Appway

Workspace Design Notes

Introducing Workspace Design Notes

If you've used PowerPoint, you know that the application has two different modes: the "edit" mode - where you see a list of slides, and can add, edit and remove slides as required; and the "slide show" view, where you actually present your deck. Activate the "slide show" mode, and the normal application window disappears. PowerPoint takes over the whole screen, and starts showing the presentation you've built.

Appway works the same way.

The Studio is where you build the solution. Like PowerPoint's edit mode, this is where you create and delete Business Objects, and make changes to them. This is where you build your Appway solution.

The Workspace is where you run the solution. Like PowerPoint's presentation mode, this is where you see your work in action, and where end users access your Appway solution.

Appway's Studio is mostly used by trained developers. The Workspace, on the other hand, is used by a broad range of company employees and company clients. Employees may be new to the company, and not know exactly how everything works yet. Clients expect clear, straightforward forms that do not require them to be in constant contact with the company in order to figure out what is required of them. Employees and clients alike want the same ease of use from Appway as from any other app they use on a daily basis.

This means that Workspace usability is extremely important. Whether a company has built a short simple solution or a large, complex solution, everybody – regardless of their skills or training – should be able to use the Appway Workspace successfully.

That's why we invest a lot of time into getting the Workspace's usability right. Appway 6.2 ships with a completely new Workspace design, and includes some completely new components that can be used to build Appway solutions.

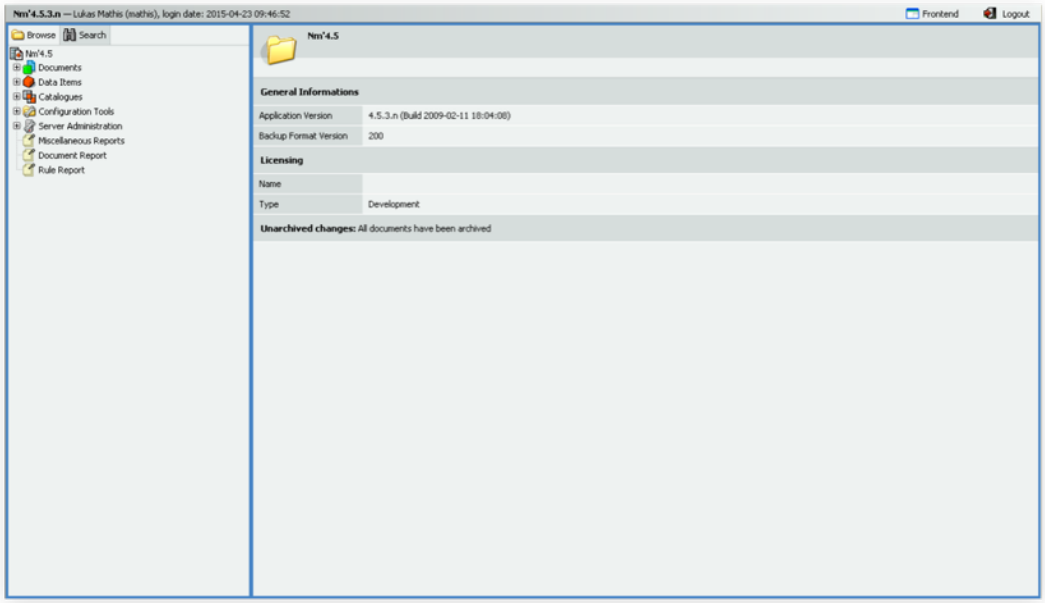
In the coming weeks, we will talk about various different aspects of the new Workspace design, and introduce the thinking behind some of the changes we've made.

Workspace Design Notes #1: Back to the Future

Let's take a trip through time, to the beginning of history. The beginning of Appway history, that is.

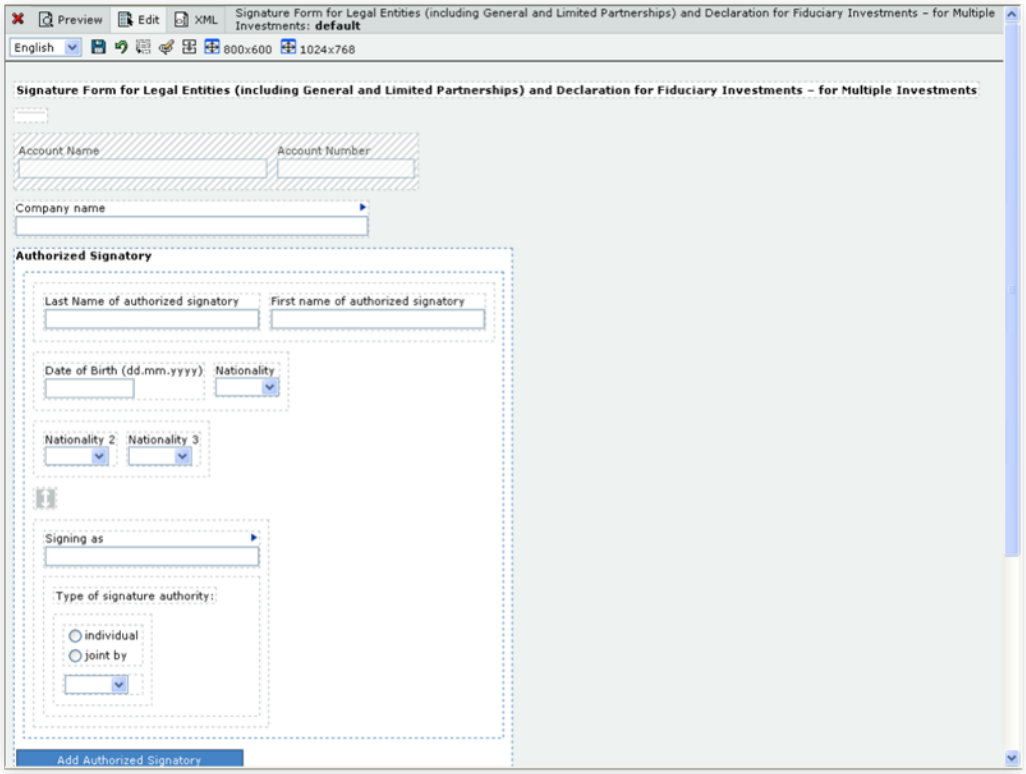
Image courtesy L. Mathis ©

Close your eyes and imagine that, in the beginning, there was no Appway. A decade ago, there was only Nm'4, the precursor product to Appway.



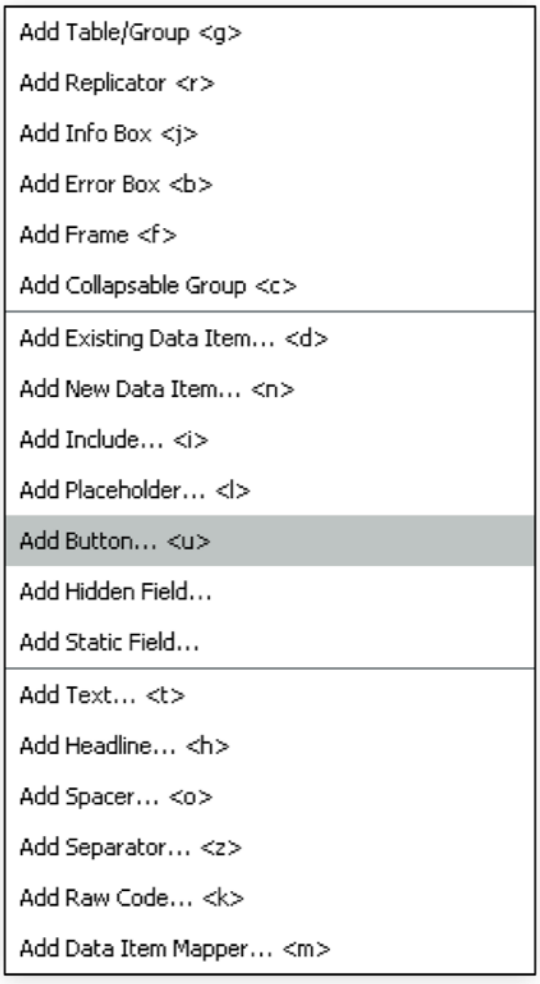
Nm'4 didn't have what we now think of as Business Objects. It had something similar, though. The "things" in Nm'4 were much less generic than the ones in Appway, and much more geared towards specific use cases. For example, in Nm'4, the portal was pretty much hard-coded, and looked the same in every installation.

Instead of Appway Screens, Nm'4 had something called "Input Forms".

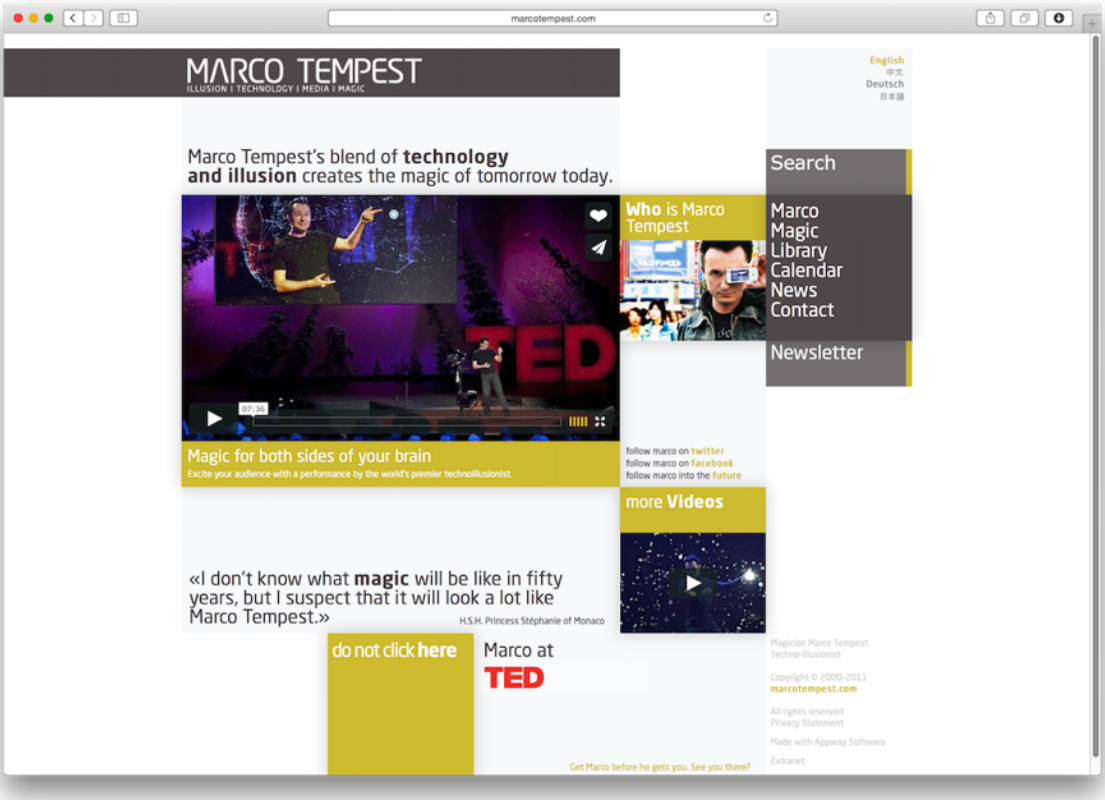


Nm4's Input Forms were much, much simpler than Appway Screens; basically, they only offered controls (text fields, dropdowns, and so on), some basic layout elements (headers, separators), and the "Replicator", a very simple logic element. Nowadays, we're used to Appway's exhaustive list of Screen components, from a simple text field all the way up to a complex border layout, but Nm'4 only provided a scant 19 different "components".

When we originally conceived Appway, we came up with the idea of a generic Screen Business Object. Pretty quickly, we figured out that this meant that we could replace all of the hard-coded stuff in Nm'4 with dynamic Screens; there was no need to have "portals" and "forms" and other such things in Appway. We could have one Screen Business Object, and implement all of Nm'4's specialized parts as Screen Business Objects.



This has many advantages. It makes Appway solutions much more configurable and flexible than Nm'4 solutions ever were. It moves stuff into the model that was previously hard-coded, and puts it under your control. It allows Appway to do much more than Nm'4 ever could — run not just account opening solutions, but also other business applications, our company website and intranet, our Developer portal, and even marcotempest.com.



Other aspects of Nm'4 (Processes, for example) went through a similar evolution. We replaced simple, purpose-specific things with more generic, more powerful Business Objects.

Power and consequences

There's a dark side to all of this, though: Creating solutions in Appway requires more knowledge than it did in Nm'4. Yes, Appway no longer has a hard-coded portal — you can now freely change and adapt it — but that means you have to know how your portal should work, and how to make the corresponding changes in Appway. By making Appway more flexible and more powerful, we also made it a bit harder to use.

Moving a lot of power into the Screen editor had other unintended consequences. It caused solution engineers to start moving business logic into Screens. One of the promises of Appway's model is that you know what's going on in your application. Moving business logic into Screens, unfortunately, runs counter to that. It hides important things at the edges of Appway solutions, inside Scripts attached to Button Actions or Script components, for example, instead of moving it to the surface, where it can easily be measured, tested, and adapted.

And, finally, it caused Screens to bloat in size. Many Screens in modern Appway installations are much, much larger than we ever anticipated.

There's a reason why many solution engineers spend most of their time in the Screen editor: Screens are hard to build, but they contain a lot of the power and logic of many Appway solutions.

One of the things we are working towards is to take a few small steps back to the Nm'4 style of having high-level, purpose-driven Business Objects and components in Appway — but without losing the power Appway currently provides. That's the reasoning behind much of Appway 6.2's new Workspace design, and the new components that come with it.

Here are some of the advantages this brings.

Better Quality

Having higher-level components improves the quality of Appway solutions. Something like the Data Table or the Workspace Collaboration components would never have been developed on a solution level; both contain years of work, and an incredible amount of expertise.

You get all of this work and expertise for free, and can just drag a single component into a Screen.

Getting Business Logic out of Screens

One of the things that can make Appway solutions more difficult to maintain than necessary are business decisions made in the branches of a solution, instead of at its heart. If business logic is in Screens instead of Processes, Data Classes, and other higher-level Business Objects, the logic can become invisible, inconsistent, and hard to control.

Having higher-level components and Business Objects can help pull some of the business logic back up into a more visible area of the Appway model.

Less Work

Higher-level components make designing Screens easier and faster. No longer will you have to assemble, say, a header using low-level components like divs. Instead, simply drag the built-in Header component into your Screen, configure it to contain whatever you want it to contain, and you're done.

Easier to Use

Purpose-driven components instead of generic components make using the Screen Designer easier. Let's say you're working on a new Appway solution, and you need to set up the basic layout — header, sidebars, things like that. Building that in a solution using low-level components is hard: It requires in-depth knowledge of HTML, CSS, JavaScript, Appway components, and all kinds of arcane stuff.

Using the Border Layout Manager, on the other hand, and you get all of this – with minimal effort. Drag it into your Screen, change some simple configurations: Done.

Coincidentally, it also makes the *Workspace* easier to use, because purpose-driven components are perfectly adapted for their use case, and because we invest a lot of time into the design and usability testing of our built-in components — much more than a typical solution ever would.

Semantic Power

Nm4's Input Forms were easy to read. You knew what they contained by looking at them, because you knew what each "thing" inside the Input Form did. With Appway's Screen Business Object, this is no longer always true. Looking at a Screen in the Screen designer often doesn't tell you much about what the Screen actually does, or what it is being used for; the components lack semantics. A div could be anything, for example, and once you start nesting stuff, it gets harder and harder to figure out what a Screen actually does.

Higher-level components have semantic power. They tell you what they are. If a Screen contained a Document Tracker component — which doesn't exist yet, it's just an example — you'd know *immediately* what it was used for. That's one aspect of what we call "self-documenting solutions".

Having higher-level components also means that you need fewer of them, which, again, makes Screens easier to read and understand.

Tradeoffs

Of course, having higher-level components doesn't just have advantages. The Data Table, for example, gives you very easy access to an incredibly powerful, sophisticated Screen component. But using it also takes a bit of power away from you: You're bound by the properties the component provides, and if you need to change something the component doesn't allow you to change, you're out of luck.

We think this is a tradeoff well worth making, though. While you don't get to control every pixel of your Data Table, or the positioning of every single button, you gain a lot in return.

The Future

Let's take another trip with our time machine. This time, let's travel into the future.

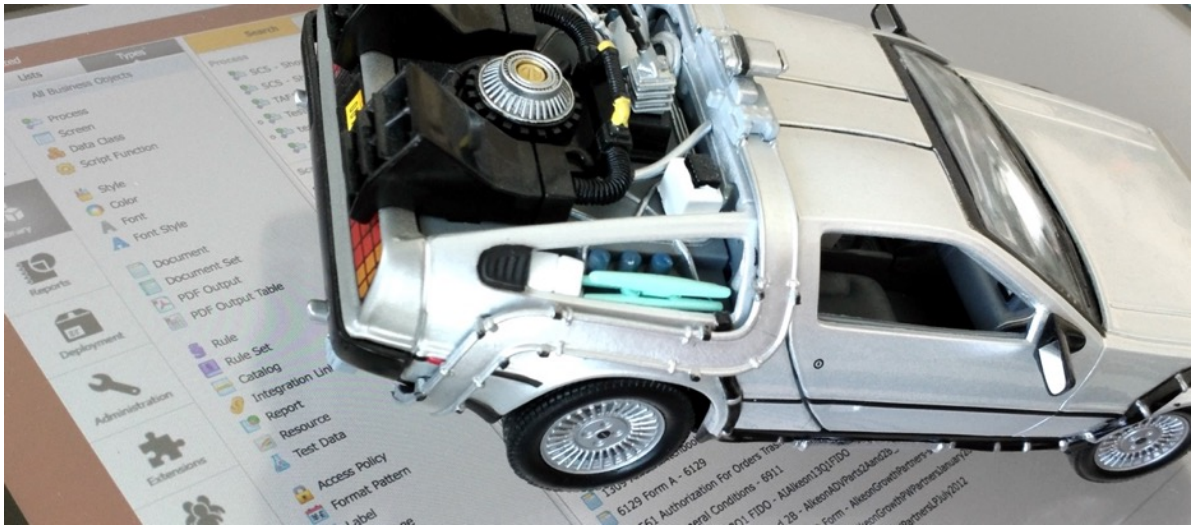


Image courtesy L. Mathis ©

The power Appway provides has been good for our company. We wouldn't have experienced the same kind of growth without it.

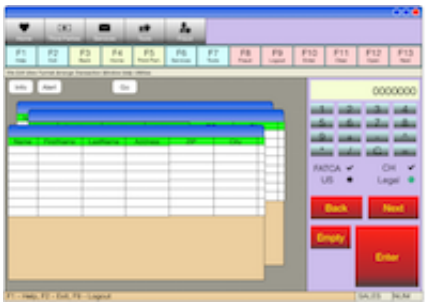
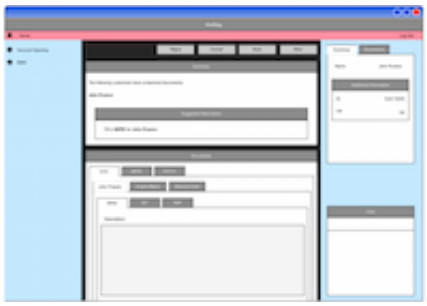
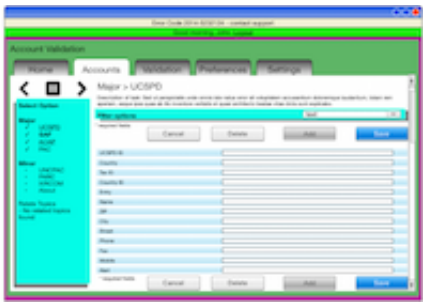
But now it's time to start working on getting back some of the simplicity and ease-of-use we've lost. If possible, we'd like to do so without losing any of the power and flexibility we currently have, of course. We think we can achieve both — power ***and*** ease-of-use — by moving business logic into more visible areas of Appway, and by offering more powerful, less generic components.

Workspace Design Notes #2: Minimizing User Decisions

Business applications have a bad reputation: "They're complicated and require a lot of training" or "they can only be used correctly by domain experts."

Business applications are often designed without much thought about user experience. They're designed by business experts who don't mind user interfaces that provide no guidance.

As a result, they end up looking like this:

The interfaces above are crowded and complex. Each screen contains a lot of information, and many different possible actions you could choose. To be able to use the application successfully, you have to know exactly what you're doing.

Unfortunately, people often *don't* know exactly what they're doing. This causes problems. Our own Client Onboarding Study comes to the conclusion that client onboarding typically takes way too long, is difficult, and fraught with errors and mistakes.

This is where Appway comes in. Our promise to our clients is that we can decrease the time it takes to do an onboarding, lessen the burden on the person doing the onboarding, and minimize errors and mistakes.

We do this by moving expertise away from the person doing the onboarding, and into the Appway solution.

The goal is to minimize the decisions users have to make, and to replace these user decisions with processes and rules that are executed correctly by default.

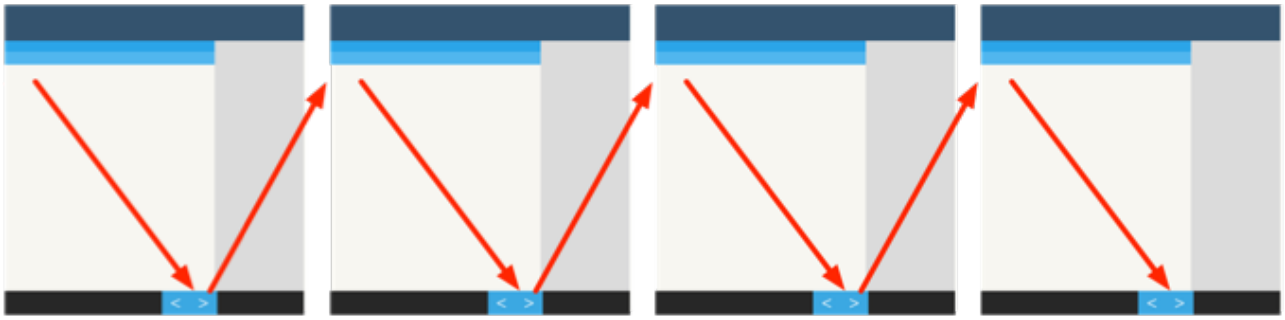
Guiding the user

One practical example of this is how we want to direct user attention using the new Workspace design.

In a traditional business application, the user needs to know what to pay attention to. For example, in an account opening, the user needs to know that the first step is to jump to the "Parties" section, and add the parties to the account;, that the second step is to make sure that all parties have the appropriate roles, and so on. The user's path through the application is guided by the user themself, and does not follow a straight line.

The user needs to know which sections of the application to visit, and in which order to visit them.

Appway's approach is different. The process makes these decisions for the user, and presents the appropriate sections at the appropriate moment. The user can simply follow the process: Appway makes sure that the user sees all of the relevant information, and fills in all of the relevant fields.



Users move through each screen from top to bottom. At the end of each screen, they click "next", and do the same thing on the next screen. As they scroll through each screen, all relevant information — and all relevant fields — are shown to them. The user never has to backtrack, switch between tabs, scroll back, jump between different pages, or manually decide what the next actions are.

Appway makes the decisions; users follow the process.

Best Practices

While working on the new Workspace design, we tried to keep the following goals in mind.

- Make sure that your users see all relevant information, and all relevant fields, when they scroll through a screen from top to bottom
- It's better to have long screens, and just allow users to scroll through them, than to add UI elements that hide content (tabs, collapsible sections). These elements can cause users to miss relevant information while requiring them to learn more complex interaction patterns.
- Each screen should have one single, obvious purpose
- Users should only have to make decisions that can't be made automatically by Appway.

These rules lessen the user's burden, minimize the potential for errors, and decrease the time it takes for the user to finish a task.

Workspace Design Notes #3: The Mobile Experience

When coming up with the new Workspace design, one important goal was that it be fully compatible with mobile devices. But rather than just adhering to a general idea, we decided to follow a few specific principles:

1. Mobile devices are not "second-class citizens"
2. Neither are desktop devices
3. We design for use cases, not for devices
4. Switching devices should be seamless, and running processes should be accessible everywhere (cross-channel processes)
5. No additional work should be required to make solutions mobile compatible

Let's start with the last point: In an ideal world, there should be no "optimize for mobile devices" step in any project. For us, this means that Appway's built-in components, and their default visual and interaction designs, should work on all devices without further adaptation.

With Appway 6.2, we have made a huge step towards this goal. If you follow our guidelines when building Appway solutions, you're 99% of the way to a fully working, responsive solution; a bit of testing on different devices will help you to work out the remaining kinks.

Cross-Channel Processes

When thinking about mobile design, people often come up with one of the following two approaches:

1. They provide a watered-down version of whatever is available on the desktop.
2. They provide a mobile version that provides specific functionality that tends to be associated with mobile use.

Both of these approaches are problematic: They confuse devices with use cases. People don't work like that; they don't use specific devices only for specific tasks. Instead, people increasingly use all kinds of different devices in all kinds of situations.

A 2013 study by Forrester Research came to the conclusion that people aren't picky when it comes to device usage. The typical split you'd expect — desktop PC at their desk, phones while commuting, tablets at client facilities and at home — is only barely reflected in actual device usage numbers. In reality, people use all types of devices in all situations: using tablets at their desk, computers at client facilities and while travelling, and phones for work when *at* work, even when they're sitting at their own desk, in front of a PC.

Importantly, Forrester also notes that most companies don't support this kind of multi-channel work style; when helping clients implement their Appway solution, we've also discovered that some firms have not yet completely adjusted to the device flexibility of their employees.

No "Mobile Version"

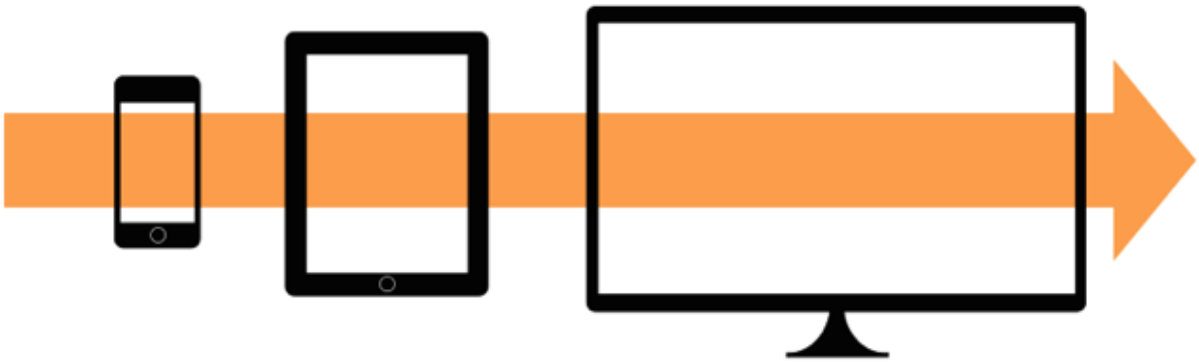
One of the implications of this is that a watered-down version of a solution developed specifically for mobile use is not going to cut it. These feature-deprived mobile versions are preventing users from doing the things they need to do, on the device they want to use.

For example, one might assume that people are not going to do an account opening on a phone. When designing a "mobile version" of a solution, one might thus conclude that the full account opening process does not need to be available. Maybe all that's needed is a simple approval process; that's a small task that works well on a small device.

However, what if users start an account opening on a desktop PC, and have to pause it because they're missing some important piece of information? Later, they might acquire that information while they're talking to the client, but might not have access to a PC right away. If the account opening process is available on the phone, they can simply use their mobile phone, open the active account opening process, and enter the missing piece of information.

In other words, even though it's true that most people might not want to go through a full account opening process on a phone, there are situations where people do have to access an existing account opening process from a mobile phone. That doesn't work if the mobile version is watered down, and only supports part of what the desktop version offers.

Instead, the same processes should run consistently on all devices, and people should be able to move seamlessly from device to device.



As usability expert Jakob Nielsen notes:

> "Most sites do understand that mobile content cannot be arbitrarily limited. The answer to a question should be the same regardless of where the question is asked: mobile, desktop, tablet."

Design for Use Cases, not Devices

Conversely, people might think about use cases that seem to be specific to mobile devices, and implement mobile versions of those. One such use case is a meeting notes application, where bank employees jot down points discussed during meetings with clients. Typically, these meetings happen at the client's residence or place of work, so bank employees usually want to use their phone or iPad to take notes.

But what if the employee takes their notebook? Or what if the meeting is held via phone, and the employee is sitting at their office desk, in front of their PC? In those situations, they want to use their laptop or PC to enter meeting notes — not their phones.

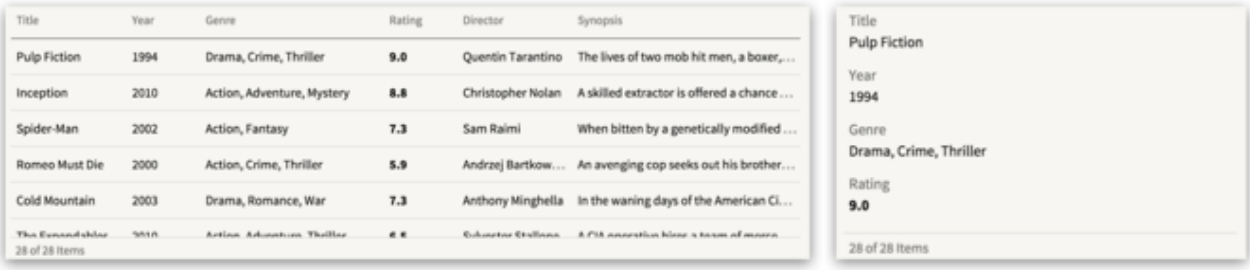
The important point here is that we should design solutions *for use cases* — "I want to take notes during a meeting with a client" — and *not* for specific devices. Instead, make sure that all use cases are accessible on all devices: You can't predict what devices your users are going to use for each use case.

To learn more about our design process, read the "Business Logic First" white paper.

What Appway Offers

When coming up with the new Workspace design, we knew that we did not want the mobile experience to be watered down. Similarly, we knew that we did not want to have mobile specific processes. Instead, we wanted all processes to be fully functional and fully accessible on all devices. We wanted cross-channel processes.

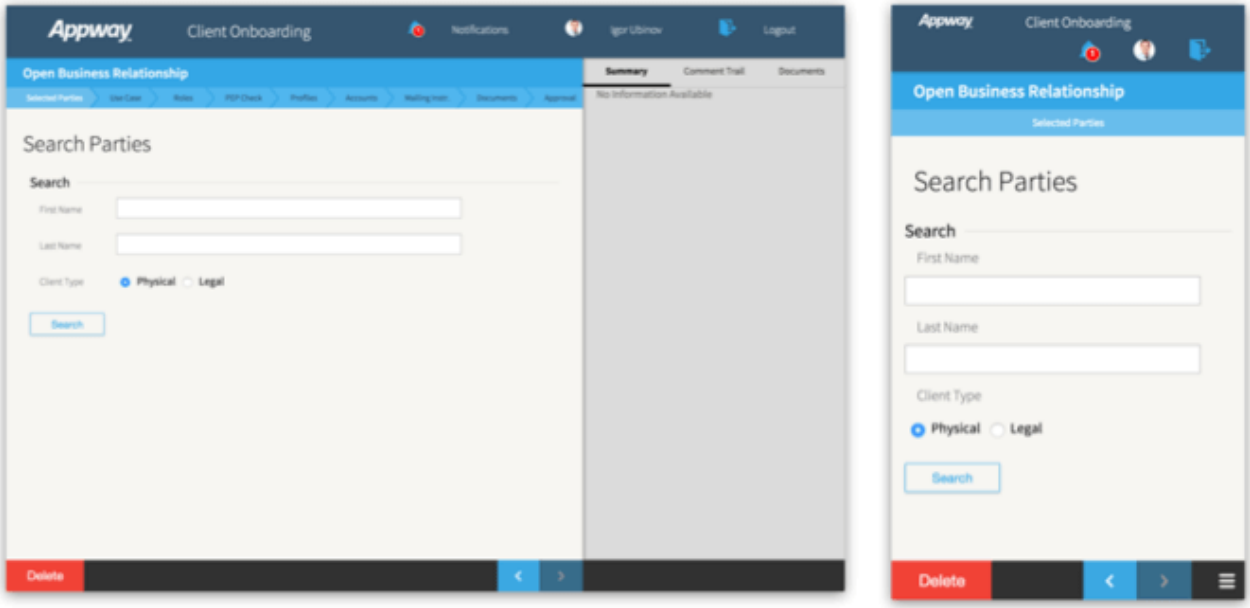
That's why Appway's new Workspace design uses a single, consistent visual design across devices. It's why all new components are fully responsive. The Data Table, for example, uses the full width of a large desktop screen, but collapses down into a narrow view on smaller devices. There's no need to design two different screens, one for desktop and one for mobile; with components like the new Data Table, the same Screen works across multiple devices.



Data Table on Large Screen

Data Table on Tiny Screen

This doesn't just apply to one individual component. Our components work together, and if used correctly, make the whole solution responsive. As an example, here are two screenshots showing the COB solution as it appears on a desktop PC, and on a mobile phone.



COB on Desktop

COB on Phone

Cross-device processes, designing for use cases instead of devices, no simplified mobile versions: these are what Appway intends to achieve with the new Workspace design.

We want to remove the barriers between mobile and desktop, and ensure that everything works everywhere by default.

Workspace Design Notes #4: Different Types of Screen

One aim of the new Workspace Design was to simplify the task of designing Screens. Appway developers often spend most of their time in the Screen designer, so that's where we can have the biggest impact on productivity.

Here I'd just like to add a quick note about "Screens". The term "screen" can be a potential source of confusion, because we use it in two different ways. A "Screen" (upper-case) is a specific kind of Business Object in Appway that is used to design screens (lower-case). A screen (lower-case) is a rendered page as it appears in the web browser.

One of the ways we've tried to simplify Screen design is by finding generic screen types that are used again and again in applications, and then design components that implement these specific screen types in a well-defined, high-level way.

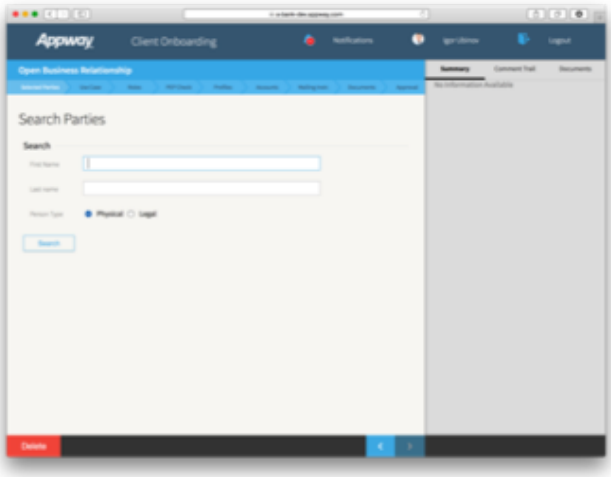
Using a typical client account opening process as our testing ground, we found that there are two main types of screens in that process:

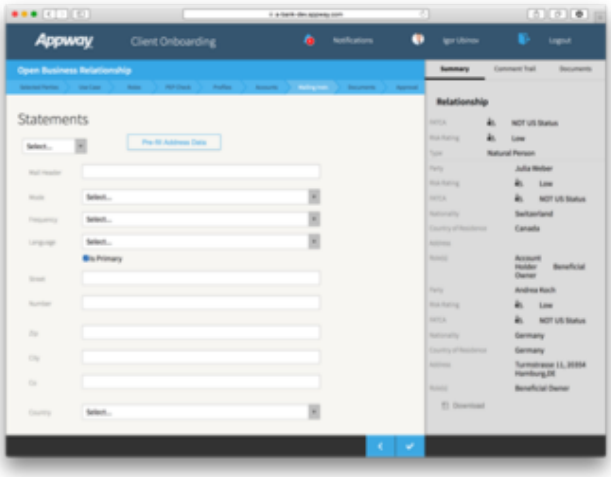
1. Form screens, where the user fills in different controls
2. List screens, where the user either picks a task from a list, or edits a list

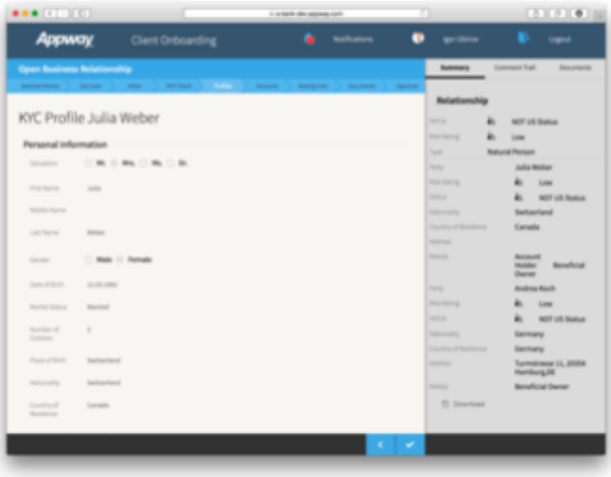
Of course there are always going to be screens that fit into neither of these two types - and there are going to be screens that have aspects of both. However, in general, thinking of screens in terms of these two archetypes will help you figure out how to build many of the screens required for your application.

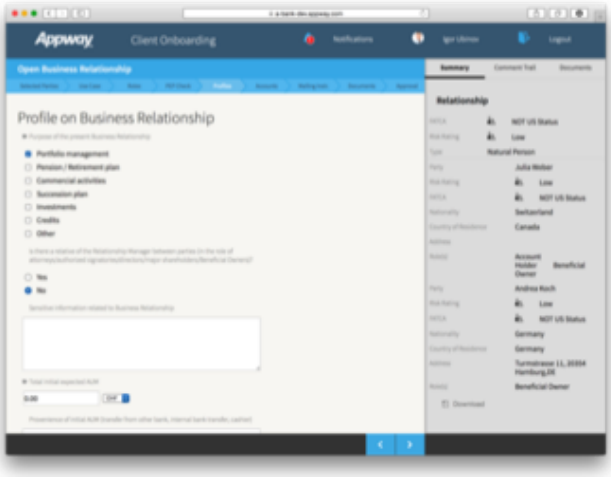
Form Screens

Form screens are the kinds of screens you typically associate with a process. They mainly contain components like text fields and dropdowns, and screen content is validated when you jump to the next screen. Here are some examples from Appway Client Onboarding.









These are all built using Adaptive Flow Layout components.

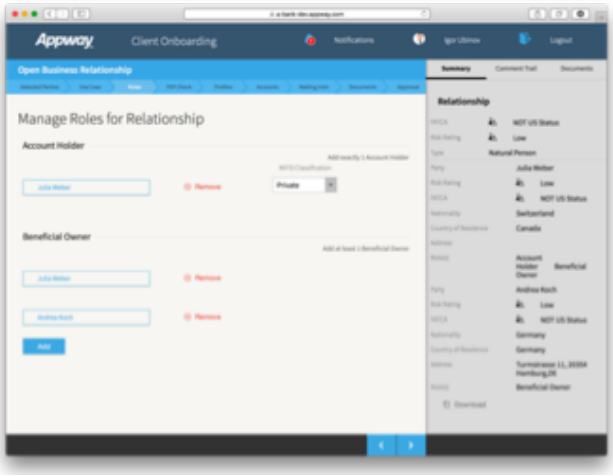
List Screens

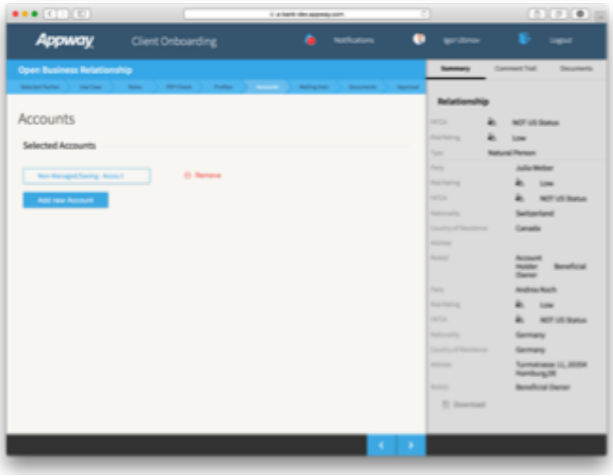
List screens contain lists. There are two different kinds of list screens:

1. Screens where the user edits the screen content by adding or removing elements from a list
2. Screens where the user picks a task from a list

As mentioned earlier, some list screens could end up combining both of the above types, but we've found that most list screens usually conform to one of the types.

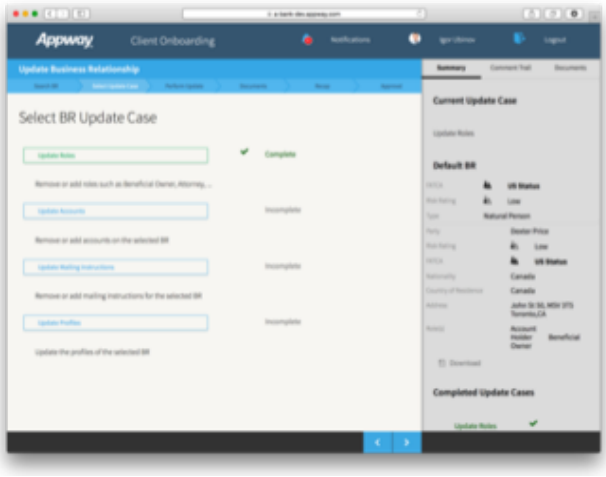
Here are some examples of editable list screens.

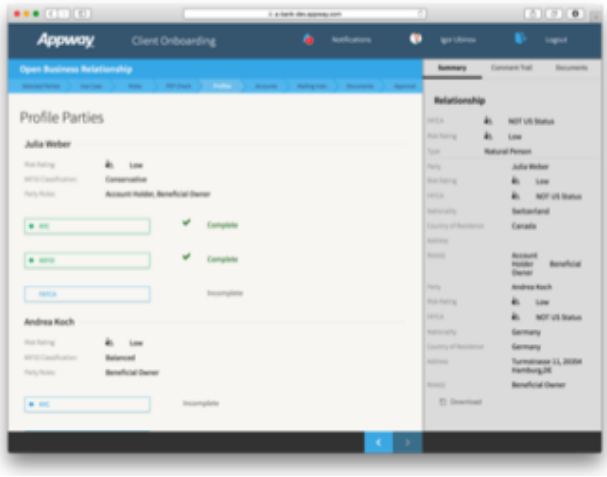




These screens are built using Adaptive Lists, and List Buttons. Each Adaptive List item has a "remove" button (and possibly other UI elements). Below the list, there's an "Add [Element]" button.

Here are some examples of task screens.





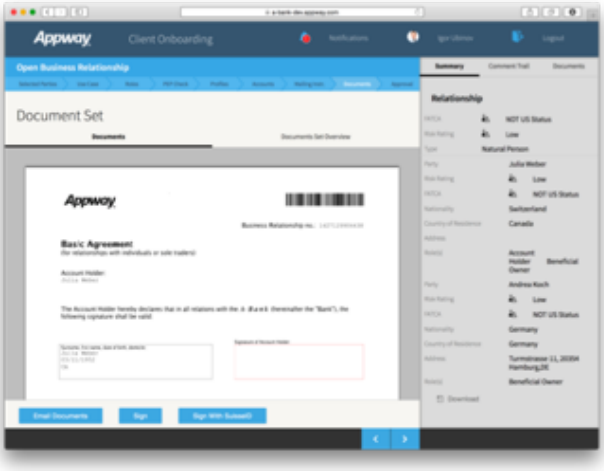
Again, the lists are built using Adaptive Lists and List Buttons. This time, though, they have an icon and a text on the right-hand side that indicates the item's status; similarly, the List Button's color is used to indicate if the task is done, available, or not yet available.

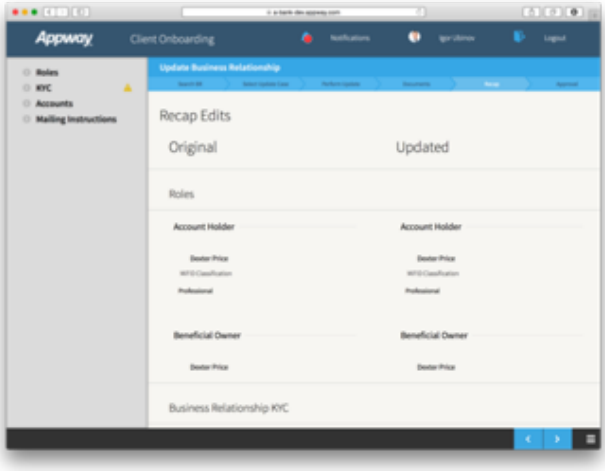
The example screen on the left shows explanatory text below the button, outlining what the task is about.

We're currently evaluating whether it makes sense to have two different components with distinctive visual styles for task lists and editable lists. The Workspace design is an ongoing process, and we welcome feedback, positive and negative.

Special Cases

Of course, we know that not all screens fit into these basic types. For example, the Client Onboarding solution contains exceptions to the types such as the "document preview" screen, and the "review changes" screen.





There are two possible approaches when it comes to "exceptional" screens. Either a component for this particular use case already exists, or one will be implement by R&D. If neither of these apply, a custom component may be required inside the solution (i.e. a Screen include that has its own CSS and, if required, JS Resource, and can easily be replaced with a built-in component in the future, as Appway's library of built-in Screen components continues to grow).

The Future

In the future, we aim to have more specific components for the different list types, and to provide more "special case" components.

Workspace Design Notes #5: Am I in a Process?

When it comes to screens and user interaction, there's a huge difference between a screen that's part of a process, and one that is not. Inside a process, user decisions should be minimized: the process should guide the user. Outside of the process, the user is in the driver's seat, and should decide what to do.

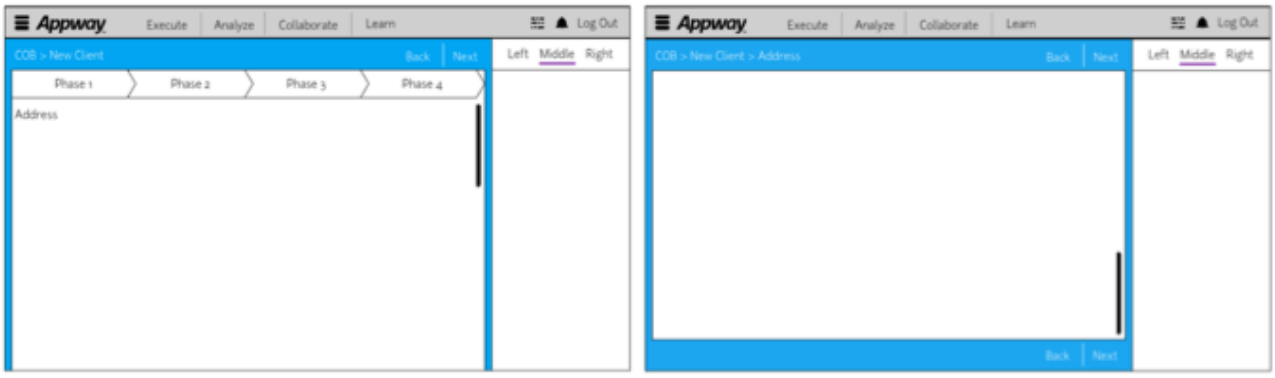
Users should therefore *know* when they're in a process, and when they're not. They should know whether they are expected to let themselves be guided by a process, or whether they are expected to make decisions themselves.

Keeping users on track

In earlier releases of Appway, the difference between a screen inside a process, and a screen outside a process, was not always immediately obvious. Depending on the design, these two different screen types could look almost identical. This is something we wanted to change with the new Workspace design.

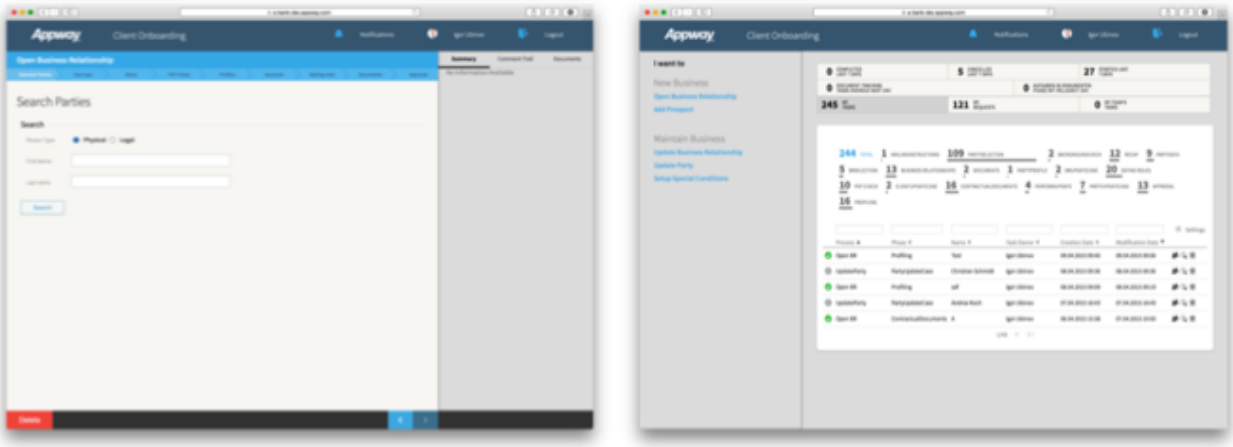
An early idea we had was to think of processes as "on rails" (i.e. you're guided along a track like a train, not allowed to roam freely like a car). This visual metaphor lead to the idea of having "rails" on the left and right-hand side of the screen when the user is in a process.

In the wireframes below, we used blue lines along the left and right edge of the content area to create the "rails".



In this design, the back and next buttons would be at the top of the screen, but would be replicated at the bottom once the user scrolls down.

We eventually abandoned this idea. Instead, we introduced the idea of the Flow Bar, a static bar at the bottom of the screen that contains the back and next buttons. We believe that this, together with the chevron at the top of the screen, is enough to visually distinguish process screens from non-process screens.



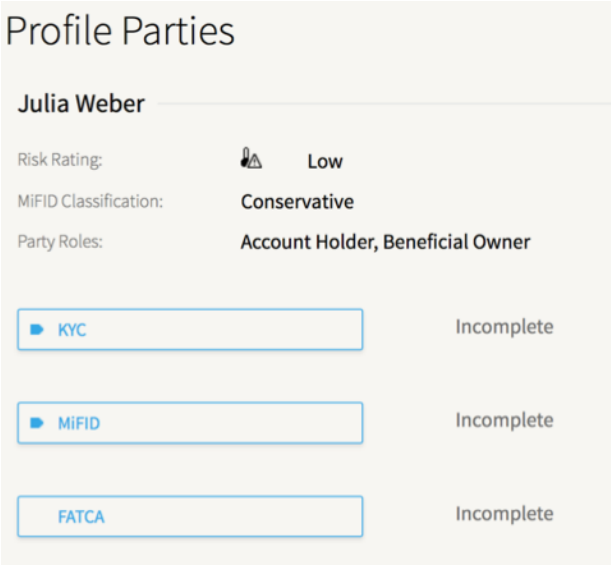
Process Screen Non-Process Screen

This is where we stand today, but it is not the end of the road: Distinguishing between process and non-process screens is not sufficient.

It's easy to think of Appway solutions as being one of two different types: either process-driven, or data-driven. Similarly, screens are often thought of as being "part of a process", or "not part of a process".

In reality it's usually not quite so simple or clear-cut. Even a solution that seems — on the surface — to be entirely process-driven has sections inside its processes that are data-driven.

For example, once you reach the "Profile Parties" screen in our Client Onboarding account opening process, you've reached a small island inside the process — a tiny embedded data-driven section. The process no longer tells you what to do: You can fill in any of the available screens in any order (or even partially fill some sections, and then move to other sections).



The current design creates a visual distinction between process screens, and non-process screens, but doesn’t yet clearly distinguish "data-driven islands" inside processes. That’s something we’re working on for a future version of Appway.

Workspace Design Notes #6: Why There is (Almost) No Drag and Drop

"We'd like to show our manager something really cool. How about adding some drag and drop to a few of the screens?"

I've heard this a few times now, and I thought it might be a good idea to outline why the Workspace has (almost) no drag and drop functionality.

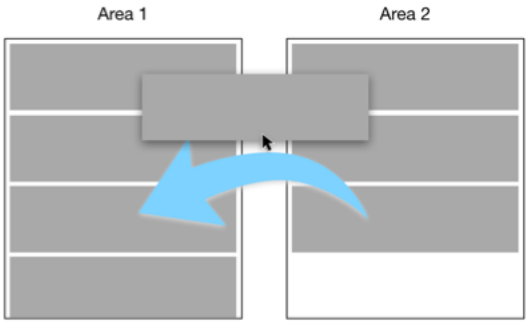
There are three main reasons why we've made the decision to avoid drag and drop interactions whenever possible:

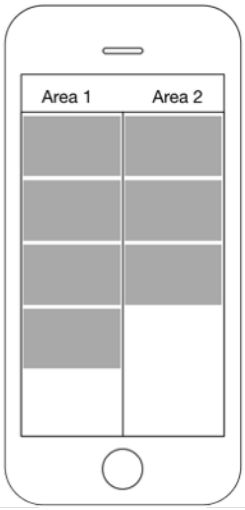
1. Drag and drop doesn't work well on small screens
2. Drag and drop causes gesture overloading on touch screens
3. Drag and drop is not accessible

Let's look at these in turn.

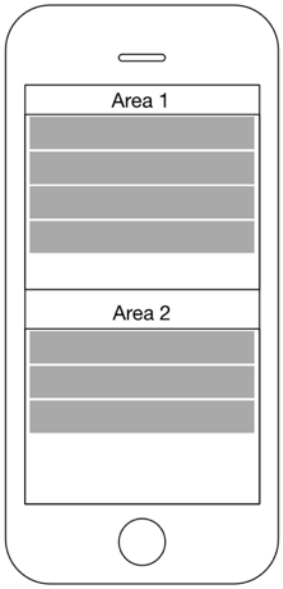
Problem 1: Drag and drop doesn't work well on small screens

In order for drag and drop to work properly, you typically need to have two different areas visible on your screen. Each area contains draggable items; you interact with this user interface by dragging things from one area to the other.



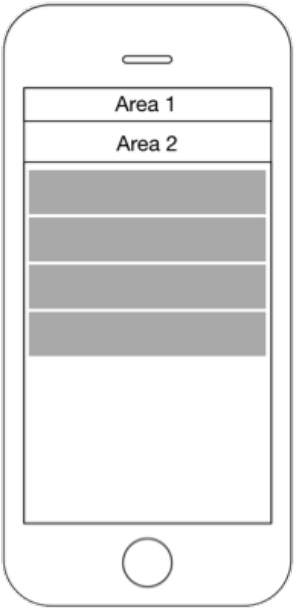


It is immediately obvious that this requires a lot of space. In an optimal situation you would want to show the two areas next to each other, so that the user can drag things from left to right, or from right to left. This arrangement is not, however, possible on a narrow phone screen. As the image to the left shows, there's just not enough space to show draggable elements with meaningful content side-by-side on a mobile phone.



An alternative solution is to put items below each other.

But even so, the user interface is very cramped – and now you're running into problems with vertical space. The wireframe above doesn't show the solution's headers or footers, yet still manages to show barely any draggable elements.



If at this point it's still required that drag and drop be a part of your interface, you could use a UI element that hides part of the screen, like tabs or accordions.

Having used an accordion, you now can't see both areas at the same time; you have to teach users that they can drag elements and then drop them on top of accordion headers.

The space problem on mobile devices is compounded by the fact that smartphones have touch screens, so interactive elements need to be larger than on a desktop system. Fingers are less precise pointing devices than computer mice, after all.

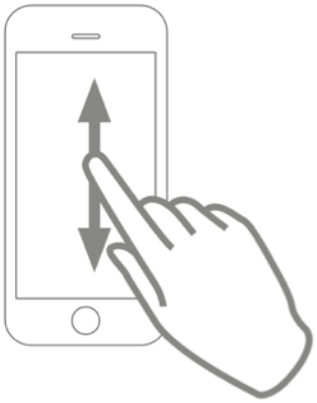
But that's not the only issue touch screens cause.

Problem 2: Drag and drop causes gesture overloading on touch screens

On a desktop PC, scrolling and dragging are two completely different, discrete actions. You scroll using the scrollbar, or the scroll wheel, or some other separate interface; you drag by clicking on an element, and moving it around.

Image courtesy renatromitra (CC-BY-SA)

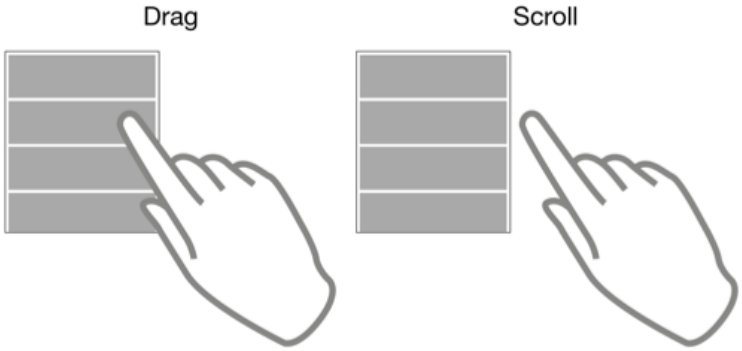
On a touch screen, scrolling and dragging *are the same action*. For both, you touch the screen with your finger, and then move the finger.



Is the user trying to scroll, or trying to drag?

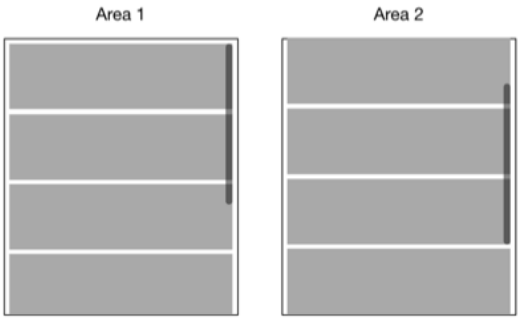
These questions mean that the user interface has to twist itself in knots to figure out what the user is attempting to do.

The simplest thing the UI has to pay attention to is what the user actually touches. A draggable element? Perhaps the user wants to drag. Something that can't be dragged? Perhaps the user wants to scroll.

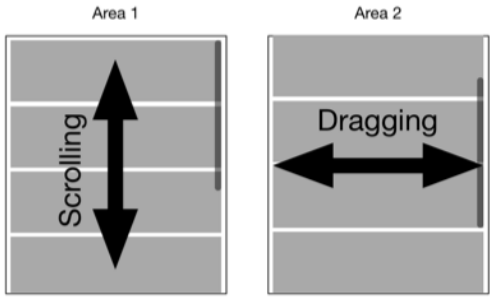


This set of decisions is already error-prone, since users don't always pay close attention to what they're doing. Particularly when they are trying to scroll, users might not notice that they can't just touch the screen wherever they want. This inattention can lead to erroneous user input, where users end up moving elements around when they simply wanted to scroll.

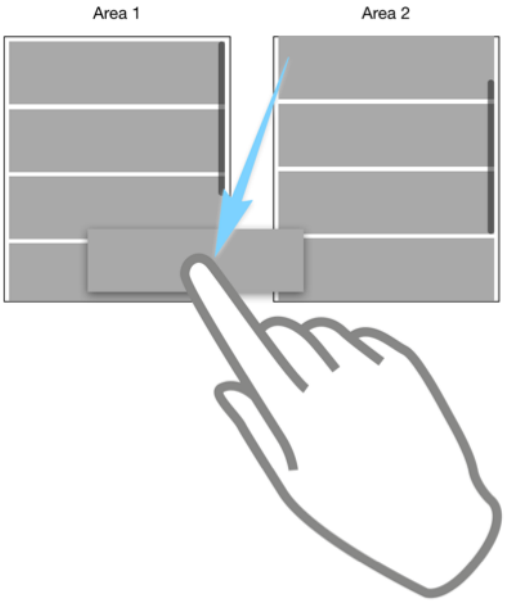
It then gets even trickier, as differentiating between scrolling and dragging based on the touched element alone does not work. Consider this user interface:



Here, the user drags elements out of a scrollable pane. It's not possible to touch the scrollable pane anywhere outside of a draggable element. In other words, the UI has to differentiate between scrolling and dragging based on the direction of the user's finger movement. Moving the finger up or down? Probably scrolling. Moving the finger sideways? Probably dragging.



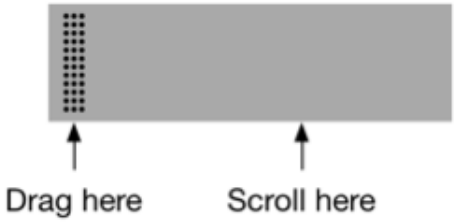
But, again, you run into difficulties. What if the user wants to drag an element from the top of the right pane to the bottom of the left pane? Now the finger is moving down, but the user is trying to drag.



Simply put, the user interface can't reliably figure out what the user is trying to do.

You could try to solve this in different ways:

1) "I'll only make things draggable if users long-tap on them". While it may sound intuitive, you're now hiding important interactions from the user; people might never figure out that they can drag things around.
2) "I'll only make part of my draggable element actually draggable". This partly works, but now you're making drag and drop even harder to use than it was before: Remember that touch screens require large touch targets, because fingers are not precise input devices, and you're now adding visual clutter (that indicates where to drag) to the user interface.



3) "I'll just avoid scrolling". This works *sometimes*. If you're designing an app with static, never-changing screens, for a specific device, it works. In other words, if you're showing drag and drop in a presentation, and you know ahead of time exactly what you're going to show, and what device you're going to use, you're fine. But in the real world, where the list of draggable elements can grow, other features could be added to the screen, and where you have to target multiple devices with potentially smaller screens, you can't avoid scrolling.
4) "I don't have two areas - I just want people to order things by dragging them up or down". Again, this sounds reasonable - but this only makes the problem worse, because we can now no longer attempt to figure out the user's intentions based on the drag direction of the user's finger. Both scrolling and dragging are now up/down gestures.

If you target one specific device with a predefined screen size, only have a limited set of draggable elements, *and* the screen itself contains few elements (so it doesn't scroll), then yes: You can implement drag and drop. That's why drag and drop works well in demos: Demos are a completely controlled environment where you know exactly what device you're going to use, and what the screen is going to contain.

But once these assumptions no longer apply (through people using different devices, the number of draggable items changing, the number of other elements on the screen changing and so on), it all starts falling apart.

Problem 3: Drag and drop is not accessible to people with disabilities or certain illnesses

Moving your mouse on top of a button and clicking the mouse is relatively easy. The whole action is divided up into two discrete, simple tasks, and users can take their time to get each task right. Move the mouse, correct for mistakes. Is the cursor on top of the button? Okay, now click.

This works almost equally well whether the user is using an actual mouse, a trackpad, a trackball, a graphics tablet, or perhaps even something like a head-operated input device.

It also works for partially sighted people, because you can zoom in on the button. Similarly, a screen reader used by blind people is able to identify the button, and allow people to trigger it.

Drag and drop is a completely different story.

First of all, it's a much more complex interaction. You have to aim and click the mouse, then hold the mouse button down, aim again while holding the mouse button down, and release the mouse button once the second aiming action has been completed.

Drag and drop can be even harder to execute on a touchscreen device, since you're now covering part of the screen with your hand, but you can't remove your hand and see what's beneath, because that ends the drag action, and drops the element somewhere you probably did not intend.

If you're an able-bodied person with reasonably good eyesight, you may never have thought about this, but drag and drop requires quite a bit of dexterity and precision from users.

This can be a problem if you're suffering from illnesses like RSI or arthritis; if you have physical impairments that make it difficult to move a mouse cursor with precision; or if you simply don't see well. You might also be using a special input device that makes drag and drop harder to execute than with a mouse.

Image courtesy lisbokt/Flickr (CC BY-NC-ND)

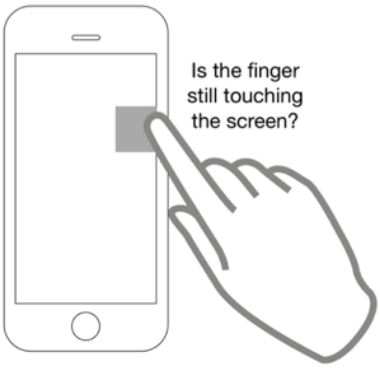
There are tools people can use to avoid the drag and drop problem, but these are typically still complicated and time-consuming. And for some situations – e.g. a blind person using a screen reader – normal buttons work just fine, but drag and drop may be entirely unavailable.

If the alternative to drag and drop is to simply offer a button that users can click, choose to use the button: A button is always faster, easier to understand, simpler to use, and more accessible.

Small Details, Large Impact

There are other minor problems with choosing drag and drop: It's a complex interaction that requires getting a lot of details right. Most web apps *don't* get these details right, which can be frustrating.

For example, you need to take another aspect of scrolling into account. Often, people want to drag an element to a place that's outside of the visible area of the current screen; if they drag the element to the screen edge, it needs to start scrolling — and web apps don't do this automatically.



"Edge dragging" causes additional issues on touchscreen devices: You've selected an item but are now covering the screen with your hand, and can't see how far the screen has scrolled. Worse, since your finger is near the edge of the screen, it's possible that the device loses track of your finger, and drops the dragged element prematurely, thus triggering an erroneous action.

Gestures and Operating Systems

Mobile browsers use gestures for built-in features (like going back to the previous page) that are also used by drag and drop. Overriding these built-in gestures in order to make your drag and drop functionality respond correctly can be confusing to users, and can cause technical problems if done poorly.

I quickly want to mention a final concern: People sometimes think that it would be nifty to hide functionality in sidebars that are shown when the user swipes in from the edges of the screen. Unfortunately, operating systems manufacturers also agree that it is nifty behavior, so more and more manufacturers are now using edge swipe gestures for built-in, OS-level features, like Slide Over in iOS 9. This means that these gestures should not be used by apps or websites anymore: operating systems already use them.

Possible Solutions

All of this does not mean that you have to absolutely avoid drag and drop in all situations. For example, Appway's slider component supports drag and drop. However, it also supports simple clicking. If you can't drag, you can simply click on the slider bar, and the knob jumps to the location you've clicked. Or you can put the focus on the knob itself (e.g. by clicking it, or tabbing to it), and use the keyboard's arrow buttons to move it. Even though the slider supports drag and drop, it is completely accessible from the keyboard.

Similarly, one spec of a drag-and-drop based component we're working on switches to a click-based user interface on smaller screens. Unfortunately, this doesn't solve all of the problems (it doesn't solve gesture overloading or accessibility) — which is why it's still a spec, and not yet a finished component.

If you do want to provide drag and drop

If you do want to provide drag and drop features, make sure you ***really*** understand the possible repercussions: Once you deploy your app to thousands of people using thousands of different devices, you're going to run into all kinds of issues. Showing drag and drop in a presentation can be a short-term win, but supporting drag and drop always ends up being a long-term headache. Once deployed, it's going to generate huge amounts of cost in maintenance, user support, and lost time due to user problems.

If there is no way round *not* implementing drag and drop functionality, it's best to think of it as a secondary interaction pattern, similar to a keyboard shortcut:

- Design your user interface such that it can be used without drag and drop first
- Add drag and drop support only when the UI already works well *without* drag and drop, as a bonus for people who prefer to use drag and drop
- Add it only on devices where adding drag and drop does not cause problems
- Finally: Sweat the small stuff. What happens if there's more items on the screen, activating scrolling? What happens if the user drags something to the edge of the screen?

Workspace Design Notes #7: Buttons

The tactile, physical nature of our world usually tells us what to expect from our actions. Understanding the difference between a toffee hammer and a sledgehammer is reasonably easy, and it's also pretty clear what happens when using either hammer.

Image courtesy H is for Home (CC BY-NC)

Image courtesy UK Ministry of Defence (Contains public sector information licensed under the Open Government Licence v3.0)

One of these hammers is small and tiny, and is used to create a small and tiny impact. The other is large and heavy, and is used to create a large, severe impact. The physical size and the visual style of the device tells you what it should be used for, and what kind of impact you can expect from using it.

These rules all go out of the window once we enter the realm of computers. On a computer screen, a small hammer can have a large impact, and a large hammer can have a small impact. There are no laws of physics governing the relationship between how things on your screen look, and what they do: Two buttons that look exactly the same could trigger vastly different actions.

When we started designing the buttons for the new Workspace design, we wanted to make sure that buttons give at least a basic indication of their power. Can I click this button without giving it much thought, because it's more of a toffee hammer, or should I really think before clicking this button, because it's more of a sledgehammer?

The button's look should communicate this information to you.

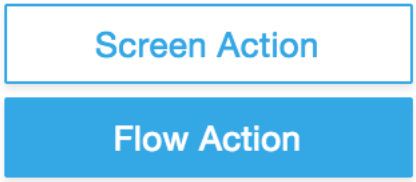
However, having different button types that are then used incorrectly only makes the problem worse. The last thing you want is a toffee hammer that punches a sledgehammer-sized hole into your table when you just want to break some candy.

Our Solution

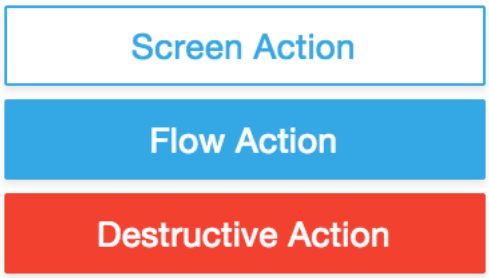
When we thought about these two goals — different button types that communicate the severity of pressing that button; clear guidance on when to use which button type — we came up with the following idea. What if we divided buttons into two basic groups based on their "reach" or "power"?

- Screen Action buttons: Buttons that only have a local effect on the current screen
- Flow Action buttons: Buttons that have a larger effect on the current Process.

The visual style of a button indicates the button's "power". The button that's only outlined has a smaller effect than the button that's filled in.



We felt that this alone didn't go quite far enough. After all, there are some actions that have particularly destructive power, and should get their own visual style: Deleting a Process, removing a party from an account, and other irreversible, disruptive, or dangerous actions should be highlighted – and visually set apart from other buttons. For these buttons, we added the "Destructive Action" type.



This leaves us with the following rules:

1. If a button's action only affects the local screen, use the Screen Action type
2. If the button's action affects the Process, use the Flow Action type
3. Overriding the first two rules, if the button's action is dangerous or destructive, use the Destructive Action type

Following these rules allows the user to make an educated guess about the impact a button has.

Workspace Design Notes #8: Icons

Recent Appway versions ship with a very sophisticated, very powerful icon framework that allows you to use icons in many different contexts, and globally switch between PNG and SVG icons. But icons need to be handled with care: they are often not as memorable as you would wish, and can have dangerous side-effects.

This post, the eighth in our series on Workspace Design concepts, discusses the impact icons can have — and how best to work with them.

What's the problem?

Icons are hard to memorize, and people usually don't recognize them. There are no standards for icons (particularly in domain-specific applications), so people simply do not understand what many of the icons mean. There's a lot of research on the topic, and it usually doesn't go well for icons.

Apart from a small set of commonly-used icons, most people have no clue what a majority of icons actually mean.

Why are they dangerous?

Icons look good at first, but once you start adding a few, you'll start to find more and more places where they seem to make sense. So you add icons for different account types. Then you add icons for different parties. Then it seems that you really should add icons for the relationships between parties and accounts — surely an Account Holder and a Beneficial Owner could have different icons…

Soon your solution will be swimming in icons – and your users drowning in confusion, unable to understand what each and every icon means, or what they are supposed to do with them.

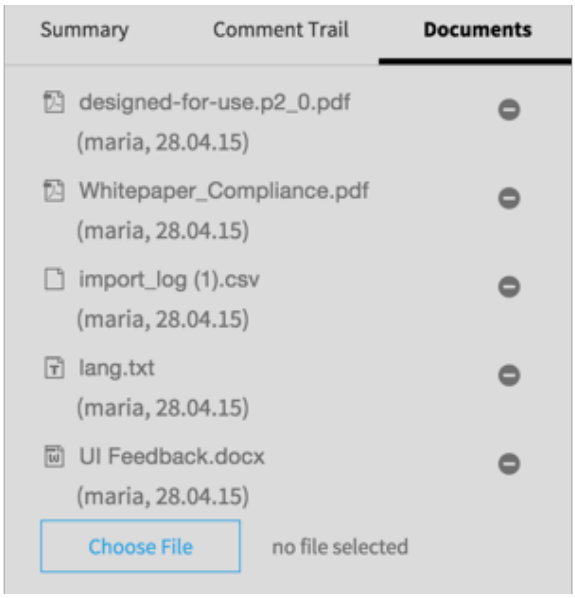
Because icons are mostly useless and often dangerous, our own Client Onboarding solution uses them very sparingly. We opt for alternatives whenever possible.

Where Icons Can Help

Icons aren't universally bad. There are some specific situations where they can add value.

Icons work well when they're commonly recognized: A lot of people recognize a delete icon, and file icons like the PDF icon, for example.

Icons can also help give a screen some structure. Taking a look at the list of uploaded documents below, the sidebar would look strange without the file icons. Furthermore, the delete icons are clear and understandable due to the context in which they appear.



Verbs vs Nouns vs Adjectives

Icons can be used to indicate verbs, to indicate adjectives (i.e. a state), or to indicate nouns. In the new Appway Workspace, we use them for all three.

- For things like the next and back buttons, and for delete buttons, icons are used as verbs. Clicking on them triggers an action.
- For things like the list of uploaded documents (see previous image), they're used as nouns. Clicking on them gets you to something, e.g. to an uploaded PDF.
- For things like a to-do list, icons are used as adjectives. They show the state of a task in a to-do list; the checkmark means "done", for example. You can't click on these icons.

In general, verb and adjective icons tend to be more abstract than noun icons, because they represent concepts, rather than actual things. Concepts like "log out" or "unfinished" are harder to represent in an icon than "user". Since these icons are harder to design, it's often best to just go with a text label instead of an icon. Text is much less ambiguous than an icon.

If you use icons for nouns, think about whether it's possible to show the actual "thing" they represent, instead of an icon. For uploaded documents, maybe it would be better to show a thumbnail of the actual document, for example.

If You Use Icons

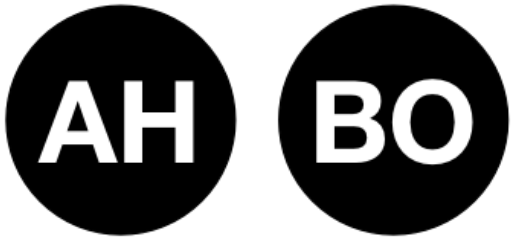
Three rules of thumb when using icons:

1. Use icons when there are no good alternative solutions, or when the solution *with* the icon is clearly and vastly better than the alternative solution *without* the icon.
2. Use icons when there is no danger of confusion, as the icon used is unambiguous and commonly known.
3. Do not use icons without labels. Apart from a small set of commonly known icons, people will not know what the icon means if it's not accompanied by a label.

Here are some of the things you might want to think about when deciding to use icons.

Alternatives to Icons

Often, icons can be replaced with a different design element that offers the same functionality, but is easier to understand. Instead of having different icons for account holders, beneficial owners, and so on, you could use typographical elements to achieve the same effect.



Unique and Consistent

Once you decide to use an icon in your solution, use it consistently, whenever the concept the icon represents occurs. Otherwise, people will have a hard time learning what the icon represents.

Make sure that the icon is distinctive, and that the same icon is not used for anything else (e.g. don't use the same icon for "close" and "delete").

Recommendation

As a general rule, do not use icons. If you do decide to break this rule:

- Use icons sparingly
- Use icons thoughtfully
- Use icons consistently

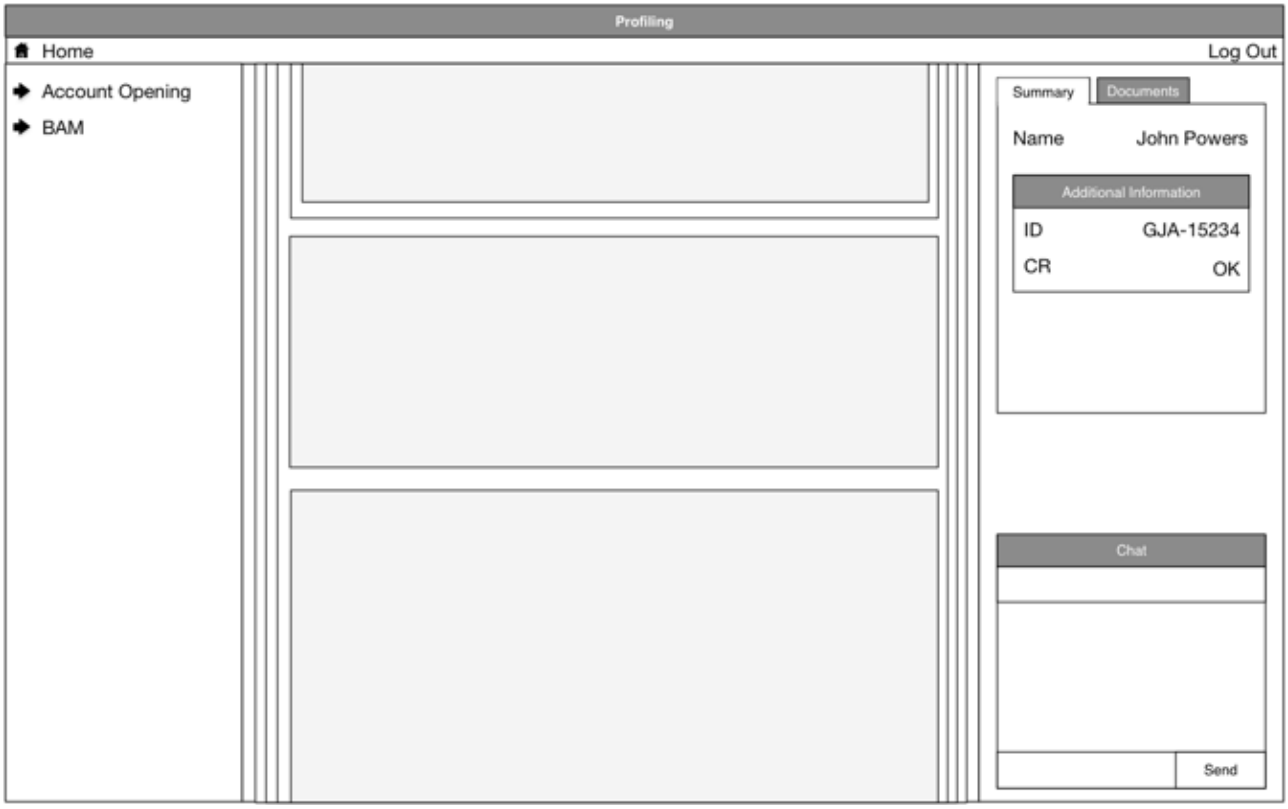
Workspace Design Notes #9: Hierarchies, Boxes and Nesting

Workspace Design Note #8 discussed icons, and how their use has been reduced and simplified within the context of the default Workspace. This post moves onto another area where the visual design has been rethought: nesting.

Appway solutions deal with hierarchical, structured, highly complex data. For example, accounts have roles, which are lists of clients, which have lists of addresses, and so on. This data often has to be shown to the user in a way that makes sense, and can be understood easily and quickly.

The most obvious way of visually representing this kind of hierarchical data is to use boxes, and nest them inside each other. But if you do that, you tend to end up with screens that look like this:

This screen is already difficult to read – and the problem only gets worse once you start to scroll down.



At this point, you no longer really see what the boxes are for.

All you have is a bunch of vertical lines that tell you that "something" is inside "something" that's inside "something else" that's inside "something else" — but no indication of what any of this actually *means*.

The basic problem here is that there's already a lot of things on these screens. Adding more stuff to organize the stuff that's already there works up to a point, but eventually starts to become highly counterproductive: All of the additional things just add more clutter, making the screen even harder to understand.

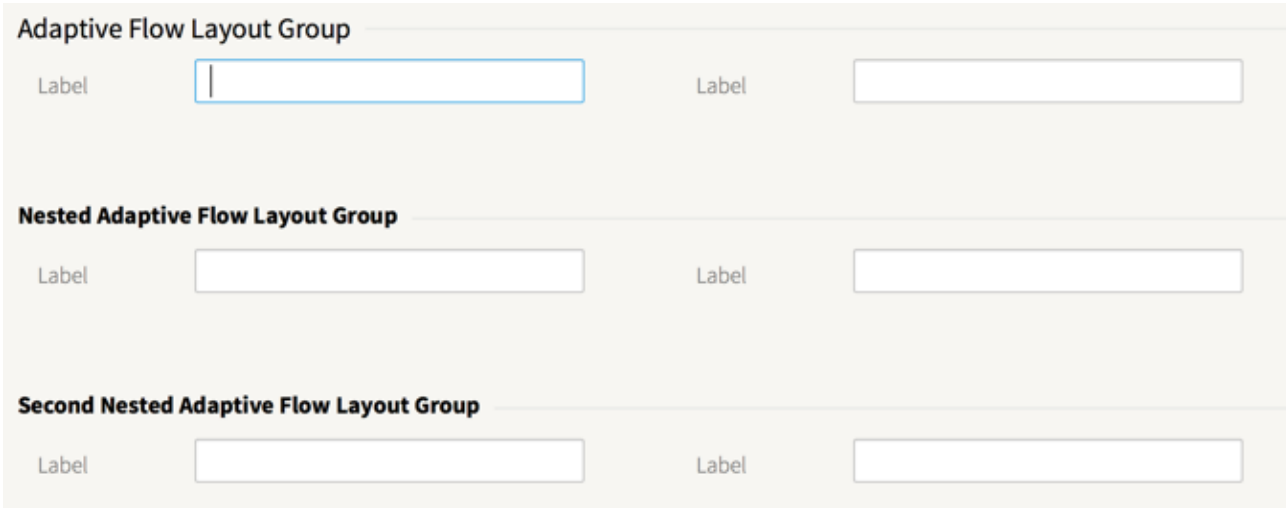
A recipe explaining how to avoid this situation is available on the Appway Developer portal, as we discovered this design set-up in more than one Appway solution; it's an easy trap to fall into.

Simplifying structure

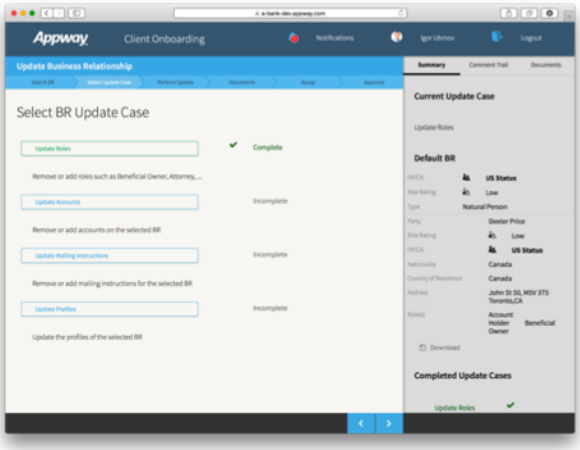
Avoiding excessive nesting was one of our goals while developing the new Workspace design. By default, we wanted to achieve a modern, flat look; a look that did not fill the screen with "boxes inside boxes inside boxes".

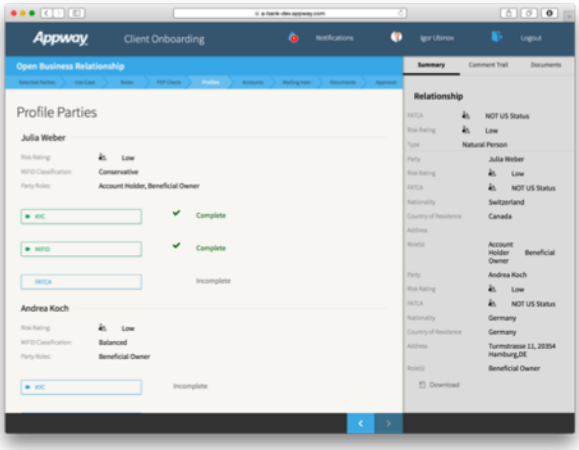
As a result of this decision, Adaptive Flow Layout Groups no longer draw boxes or indents, for example. Instead, they just add titles. What's more, there are only two visual levels of nesting to avoid deep, difficult to understand hierarchies of elements.

In the following screenshot, each Adaptive Flow Layout Group is *inside* its parent group. As you can see, the second-level nesting is no longer visually distinct from the first-level nesting.



With this approach, the COB screens all look clean and flat, despite sometimes containing relatively complex data.





Of course, we realize that while a completely flat design would be preferable, it is not always possible to avoid visual nesting in real-world solutions. But by avoiding it in the *default* Workspace design, we're giving you a little bit of room — when absolutely, unavoidably, unrelentingly necessary! — to add a small amount of nesting to your screens, without immediately causing your screens to turn into the digital equivalent of a Matryoshka doll.

Image courtesy Bradley Davis (CC BY-ND)

Workspace Design Notes #10: Why Use the Border Layout Manager?

Our Workspace Design documentation contains clear instructions: "To create the basic [screen] layout, use the Border Layout Manager." It tells you to "use the North region of the Border Layout Manager for headers" and "use the South section of the Border Layout Manager for footers."

When needed, we added some additional rules — if you use a Table of Contents component, put it into the left sidebar, but make sure that it appears *below* the process header, for example.

Saying "what" to do is not the same as explaining the reasoning behind this choice: We've been pretty circumspect about providing detailed explanations about *why* we think the basic screen layout should be built using the Border Layout Manager. This post remedies that situation.

There are two different reasons for the recommendations on usage of the Border Layout Manager.

- Separation of concerns. Each area of the Border Layout Manager should have a clear, well-defined, consistent reason for existing.
- Hierarchy. The Border Layout Manager implies a visual hierarchy; the things we put into the Border Layout Manager need to be consistent with that visual hierarchy.

Separation of Concerns

Any good layout has to be driven by the content. The main goal is to have well-organized content that helps users get their bearings: Where am I? What's on this page? What do I need to do to achieve my goal?

The first step in defining a layout should be to assess what content needs to presented on screen, and then identify a suitable way to structure it.

A typical Appway solution has five different categories of things it shows to the user.

- Global functions or "utilities" (logout, show notifications, search, user settings)
- Global navigation (home, start account opening process)
- Forms or "content" (enter data into a bunch of fields on a screen)
- Local navigation (go to the next screen in this process)
- Context information (Who are the parties in this account opening? What's their risk rating?)

It's usually best not to mix these categories. You wouldn't want to put the button that starts an entirely new account opening next to the button that moves you ahead a step in the current account opening, for example. It would also not be advisable to put the logout link next to the field where you enter a party's name.

What's more, you want people to clearly and easily understand what something is. Is this a button that moves me ahead in the current process, or is this a button that takes me out of this process and back to my portal? You can greatly improve clarity by giving each category of thing its own location; that way, people can easily understand *what* each thing is, based on *where* it is.

Coincidentally, the five categories of things a typical Appway solution shows mostly map to the five areas of the Border Layout Manager. In order to help people understand Appway solutions more easily, we can therefore just map each category to one of the Border Layout Manager's areas. The mapping is intuitively obvious for most of these areas. You probably *need* to put global functions into the header, and forms into the content area, right?

Hierarchy

The second reason for the recommended usage of the Border Layout Manager — and the missing clue that fully explains where everything needs to go — is the hierarchy of the screen.

The five different categories of things in Appway are hierarchically related to each other. Global functions and global navigation are at the top of this hierarchy: They're always applicable, and are independent of other content.

The content area itself is therefore below the two global elements: What's visible in the content area depends on the global action or global navigation that the user triggers.

Context information could be yet another level below — or on the same level as the content — depending on what it is about. If it's just context information about the current content, it would be below it; if it's context information about a process (of which the content area shows one step), it would be on the same level as the content area.

As for local navigation, it probably makes sense to think of it as residing somewhere between global navigation and content. Local navigation influences what's visible in the content area, but only to a limited degree; it can't entirely change what the user is doing, just move the user through the running process.

If you think about the relationship between these things in this way, you'll end up with a hierarchy that looks something like this:

- Global Actions/Global Navigation
  - Local Navigation
    - Content Area
      - Context Information

Stated very briefly: "Things above influence things below".

Two different elements reside at the top of this hierarchy; we should be able to distinguish between them, too. Global Actions need to always be visible. You always want to see your notifications, and you always need to be able to log out. Global Navigation, on the other hand, does *not* need to be visible when you're inside a process. You probably don't want to start another account opening from inside a running account opening, so it's therefore okay to only show Global Navigation outside of running processes.

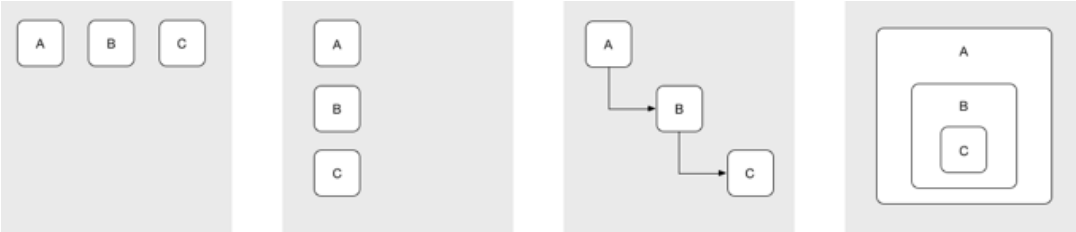
(The only exception to this is the Home link, which needs to be available at all times. This is why we've taken "Home" out of the Global Navigation section, and put it into Global Actions.)

Based on this reasoning, Global Actions are above Global Navigation in the hierarchy.

So now we've organized our five categories into a hierarchy. What exactly did that buy us? Well, the Border Layout Manager *also* implies a hierarchy. If we think that Global Actions should be above Global Navigation, for example, then they should be visually organized that way in the Border Layout Manager as well.

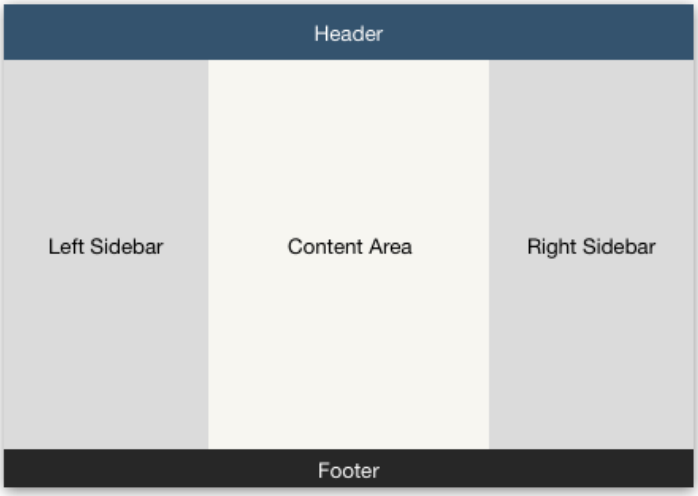
Visual hierarchy

What's a visual hierarchy? The way visual elements are arranged on a screen conveys a hierarchy to the user. The "Western" reading flow (top-down, left-to-right) implies a hierarchy. Things on the left are "above" things on the right. Nesting (outside-inside) also implies a hierarchy. Things outside are "above" things inside. And so on.



This ties in with our goal of guiding users. If people order their reading of a screen in a standard way, we want them to start with information about where they are (the Phase Chevron, for example), followed by the content they're supposed to look at or fill in (a form on a screen, for example), and eventually end up at a button that brings them to the next screen (the "next" button in the Flow Bar, for example).

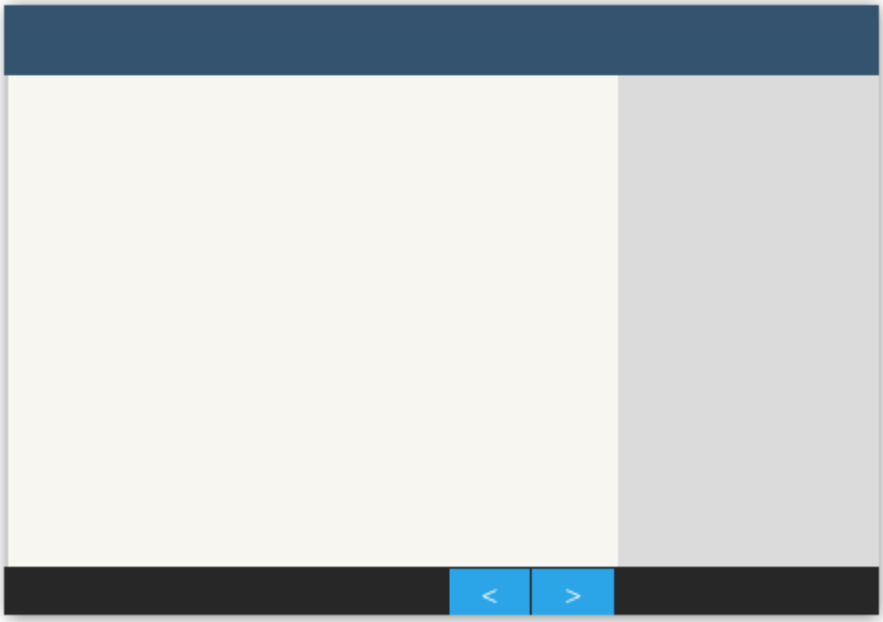
Now, what kind of visual hierarchy does the Border Layout Manager imply?



The header is clearly at the top of the hierarchy. Since the footer encompasses the full width, I would put it second. It is important to note that this is an instance where users could either read the footer as "below" the content, or the content as "inside" the footer. This changes the perceived hierarchical relationship — but visual elements like shadows can help push users into one or the other direction.

For the two elements in the center, I'd put the left sidebar visually "above" the content area. The right sidebar could be either "above" the content area, or "below" it, depending on context.

As an aside, something else to consider here is that the next and back icons should be aligned with the content area; that way, even though they're hierarchically "above" the content, the alignment conveys the idea that they have an effect on what's visible in the content area.



Now that we have a hierarchy for the categories of things in Appway, and a hierarchy for the areas in the Border Layout manager, it's a simple matter of matching the two.

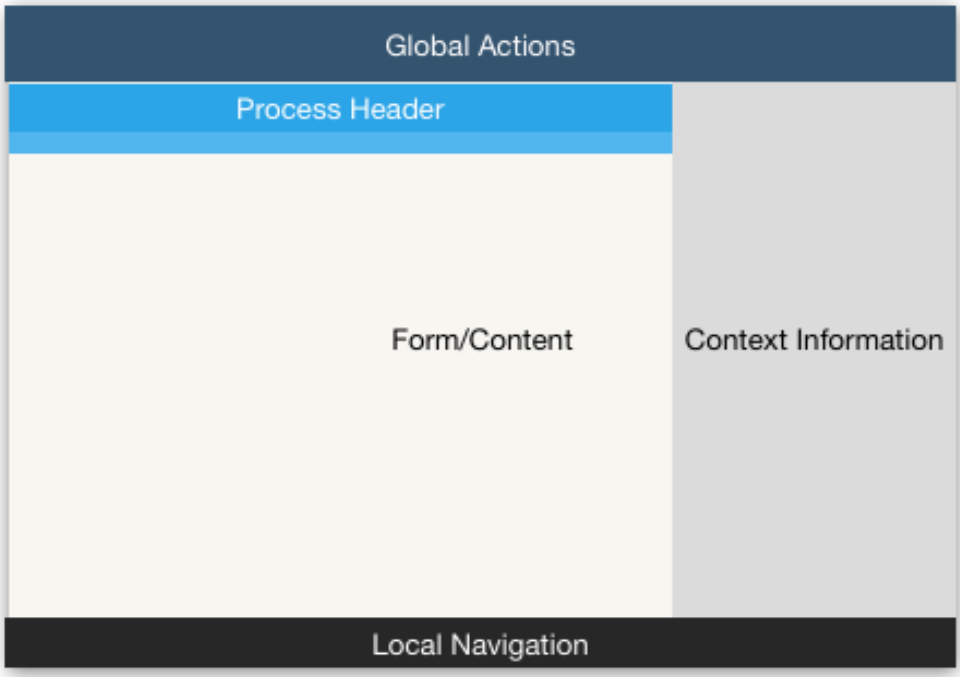
The portal doesn't need the context information and the local navigation, while the process screens don't need the global navigation, so we end up with the following two arrangements:

In addition to marrying our category hierarchy with our Border Layout Manager hierarchy, this arrangement has the added benefit of making process screens visually distinct from non-process screens. In other words, users always know whether or not they're in a guided process.

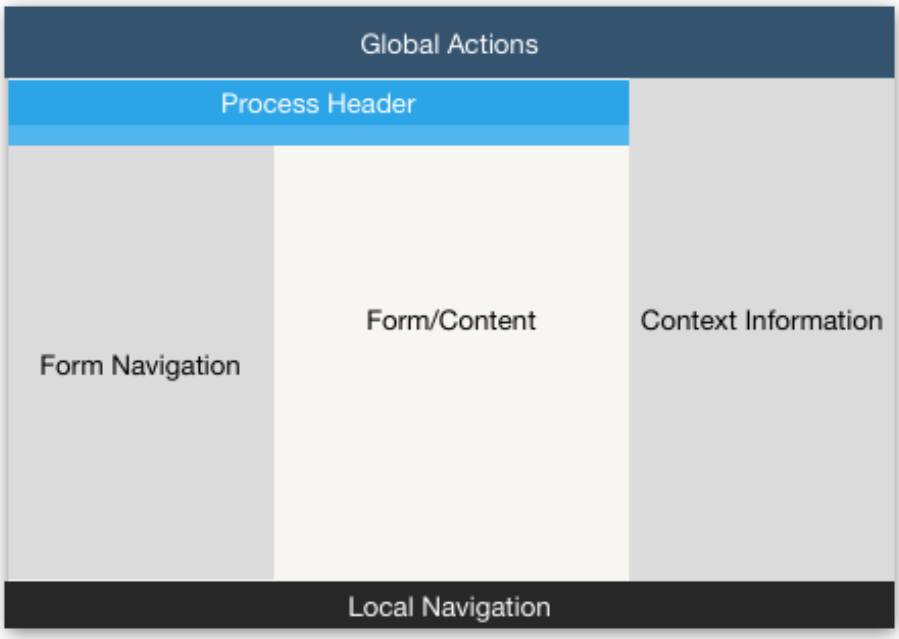
Finishing Touches

There are a couple of things remaining: What about headers within a process? What about in-screen navigation? The decisions on where to place these items build upon the design outlined above.

When we need a second header inside the process, which tells you which stage of the process you're currently in, where does it go? Well, hierarchically, it's above the content (it tells you information *about* the content), but not necessarily above the context information (which can be about the process, not just the current screen). That puts it here:



On longer screens, there is sometimes a Table of Contents component that allows users to jump through a screen. Where does *that* go? Well, hierarchically, it's above the actual content of the screen, because it allows you to navigate through it, but below anything else. This means that it needs to go *below* the process header.



Conclusion

While explaining the reasoning behind our Workspace Design guidelines is interesting, it's also important that the final design decisions don't only come down to logical arguments. Instead, design is about what works.

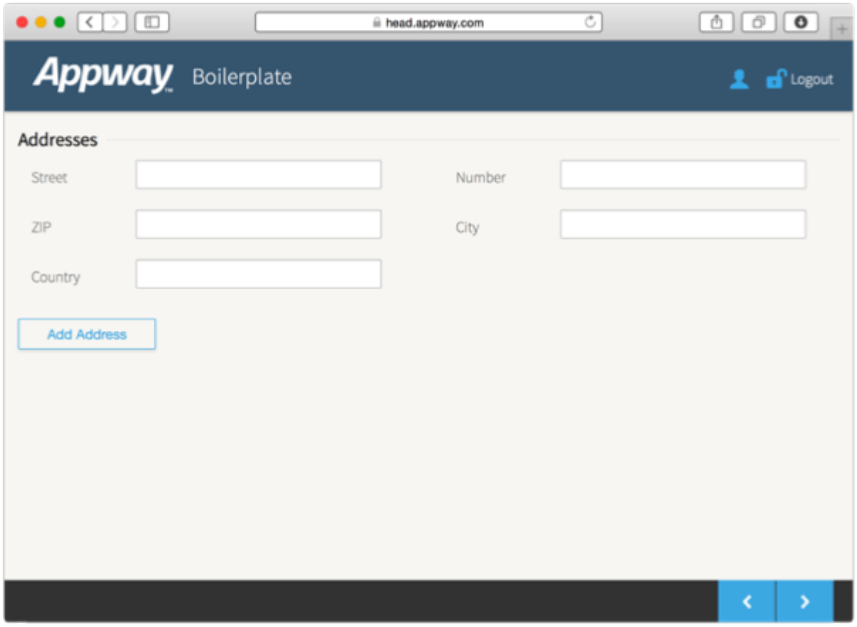
So the question isn't really "Why did we put context information inside the right sidebar?" Instead, the question is, "Do our users understand how this works?"

Our user testing activities suggest "yes".

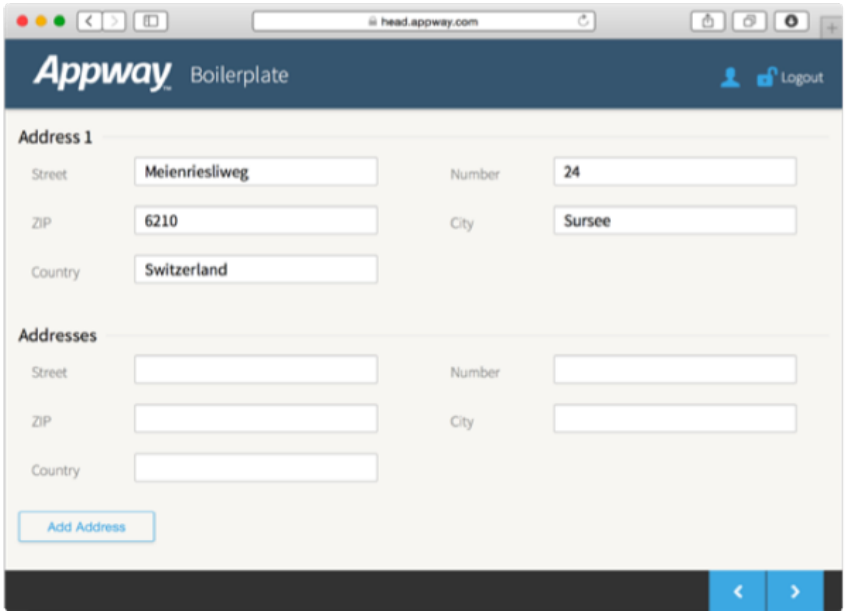
Workspace Design Notes #11: The "Adding Elements" Trap

This Note focuses on a very common usability issue: How to design a user interface that allows the user to add a new element to a list in a way that isn't too confusing. But that doesn't sound like it should be difficult, right? And it also seems like something peculiarly specific to be concerned about. Well, it turns out there's more to it than initially meets the eye…

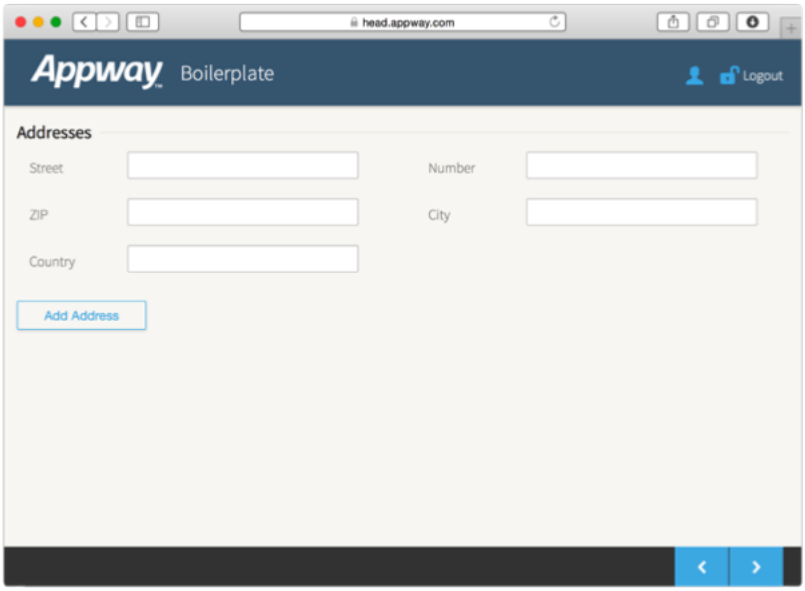
Let's say you're designing a screen that allows the user to add an address to an address list. Your screen might look like this:



Once the user fills in the form and clicks "Add Address", the new address is shown at the top, is editable, and a form to add a second address is shown below it. The screen now looks like this:



All of this seems very logical. Unfortunately, it's not. Pretend for a second that you're seeing this screen for the first time:



You've just opened it, you're in a hurry, you need to quickly enter a client's address, and your colleagues are already discussing where to go grab lunch.

"Okay, I'm on the *Addresses* screen. Let's see... here's the form, enter the new address. Do I need to click the *Add Address* button? No, I only need one address, and I've already entered one. Click 'next', and I'm done. Time for lunch!"

Unfortunately, the user just lost the entered data; since "Add Address" was not clicked, the address was never stored.

Breaking the Pattern

It's very easy to either miss the "Add Address" button, or misunderstand what it does. On most Appway screens, you can just fill in form fields, and click "next". Everything you've entered is automatically stored. But not on this screen.

This issue occurs on all screens that show forms to the user, forms where information is only stored if the user clicks on a button other than "next" or "previous".

In fact, it also occurs outside of the Workspace. Have you ever tried to add an inline style to an Appway Screen component, saved the Screen, tested it, and wondered why your inline style was lost? Well, the Appway Screen editor Style tab behaves the same way as the example above. It's easy to misunderstand what the "Add" button does, or just not notice it at all.

It looks like the selected component has a "MyCSSClass" class — but it does not.

Note: The reason why we decided not to alter this yet is because we plan to replace inline styles with a better system in a future version of Appway.

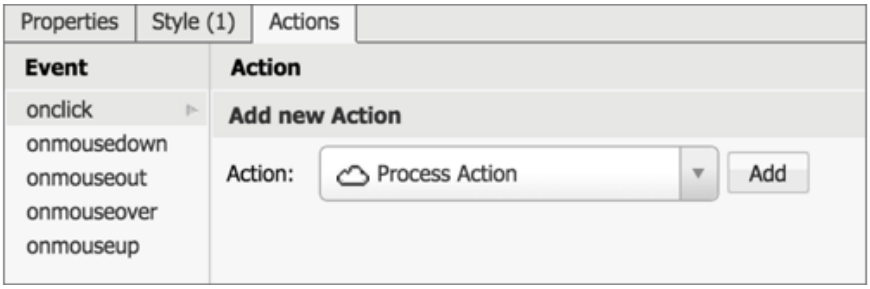
But let's return to the most important question: How *do* you fix this problem?

There are two options. You must either *force* the user to click the button, or make clicking the button unnecessary.

Forcing the user to click the button

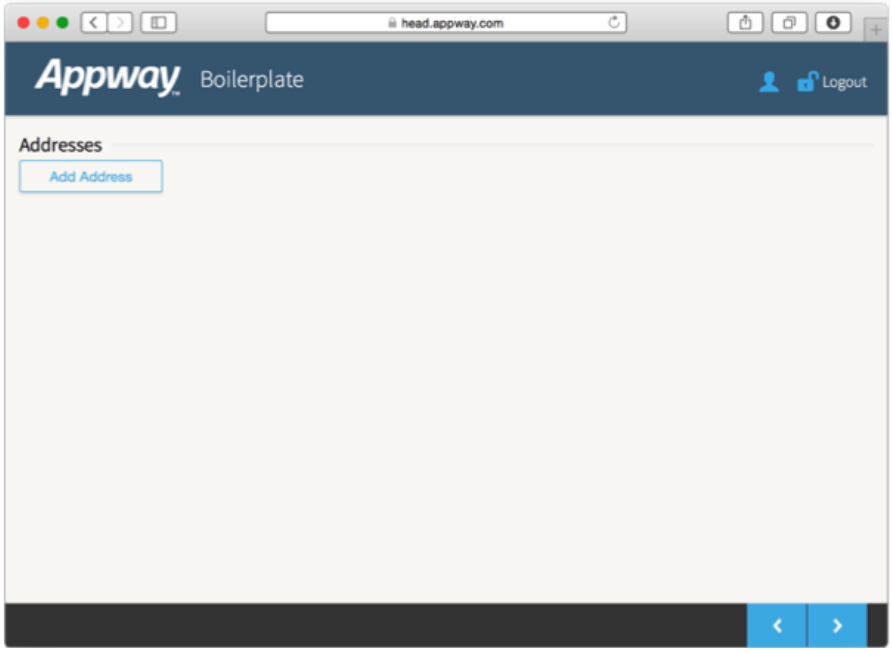
In both the examples above, the user doesn't click the button because it's possible to fill in the form without clicking the button. Therefore, the easiest solution is to *not show the form* until the button has been clicked.

For example, if you want to add an action to a Screen component, Appway will not allow you to edit the action's properties until after you've clicked the "Add" button.

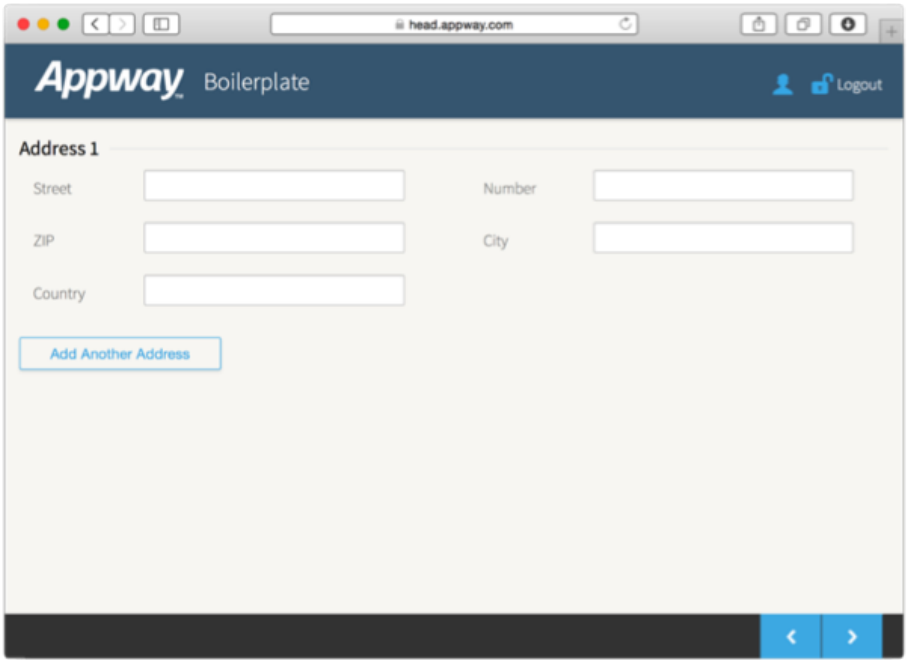


Unlike styles, you've never accidentally lost any actions; it's impossible.

If you apply the same solution to the address example, the user will see this upon first opening the screen:



At this point, the user's *only* choice is to click "Add Address". It is no longer possible to forget to click the button. Once clicked, the screen now looks like this:



This design makes it impossible for the user to accidentally lose data.

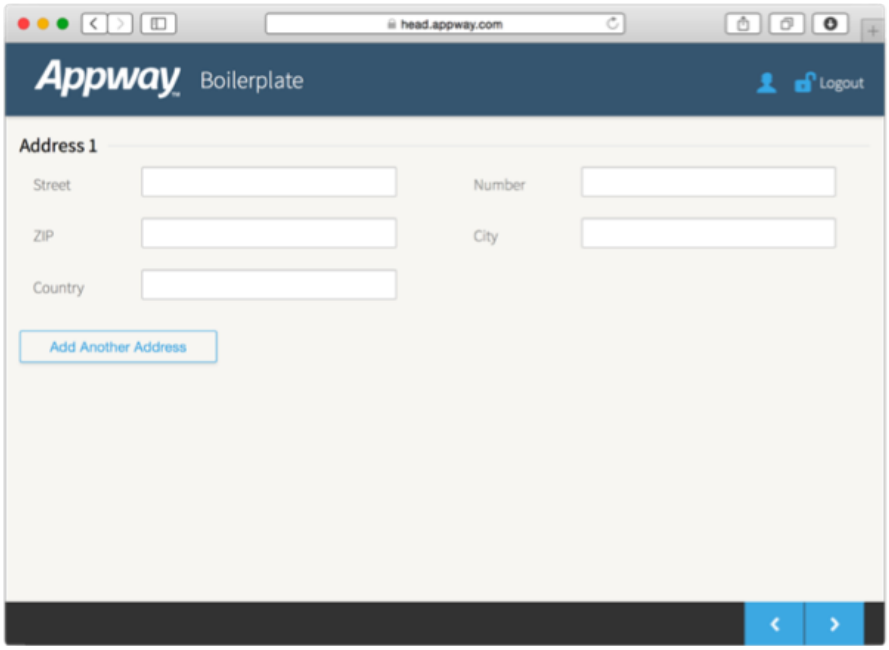
One last detail: After the user has clicked "Add Address" once, the button's label changes to "Add Another Address", thus ensuring that the button's meaning is obvious. Users only have to click it if it they want *more* addresses to be added, not if they want to store the data they've already entered.

Forcing the user to click the button is one solution to this problem. Another solution is to make clicking the button unnecessary.

Make clicking the button unnecessary

Notice how in the solution above, when users first reach the Addresses screen, they *have* to click the Add Address button? There's no other choice — unless they don't want to enter any addresses.

If you know that the user has to add at least one item to a list, you can skip that first screen, automatically add an empty element to the list, and immediately jump to this screen:



This way, clicking the Add Address button is no longer necessary; the user can fill in the form, click "next", and the address will be stored properly.

Conclusion

Avoid screens that show forms where data will not be stored when the user clicks "next" (unless they are forms for data that never has to be stored— search fields, for example). Such screens are particularly common when users have to add and remove elements from lists.

For those screens, use one of the two solutions described above, depending on whether or not the user has to add at least one element to the list.

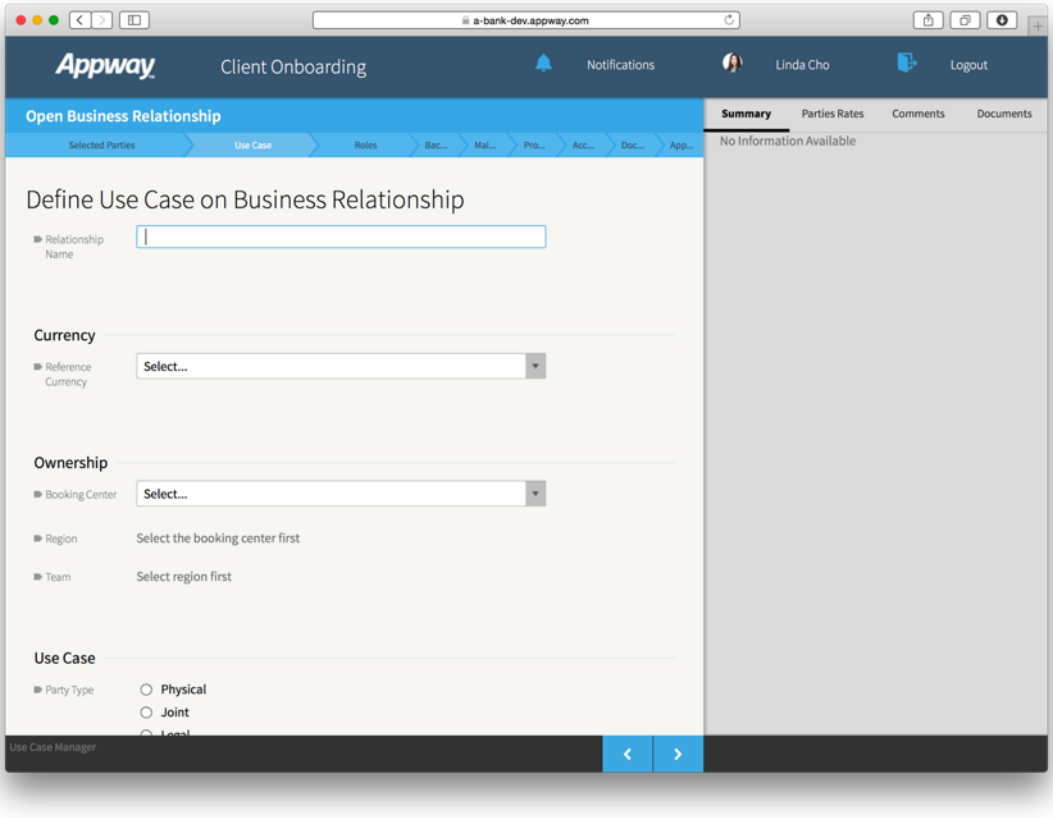
Workspace Design Notes #12: The Mystery of the Disappearing Chevron

It's hard to predict the kinds of things people will notice in a user interface. When working on the new Workspace, I don't think anyone would have predicted one of the most commonly asked questions:

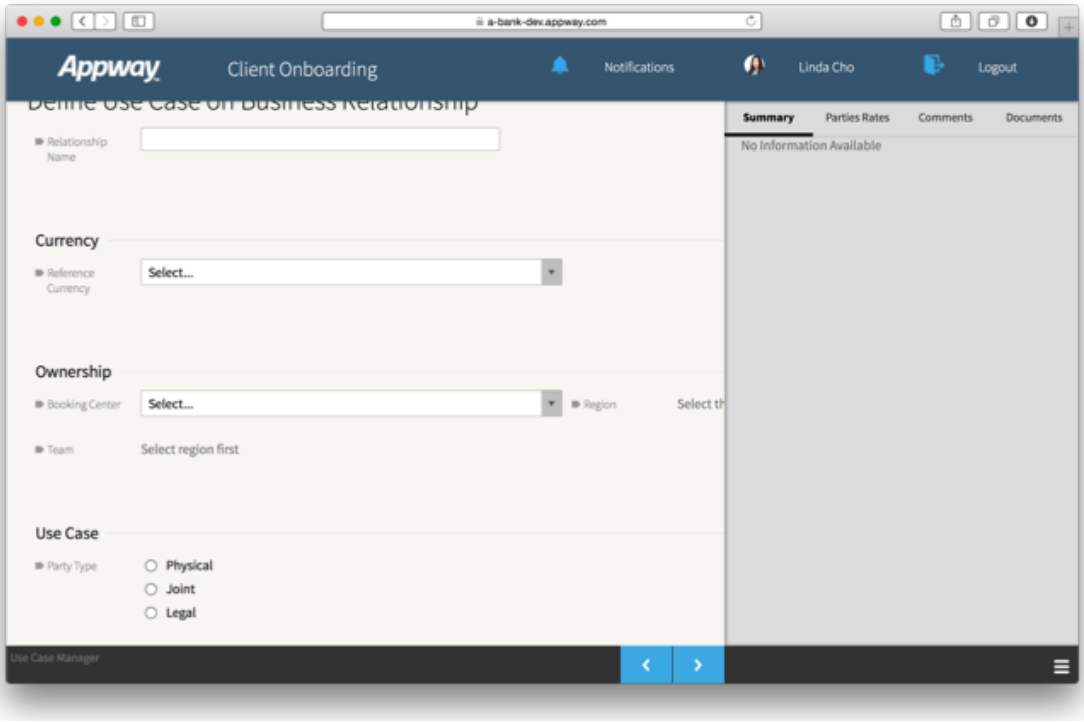
Why does the Phase Chevron scroll away?

When you're in an Appway process, there's a phase chevron at the top of the screen that tells you the title of the current screen, the phases the current process has, and where in the process you are.

In the screenshot below, the user is in the second phase of the process, "Use Case".



Once the user scrolls down, the chevron scrolls out of the screen.



This doesn't happen with the normal, main header. It stays put. It's only the chevron that disappears.

Why? For two reasons.

Reason 1: A clear, focused design

Once users start scrolling down, we're assuming that they want to fill in the form. This means that they know where they are, and what their task is. Hence, showing the chevron is no longer necessary; it shows information the user is already aware of. To make sure that the UI is now focused on the user's current task — filling in form fields —extraneous visual clutter is removed, and the chevron scrolls away.

Reason 2: Mobile

On your average desktop PC, the main header and the chevron combined take up, at most, a sixth of the window's height. That's fine; there's still plenty of room for the other UI elements. But on a mobile phone, the situation is rather different. Here, the two headers combined can easily take up a third of the screen. On these devices, allowing the chevron to scroll away provides some much-needed space.

More Complex Behavior

We'd like to support more complex behavior in the future; maybe shrinking the main header when the user starts scrolling. This requires a lot of thought – any feedback on this topic, as on any other of the topics covered in the Workspace Design Notes, is very welcome.

Workspace Design Notes #13: Resizing for Touchscreens

You've probably noticed that the new Workspace renders fonts a little bit larger on smaller screens. On a mobile phone, for example, labels are rendered with font size 16px, but on a desktop, they're rendered at 14px.

At first blush, this seems exactly the opposite of what you'd expect. Smaller screens have less space - wouldn't you want to make things *smaller*, so more stuff fits?

Image courtesy Death to the Stock Photo

There are two reasons for making things larger.

First, mobile phones already render everything quite a bit smaller.

Second, computer mice are more precise than fingers.

Let's look at each of these reasons in turn.

Reason 1: Mobile phones already render everything quite a bit smaller

If you specify your element sizes in pixels (as we do in the Workspace), the same pixel is going to look quite a bit smaller on your average phone than on your average laptop or desktop PC.

Here's the same red square as it is being rendered on my laptop, and on my phone:

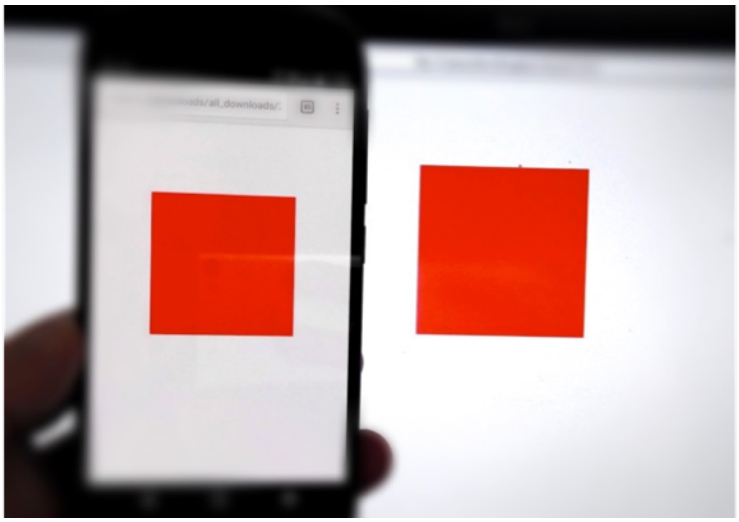


Image courtesy L. Mathis ©

As you can see, the red square appears larger on the laptop's screen in the background, compared to the mobile phone's screen.

Increasing the size of elements on smaller screens is intended to counteract this effect. It won't completely make up for it; even though we increase the font size on smaller screens, fonts still appear smaller on your mobile phone than on your desktop or laptop.

Reason 2: Computer mice are more precise than fingers

You've probably noticed this: on your Mac or Windows PC, UI elements like buttons tend to be quite small. By comparison, the same kinds of elements on an iPad or mobile phone are much larger. Why do Apple and other manufacturers do this? Why not make things smaller on smaller screens, in order to fit more on the same screen?

Mainly, it's because you use your Mac or Windows PC with a mouse or a trackpad, but you use your iPad or phone with your finger.

A mouse or trackpad is a precise input device. Your fingers, on the other hand, are very imprecise. It's easier to hit a small target with a mouse than with a finger.

By increasing the size of elements on smaller screens, we're making it easier for people to hit them with their fingers.

Of course, increasing the size of elements also makes them easier to use with a mouse, so whenever possible, *we show larger elements on all devices, instead of dynamically adjusting the size*.

Other reasons

There are a few other reasons. For example, if you don't see perfectly, it's easy to adjust the text size on a desktop device. It's much harder on a mobile phone. Making things a bit larger by default makes it less likely that people will run into accessibility issues.

There's just one thing...

There's just one slight downside with the current Workspace implementation of this feature: it doesn't distinguish between desktop devices, and touchscreen devices. In some ways, this is because it's impossible to do this correctly. For example, a Surface Pro 3 has a touchscreen, but can also be used with a mouse. Should we increase the size of elements on this device, or not?

For the first release of the new Workspace, we could be accused of taking the easy way out. We're currently increasing the size of elements for all screens that are 1024 pixels wide or less. Why 1024 pixels? This is the logical width of the iPad's screen. By setting the threshold at 1024 pixels, we ensure that iPads get the larger sizes.

This has the side-effect of also increasing the size of elements on desktop PCs, if you use Appway inside a small window.

For the future, we intend to be smarter about how we adjust the sizes of these elements.

Workspace Design Notes #14: The Luxury of Space

Users should trust the applications they use, and visual design is an important tool for building that trust. One visual detail that conveys things like stability and quality and value to people, and thus allows them to build trust, is empty space.

Empty space conveys an impression of efficiency, thoughtfulness and focus. Not sold on the idea? Then let me briefly demonstrate what I mean.

Here are two pictures of clothes stores.

Image courtesy thinkretail (CC BY-NC-ND)

Image courtesy Brad K (CC BY)

Both pictures show stores selling clothes, but the associations they trigger are very different. Picture 1 shows a store that clearly portrays an air of luxury, quality and high value. Picture 2 is attempting to communicate that the products it sells are cheap, and cheaply made.

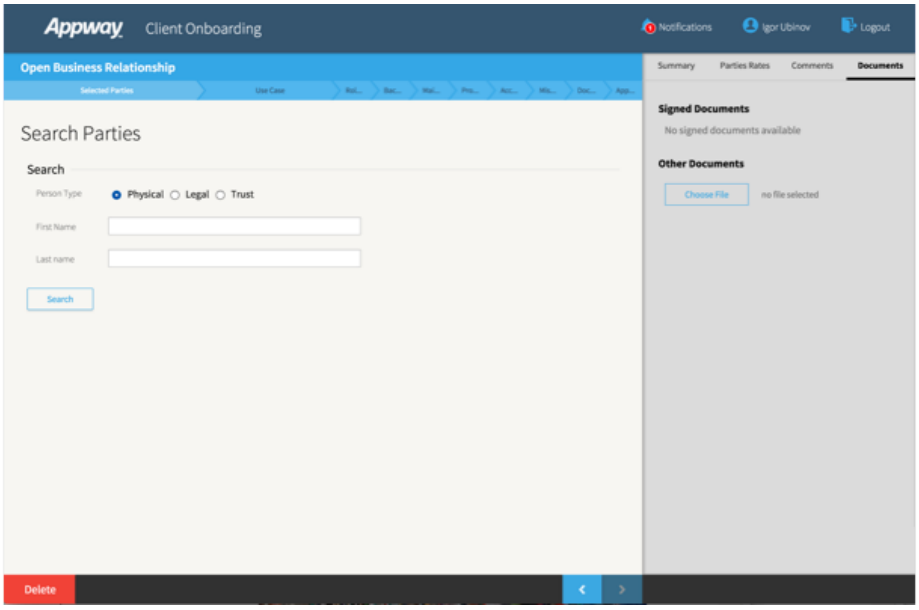
Please note that I'm *not* saying that the store in picture 2 is poorly designed! Both stores are designed really well; they both communicate exactly what they want to communicate. If you're looking for luxury, you're not going to go to the store in picture 2. Similarly, the reverse is also true. Both stores' designs are successful at communicating what they are offering, and align customer expectations with customer experience accordingly.

One way these two stores communicate this is by their use of space: Picture 1 shows a lot of empty space; picture 2 has no empty space.

Appway as a company, and Appway as a product, fall into a similar category to the store in picture 1. Appway is a quality product.

The same use of space that made the store in picture 1 look luxurious, and the store in picture 2 look a little overwhelming, can be used to influence the kinds of associations people make when they use solutions built with Appway.

That's why the new Workspace design uses space liberally. There's a lot of empty space in the new Workspace, we don't put stuff into every nook and cranny, and we have visible margins between elements. All of this is completely intentional.

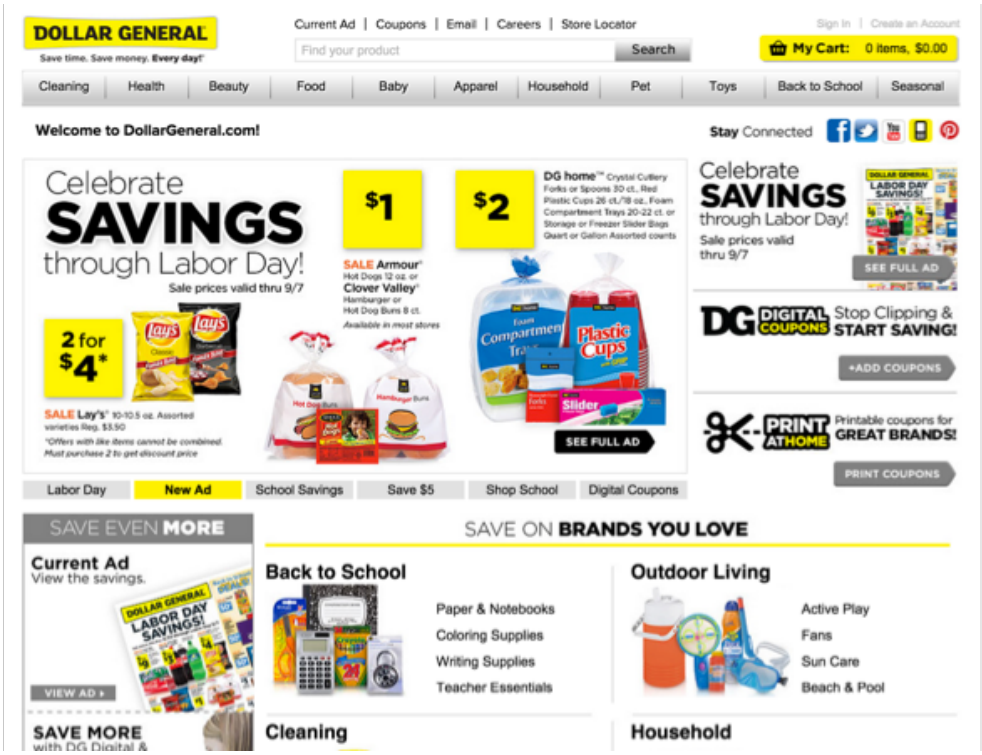


There's a basic human need to fill empty space with stuff. This need is so common that there's a term for it: Horror Vacui.

"Horror Vacui"

Horror Vacui. are one of the reasons why solution engineers often look at the new Workspace design, and come to the conclusion that it needs more stuff. And that's okay. If anything, the new Workspace design erred on the side of making things too spacious, so it's okay to dial that back a little, and add some more stuff.

But always keep in mind what kind of value perception you're generating in your users. The last thing you want to do is add so much stuff that your solution ends up looking like a dollar store or classified ads website.



We want to avoid this not because these websites are bad, but because we're in a different business, and think a different message should be communicated to users.