page. They are closely tied to the haptic aspects of the element.

While the typographic style is fully scalable, the chrome and object style are not. This ensures that the control size is constant. Since device display ratios are typically ×2 or ×3, we can design with a 2-3 subpixel resolution, and make lines (of any thickness) touch CSS-pixel boundaries at least at their outside. For example, a 0.5px border in a box with box-sizing:border-box that is itself at an integral position and of integral dimensions will be sharp at the outside and blurry at the inside in the case of an odd device pixel ratio, while a 1px

border will be sharp in any case. The resolution of diagonal line-pairs is lower.

typographic style

indicates copiable, searchable text and a hierarchical, TOC-accessible, overarching structure. Chrome embedded in typographic DOM nodes such as `article` or `p` may morph into a typographic form, whereas nodes typically associated with typography may instead acquire a chrome style when placed in a form. The boundary is blurry and depends on the amount of copy and on the intended feel of the app. The typographic style may define its own highlight color, employ inversion, or adhere to the system-supplied accent color. It looks 'flat'.

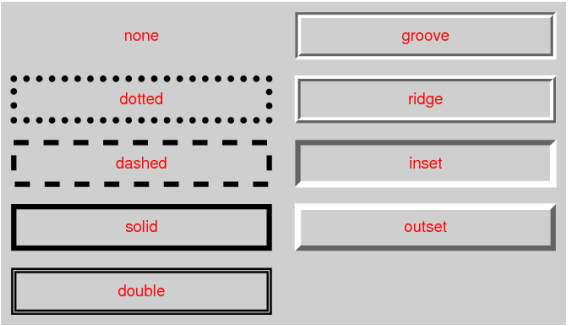
inline controls

Links and inline toggles are exempt from the target size rule and may be styled according to specific conventions.

chrome style

indicates direct manipulation associated with objects. May define its own highlight color, employ inversion, or adhere to the system-supplied highlight color. May employ extravagant features such as skeumorphic beveled/chiseled edges, or double-exposure and light effects when used on a dark background (e.g. a H.U.D.)

Chiselation



See the borders on the right half of this image from the W3C. This style dates back to 1996

object style

is the look-and-feel of the representation of data items (see object). For each domain (vector graphics, sound waves, 3D, video, pixel graphics, spreadsheets) there are rich conventions. If objects are light or text-like (markdown, calendar date and time, location), you can expand upon the typographic style. In the early 90s, it was fashionable to build upon the chrome style instead, and highly interactive or mutable domains such as calculations, flow diagrams or network representations may profit from such an approach.

page permutations

As far as a page is tree-shaped, every permutation of a node in the DOM may be called a mode of that node and must be transparently reversible.

Ui modes

Choices such as accessibility options or toolbar arrangements pertain to a user rather than the data. Many conventions have arisen to visually distinguish Ui modes from data manipulation, and to discriminate between persistency tiers. Compare the mode controls with the data controls. Here are some common modes in order of ascending persistence:

mode tier	persists with	examples
input	input devices	mouse, pen
short-term	modifier key	Shift, Ctrl
spring-loaded	other mode	drop target
tool	cursor	brush, circle
stickaround	just 1 click	menu, popup
collapsible	user; until closed	search widget, prop. sheet

sequential	Url fragment	wizard steps
config	Url parameter	viewmode
multi-dataset	Url parameter	tabs
site nav	Url path	web pages
setting	user	nickname

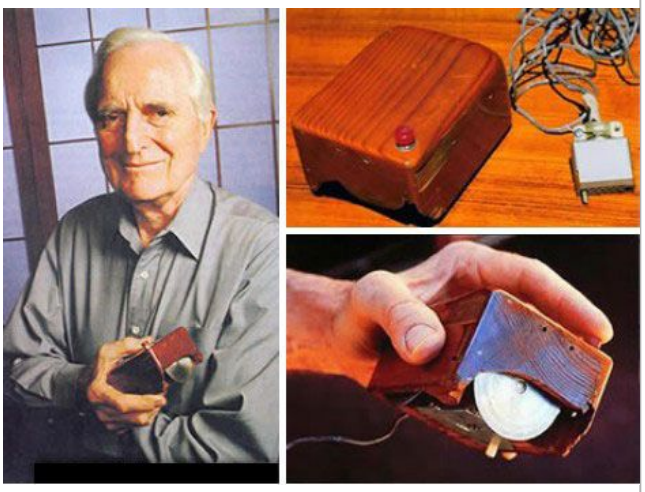
Note that for each tear, there are single-select controls for binary choice, choice among 3-4 options, 5-11, and 12+, and there may exist multiple-choice controls for these categories, too. Check the tables under mode controls for details.

Pink: typically in typographic style.

Grey: chrome style.

input device mode

Ever since smartphones have replaced PCs completely, only the nostalgic still cater for the touchpad enthusiasts, and of those, a handful of lunatics still remember the venerable mouse:



Most devices can be equipped with mouse, keyboard, and numerous other input devices, so it may be wise to offer a menu for switching operating modes.

Check out pointer for a detailed ontology of input gestures!

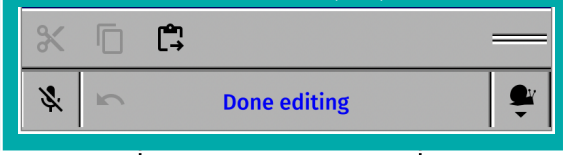
In the following example, I want to outline how the input device would affect interaction patterns in a beautiful collage app.

	touchscreen	touchpad	mouse
toggle mic	click toggle-button in toolbar		
reveal avatar sheet	click your avatar in the center of the screen, or click avatar-icon in toolbar		
	focus avatar, then press space		
share	button in avatar sheet		
browse history	use the single-select *'now, 1 minute ago, yesterday...'* in avatar sheet		
close any sheet	tap skrim	click ☒	
	drag the toolbar down		
	press ESC		

move avatar & viewport	**scroll** (UA default behavior)	
	focus avatar, then press arrow keys	
zoom	UA default behavior	
rotate focus	**tab key** (between avatar, object, and sheet) **arrow keys** (through objects, or sheet controls)	
walk to & reveal comment sheet	tap object	click object
	focus target object with tab and arrow keys, then press space. (focus will automatically switch to comment sheet)	

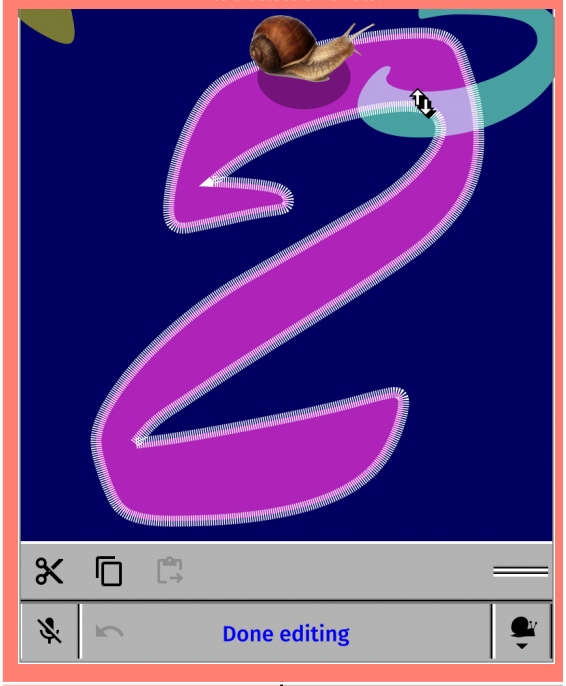
activate edit mode	click ‘edit’ button	
	press enter when focus is on avatar or object	
	edit Url path from /explore to /edit	

While in edit mode (/edit):

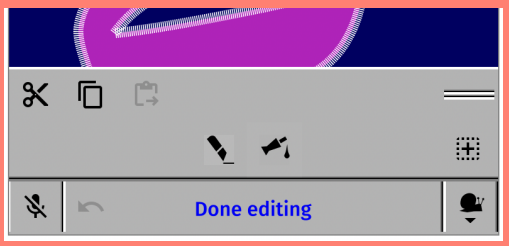


move avatar & viewport	2-finger pan	MMB
	arrow keys (when avatar has focus)	
	modifier + arrow keys, when focus is on object, toolbar, or sheet	
select single object	tap or double-click unselected object (double-click implicitly opens property sheet)	
	initiate pinch gesture	
	focus an unselected object, then press space or enter (enter implicitly opens and focuses the property sheet)	
select many objects with a lasso	tap on the canvas and draw a lasso around objects. The canvas scrolls at the edges.	

While a selection exists:

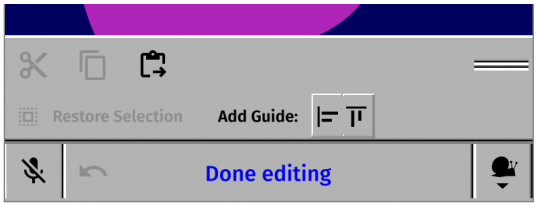


show ‘select up to’ togglebutton		
also (de)select	with first finger down on selection, tap more objects or	
	islands	
	tap objects with modifier key: Ctrl/Cmd	
(de)select contiguous (‘up to’)	‘select up to’ togglebutton	
	modifier key: Shift	
copy, cut	click ‘copy’ or ‘cut’ button in toolbar	
	Ctrl/Cmd+C or X	
reveal property sheet	click the fold under the toolbar	
	double-click any object (implicitly selects it)	
	drag the toolbar up	
	focus the toolbar, then navigate down with arrow keys	



use knife	activate the knife tool in the property sheet, then click any of the high-contrast fissures on the most-recently selected object and tap the scraps you want to discard. Discarded scraps appear semitransparent. Click the discarded scraps to restore them; click cut fissures to heal them.	
	keyboard navigation (tab, arrow) will cycle the focus through scraps and fissures. Press space to toggle (cut/heal, discard/restore).	
use glue	activate the glue tool in the property sheet, then click any overlapping region around the most-recently selected object. Glued objects will merge into the currently selected object. If they were not previously part of the selection, they become now.	
	keyboard navigation (tab, arrow) will cycle the focus through overlapping regions. Press space to toggle glue.	
while using a tool	*A skrim appears that darkens and freezes all but the most recently selected object.*	
exit any tool	click skrim	
	ESC	
scale individual object	pinch and spread gesture	drag and drop overlaid scale handles
move n islands individually at once	pinch and spread gesture	
move selection	drag and drop object (auto-scoll at edges)	

	reveal property sheet, then activate location input popup, then alter with arrow keys or input numeric value	
rotate selection	reveal property sheet, then activate location input popup, then alter with arrow keys or input numeric value	
rotate 2+ islands collectively	2+ finger rotate gesture	
Deselect all	click the canvas	
	with 'also deselect' or 'deselect up to' gestures	
	press ESC when no sheet is open	
undo	'undo' button in toolbar	
	Ctrl/Cmd+Z	
paste at avatar location	click 'paste' button in toolbar	
	Ctrl/Cmd+V	
reveal toolsheet	click the fold symbol on the toolbar	
	double-click the canvas	
	drag the toolbar up	
	focus the toolbar, then navigate down with arrow keys	



soft undo (restore previous selection)	'restore selection' button in toolsheet	
	Ctrl/Cmd+Shift+Z	
add guide	button in toolsheet	
switch to normal mode	'done editing' link in the toolbar	
	press ESC when no sheet is open	
	edit Url path from /edit to /explore	

pointer

is a hardware-agnostic representation of a 2D coordinate input. Check touch-action for fine-grained enabling and disabling UA gestures with CSS, and pointer-events for an overview of the JS side.

short-term mode (hotkey-persistent)

The Aqua HIG gives the example of a modifier key (Shift) to alter the behaviour of Drag&Drop. Which itself is also a short-term mode.

modifier keys

Keyboard maps and hotkey previews (for example on Ctrl/Cmd down) aid discoverability.

drag&drop

is in a bad shape on the web. Yet, it's ubiquitous in the WIMP paradigm and very nice for creating 1:1 relations and for arranging stuff in 2D. Check recepticle for potential drop targets and remember to design recepticle and dragging states when applicable. These drag&drop cursor shapes aid discover draggability:

`alias`, `copy`, **`move`**, `no-drop`, `not-allowed`, **`grab`**, **`grabbing`**.

Once we add contemporary multipoint touch input devices, we can implement lots of gestures of which drag&drop is only the most ubiquitous.

gestures

We have an ugly situation (as of 2021) where a **pinch zoom** on a touchpad will show up as a Ctrl+scroll event. Safari has its own gesture-event. There is a somewhat straightforward listener implementation. Gestures are timed, so Ui elements can change their appearance or even function during a gesture. For example, while zooming, all object overlays may go inactive or hide.

spring-loaded mode (mode-persistent)

This is a mode that is activated in synchrony with some global mode, and deactivated once the global mode finishes. The example for that is, of course, that once a drag short-term mode is activated anywhere, drop targets may switch into a spring-loaded 'recepticle' mode until the inevitable drop.

recepticle

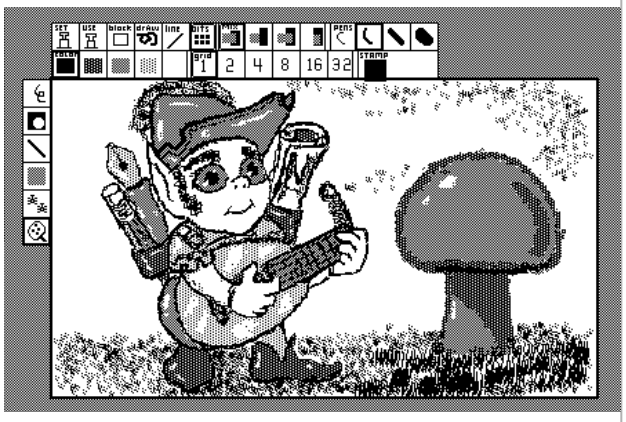
accepts a drop.

target mode

is rarely seen today. Uses include earlier designs of WYSIWYG and Hypertext editors where you would select an object, then a binary function, and then you'd need to specify a target object, or else you lose. For example, in CorelDraw you can subtract a path from another one. The mouse cursor would change and become very large in target mode. It was confusing.

tool mode (cursor-persistent)

When you have a palette, you choose your tool :-)



It is of note that traditionally, a tool mode may follow the cursor, i.e. here the state affects the mode! And the cursor may change its own shape depending on the current tool mode.

toolbar

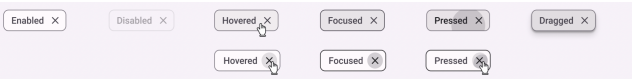
ordered set of controls pertaining to all the data below the cursor (including the page itself). If multiple objects are below the cursor, their respective toolbars are concatenated.

toggle button

In addition to the action that a toggle button controls (its status is one out of {Pending + Success + Failure + NotAsked}), its relation to the cursor implies more states, which are fully orthogonal to the data status. I will use **+** to indicate exclusivity/parallelity and **×** to indicate cumulative/orthogonal features. If every word in the equation is 1, you can count the cardinality, or number of permutations that a toggle button can possibly attain. Obviously, a touch interface differs from the traditional single-cursor model in several details (see touch).

```
toggle-button-mode
  = ( Focused + NotFocused )          2
  × ( Inoperative*                    1
    + Operative                       1
      × ( On + Off )                  2
      × ( action status = 4 )         4
      × ( Hovering
        + Pressing
        + Released
        + NotUsed
        )                             4
      × ( recepticle = 2 + Dragging
        )                             3
  )
```

The cardinality of **196** may seem excessive, but usually a designer would merge some states to arrive at asmaller number, as you can see in the common design systems. For example, focus may imply hovering.



* A thing to note is that 'disabled' traditionally removes a button from the tab ring, which often causes a11y issues, so we prefer aria-disabled (here calles Inoperative) which retains the tabbability and then we can mandate a tooltip to explain the reason behind inoperativity.

multitool

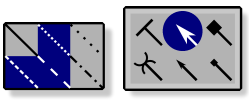
A multitool merges several toggle buttons in that either one in the group may be active at once. The data structure is a One-hole Zipper over a set.

A multi-select multitool would look like this:

Often, multitools are arranged in a row, and reside within toolbars.

palette

This is a grid-shaped version of a multitool, again either multi-select or single-select. Usually, palettes are the only control in a tool expansion.



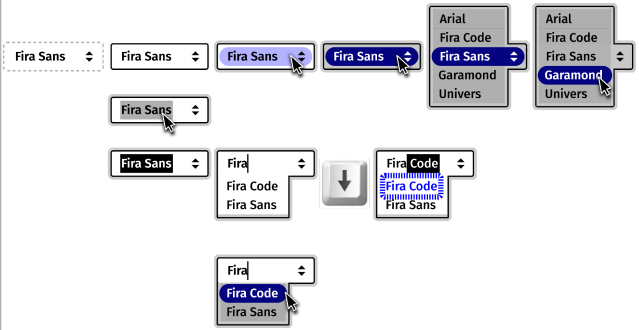
stickaround mode (volatile mode)

isolated to a single control within the Ui tree, and always binary. For example, a menu is sticky. Clicking a menu heading opens the menu and it sticks around until you click on a menu entry or around it, that is, on the implicit skrim. Stickaround things can be nested hierarchically, such as submenus or sub-dialogs. On the web, dialogs tend to be sticky, whether they are centered or pulldown or dropdown or pop-up or pullout or something weird. Overflows from toolbars and other Ui components are often sticky. On ShellCongress.com, all the windows are sticky. Examples for controls that stick around:

- pop-up and drop-down selector
- scrolling-list selector
- overflow
- most dialogs
- floating hints

pop-up selection

will choose among 5-11 exclusive options. Token may be editable.



overflow

is what you get when you put more than seven items in a toolbar.

stickaround dialog

Buttons that *toggle* a sticky dialog should not have bevels like the action and tool toggles. Instead, we can use a *typographic style* for them. If you remove the skrim and add a close-button, you get a manually collapsible dialog (property sheet).

Conversely, if the button is textual, we add an ellipsis (…) to indicate that it opns a stickaround dialog.

Examples include:

- volatile calendar (date picker)
- color picker

```
typographic-toggle-mode
  = ( Focused + NotFocused )          2
  × ( Inoperative                     1
    + Operative                       1
      × ( On + Off )                  2
      × ( download status = 4 )*      4
      × ( Hovering
        + Pressing
        + Released
        + NotUsed
        )                             4
      × ( recepticle = 2 + Dragging
```

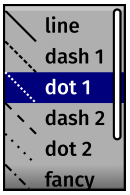
```
        )                                   3
    )
```

Like in case of the tool toggle, the number of combinations is **196**.

* In this case, status would usually to the loading of remote (or, in rare cases such as media processing, the parsing and processing of large local files by a webworker), and this process be be initiated without user interaction (preloading). Check the elm-pages library that we use for the implementation of deferred loading and progressive web apps (PWAs).

scrolling list selection

provided by the browser or device Ui toolkit.



skrim

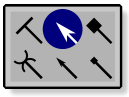
The 'rest of the screen' besides a stickaround item. In addition to a skrim, you can use the history API ('back') which often has convenient gestures such as swiping or hardware buttons, and the venerable OK button for the silver surfers.

collapsible mode (must close)

like stickaround mode, but without the skrim. Instead, a collapsible sheet or dialogue has a little close icon or proper OK and Cancel button.

tool expansion

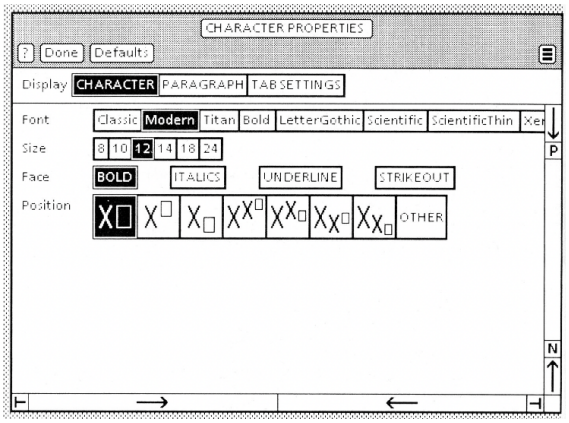
is a tool's backyard. Example: the brush settings dropdown expands the brush tool. TrueSpace had a nice Ui where you would establish sub-toolbars with the right mouse button. *Intention*: progressive disclosure.



property sheet

was heralded in the Xerox Star interface and offers a form-like representation of any object, in a collapsible sheet. Other monikers for this control are *Inspector panel* (as seen in creative Resolve) or simply *Dialogs* (inkscape).

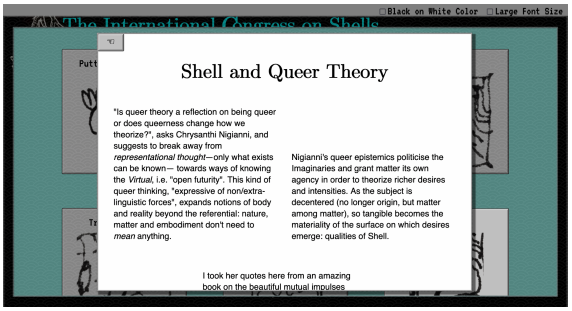
- There is at most one property sheet on a screen, corresponding to the selected object.
- indispensable in WYSIWYG apps and for tasks where a document has several sides, such as comments
- may be object-persistent, i.e. dismissed on deselect.



info window

displays a longer definition, a sub-scene or a preview of a link. So this fits a hierarchical domain organisation as well as a multi-path rhizome. In contrast to the property sheet,

- the info window has a skrim that covers the whole page and renders it noninteractive;
- it has its own URL (history API node), and dismissing it (through skrim, back or OK button) pops it together with all subsequent nodes;
- it can be nested (see picture below).



config mode (tab-persistent)

will persist throughout the tabsession (Url). Usually, representational modes are tab-persistent. In the recent years, icons and complex gestures have replaced the dedikated keys that would toggle config modes:

representational choice, or viewmode

An object may toggle through several representations. The classical example would be a document that can be represented by an icon, by a print preview, a WYSIWYG or markdown editor. The printout is not a mode because you edit it with nondigital tools. All digital representation modes are grouped in a configtoggle, which looks like a multitool but is smaller:

View:

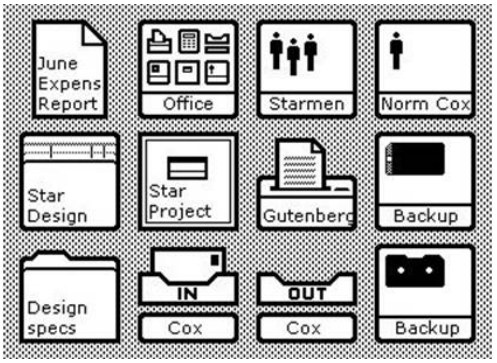
multiconfig

This is the variant for multiple choice:

Show:

iconification

Well, an icon can expand into just any other mode.



radio

single choice for tools and forms

- ◯ bright
- ◉ dark
- ◯ murky

checkbox

binary choice that does not trigger anything, only affects future tool-use or form submissions. For tools and forms.

- ☑ Hue
- ☐ Saturation
- ☑ Lightness

Consult this exhaustive tutorial by Stephanie Eckles for that covers ally considerations and CSS.

token

HIG: token; MD: chip. A chip group allows to assemble a *set* of predefined or user-generated tokens in a user-defined or preset order.

Politics | Household × | Music | Transport | Education | ...

The MD3 specs distinguish between

- Assist → start a 'smart action'
- Filter → compose a set of filters
- Input → collect data and add it to the set
- Suggestion → fill an input with this

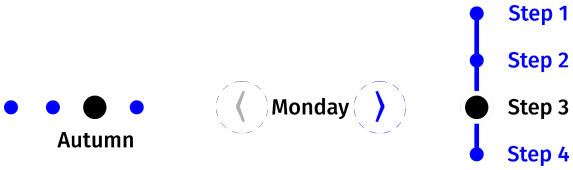
They are grouped and contextual, unlike buttons which are static and predefined.

sequential navigation (per Url-fragment)

through TOCs, paginated compositions, walkthroughs, carousels, or multi-page forms. This 'mode' thus pertains to a fixed place in the app (spatial) and represents a limited sequence.

stepped sheets or dialogs

have a stepper below or to the right, with links to all steps or just the adjacent ones,



TOC

is a tree-shaped, searchable navigation tool for hierarchical data.

multi-dataset navigation (tabbed, Url-parameter persistent)

used when different datasets can be edited in the same place. Implemented with tabbed windows. This 'mode' thus pertains to a fixed place in the app (spatial) and represents its data choice.

tabbed sheets or dialogs

ribbon (tabbed dropdown)

is not an interesting control. Basically it's just a tabbed configuration sheet.

site navigation (Url-path persistent)

would switch the whole screen. Each site can have a totally different layout. A sitemap represents the site navigation to our friends, the bots. Some typical sites include:

landing page

usually just has some links and animations, and lots of S.E.O.

preferences

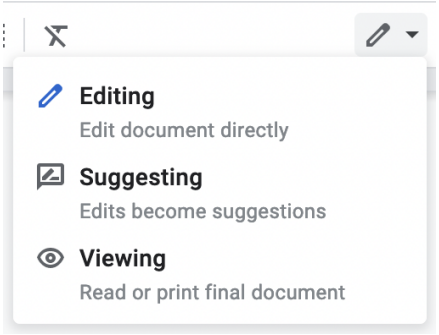
a separate page for the logged-in (or anonymous) user to change their own config. Each individual setting implements the setting mode.

global tool mode

Url: xyz/edit

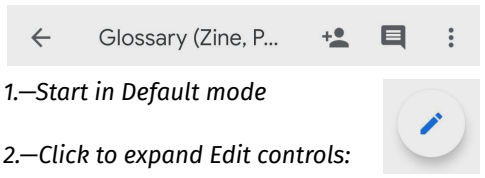
A blinking cursor or other affordance should make the distinction clear; otherwise a WYSIWYG composition would render in the same way across the modes.

Example: Google Docs, Desktop



Basic interactions (touch, modifier keys...) may be altered across global modes. In this sense, the different modes can be interpreted as global, i.e. page-wide, tools, with the cursor pointing to the Url as its object. In contrast, a tabbed property sheet would not alter the global hotkeys!

An edit mode can exist next to other modes, like in the desktop version of Google docs, or implement progressive disclosure pattern, as a tool expansion on the most global level (think 'global edit' as an *expansion* of the 'global view' *tool*), like in the *Google Docs mobile version:*



3.—Edit, then press Done ✓ to collapse them.

edit mode

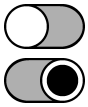
prime example of a → global tool mode

setting mode (user-persistent)

will persist across user sessions. It will not persist across different users because it strictly pertains to the Ui and not the data (that would be state). Settings include preference of input devices, accessibility aids, custom hotkeys, macros, toolchains, verbosity levels. Settings often have no obvious representational object on the screen, so a separate page or dialog bundles them. Settings may be migrated and merged across users. If a setting pertains to a specific tool, it may appear adjacent.

switch

in contrast to the radio, a switch is a binary data input, not a tool config. So it'll be synced asap.



object-contextuals

Many objects have some sort of contextual Ui attached, be it handles, contextual additions to the global menu, or toolbars. We discuss the controls that are displayed relative to the object representation under overlay. In contrast, the following controls are laid out as part of a page layout. The choice between overlay and fixed positioning is sometimes hard, as both have unique advantages.

- *Overlays are great for big-screen, mouse-device, graphical tools*
- *Fixed-position object contextuals such as contextual additions to the global menu or fixed-position toolbars are better suited on small screens and on handheld devices.*

non-dismissable, contextual toolbar

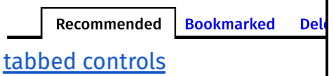
a row of mostly single-select multitools.

non-dismissable, contextual palette window

in principle, this is a grid-shaped toolbar, but traditionally, palette windows have also been used for such purposes as tool expansions or property sheets, but in these cases, they are dismissible.

mode controls

Which control to use to switch between modes depends on the mode behavior and the number of permutations. The following table lists the appropriate control for each, given *exclusive choice*:

exclusive	binary	3-4	5-11	12+
tool	 toggle button	 multi tool	 palette	 list
sticky	 dialog +skrim	menubar		
collapsible	collaps. dialog	ribbon, tear-off menus		
multi-dataset	 tabbed controls			TOC
sequence	 stepped controls			
config	View: 			
setting	 switch	 radio	 pop-up menu	 list
site	hyperlink			

If several choices can be combined, use these controls:

multiple-choice	2	3-4	5-11
tool	multiple-choice multitool		palette
config	Show: checkboxes or chips or multiconfig		palette or chips
setting	Hue, Saturation, Lightness checkboxes	Politics, Household chips	

Note to developers: You can find a great collection of controls that adhere to A11y best practices on Scott O'Hara's github.

focus

The first funny thing about focus is that it spans all elements of a page, including data (as far as it's selectable). The second is that selection and focus sometimes blur into each other. The third is that losing focus is called 'to blur'. Do you find that funny, too?

focus indicator

We distinguish a focused datum from a focused Ui control, and modern CSS engines distinguish users that need to see a focus indicator `:focus-visible` from those that don't `:focus`. While objects would require some sort of overlay Ui to indicate focus, Ui controls can be adorned with an extra outline or a cherryshade of color.

Deny Allow

The CSS-UI spec leaves the `outline` property to the user agents so they can implement something geared specifically to their environment. `outline-color: invert` is a unique value that may come in handy here. Outlines are drawn after the fact, on whatever is the pixels.

focus ring

either denotes the typical focus indicator *or* the tab ring.

tab ring

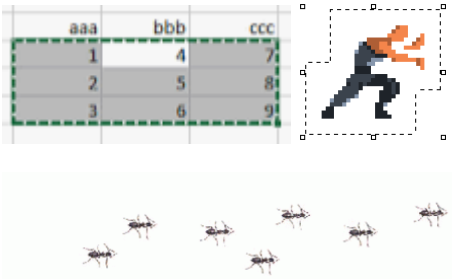
the order of elements that you can tab through in the browser is a ring. You can walk clockwise (Tab) or anticlockwise (Shift+Tab). tabindex determines whether or not a node is focusable. As by ipadOS guidelines in 2021, list items should be arrowkey-transversable while tab cycles through groups.

overlay

Within the materiality of data objects, Ui controls can be overlaid so that their proximity to the object conveys the target of their operation.

linear selection

marching ants selection



handles

HUD

Generic commands

Cut, Delete

Copy, Paste, Paste-in-Place, Duplicate

You know how these work. Advanced users may want to be able to peek into the clipboard, and even browse the clipboard history across sessions.

clipboard

is where copied and cut stuff lives. Handled by the system; traditionally quite forgetful.

(De/Re)activate

Undo, Redo

Find, Replace

Cancel, Submit

Bookmark

Comment on

Close

Alt Shortcuts

Data

datum

unit of data. Usually, you want to be more specific than that, though.

object

datum or composition that have a materiality. In addition, selections of objects are objects themselves.

selection

a subset of objects, being an object itself. Only one selection is 'active' at a time, although selections may be stored and recalled, and even represented among the objects.

group

manually gathered, potentially noncontiguous and potentially heterogenous subset of objects.

dynamic selection

momentary result of applying a filter on the totality of objects.

selection growth and shrinkage

are operations on the selection.

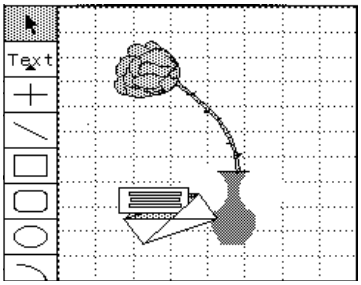
materiality

an emergent property of objects. Materiality is determined by dimensionality, extent, tools, and atom.

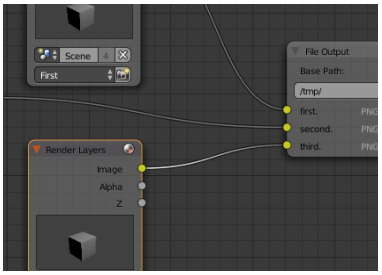
- **text** is linear, wrapping in a paginated 2D space (or through frames), tools are insertion and deletion, and its atom is the letter.
- a **pixel** graphic is 2-dimensional, clipped canvas. Its atoms are a pixels. Tools include painting and effect brushes, a fill bucket, and cropping and selection tools.

On a third dimension, we may have layers (with pixels, fills or text) and nondestructive filters as atoms.

- a **vector** graphic is usually 2-dimensional, infinite, and its atoms are vectors, texts, groups, and imported pixel graphics. Tools include matrix calculations such as shearing, scaling and rotating, and primitives such as squares:



- an **audio** wavegraph is 1-dimensional (or 2-dimensional, if frequency is the second dimension), clipped, and I don't know what its atoms are.
- a **compositing** graph is 2-dimensional, infinit, and its atoms are nodes and their relations.



dimensionality

is the set of axes that constitutes the object space.

extent

is the boundary of the axes of the object space.

contiguity

is adjacency over any of the dimensions. A contiguous selection is achieved with the hotkey *Shift* in the presence of a keyboard, and with a collapsible toolbar with selection tools on pure touch devices. Discontiguous selection is achieved with the Ctrl hotkey, respectively with little checkmarks in the corner of objects as overlays.

Let it flow!

stream

the data situation of a tab or a sub-item within a tab, at a time. Streams have a source and a window. The source delimits which data nodes are retrieved, and the window defines which ones are visible and accessible. Nodes adjacent to the window should be prefetched. The window should be encoded within the URL so that it can be easily shared. Sources can be substituted by filters. A stream is itself a source. A tab subscribes to a stream, and the p2p backend is responsible for pushing relevant updates.

window

the corners cropping streamed data. A time window would only show recent nodes in a blog, a geospatial window would crop the surface of the earth to a near-rectangle, etc.

filter

combinators over metadata selectors, full text search expressions, and sources. One filter outputs one source. Should be URL-encodable for easy sharing and manual composing. Add unary functions/currying, composition `∘`, and a pipe `▷` operator. Voilà une mini-language.

channel

a curated stream source.

hashtag

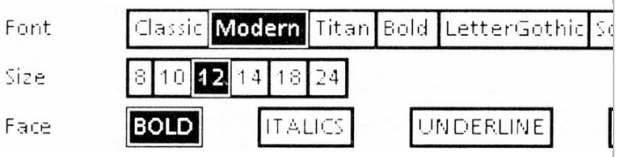
metadata for establishing implicit categories. Any item and channel may have n hashtags, where n should probably not be infinite.

States, Stages, Stadia

Temporal or spatial intervals structure the relations between

state

While this word is arguably used in many contexts, it may apply here to a momentary form of a datum. In contrast to a mode which pertains to a tab (persisted via URL) or a user, the state is eventually consistent for all users. Yet, state can switch tool mode, as in the case of the classical monument to bold/underline/italic which is described here and implemented there:



data controls

To manipulate data, we use different controls than for mode. This is because modes are limited to one user while data manipulation affects everyone. Conventionally, the button is the tool of choice for data manipulation as it conveys a sense of irreversibility and urgency. A button click functions like a stopwatch control in that it stores the present time Obviously, a button can only cause one-time effects, or, in the case of a togglebutton, intervalled effects. For everything else, we use tools (which are bound to the 2D cursor) or specialized inputs.

	data	validity
button	time × user	from now on
togglebutton		from..until
tool	vector	always
input	other types	

button

increments the state of an object immediately. may be cursor-persistent. Examples: “Send” (E-Mail), “Cancel” (Thermonuclear War). *Note: in Html,* `button`*s force an* `inline-block` *layout, i.e. they can neither be displayed full-width (block) nor in line with text in a paragraph. To cover these cases, you can create* `custom-button` *elements.*

togglebutton

toggles the binary state of an object immediately. Cursor-persistent. Examples: “Invisible” (toggling my avatar status), “Block” (an annoying peer).

tool

previously selected in a toolbar or palette. Cursor-persistent. Generates a vector or timed curve on a canvas. For example, a freehand selection, an inertia, a geolocation (on a map), a choice (in a puzzle game).

input

Cursor-persistent. Generates some datum, such as a time interval, a unicode character, or an HSLA Rec.2020 color. Or a new password. It may be smart to use the Html5 builtins.

target

is what a data control would affect. We strive to convey this relation through proximity. It is important that the target of an operation be unambiguous. For a general discussion of proximity and expectations regarding the placement and potential effects of a control within a page, see context. A great strategy is to limit the power of any control as much as possible. In the case of data, we call the object of a toolbar action or a property sheet the ‘selected object’, where the ‘object’ often is a group or a window into the composition rather than a specific data instance. The selection is implemented as an overlay.

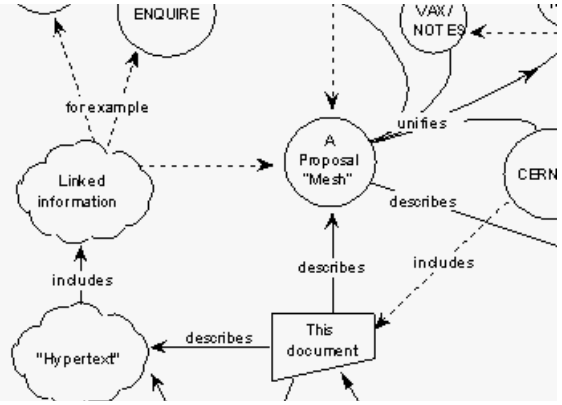
history

is the past of an object. You can indirectly bring back a past state by undoing the most recent tab action, but the history allows you to compare versions. Since each client experiences the mutations of shared data in a differen order (eventual consistency notwithstanding), none will record all potential histories. Thus, to share a history, it must register the mutation order locally.

Mesh

mashup

A Web 2.0 technique to combine different data sources or microservices at a single site. Tim Barners-Lee would have called it *Meshup*; his idea was that instead of ingesting data into a new system, the *mesh* (later: www) should instead live orthogonally to the existing webs.



status

Remote nodes in a web may be offline or online, and we may be waiting quite some time for their answer to a question we posed. Or perhaps we asked no question at all. Here is an Elm type.

```
type RemoteData e a
    = NotAsked
    | Loading
    | Failure e
    | Success a
```

acentrality and decentering

A mosaic does not have a central tile, no 'content' around which it is built. A fairy-tale is authored, and continues to be authored indefinitely, by a multitude, which itself varies over time and space, and the themes and tropes of fairy-tales and of art at large are leaking and osmoting between such multitudes. The various ways to lay out a picture before the canonization of central perspective had no fixed point for the onlooker.

Three metaphysical conventions — content, authorship, and perspective — were instituted in the times of what Marx called primitive accumulation, and which saw the establishment of the current form of capitalism.

eventual consistency

guarantees that all clients converge in terms of their shared data-model.

Design

Analysing existing apps

In this part, I am collecting a few sites' approaches to the Ui patterns listed above.

ultra.bs

https://ultra.tf/

is a simple demo where you can paste tiles of many types onto an infinite canvas, stored locally. Scrolling and zooming work as expected.

Verdict: This Ui is evil and makes any user's life miserable. Here is a list of sins instead of a feature analysis because nearly every feature of ultra is an antipattern...

There is exactly one tool mode, and it's an implicit and sneaky one, toggling between text range selection and text-tile dragging behaviour. The mouse pointer changes to a text-cursor on hover, but dragging does not select text but moves the tile. Only on press, the tile becomes editable. This is very weird: once you have dragged a text tile around, it invisibly switches into editing mode, and when you try to drag it again, the cursor instead selects text.

Since you cannot select anything, you cannot copy stuff out of ultra.tf. This makes that app a giant black hole.

Drag&drop an image from the OS shell works as expected, but Copy&Paste only pastes the filename.

Cropping images is restricted to one dimension (where images would remain centered), whereas videos are letterboxed on scale (in transparent color, which looks funny). The evil thing is that the chrome of the letterbox looks the same as the tile border, but when you try to grab it for dragging, you realize it's interpreted as part of the video instead. There is no way to restore the native aspect ratio of a tile.

All tiles are rectangular and have a border, which makes this unsuitable for collage, and more suitable for something like an album style composition.

Some tiles have an editable title above. Websites also have something like a subtitle, but that is not editable. Confusing!

Scrolling is only possible with the 2-finger gesture. A11y is totally missing from this app. That's evil.

On pointer-release, an item switches to topmost position. This is awkward, for example when you drag an item under other items and then it suddenly pops up.

Youtube videos don't bubble pointer events, so when you try to drag them, they start to play instead.

Scrolling is jitter-free on a MBA M1 whereas zooming stutters. Surprisingly, ultra.tf does implement the infinite canvas pattern without shooting itself in the foot.

There is no way to pause repeating animations. Very cyberpunk?

☒ (a boxed 'X'), established like 40 years ago as the conventional icon for closing a window, does delete a tile forever instead. This is eeeeevil.

relate

https://blog.relate.design/design-and-develop-at-the-same-time

is a WYSIWYG editor for web design which aims to cater for both developers and visual designers. It's still in development. They tackled the easy stuff first: purely visual elements. I wonder how well their approach will, in the future, translate to web apps that handle remote data. For now, it's an elegant proof of concept that eschews some of the historical metaphors and indirections that burden design communication the most.

PushPin

https://automerge.github.io/pushpin/

is very similar to zine in scope. UX is haphazard and incomplete. Backend seems to be the most solid out of all comparable apps (hypercore+automerge).

AffInE

https://github.com/toeverything/AFFiNE

Live demo: https://livedemo.affine.pro/AFFiNE/a3007697-7fed-5671-95d8-1159c886a5b4/edgeless

AFFiNe is a very ambitious project that will need a lot of work to be completed because it aims to integrate all project management models. It lacks basic functionality such as Copy&Paste, and instead of WISYWIG it opens an overlaid windows for rich text editing.

This is the most mature among those I tested, and it seems to have a solid internal model, but it uses a very haphazard Gui framework that makes me wonder why people in 2022 are reinventing WIMP conventions wrongly that exist since 1979. Here are some of the bugs I encountered in the first five minutes:

- When Scrolling, as soon as the cursor hovers over an overlay button, the scrolling is interrupted

- "More Actions" button does nothing when a paragraph is highlighted

- Some text editing features such as background color don't work and send the cursor to the end of the whole text

- Text editor overlay closes even when click on skrim is cancelled

- Sometimes I can't select a box, i.e. it doesn't get the edge boxes for scaling and it doesn't move on dragndrop: namely when I have started dragging a box in a box around. In this case, I also can't scroll inside the box.

- When choosing connector tool, I cannot select 'arrow' because it's occluded by tooltip 'line'

- When dragging cursor out of window during scaling operation, releasing mouse, then coming back, it continues scaling until I click somewhere

- Cannot undo delete a block

- Language dropdown cannot be closed

- Cannot select items on click

- Some dropdowns (for example line style selector) close when I drag out, which makes selecting a thick line nearly impossible

- In the paragraph format dropdown, entries are triggerbuttons instead of select options: the dropdown remains open

- Hand move tool has wrong icon

- Ctrl X cuts, but there is no paste functionality, so it's effectively gone

- Objects that can't be rotated still have the rotate knob visible

- Text cannot be moved around (you need to grab a potentially invisible border)

- It is very hard to find out how to create lines: first click start, then click end, instead of conventional dragndrop drawing.

- Overlay buttons can occlude scale knobs

- performance can get laggy when scaling large objects (even on an M1 macbook air)

- minimap is excellent though

PicCollage

commercial software. Native (android&apple). Appropriate use of Ui controls hooray. No infinite canvas. Scaling and rotating with two-finger gesture as well as a large dedicated overlay-knob (top right). Two-finger gesture also triggers when one finger is out of target area, which can be confusing. Drag&drop with one-finger gesture. Subtle snapping. excellent integration of video and photo. Clumsy modal layer management (one layer per photo; dragging reference items in a list; no swapping). Delete via trash-zone (top right of scene). No way to select lower layer items. Freehand vectors are lo-poly (see screenshot).

During 2-finger scale&rotate gesture, the top toolbar is greyed out but the bottom toolbar is visible. The cyan rectangle in the screenshot is actually a knob to change the width of the text box. Text is centered and behaves like a block in the css 'flow' model.

Based app design

Designing is always creating spaces for cultural **practices**. These practices are **virtual**: they don't need to be realized to be there, but they are more than only potential.

There are as many approaches to design as there are possible spaces. The following ontology is somewhat arbitrary; any project will leverage many approaches, mostly unexplicated. The ontology may help when you hit a roadblock or feel like your process is steering in a weird direction.

Protocol-based

A traditional, **top-down approach** to design might initially envision a linear process as the targeted practice. Then, a **protocol** would open a linear space wherein the practice can take place. Thile this approach is the most straightforward for hierarchical institutions to produce, and is indeed the most ubiquitous, it has three widely-acknowledged fallacies:

- **Protocols are brittle**, to the extent that they break at some point. Polyglot designs such as the federation will prevail.
- **Tunnels are frustrating for people**. Linear spaces reduce my agency and make me cynical.
- **Systems evade understanding**. Which means that any tunnel will leak in weird places, inviting capital-destroying creativity and fun.

Practice-based

Building up from the practice itself, it is easier to find and assemble best-practices and delightful patterns. And since the agency space is not artificially sealed, such apps tend to bud unforeseen practices even years after their creation. Even very tight spaces can provoke such practices, as in the case of emoticons (BBS) and image-based

memes, given a creative community and joytime. But this approach is limited:

- **Practices are hard to invent.** Often, a practice is retconned into a technology that was invented with a different practice in mind, and new technologies that capitalise on practices that grew in the milieu of older tech feel contrived and even jarring. In this sense, a bottom-up app is always weirdly nostalgic.
- **Practices that exist already have their app**. Why even create a new app when a practice has developed in the bosom of the old system?
- **Designers don't understand practices**. This has many reasons, one of which is that designers are solitary whereas practices are always, to some extent, collective. Collective design is nontrivial.

Community-based

Community-based design (not community-driven!) is the collective design that actually is always happening in any community. A designer may decide to create some tools to augment the agency of the collective in terms of iterating on their design. The outcome of such a process is absolutely unpredictable, and may be quite amazing. If apps had good feedback loops from communities, they'd become as common and indispensable and invisible as all the contraptions we are so used to: knives, storytelling devices, bowls, consent techniques, costumes, protective clothing, tents.

Controls for a comprehensive and restrictive GUI

This section explores a schema for adding a GUI on data. Conventionally, a designer tries to understand the data model and the workflows, and then selects appropriate pages, objects and control fields. Since this is a chore, and designers reinvent the wheel all the time, but they miss so many important details, we want to make this process very restrictive. What does a library need to know about data and workflows in order to generate a comprehensive and appropriate Ui? Traditional approaches would create endless forms, but we can do better — with the accumulated wisdom of 40 years of WIMP.

Persistence Models, Four Dimensions

Any application maps to n persistency models, where each model, pertaining to a persistency tier, consists of objects that interact with each other, but can't affect objects from other dimensions. Each such object is projected to the client in four dimensions, which are cycled with Tab:

Handle: An iconic, or proxy representation of the pivotal object in each persistency tier. For example, the current user, the selected text, the selected shape in a drawing, the current workspace layout, the clipboard, the multiselection group (on a small screen), the sync-state or database connection, the locale. These are usually icons on top of the page and either offer drop-down fields or switch to a different global view.

Scene: Objects that can be shared among users in a file, and that are considered the creative output of an app. Displayed in the central region of the screen.

Control: Object properties, tools and tool properties. Typically displayed below or above the scene. On large screens, config windows may appear as overlays, context menus or HUDs along the objects of the scene, but in a higher z dimension.

Info: Toasts and Snacks and the like. They spawn from top to bottom, on the right side, or they get their own 'status bar' (what an ancient thing!)

In the API, we call the vector of four dimensions in a specific persistency tier a Gui, and we allow nesting lists of Guis in each dimension so that we get a fourdimensional tree.

Example: a single-user Collage app

A *User* has a **handle** with their avatar, a **control** field with their settings (some of which, for example locale or workspace layout, may have their own handles) and for logging out and switching users, and a **scene** which contains a single collage.

The *Collage* has a **handle** that allows choosing a different collage to work on, or to share or transclude it. Its **scene** contains a set of positioned shapes, and potentially a subset of selected shapes. A collage **handles** its own database connection (Sync).

This *DatabaseConnection* asynchronously syncs with the backend and shows its own **handle** to indicate whether it's online or offline. A **control** can offer options to duplicate, delete, de-offline its data.

The optional *Selection* has a **handle** for expanding or cancelling the selection, and a **control** to group/ungroup, copy, delete, paste/replace or cut it.

A *Shape* has two possible viewmodes: selected or unselected. In both modes, it has a scene which renders its contents. In the selected mode, it will also spawn a **control** field with options to choose and modify color, outline, z-order, distortion, etc.

Viewing a global Gui requires an integration step, and a 'GUI state'. Each handle doubles its tree's cardinality. An active handle occludes the dimensions of its sub-trees, so for example, clicking the avatar will show only the log in/out controls. Active handles on the same level co-exist, so for example Sync, Selection and Shape are direct descendents of Collage, and their respective scenes, controls, infos and handles will co-exist. You can define other cross-relations by altering the constructor depending on the handle state, as is the case of Shapes, which will only be editable if either one or some are selected (see the pink highlight in the table below). In our example, we get the following tree (where → denotes nested GUIs):

	Handle	**Scene**	**Control**	**Info**
User	Avatar	→ Collage		(Server response)
	Avatar		Log in/out, ...	
Collage	Current Collage	Shape*s* + → Selection		
	Current Collage	(→ My Collages)	Share, ...	
	→ Sync			
Sync	Online Indicator			(Server response)
	Online Indicator		Persistence settings	
Selection	(None)			
	One		Copy, paste... +	
	Some		Copy, paste... + Group/Ungroup + Align	
	Add/Remove	(overlaid checkboxes)		
Shape		(its contents)		
Selected Shape		(its contents + scale-overlay)	edit properties	