Jesse Freeman · Follow
Jul 4, 2020 · 12 min read ★

What the Apple Newton taught us about UX 27 years ago

One of Apple's biggest failures was years ahead of its time.

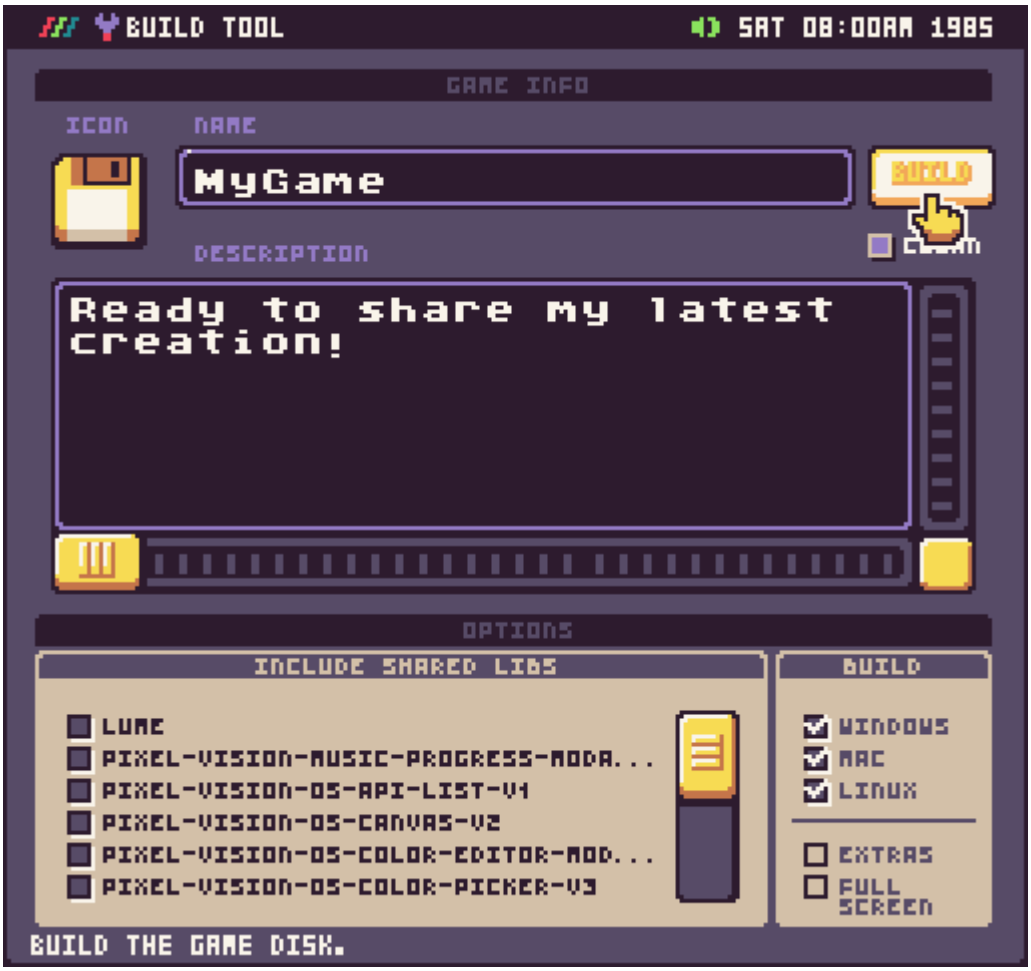
I have always loved the UI of Apple's Newton. Considering that nothing like it existed at the time, it's incredible how well thought out everything was. Apple doubled down on the unique handwriting recognition, which eventually became the Newton's undoing. Also, Steve Jobs famously hated the use of a stylus and, for a multitude of reasons, killed off the project when he returned to Apple in 1998.

> It was the stylus. I killed the Newton because of the stylus. If you're holding the stylus, you can't use the other five that are attached to your wrist.
>
> - Steve Jobs Movie, 2015

While there were multiple reasons for shutting down the Newton project, some of its more unique UX paradigms exist even today. Apple had a long history of building pixel-perfect UIs for black and white screens, so it's no surprise they were able to pull off a simple design aesthetic

that was consistent across all of the UI. The Newton's OS had the personality of the original Macintosh and leveraged the hardware limitations the best that it could.

When I started working with Christina Antoinette Neofotistou to design Pixel Vision 8, I referenced a lot of the Newton UI for inspiration.



Pixel Vision OS's Build Tool, which shares similar components found in the Newton's OS.

Pixel Vision 8 is a Fantasy Console, which is a self-contained game development environment reminiscent of early game consoles and computers from the '80s. The most popular Fantasy Console, Pico-8, is similar to a Commodore 64 in the respect that when you boot it up, you are presented with a command line to get started. As a child that grew up using a Macintosh, I wanted Pixel Vision 8 to be a hybrid of the early Mac OS and the Newton.

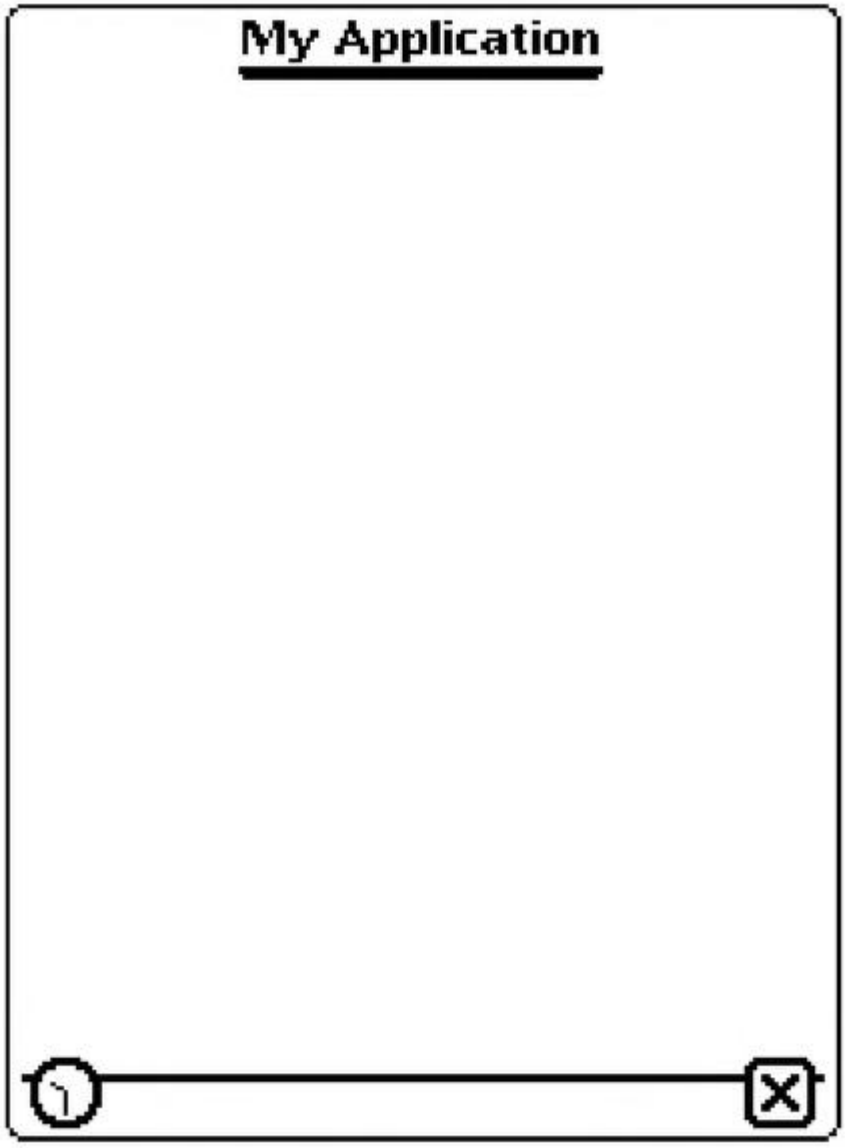
I thought I'd share a collection of reference material that inspired me to build an OS from scratch for Pixel Vision 8. A lot of these images come directly from the Newton 2.0 Programmer Reference Guide. It is incredibly detailed at over 1,390 pages long.

Let's take a look at all of the UI components that not only allow users to interact with Newton applications but also helped give the device its unique look and fee.

. . .

Application View

The **proto** view was the building block of the Newton's base application. The Newton used a series of screens to display an application's UI, just like we see today on mobile phones and tablets. The default view would have a title at the top and nav bar at the bottom.



In this example, there is a small pop up clock button on the left and a close button on the right. Since the Newton had a dedicated navigation area at the bottom of the screen, most navigation UI was towards the bottom of the application's view.

Headers

You can add a **header** to the application. This usually consists of an icon on the left, some text, and a horizontal line that goes across the rest of the screen.

Figure 3-11 A page header

This helped display the title, a custom title, and could even have additional icon buttons for contextual actions.

Figure 3-12 A roll header

For all of the forward-thinking that went into the Newton's UI, there were no rules around how an application could look or feel. So most developers implemented custom looking layouts, which meant that things we now take for granted like a consistent location for the time, date, and notifications did not exist.

Folder Tab

Applications could also make use of a **folder tab** that went along the top of the view.

Figure 3-7 The plain folder tab

newtClockFolderTab

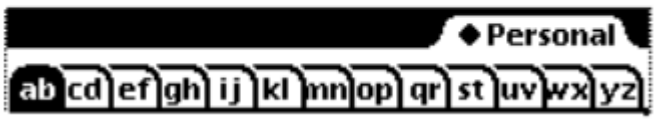
This folder tab incorporates a date and time indicator. It is automatically updated if the current folder is deleted. When the user taps the folder tab, a picker containing the list of folders available to your application displays. If you want filing to operate correctly in your application, it must use either the `newtFolderTab` proto or the `newtClockFolderTab` proto, shown in Figure 3-8.

Figure 3-8 The digital clock and folder tab

While this is commonplace now thanks to web browsers, back in the early '90s, this was a new concept. The newton actually implemented this as a drop-down menu instead of the way we know of tabs today. To make use of the space, developers could also include the clock, which wasn't a fixed UI element along the top of the screen like we have today.

In addition to the folder tab, developers had access to sub-tabs for jumping between data views in an application. Here is an **AZ tab** navigation example typical on the Newton.

Figure 3-6 NewtApp A-Z tabs



Since the Newton was limited in how many pixels it could display on the screen with its 320 x 240 pixel resolution, designers had to cram in as much as they could. Here you can see that the tabs automatically increment two letters at a time since it wouldn't be possible to display 26

individual tabs at a time. Some letters actually touch the border of the tab because there simply isn't enough space to have consistent padding.

This wouldn't acceptable with today's UI design. Most likely, tabs would be missing to allow for scrolling left and right, so each letter would have the same spacing and padding.

Manu Bar

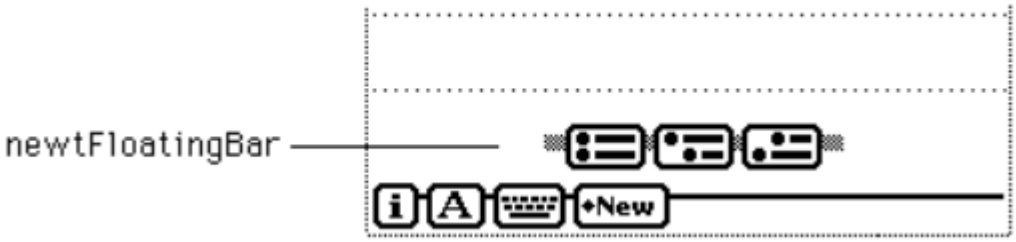
As we saw earlier with the default application view window, the bottom of the screen was reserved for an **application menu bar**.

Figure 4-1 Calls application menu bar

There was always an information button on the left to display help pop-ups for users. Next was usually the font and keyboard input options. Then it was up to the developer what they wanted to display. Since there was no drop-down menu like you would have on a desktop computer, most actions were buttons in this menu bar.

When there wasn't enough room to display all of the actions in the static menu bar, developers used a floating bar with additional options.

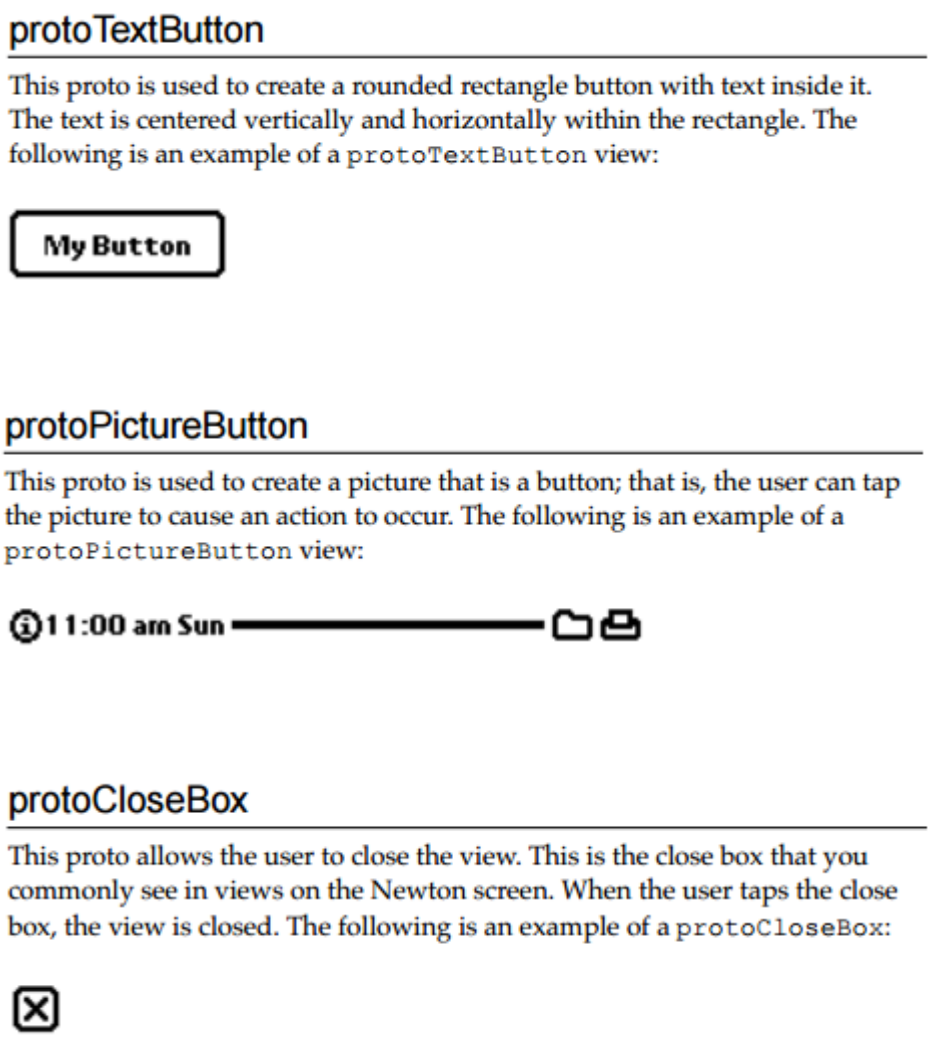
Figure 3-10 A floating bar view



Finally, all applications needed a way of closing, so the exit button was always the last opinion along the far left of the menu bar. A lot of thought had to go into minimizing the actions of an application depending on what kind of buttons you used in the menu bar and how much space was available.

Buttons

Buttons are the building block of any UI, so it's not surprising that the Newton supported several types of buttons from simple text to borderless icons.



One thing that was rare on the Newton was compound buttons that combined an icon and text. This was probably due to the limited space on the screen, especially when it came to action buttons in menus.

Input Fields

I could write an entire article on the Newton's text input components. The most basic one was just a simple input field with a label no the left and a dotted line indicating it could be edited.

Figure 3-15 A NewtApp label input line

Because of the Newton's unique handwriting recognition, there were some additional options that became available when access text input fields allowing you to toggle between script or the virtual keyboard.

Radio Buttons

The Newton had **radio buttons** developers could also include in their user choice options.

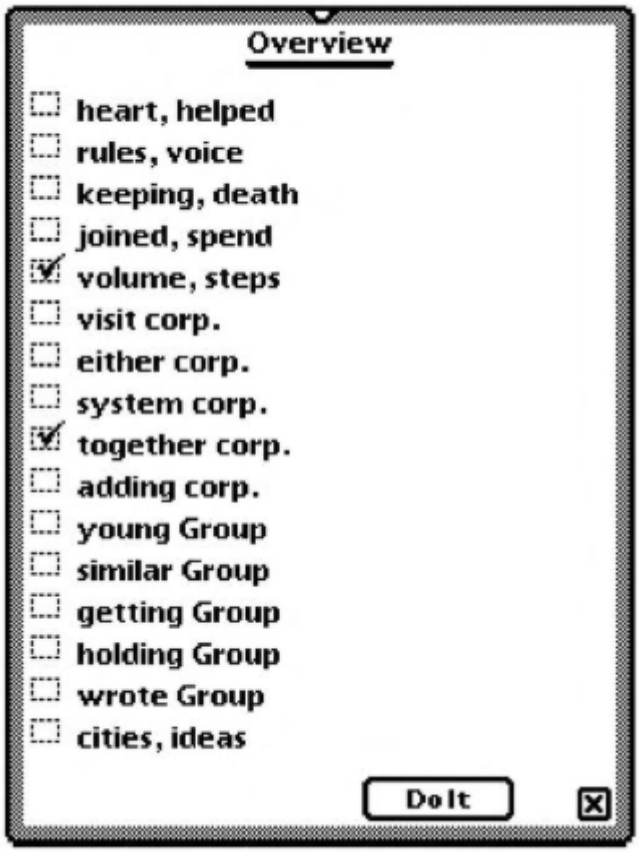
bull's-eye when it is selected. It is labeled to the right with a text label, as shown in the following view:

○ 9 pt
○ 10 pt
○ 12 pt
● 14 pt

These radio buttons were minimalistic, using a dotted outline for an unselected option and filled circle for the current selection. Today, the dotted line circles would read as disable or not selectable, and I wonder why they didn't go with standard circle outlines? Perhaps with all of the other think outlines, it would have been too difficult to stand out on a crowded screen.

Checkboxes followed a similar design with dotted rectangle outlines for unselected options and a small checkmark on top for selected.

Figure 5-39 Example of a soup entry proto

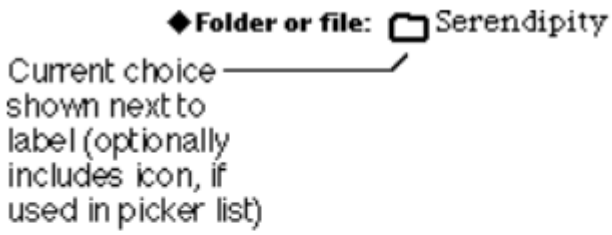


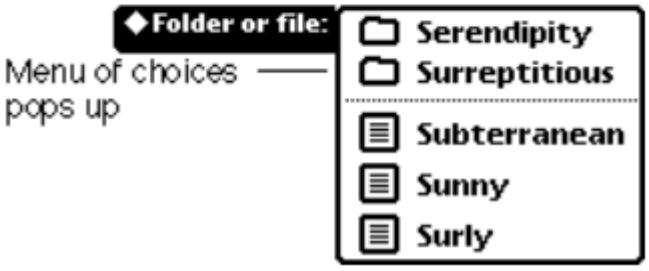
Pickers

Pickers are usually a combination of a button and some kind of scrolling list. The Newton had support for pickers as well, and these often took on the appearance of what we would consider contextual pop-up menus that appear when right-clicking on today's computers.

These pickers were always triggered by a button and could contain icons, text, checkboxes, and even horizontal dividers.

Figure 5-3 A `ProtoLabelPicker`





Based on the location on the screen, the picker would appear below, next to, or above the button.

Figure 3-1 The Information button and picker

The following methods provide default handling for items in the picker menu of the `newtInfoButton`.

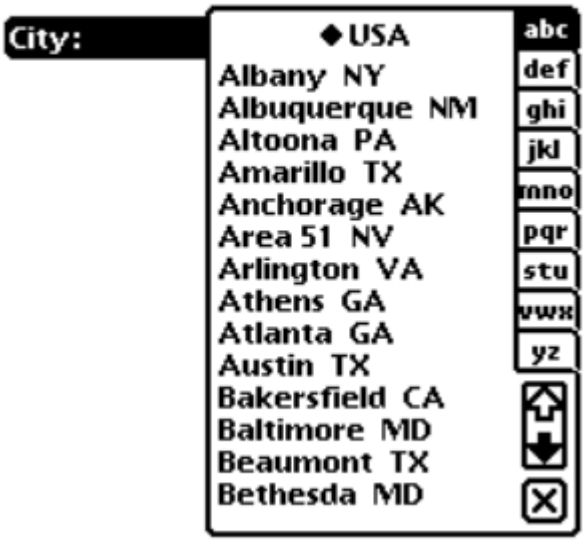
The Mac was well known for its use of icons throughout the OS, so the Newton leaned heavily on clear iconography wherever possible. I like this example of a person, company, and group used in conjunction with the text to add more personality to the picker.

Figure 4-2 `newtNewStationeryButton` in Names

Finally, additional UI elements could be combined with pickers to create more complex interactions. In this example, there is a title, a list of text, sorting tabs on the right, scrollers

below, and a close button.

Figure 5-23 Example of a city picker

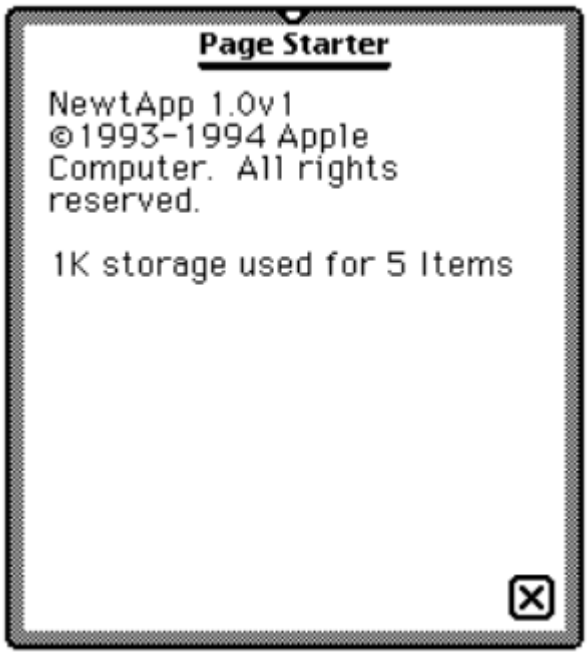


Today this would be too much UI for a single picker, especially since the close button is redundant. Now we expect to close a pop-up picker by clicking outside of it or making a selection.

Modals

Modal and pop-up **views** were a big part of the Newton's UI. These could be simple display screens telling you about the current application or show some kind of help message.

Figure 3-2 The NewtApp About view

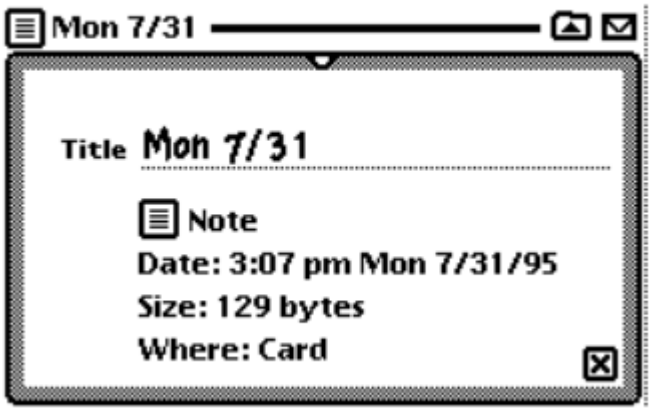


Modals could also promo the user for action or modify settings.

Figure 3-3 A NewtApp Preferences view

While a modal could contain most of the standard UI components, a typical application was only able to display one at a time.

Figure 3-13 A NewtApp Information slip



Finally, the border, title, and close options were consistent across the OS.

Scrollers

It wasn't particle to expect the developer to be able to display all of the content a user needed on a single screen at a time. So the Newton had a **scroller** that could be tied to lists for paging up and down.

protoUpDownScroller

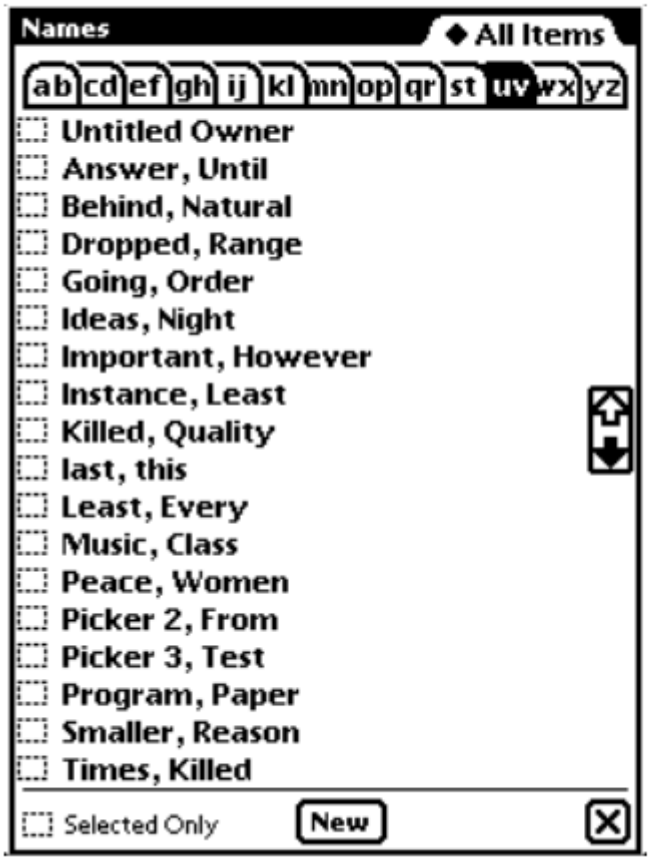
This proto is used to include up/down scrollers, centered at the right side of a view. The following is an example of a `protoUpDownScroller` view:

Scrollers typically sit to the right of the UI component they control. Unlike desktops operating systems, these scrollers didn't extend to the hight if the viewport.

Figure 5-7 Scrollable list of shapes and text

While this may have been efficient in small lists, using these scrollers for lists taking up the entire screen was not ideal.

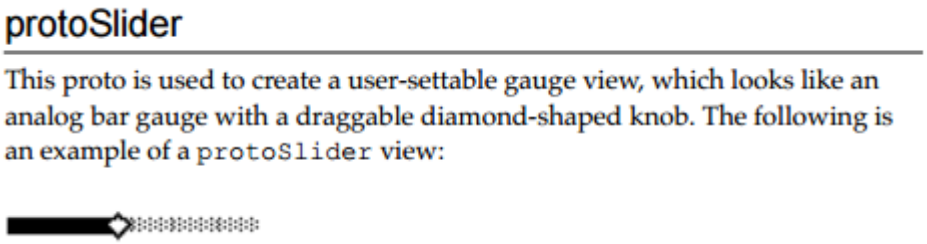
Figure 5-40 A `protoListPicker` based on `protoPeopleDataDef`



There was no "smooth scrolling" on the Newton. Lists refreshed slowly, line by line when you scrolled up or down, making this a chore in most applications.

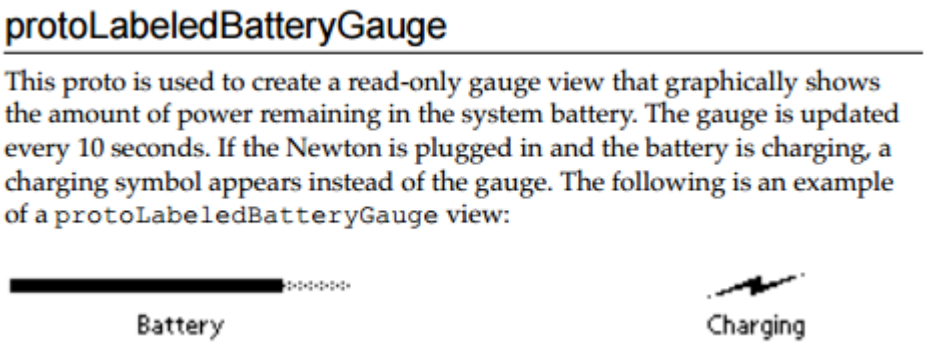
Sliders and Progress Bars

Visually, **sliders** and progress bars were identical. The only difference being the handle on the slider.



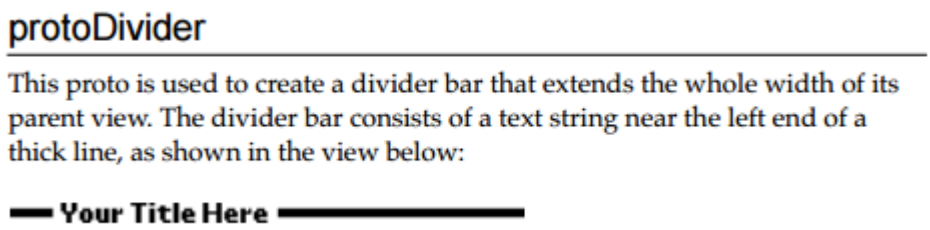
One downside of the slider, and most of the Newton's UI, was the size of interactive elements. In the case of the slider's handle, it was difficult to click on even with the stylist. Components requiring dragging tend to be a bit frustrating to use, especially with the Newton's slow redraw rate.

For progress bars, used in the battery gauge example, the slider's handle was removed and managed programmatically.



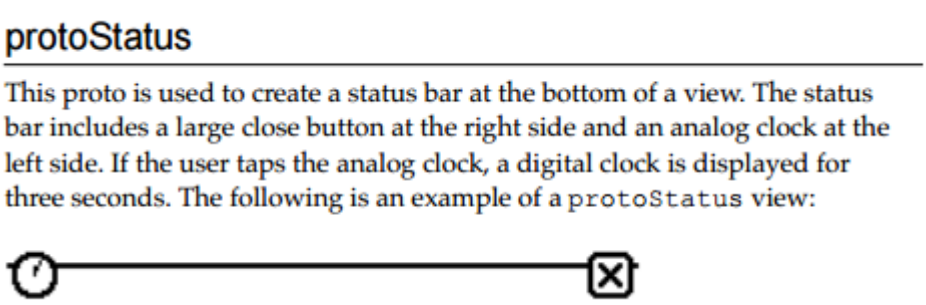
Dividers, Titles, and Status Bars

While I touched a bit on some of the standard title bars an application could display across the top of the screen, there some additional UI components a developer had access to for breaking up the visual flow of a screen. Even back then Apple knew the importance of grouping UI together in order to create a readable visual hierarchy. All of these elements shared a similar design aesthetic stating with the divider.



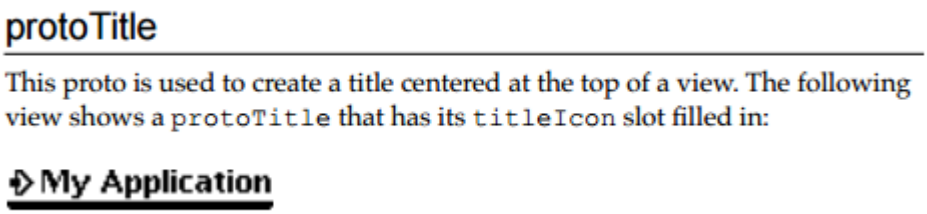
The Newton UI used a lot of heavy horizontal lines to separate elements on the screen. The **divider** was unique in that it allows text to in the middle of it, unlike the traditional horizontal rule we use in web design.

The same design aesthetic applied to the **status bar,** which we saw in the beginning.



Unlike the divider which breaks around the text, the status bar's icons sit on top of the line. This was a strange design choice and made these status bars feel a bit cluttered when a lot of options or icon buttons were in them.

Finally, there was an individual **title** that supported a small icon, text, and a line below the two.



I wasn't able to tell if you had any control over changing the thickness of these divider lines. It appears that each component had a set line thickness, and it was up to the developer to place them according to what works best for the visual weight on the screen.

Fonts

The Newton had an impressive collection of built-in **fonts**. Users were even able to adjust the size of the fonts as well. In total, the Newton had around 12 or so fonts that were common on the Mac OS of the time.

Also, developers had access to similar font display options you'd expect today.

Constant descriptions

`kFaceNormal`	Plain font face
`kFaceBold`	Bold font face
`kFaceItalic`	Italic font face
`kFaceUnderline`	Underlined font face
`kFaceOutline`	Outlined font face
`kFaceSuperScript`	Superscripted font face
`kFaceSubScript`	Subscripted font face

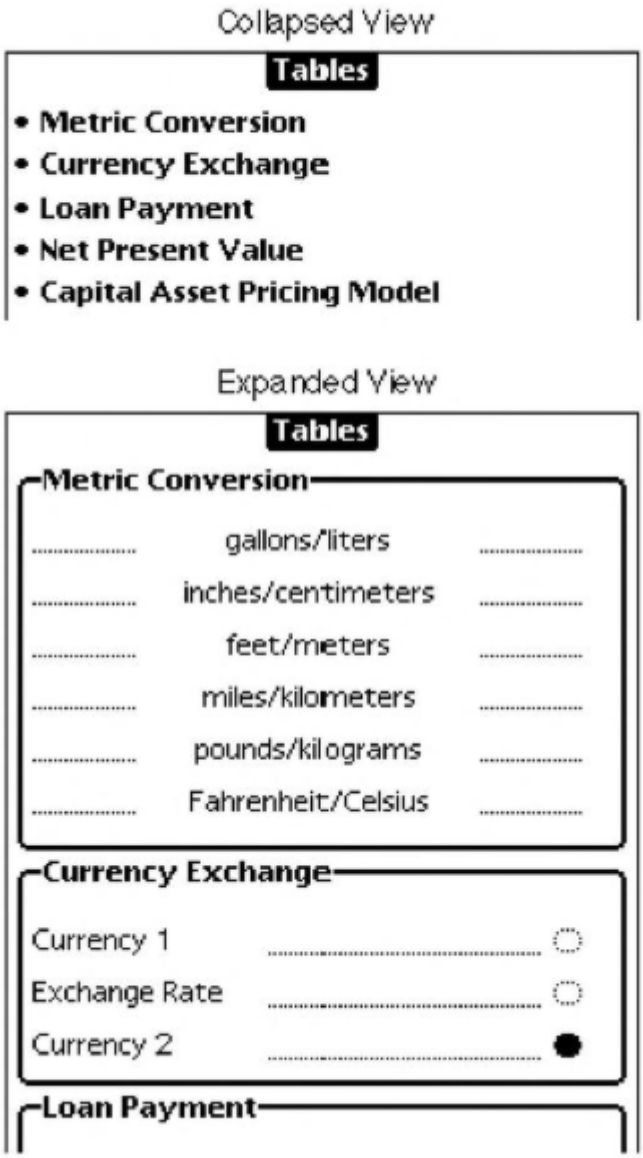
While some of these fonts were better than others, text elements in the core UI components all shared a similar font and set size based on the default design of the component. As far as I know, these could not be changed, and there was no way to increase the system font size for people without 20/20 eyesight. Overall, the UI fonts on the Newton were tiny to display as much information as possible on a single screen.

Everything Else

There were a lot of additional UI components I didn't go into detail above but wanted to call out quickly.

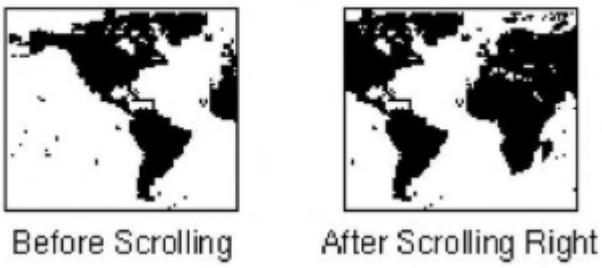
There were different types of expandable and collapsible **rolled lists** as they called them:

Figure 5-42 Example of a collapsed and expanded rolled list of items



Outside of accordion lists, there were **scrollable images** that were masked off inside of a viewport:

Figure 2-1 `SetOrigin` example



For tabular data and help with laying out content in a view, developers had access to **tables**:

Figure 2-2 `LayoutTable` results

1	2	3
4	5	6
7	8	9
10	11	12

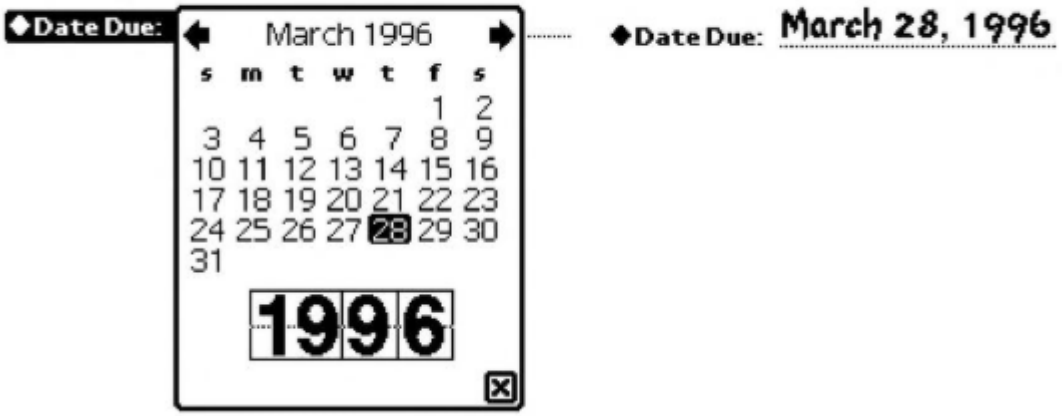
The *columnStart* and *rowStart* parameters are set to 0,0

5	6
7	8
9	10

The *columnStart* and *rowStart* parameters are set to 1,1

And finally, there were several ways to visualize the date and time from simple **calendars**:

Figure 3-18 A NewtApp label date input line



Along with analog and digital **clocks:**

Figure 5-33 Example of an analog time pop-up view

Figure 3-22 Time input with picker-only access

You can start to see the early shift toward skeuomorphic design in their UI. The digital clock example is reminiscent of an old alarm clock radio that used number cards attached to a wheel to display the time. You can see this reference in the dotted line across the middle of the digits.

. . .

I know this was an exhaustive list, but if you are going to design a UI from scratch, you will need to cover almost all of these components and maybe more. Even 27 years later, we still use the same standard primitive components:

- Buttons
- Input Fields
- Lists
- Scroll Bars
- Sliders
- Pop-ups

This was a massive inspiration for working out everything I needed to create to build a complete OS from scratch. In the end, Pixel Vision OS deviated from the Newton's design to stand on its own. Even today we owe a lot to the early UI innovations in the Newton to what we currently see on our phones and tablets. Maybe the Newton wasn't as bad of a failure as history will lead you to believe?