Coding competition overview

Coding Competition

Welcome App Developers!

- The Competition Brief
- The Technical Overview (rules & FAQs)
- The *Future Bank* Reference site

Files to take home

Download the file TopCoder-August2012.zip. It contains The Sample Data, some sample Apps and the Developer Container.

Submission and socialising

- The competition submission process
- Managing registration info
- Discussions on Yammer
- Editing or commenting on this Wiki

Building Apps

Getting Started

- Setting up your machine
- Containers & Apps
- Framework component overview
- Hello World - Sample App
- The Developer Container
- Coding Comp Sample Dataset
- Doing it with style

Developing

- Building your App
- Dealing with data
- Dealing with responsive design

Guiding principles
API Documentation
JavaScript best practices
Code samples
FAQs, common issues & errors
UI controls
Instrumentation & error handling
Validation
Naming Conventions

Advanced

Context Editor App
Including external resources
Guidance for extending the capabilities

Core Concepts

Here's what you need to know about what's special about Project Helix Apps:

Design Guidelines
Apps are managed and released independently
Building once for multiple devices and form factors
Building once for multiple user types
Federated IT capability
Legacy integration
Outage-less deployments & changes
Scalable UIs using Responsive design
Tailoring to a subset of users and devices

The Competition Brief

Coding Competition – The Brief

The primary objective of the coding competition is to allow participants to demonstrate technical aptitude by creating a banking-centric technology solution for a given problem. The intent is for the participants to build an ‘app’ that could be of use to the Commonwealth Bank Group, its subsidiaries and / or its customers.

The App Store – a Developer preview

We want you to build an app that can reside in the future CBA App store, but there's some caveats. The CBA App store is still under development, and we're not ready to show the full picture yet. We also have had to make some adjustments to make sure that people can develop working software outside of the CBA, so we've partly compromised the approach. What we **do** have is an early preview of the user interface layer that will allow you to write a web site / web app that can begin to test out how apps will interact with each other, how they'll inherit rudimentary theming, and use elements of the Responsive Design framework.

*Please note – if the app framework feels unpolished, **it should be**. We have another 3-6 months of activity before we'll be using this on a 'real' project.*

Problem Summary

The Bank has created a new App Store framework, summarised in the The Technical Overview. We want you to build an app that lives in this app store. **We want you to use the APIs provided to help create an interesting customer experience**.

The first apps are built around a Future Bank portal. The Future Bank portal is a dynamic web site that allows customers of CBA, BankWest or ASB to interact with the bank online – on their iPad, on their mobile phone, on their PC, on their TV. The Future Bank portal currently holds a few rudimentary apps to frame the problem: people can log on, view a summary of their accounts, drill into the transaction history of their accounts, and complete a payment.

The apps built so far are fairly rudimentary. We'd like to invite you to solve any one of the following problems:

- **Replace the "Home page"** with a more compelling solution. This is what a customer sees when they log on. You might want to organise the view differently, display accounts differently, or produce a more interesting composite view of their data by mixing in different data from other sources. You might want to show a graph of their total account summary, you might want to give them a vastly different view of their accounts.
- Using the **Data APIs** that are available, you can **create a unique payments experience**. Payments innovation is a key focus at the moment for the group: if you can use our payment APIs to build something truly different, we'd be impressed.
- **Build something using our data APIs.** We have collapsed all of our data services into a single, very rich data API for the purpose of this competition. This has a very wide range of sample data. You can use this as the basis for any kind of application. For example, you might want to look at the address book data and display a unique view of people you pay. You might want to use the Customer Messages source and show how you can make it easier for customers to read their messages
- **Build something using entirely different data sources that makes sense for customers in an online banking experience**. Our APIs are centred around banking, using data that is available to us. However, you might want to show a set of financial data from other institutions. You might want to show a calendar of financial events, hooking into a Facebook calendar. If our sample data isn't sufficient, create your own data.

Format of Entry – How to build for an app store

The *Future Bank* portal is, really, just a simple web site. Apps that reside 'in' this app store are (for now) just simple web sites. If you can build a web site, then this can be an 'app' that can be accessed through the App Store. Details of specific things to include are outlined in Building your App.

Technology

We've specified we want people to build web sites. Importantly, ***we don't care what technology you use*** so long as the UI component is accessible in a web browser. HTML, HTML5, Javascript, Flash, Silverlight, other RIA technologies are all fine. If you want to write your web tier in Java, C#, PHP, Ruby – it doesn't matter. If

you want to use open source or 3rd party plug ins, that's fine too. The App Store aims to be able to support any server-side technology you want to use.

Format of Submission

To submit your app, please review The competition submission process. We're expecting you to submit:

- All code files
- If your site is hosted on an accessible server (public, or on our intranet), the URL. This is not mandatory at all, but it could be very helpful
- Installation instructions on how to get your app running

…Doing something entirely different

We've spent a lot of effort to make this competition a bit more focussed, and try to allow competitors to focus predominately on a narrow technical problem. With this comes some compromise – we've made the problem narrower than last time, we've asked for people to build web applications.

However: if you really want to code to a different problem or solution, please don't be put off. Maybe web solutions aren't your thing, or maybe you have an idea that won't 'work' as an app. If you have another solution in mind that could be used in a Future Bank concept, please go right ahead. And if you find the sample data too broad or not right to demonstrate your needs, please use your creativity to do whatever you want.

Resources

All the resources are declared in TopCoder-August2012.zip -file contents

The Technical Overview

```
import general.initiative;
import general.technical.coding;
import general.language.*
import general.time.spare;
import CBA.ES.People;
public class CBA.CodingCompetion.Instructions {

private string[] sections;
public Instructions()
{
sections = new string[]{"Introduction",
"Competition Overview",
"Submissions",
"Frequently Asked Questions"};
Navigate.To(sections[0]);
}
public void Initialise()
{
Navigate.To(sections[1]);
}
public void BeginSubmit()
{
Navigate.To(sections[2]);
}
public void Query(string query)
{
Navigate.To(sections[3]);
Document.Find(query);
}
}
```

Tim Hogarth
Version 0.02

Introduction

Welcome, and thank you for entering the second CBA Coding competition, sponsored by the Chief Technology Office. This competition is designed to be a fun, interesting and ultimately rewarding challenge that will hopefully discover and promote hidden talents in the art of writing software code, within any part of our organisation. This is our second coding competition – we received such an enthusiastic response last time, we figured it made sense to make it even bigger this time.
This document is meant to be a technical introduction to the event, and help you prepare. This will not be the only information you will receive, but this document should be your initial guide to help you get a feel for what we are looking for, what you need to do to earn a chance of winning, what you need to do to impress the judges, and how you can best make use of your time.
Ultimately, participation is one of the best things we're looking for. We're proud that you've volunteered to show off your wares. We're hoping the challenge is both interesting and achievable. As one of our official entrants, we will also extend an exclusive invitation for you to attend a celebratory event where the ultimate winner will be announced. Entry will be limited only to those that applied and I encourage you to attend what will be a fun and exciting night.

- *Mok Choe*

Chief Technology Officer

Competition Overview

Introduction

The coding competition is designed to be an event where employees of CBA can compete for a prize by producing the best technical solution to a problem proposed by the CTO. This competition will commence at 9am on Monday the 20th of August. There are two classes of entry: the **Fun Run** and the **Marathon**. For the fun run, you can submit your entry at the end of fortnight. If you want to polish your solution to its very best, you can submit after four weeks (the marathon). We'll be awarding prizes to both.
The Fun Run submissions must be received by 11am, Monday the 3rd of September, and Marathon entries by 11am, Monday the 17th of September. All submissions must be include a copy of the code written for the challenge. Judging will take place over a period of 2-4 weeks. A shortlist of finalists will be invited to present their code, design and approach to a technical panel for further evaluation.
The expected submission will be a technical creation that demonstrates a concept along the lines of a specific problem. This creation is expected to be a working version where both the raw code and the working run time can be inspected by the judges.

Key Dates

Phase	When	What happens
Technical pre briefing	*To be supplied*	Official entrants notified of their entrance status. High level overview of competition and FAQ distributed to entrants
Competition opens	*20th of August*	Official entrants are emailed formal competition instructions, including the **problem description,** sample data and supporting technical documentation
Submission acceptance opens	*27th of August*	The competition submission process is made available for entrants to submit their coding samples
First round of submissions close	*3rd September*	The "Fun Run" submission is locked down. No more submissions can be received.
Submission acceptance closes	*17th September*	The submission process is locked down. No more submissions can be received.

The Challenge: Introducing The App Store

Last time, we gave entrants a simple data file and asked them to use it for their own purposes. This time, we're doing **something a little different**. The CTO group is in the progress of designing the next standard approach for software development for CBA distribution channels: an internal, peer reviewed app store that will allow internal and partner development teams to build applications in the "app" model espoused by Apple, Google and others. In 2013, we expect to be building our newer CBA applications in this way – we'll move from adding yet more pages to NetBank / CommSee / CommBiz to adding 'apps' to those platforms: safely, incrementally and as

fast they're ready.
Our App Store is still in development and not feature complete, but we'd like to invite you all to help us 'kick the tires' to prove this approach out. We will be asking you to write creative solutions that can 'live' in the CBA app store.
To make our app store feel a bit more 'real', we're going to have a sample suite of apps that make up a customer self-service portal – something our customers can use on mobiles, tablets or PCs. This portal will be all things any customer might use with the CBA Group, and combine elements of some things we do today and some things we don't. We'd like to make this portal a compelling customer experience and we invite you to add an app to this portal.
We will be giving more specific goals around what kind of app we'd like to see, and the kind of inputs you can use to interact with this data. We'll release the specific details of this scenario at the opening point of the competition. The details will be revealed to all contestants at the same time, in the interests of fairness. However, in the interest of giving, we can reveal that the problem:

- Is a real business problem, selected by the CTO.
- Involves using a specific data set of customer-centric data.
- Uses mock data. The dataset we will distribute will include sanitised, artificial data that is representative of 'real' data.
- Requires the entrant to use this data to 'do something interesting'. This will be scoped to a couple of specific business problems, but will allow for a high degree of creativity.

More details will follow when the competition formally opens.

Writing The Code – How to Get Started, What Can You Use

The coding competition will require you to write code: we're expecting entrants to have a technical capability to write code, and in the interests of gaining widest possible participation, we're trying to avoid setting a specific programming environment. An important difference this time is that we're mandating the entry is rendered as a web site We'll be using Chrome as the default web engine to review all web solutions, but we don't care what language / tools you use to build the site.
While this does let you use whatever you want (and hopefully, let you show your individual flair), there are some side effects to this.
Firstly, to be crystal clear, any language is eligible. Java, C#, VB.Net, VB6, Cobol, Python, Perl, Javascript, Excel VBA, VBScript, Flash action script, etc. However, with great choice comes responsibility:

- **Your choice, your problem**. Don't expect any help from the organisers getting you started or helping you with a specific language problem. If you can't work out how to read our data set using a Fortran text reader, that's your problem (and opportunity!). If it turns out that getting VBScript to parse large record sets (if that's the problem) is unachievable, pick a different language.
- **Your choice, your environment**. Given we're opening this for any language, and that we're aiming for the competition to run for four weeks, it's up to the entrants to work out their programming environment. We figure that most people who would enter a programming competition would already have access to a mechanism to write code, be it on their work PC or somewhere else.
- **We're going to need to run your code somehow**. If you like coding in Ruby on Rails with JDK 1.4 with a specific service pack, or you're going to rely on new Silverlight features in .Net 4 that only work on Windows 7, or you're into extended use of emerging HTML5 features, then you need to consider how we're going to run this. Check the submissions section for details, but use a bit of common sense: if it takes use more than 10 minutes to install the pre-requisites to get your code to start or if we need to buy hardware, we just won't see your code running. You'll still be considered, but we're going to be just that little bit less confident in your entry.

Secondly, re-use is good. If you want to leverage some existing bank code, or you want to borrow some open source libraries, or you want to grab something else to help you move ahead, then that's fine. However, a couple of common sense guidelines.

- **Attribute it**. Don't present someone else's work as your own – make sure you declare where you've leveraged code or other 3rd party functions. If you don't, and we find out, you'll be ineligible to proceed

and probably ostracised far and wide.
- **It's got to be legally usable.** Should go without saying, but just so we're clear: you'd better have the right to use the software you want to leverage.
- **It can't be in breach of our policy or represent unreasonable risks.** The bank regularly uses open source code. However, clearly we don't want you download dubious black-box executables that carry a risk of unwanted malware. Common sense should prevail.

Writing to Fit In To The App Store

We want your apps to run in our app store. This means your app will need to exhibit a few key characteristics.

- Must be a web application. It can be written in any language, but the UI must be a web site
- Must have a CBA-formatted manifest
- Must reference a few CBA JavaScript files
- Must make some minimal JavaScript calls to interact with the system, and implement 1-2 calls to receive instruction from the system
- Should follow key style guidelines. Note this is not about aesthetics, it is so your app could react to the responsive design of the site.

All said and done, this is mostly about building your submission as a web site. The App Store team have built a dozen apps out of existing web sites, most of them without accessing any of the original web site code. Adding the App store "wrappers" is a relatively tiny overhead.
We are finalising the developer documentation; drafts will be made available early on.

The Work

No, we can't give you a time code. No, we can't give you leave to work on this. This is a 'spare time' activity – before work, after work, lunch time.
This will be challenging, we know. That's part of the problem we're giving you: those with the ability to judiciously manage their time will have an edge.

Submissions

You will need to submit your code as a package as per the competition submission process. The submission will need to include:

- An abstract; 1-2 paragraphs describing what you are doing.
- Screenshots, if available
- The source code
- The runtime binaries.
- If you have hosted this on a server – internally or externally – we'll just need the URL. Note that we found people who exposed their solutions via this technique last time were much easier to review
- A setup script / document (if required)

Submissions can be sophisticated or impoverished: we'll judge you on a range of criteria, but submitting nothing at all is not encouraged. If it's unfinished, or it doesn't work reliably, or you can't get a setup process to repeat, then we still want to see what you've done. We're proud of all entrants.
A special note on submissions: depending on your code sample, it may be complex or impossible for us to run (for example, if it turns out you've decided to build a web app that requires some open source middleware and the judging panel can't easily download & configure that app server). Don't worry – you're not out. We'll still consider your submission, and *may* arrange for you to give a demonstration.
For **all** contestants, it is vital you do not continue to evolve your code post the competition close. We may well ask any of the participants to demonstrate their solution during the judging phase: if we find a discrepancy between the original code submission and the demo, we'll be cranky and probably count you out of this competition and the next one.

Judging Process

The judging process involves a number of different activities over a period of weeks: we want to give each submission a fair assessment.
Criteria The criteria the team will judge on include:

- Technical prowess. How clever is the assembly, how well structured is it, is logical and soundly assembled
- Code elegance. How well written is the code? Is it clear, concise and robust? Or clumsy, verbose and poorly laid out?
- Viability of idea. How much value will this idea add to the group?
- Technical achievability. How viable is the approach? Will it scale up? Is it deployable?
- Originality. How unique is this idea?

Judges There will be a judging panel of people picked by the CTO. This panel will be responsible for initially reviewing all entrants, and picking the finalists. This includes the prize winners of the preview competition.
The panellists will be joined by the CTO for a formal review of the code, with the respective entrant 'pitching' their code. They'll be asked questions on their technique, their approach, what they want to do with the idea, etc. They will also be asked to demonstrate their code.
The final judging will be overseen by Mok Choe. His decision is final.

Frequently Asked Questions – App Store

What is the App Store?
Simply put, the CBA app store is a technology framework that allows separately developed applications to run 'together' within a single web site. Similar to the Facebook or Apple or Google models, the CBA app store permits independent software development teams to build unique experiences that 'plug in' together and allow a customer or end user to use all of the applications, perhaps unaware they were all authored differently.
More specifically, the app store is a technology architecture that permits a series of applications to be bound together through a combination of strong naming conventions, declarative intent through the use of manifest files, a client-side message bus, open standards, responsive design, an incremental and automated approach to deployment, strong versioning, and on-demand activation.
Try to be more specific – give me an example
Think of the next versions of NetBank or CommBiz or CommSee. These platforms start to exist as web sites that one logs on to. As a user, you can see a set of 'apps' within these platforms – Transfer Money, View Customer Relationship, Open Account, Pay Bill. Other apps are also available – Kaching for Netbank, or a staff phone directory for CommSee users, or perhaps a foreign exchange app that works across all of these.
These apps differ from the normal functions of the web site because they:

- Can be dropped in and dropped out without changing the entire platform. The interdependencies are greatly reduced, so I can swap in and out different apps
- Can permit greater user customisation; each user can have a different suite of apps based on their needs.
- Can support multiple versions of everything; we can have 4 different variants of transaction history
- Are transportable; apps in NetBank can be surfaced in CommSee and used by staff members, saving us money on wasteful re-writes
- Are transformable; instead of building everything for each and every form factor, the App Store utilises responsive design and reshapes apps based on the size and function of each devicve

How does it work technically?
At the simplest level, apps are written as HTML5 web pages and plug into a 'container' web site via an iFrame. The apps are bound to the web site by adhering to a few rules:

- They create a manifest file in the root of their location which follows a specific format. This allows the app store to discover and understand application intents.
- They link to a couple of common JavaScript and CSS files. These allow the app to communicate to the

container, and vice versa
- They use the JS APIs to perform various functions
    - Read the security token to understand user context
    - Publish an interest in specific context types – Customer, Account, etc.
    - React to events pushed by the container
- They follow the style guides – implemented through a combination of process and JS rules. This permits the app to be resized, used on devices with different characteristics or change brand.

A concrete example might be a transaction history app. In addition to above, this app would be implemented with the following aspects:

- It would call a service to get transaction data. It might page through these per account, or show a combined view of multiple accounts – it doesn't matter.
- It would render the transactions in a graphical way – perhaps tabular, perhaps in a summary view
- It might have a form that allows people to search for transactions

Once we look at this pure function, the app could realistically be used in a range of circumstances – for small business customers, for retail, on iPads or iPhones or PCs, or in a branch on an iPad or in a call centre on a PC. To support this, the app is then referenced by the various web site containers that are used by different user groups, and manages all of the brand, form factor, navigation and other issues.

OK, so how does a developer write code for the app store?

For the coding competition, we'll produce a developer-only container. This is a desktop website structure that allows one to simulate how your app will interact in an app store: you can test different sizes, different brands, etc. Your app will run in an iframe, and rudimentary javascript stubs will be your interface. These stubs will contain sample data but mirror the kind of data you might expect in the production environment – for example, a sample security token reader for the web authentication.

In addition, we'll create some data-specific JSON stubs that will contain sample data you can use programmatically.

We'll also have a Manifest-Generator. This will allow you to create a manifest that accurately describes your code. The kind of information you'll need to supply will be the name of the app, your name, an icon for your app, a description – that kind of meta data

How do I submit my code to the app store?

Long term, we'll be building a range of developer / SDLC tools that will permit app management. However, given the scope of the coding competition (and the need to keep entries private from one another), we'll be expecting developers to submit as per the competition submission info.

Frequently Asked Questions - General

What's the problem?

All will be revealed on August 20

Go on – give us a hint. I'd like to know so I can my development house in order, hit the ground running in August

The challenge is designed to be a problem where you take a data set that we give you, and do something with that data that meets our high level request. The exercise will largely focus on read-only access to the data set, but your idea might involve writes. The data set will be small enough to be used as a simple text file, but interesting enough to give you a range of different scenarios. The data will be anonymised, 'cleansed' data which carries no risk of sharing.

What if I already have an idea that is markedly different?

Feel free to develop your idea – innovation is key – but it won't be considered as part of this competition. You might want to propose your idea for the next competition.

Can I use truly any language / technique / tool to build this?

Yes. We have no particular preference around your target programming language. However, we are narrowing the entries to be exhibited as web pages that can be run in standard web browser (Chrome). This lets you use

HTML4/5, Javascript, CSS, Flash, and a range of other browser technologies. Serverside you can use any language you want: PHP, C++, C#, ASP.net, Java servlets, Ruby on Rails, etc.
You will need to have very rudimentary JavaScript skills to complete very light integration to the app store. We'll provide sample code and documentation for this – we're not expecting this to be onerous at all.

So I can use my work computer to do this?

You are free to develop in any way you see fit. If that works well for you, then that's a good approach.

If I currently use a specific technology in my job, do I have to use that?

You are free to develop in any way you see fit. It might be advantageous for you to use your existing environment, but you can do whatever makes sense

I want to develop in technology X, but I don't have it on my machine – it's blocked by the proxy/security rights / there's licencing costs / there's a conflict with existing software on my machine / doesn't run on Windows XP. Can the organisers arrange for those impediments to be removed?

You are free to develop in any way you see fit, but we are not setting up specialised development environments for participants. There are a couple of reasons for this.
Firstly, we'd like to allow all participants to choose their language of choice. We're expecting a wide range of preferences.
Secondly, being a large financial services organisation, we have a series of controls in place that are designed to produce standardised managed desktops. These controls exist for sound business reasons, and circumventing or relaxing these controls to support a wide range of programming languages is not our goal. We also have the explicit goal of not interfering with normal day-to-day operations: it would be unacceptable if a participant was unable to complete their normal day-to-day operation because of an environmental conflict.

I want to develop in technology X, but my computer isn't powerful enough to run the supporting development environment. Can I get a processor upgrade / memory upgrade?

No.

Can I develop on my home machine?

You are free to develop in any way you see fit. If that works well for you, then that's a good approach. Of course, if you are running on your own equipment, you need to be mindful of the normal policy around non-bank-issued equipment. However, we can think of many scenarios where such an approach will work well.

I had a great idea but I ran out of time. Can I submit something that's unfinished?

Sure – but please mark it as such.

What environment will run you my code in? Will it be on the bank network? Will it have Chrome installed? Can I depend on a particular runtime?

Give us your target run time recommendations when you give us your code. Our base environment will be a normal bank SOE 4.0 or 5.0 machine, plus we've got access to a normal 'developer build' machine used by ES Development. Windows XP is the current OS, which includes a Java run time (Java 6 update 22) and .Net (1.1, 2.0, 3.0, 3.5, 4.0). We'll have Chrome, and Flash. We'll be on the normal bank network – normal bank network access rules apply. We'll have internet access.
What won't we have:

- Machines with high end video cards
- Machines with 64bit OS.
- Machines running Win7, Linux variants or MacOsx
- Spare servers to run up a custom server image

Why isn't this coding competition giving me specific support for technology X? Surely with the resources at the bank's disposal we can set up any amount of virtual images that should allow me to have a tailored environment?

Our last competition provided no specific server images, and we received around 39 entries. Of these, we saw over 15 different languages and even more server side variants. Creating this kind of permutation of environments in advance with the right level of configuration and access simply isn't practical.

```
using System.Threading;
public class CBA.Competition
{

public Competition(string[] CompetitionEntrants)
{
for (int i=0;i<CompetitionEntrants.Length;i++)
{
MakeIt do = new MakeIt (CompetitionEntrants[i]);
ThreadStart ts = new ThreadStart(do.MakeIncredibleStuffHappen);
Thread t = new Thread(ts);
t.Start();
}
}


public class MakeIt
{
string _entrantId;
public Do(string entrantId)
{
_entrantId = entrantId;
}

void MakeIncredibleStuffHappen()
{
Register(_entrantId);
GetCompetitionRules();
Idea i = Idea.Generate();
Code c = i.Implement();
}
}
}
```

Future Bank

Overview

Future Bank is a demonstration of how Apps and a Container work together to create a unified experience. In order to focus on the architecture rather than any particular innovation in user experience or business process, it has largely replicated the functions offered in NetBank Tablet as individual Apps:

1. Whats New - combined transaction history of all accounts + scheduled payments for all accounts
2. All accounts - account balances and total position
3. Transfer money - transfers and payments to address book entries

4. Transaction History - using the winning entry from last coding competition (Natural Language Search by Ed Gallimore)
5. Feedback - new
6. Help - content from Adobe CQ5 content management system as an App

The goal is for your App to appear in this list too.

Seeing it in action

> The current version of Future Bank only runs in Chrome (compatibility with other browsers is in future releases). Most users in the Bank can install Chrome without being an administrator.

Using Chrome: to see Future Bank running go to http://appstore.dev.cba/Containers/cba.developer.v1/Default.aspx?env=http://appstore.dev.cba/EnvironmentDefinitions//Womble55.BADEV.xml

You don't need to type any username or password to login.

What to look out for

We want to show you 3 concepts with this site:

1. The target experience that your App will be running in after it's been submitted. Your App could be a new menu item or replace one of the existing Apps.
2. Seeing the The Developer Container in action
3. Demonstrating theme injection and responsive design

Screenshots

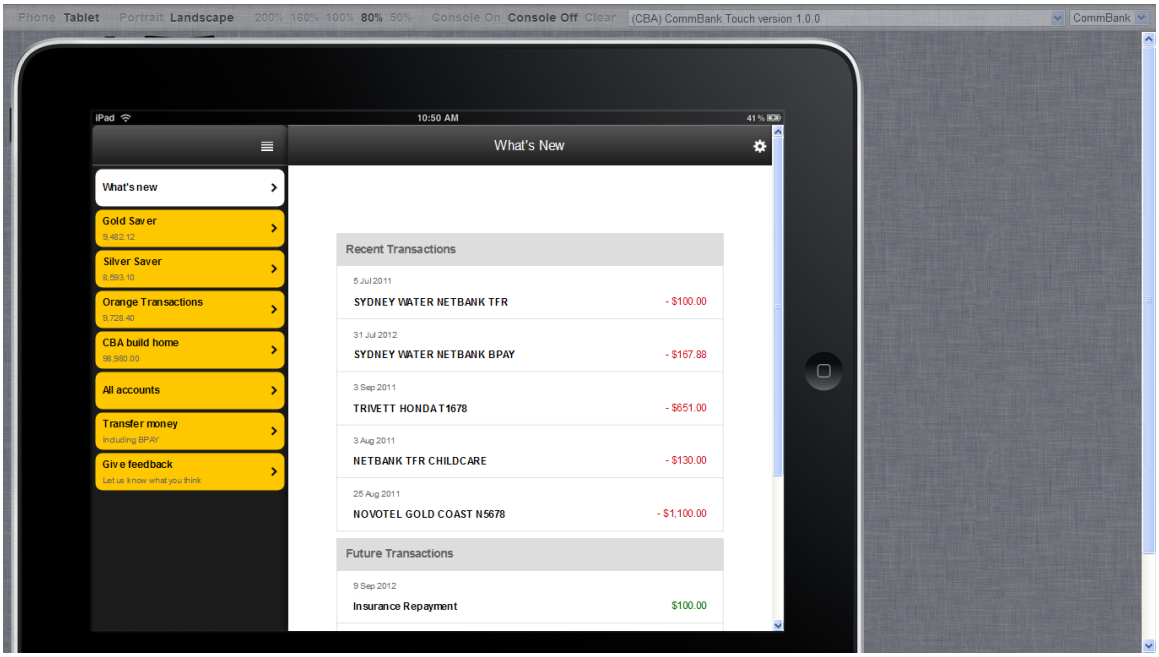
If you are not on the corporate network, here are some screenshots to illustrate the concepts. (Click on each one to see a larger version).

General

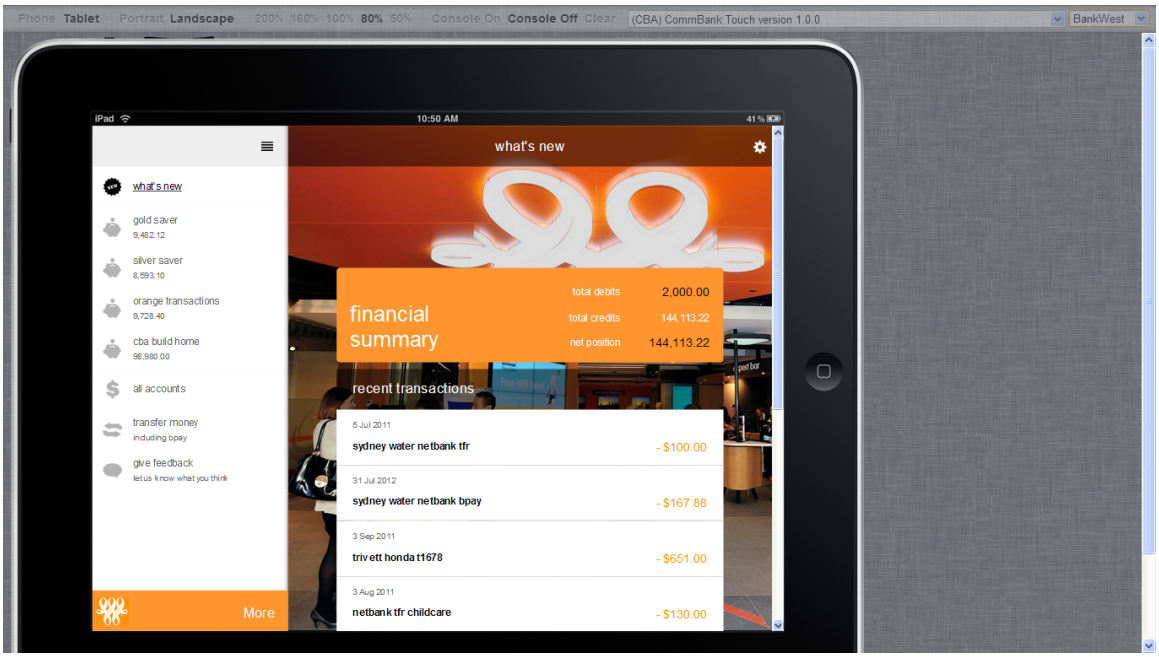
The menu item labelled 'More' on the bottom left is where your App will be discovered from

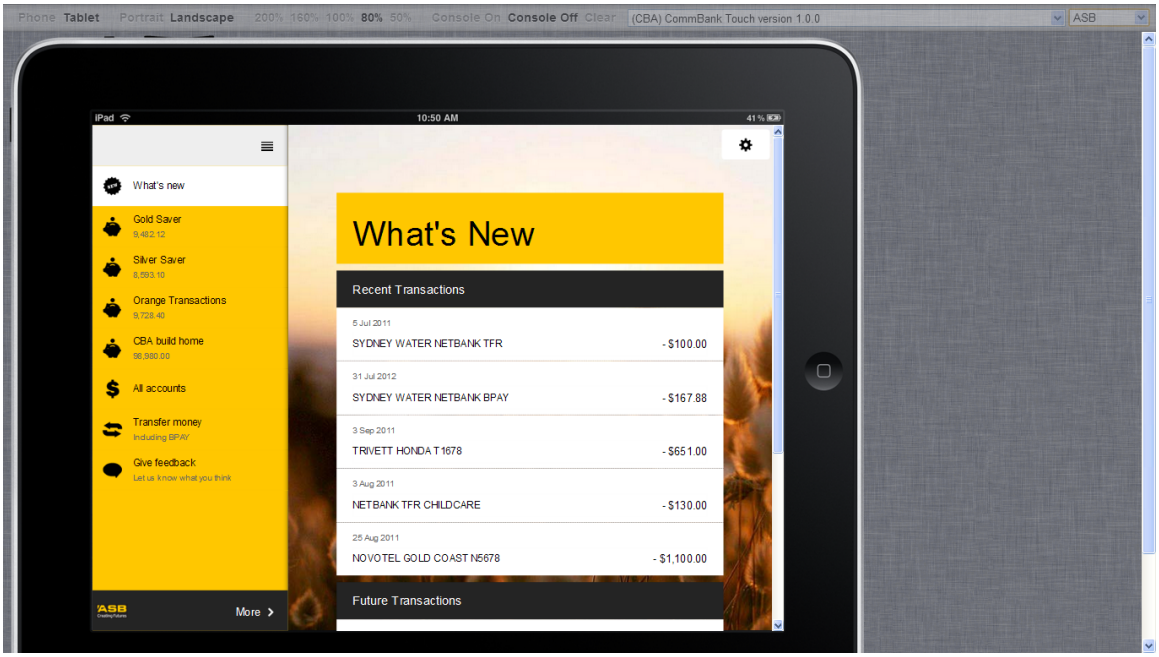
Theme injection

The same UI is rendered using the existing NetBank Tablet style, then by changing the Brand dropdown in the top left corner of the Developer container, it use Bankwest or ASB Bank's themes



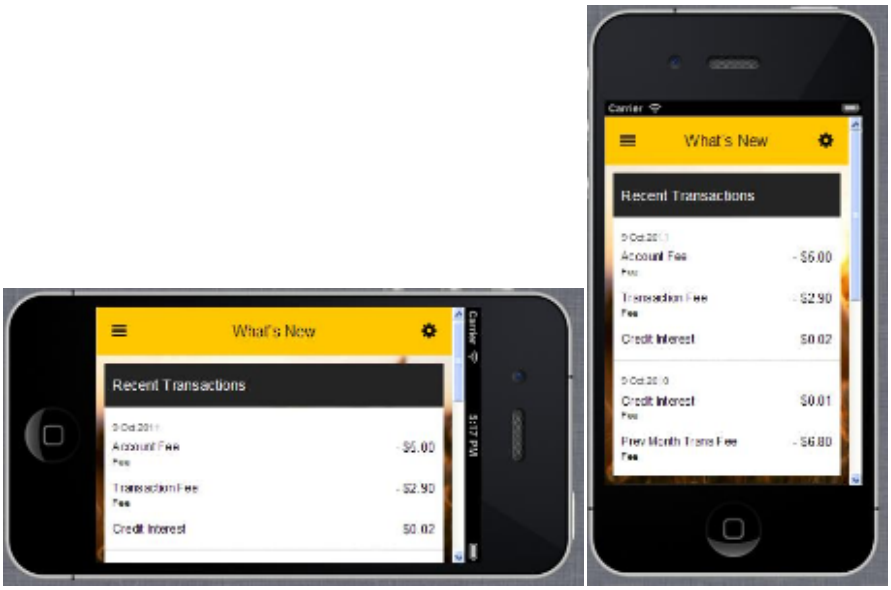
The Bankwest theme shows the Financial Summary information. The other two themes do not. This illustrates how theme injection can be used for more than just colour changes.



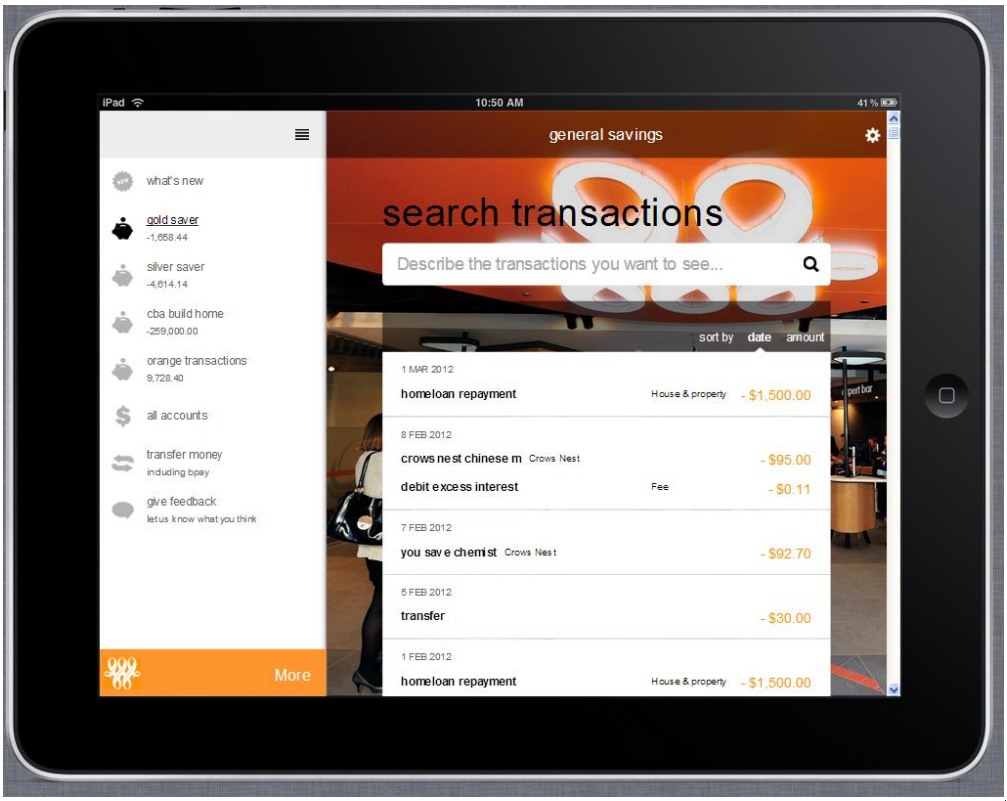


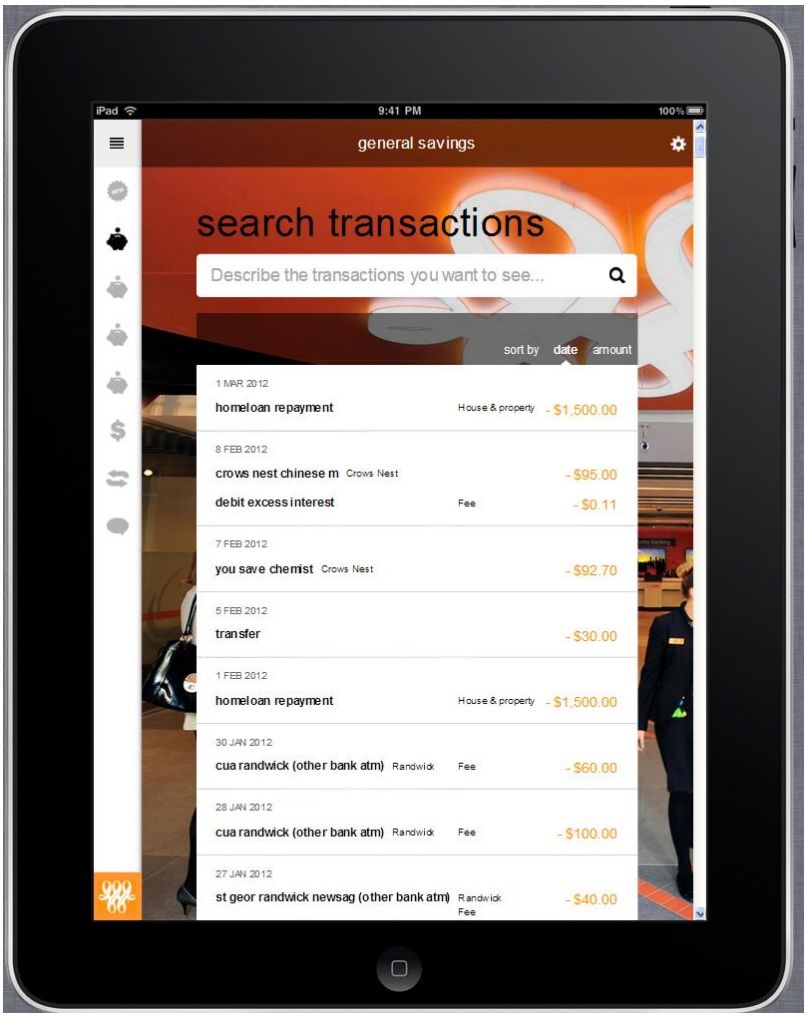
Responsive design

Using Responsive Design principles you can see the design scaling down to a phone and different content layouts for portrait vs landscape



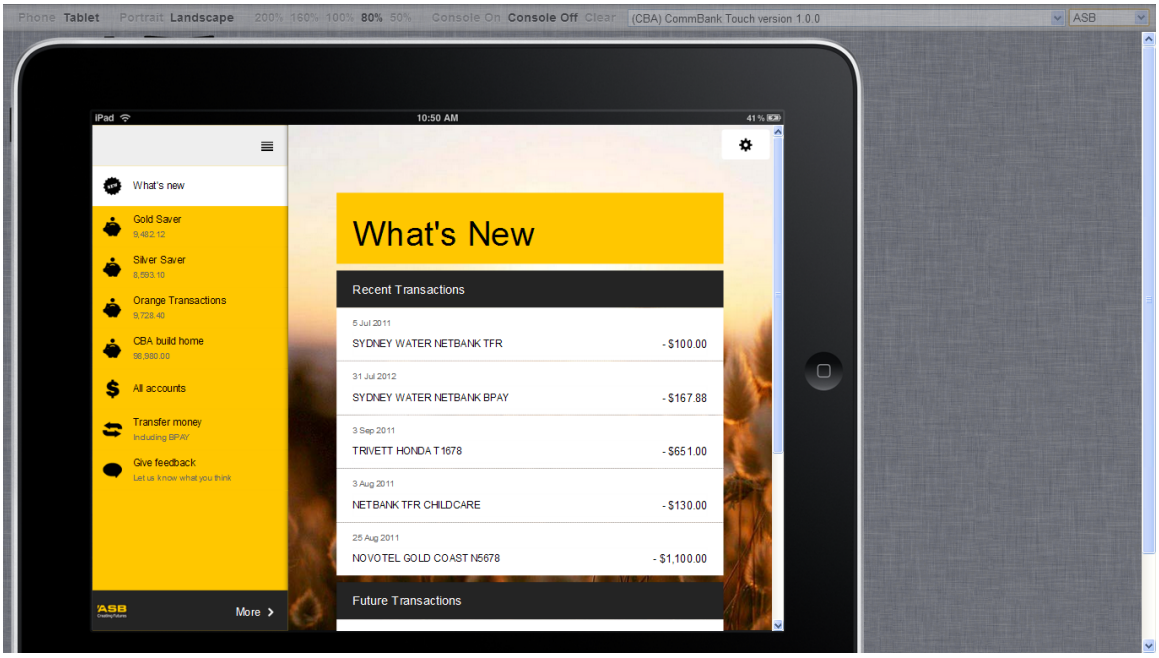
Here, Transaction History on tablet shows how the menu shrinks in portrait view:





The Developer Container

Shows the device emulator and the controls on the top menu for setting themes etc



TopCoder-August2012.zip -file contents

Download

Download

Name	Date
[illegible]ugust2012.v.1.6.zip	
[illegible]on.v.1.6.zip	
[illegible]v.1.0.zip	
[illegible]ntainer.v.1.6.zip	
[illegible]ugust2012.v.1.5.zip	

Change history

Version	Date	Release Notes
1.7	29/8/2012	Changes • Fix for paths when using custom themes • Fix for problem with theming after reloading Navigation Example app • Fix for 404 error showing port 0 (http://localhost:0/...) on Safari and older versions of Chrome • Less trace messages to browser console.log from container (you can still see them on the Container Trace tab) • Fix for initialisation failure in Firefox • Added AppStore.parseURL() helper function

1.6	24/8/2012	Changes • New Navigation Example - Sample App • Changes to resolve problems running in case-sensitive environments like Linux • Some internal changes to error handling and messaging • Moved AppStore.Theme.js into AppStore.Common.js (already included in Framework.js) • Improvements to some theme elements for the reference Apps on Future Bank

Update instructions

1. Back up the /Environment.xml file from your current installation
2. Unzip the .zip file in a new folder
3. Copy the contents of /DeveloperContainer over the existing /DeveloperContainer folder
4. Edit the Environment.xml file to add your app back in
5. Copy the new version of Framework.js from /Apps/HelloWorld to your app folder, if your app is using it.

Explanation of contents

The zip file contains:

1. This documentation site as a single PDF (including competition brief, submission process, help on writing Apps and Helix concepts)
2. The Developer Container To develop this at home or on your own machine, you'll need a way to simulate the experience. To that end, we've created a standalone portal that you can run on your own machine. This will allow you to put your app into a skeleton framework, and test how it works on different devices, resolutions, brands and themes.
3. Hello World - Sample App - shows the minimum required to create an App
4. Forms Example - Sample App - shows additional theming to make data entry forms look consistent.
5. Navigation Example - Sample App - shows the additional steps to handle page transitions within the container.
6. Context Example - Sample App - shows how an app can listen and respond to context changes.
7. Context Editor - this is a utility app used by the developer container to allow injecting arbitrary context.
8. Coding Comp Sample Dataset - The explains the breadth of data that is in the sample data set. Don't worry about how comprehensive it is! This is only to help you navigate the data if you want to use it.

```
/TopCoder-August2012
 /DeveloperContainer
  /Apps
   /HelloWorld
   /HelloWorldForm
   /ContextExample
   /NavigationExample
   /ContextEditor
 /SampleData
 /Documentation.pdf
```

The competition submission process

The competition submission process is now live.

What to submit

You need to provide **a single zip file** containing:

1. your app or entry
    a. source code
    b. manifest file
2. screenshots, if available
3. readme.txt containing:
    - an abstract (1-2 paragraphs describing the app)
    - setup script instructions
    - the URL (if hosted internally or externally)

Process

If you have access to the registration site

1. Go to http://appstore.dev.cba/Containers/CBA.MissionControl.v1/Default.aspx
2. Search for your team entry.
3. The submission info is now shown on the team info page. Click '*upload submission*'

If you are external with no network access to the registration site

Send your submission to Ricardo Correia via ESCommunication@cba.com.au

If all else fails

Email gateways can sometimes be finnicky – if your submission is too large for email, or you get a bounce back for other reasons, you'll need to fall back to the physical world. Please deliver your code via a USB key to one of these individuals:

- CBA: Ricardo Correia
- BWA: Peter Boni
- ASB: Geoff Fitzgerald

App Development
Setting up your machine

You can use any computer to code, so these instructions are vague enough to be appropriate for most types of operating system.

Overview

Your App can be written any web code you like (client side only or client + server), however to make sure it will work as an App inside a container and respond to theme and context being passed into it you will need to test it in The Developer Container.

We've provided the code for the The Developer Container as part of the zip file for the competition. So download it onto your computer first.

While The Developer Container we've provided for the Coding Comp has no server-side code (i.e. it is pure HTML, CSS and Javascript), it **won't work** when run as a file based URL (i.e. file:///D:/TopCoder-August2012/DeveloperContainer/Default.htm) due to the way browser security treats Javascript files. Rather, it needs to be hosted under a http location (i.e. http://localhost/TopCoder-August2012/Default.htm).

Because of this, this needs to be deployed onto a web server – any web server. It will work on Apache, Tomcat, Websphere, IIS. Basic instructions on installing a webserver can be found on various websites.

Installing a webserver

- **IIS** - help on installing IIS on Windows 7 or Vista

Once IIS is installed, the easiest way to make a new virtual directory in IIS is from Windows Explorer:

1. Right click on the folder eg *D:\TopCoder-August2012\DeveloperContainer*
2. Select "*Properties*'
3. Select '*Web Sharing*'
4. Click '*Share this folder*'

- Or you could use **Apache** - we have some help for Installing Apache for Windows

Launching the site

Once your webserver is setup, you can run the Developer Container from your favourite modern browser. We recommend Chrome, Safari or Firefox.

You should now be able to point to the folder with a URL like http://localhost/TopCoder-August2012/Default.htm

Now you can move on to understanding the components of the Helix framework, the Hello World - Sample App a nd Developer Container.

Installing Apache for Windows

1) Follow the instructions to download and install Apache from http://httpd.apache.org/docs/2.2/platform/windows.html

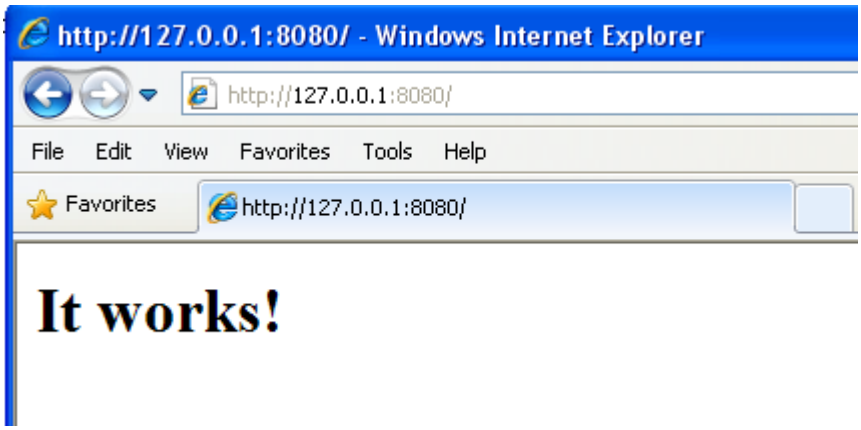
Select an MSI installer e.g Win32 Binary without crypto (no mod_ssl) and run the setup.
For this example it was selected to install for the current user listening on port 8080 and manual start.

2) Follow the instructions for starting Apache
e.g. to run as Console, open a command prompt :

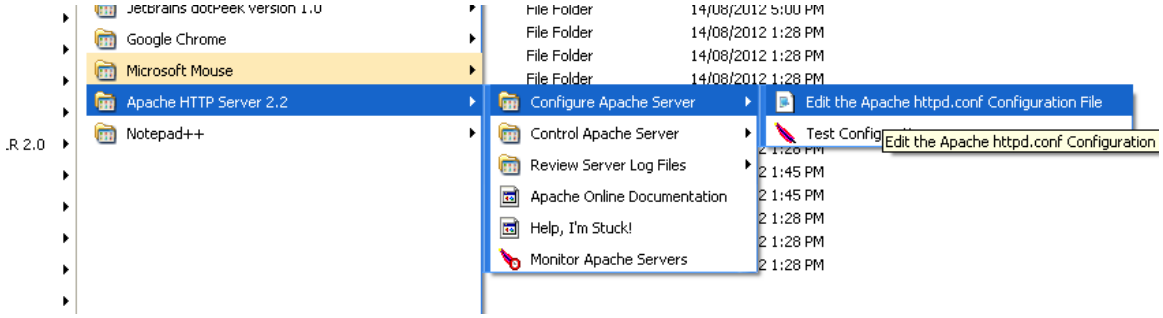
```
c:
cd "\Program Files\Apache Software Foundation\Apache2.2\bin"
httpd.exe
```

3) Test that Apache is working by opening a browser and navigating to the loopback address. Add the port if it is listening on a different port e.g.http://127.0.0.1:8080

A successfull installation will be indicated by:



4) Edit the .conf file to point to the website that you want to host



5) Change the "DocumentRoot" to point to the physical path where the website is hosted

e.g (note use / instead of \ in the path)

```
DocumentRoot "d:/AppStore/src/Stable/Containers/DeveloperHTML"
```

6) Change the "<Directory.." element attribute to the same path as specified above

e.g

```
<Directory "d:/AppStore/src/Stable/Containers/DeveloperHTML">
```

7) Change the default start document

e.g

```
<IfModule dir_module>
DirectoryIndex default.html
</IfModule>
```

8) Use the *Test Configuration* tool to verify the changes

9) Verify that your web site is being served by navigating to the URL as mentioned in Step 3.

Containers & Apps

The two major components of the Helix architecture are Containers and Apps. Before understanding the rest of the Framework component overview, its worth understanding these two in a little more detail.

Container

Our definition of a Container is that it is the icon that the user selects on a phone, or the website URL that they type in to their browser. It is the boundary of the user experience. This can be a little confusing for people thinking about iOS or Android Apps because an iOS or Android App on the phone's home screen = A Container in Helix.

What is a container	What a container is not

• It's the user identifiable experience for a channel and device
• It provides the security boundary around the experience
• It assembles multiple Apps into a cohesive experience
• It controls when Apps are loaded and unloaded
• Is responsible for:
  ◦ the menu and navigation
  ◦ page furniture (eg headers and footers)
  ◦ user context
  ◦ message passing between Apps
  ◦ user authentication
  ◦ coarse grained entitlements
  ◦ piloting of Apps
• Exposes APIs to enable Apps to access device features.
	• It doesn't perform business functions.

App

For the purpose of Coding Comp, one entry would ideally be one App.

Conceptually, an App broadly represents a business function. It could only cover part of a business domain as long as it accomplishes a task that a user would want to achieve in one interaction. It can also depend on other Apps to help complete a larger task. The exact dimensions of an App change both over time and for the function they cover. It could also be considered to be a Unit Of Commissioning (like a project)

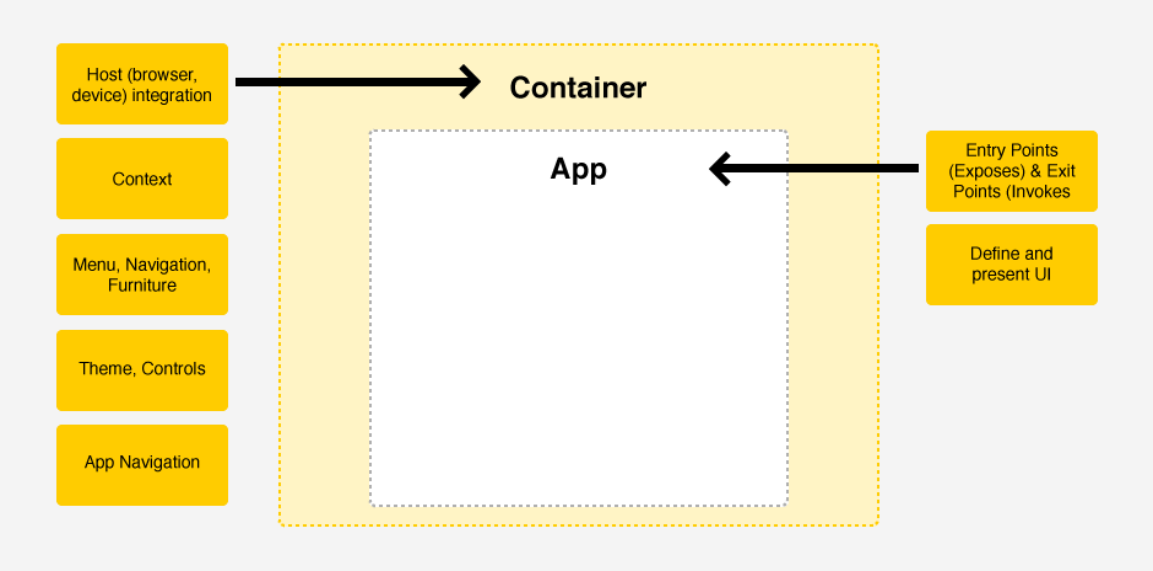
What an App is	What an App is not

• It provides the user interfaces for a user to perform a business function • It may consist of 1 UI view or multiple Views depending on how complex the business function is. • It may have UI views that only a subset of users can see, or it might be tailored to a subset of users with other Apps targeting other users too.	• A single reusable code component. It needs to be big enough to perform a business function. ie to act as a menu item that the user understands.

Apps could be implemented as 1 page or multiple. There are some differences in capabilities for Classes of Apps that use multiple pages.

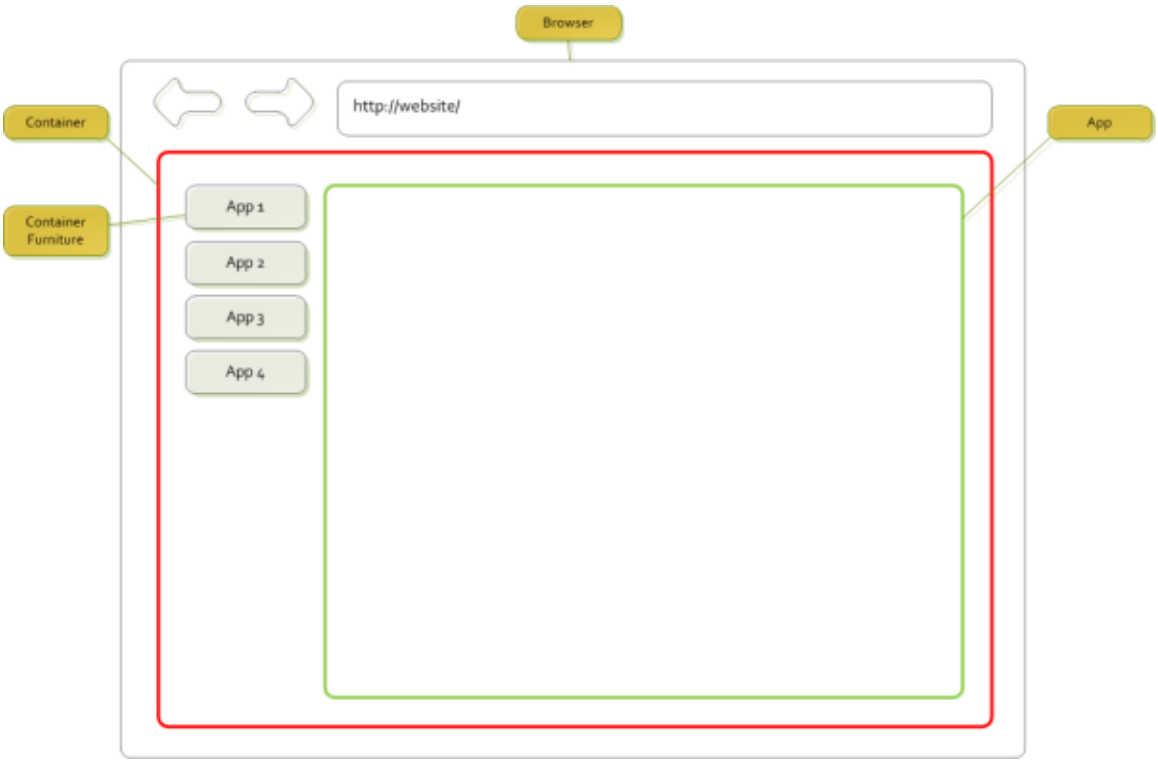
Responsibilities & Relationships

A Container and App have different responsibilities, which determine their relationships.
In some respects a Container is an App with extra responsibilities.
A Container is responsible for hosting Apps.
An App runs inside a Container.

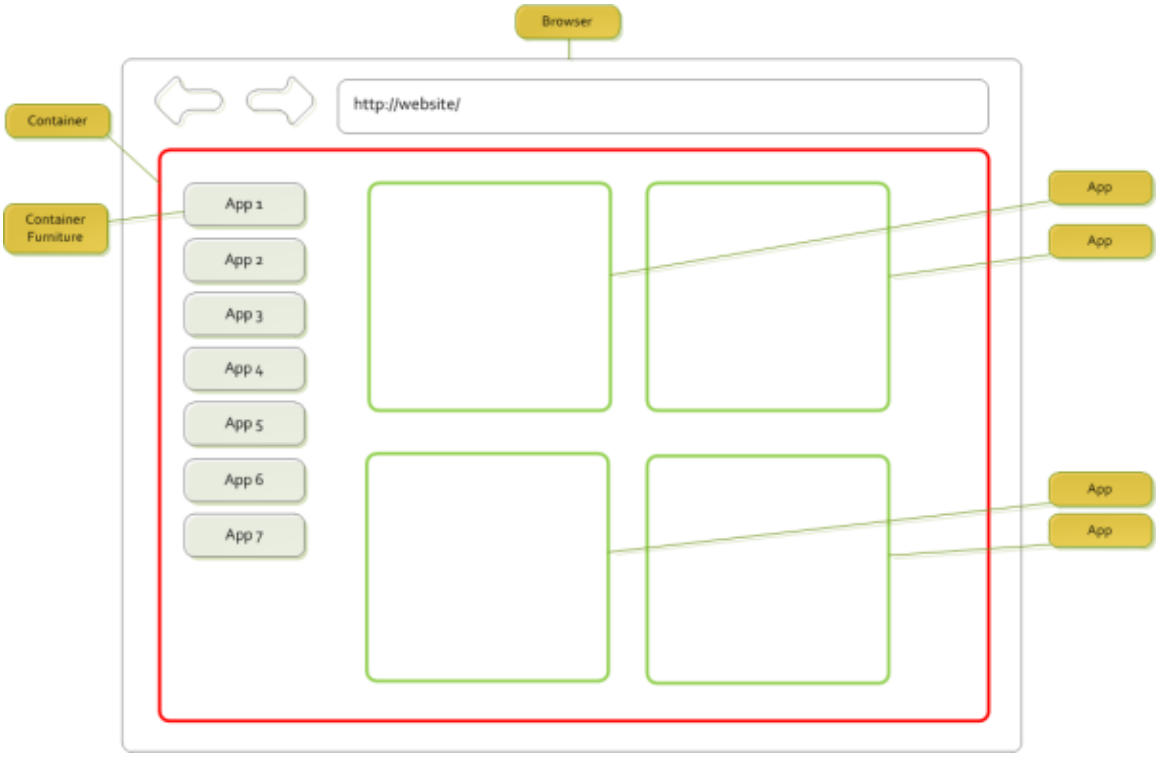


Visual

The relationship between Containers and Apps is best represented visually as below.
The Container runs inside the Browser.
The Container has furniture, which in this case is navigation between different Apps.
The App is hosted in a particular region inside the Container.



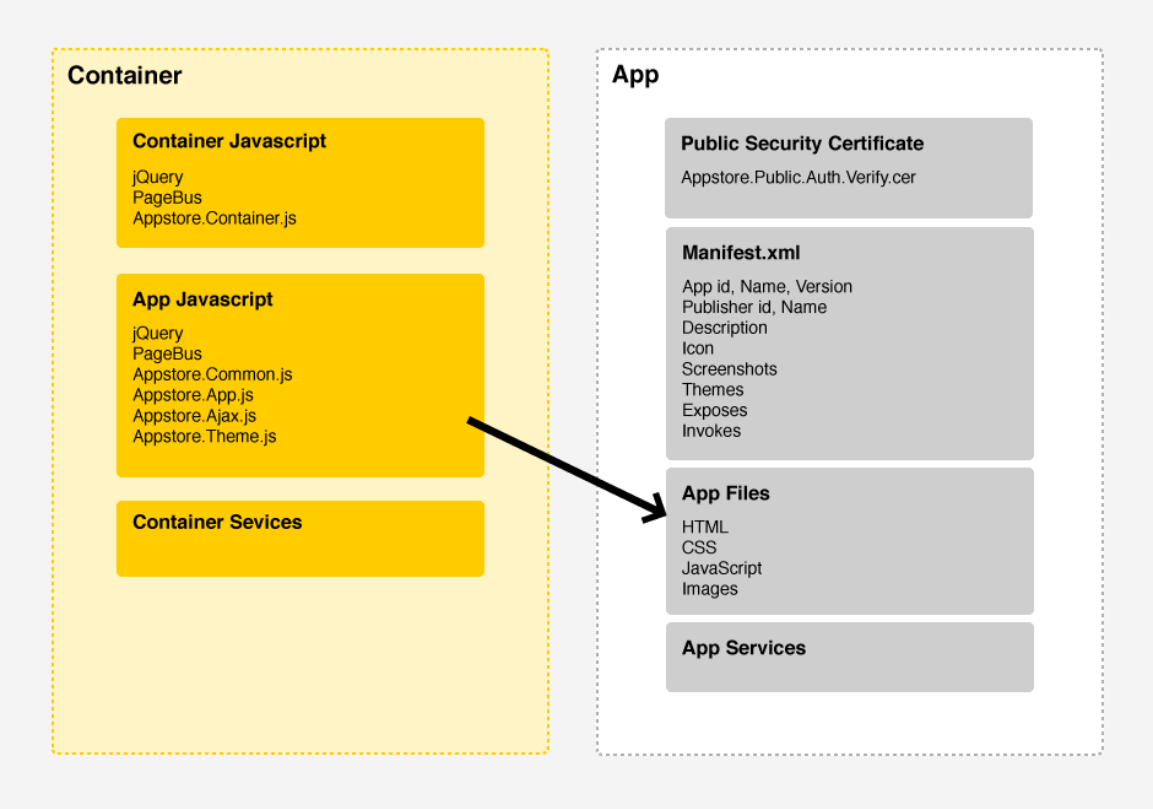
Containers can host more than one App.



Elements

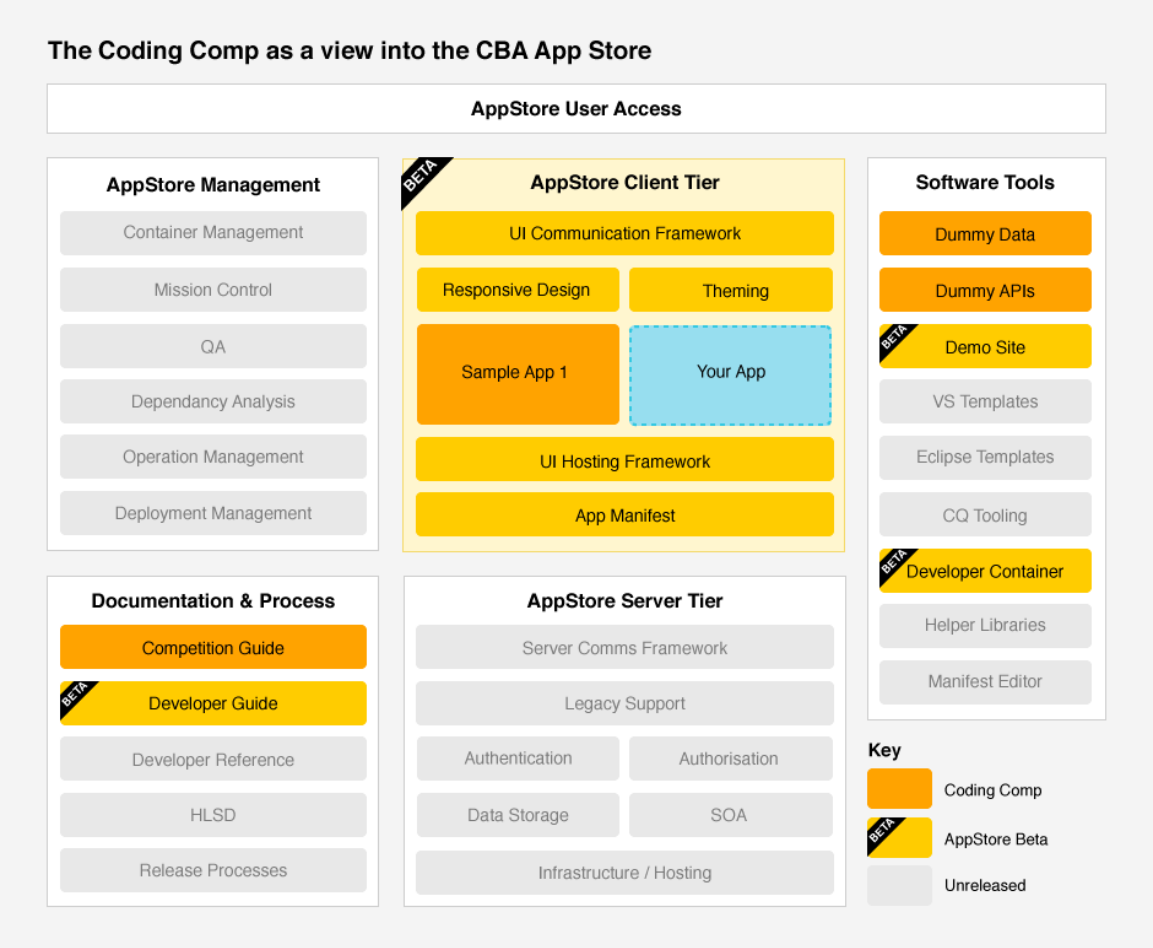
A Container and App have the following elements.
A Container has "App JavaScript" that Apps that run inside it use.

App JavaScript is standard frameworks / libraries, as well as App Store framework JavaScript that all Apps should use to gain maximum benefits / consistency, e.g. themes, controls etc.



Framework component overview

This diagram shows the major components of the Helix architecture. For coding comp, we are only providing a subset of these + some additional ones that are coding comp specific.

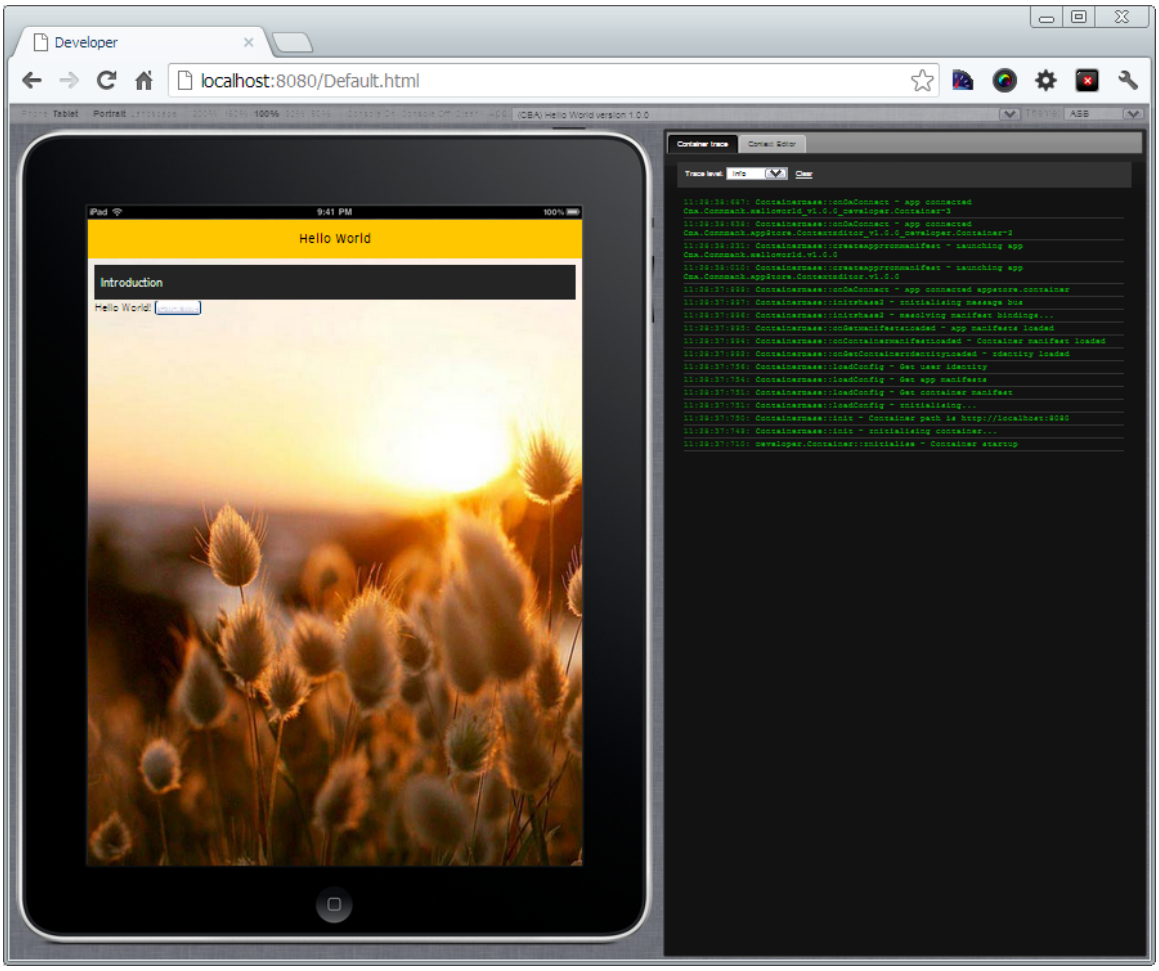


The Developer Container

Components

The Developer Container is really 3 things:

1. A way of loading your App inside a Container sandbox that runs on your machine
2. A device emulator to test how your App handles [Scalable UI's using Responsive design](#)
3. Diagnostic information on how your App is running



Why it exists

The App Store architecture exists to enable software developers to build apps that are logically isolated from one another, but still inherit common cross-cutting concerns. Apps can be functionally quite varied, but should still have some commonality. This has the following implications for a programmer

- Apps cannot directly link, reference or reuse any components that belong to another app. They must be isolated from each other to enable their independent development, testing and deployment. Sharing is strictly limited to only components that live in a SOA layer.
- Apps must use specific APIs to communicate to the host container
- Apps must implement specific APIs to allow the host container to communicate with them
- Apps should inherit the form factor constraints

To allow teams to do this and run on their own volition, a developer sandbox that simulates most of this is made available to any app developers. The developer sandbox is a tool that can be run on any web server and host any web content that is to be packaged as an app. The developer sandbox has these features:

- Simulates multiple device form factors: orientations, screen size, dimensions, etc.
- Simulates multiple themes and branding
- Simulates Container APIs
- Allows a developer to manually set data context that would normally be triggered by external events.

High level architecture

The Developer Container is effectively a web page that holds a couple of iFrames, with a number of included Javascript libraries, images and CSS files. The Javascript libraries ‘drive’ most of the site, allowing for:

- Screen manipulation to simulate device type, size, etc.
- API simulation for event context setting

- Presentation elements to drive theming

The container allows a developer to write a web site, and host that web site content within the boundary of a single user experience.

User Guide for driving it

1. To start the developer container, load the URL that was setup in Setting up your machine.
   This gives you the empty container experience which shows a device emulator on the left, instrumentation on the right and a menu on the top.
2. Now that your App is displayed, you can test how it responds to orientation and size changes by toggling the Phone/Tablet & Portrait/Landscape links.
3. In order to check for layout discrepancies, you can change the zoom level or toggle the instrumentation console on or off.
4. The dropdowns let you select which App you want to load and which Theme you want to display it with.
5. Console - to help see what the container is doing when on your home computer, the *Container Trace* writ es out info as it loads apps and fires events.
6. Context Editor App allows you to set and manipulate context such as Account Number to drive an App

Common issues & errors

- Check out the FAQs, common issues & errors page

Code samples

Sample Apps illustrate the core concepts.

- The simplest is Hello World - Sample App
- Forms Example - Sample App
- Context Example - Sample App

There are also visual examples of Style & Theme Implementation Samples to help work out how to visually lay out content in an App.

Hello World - Sample App

To make it easier to understand what your App needs to implement, we've supplied the files required for a Hello World App as part of the TopCoder zip file.

This Sample App has all server side components commented out so you can run it at home without any external dependencies. Real Helix Apps have server side components to take care of Security and consuming and creating data

Hello World implements the Responsive Design principles and Theme injection capabilities. Other samples demonstrate Creating Forms and Using Context.

The contents of Hello World are:

1. hello_world.js - App specific javascript functions

hello_world.js

```
function clickMe() {
    alert('Hello');
}
```

2. Default.htm - the web presentation logic

Default.htm

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Hello World</title>
</head>
<body>
    <div class="background"></div>
    <div class="app">
  <div class="header">
   <h1>Hello World</h1>
        </div>

  <div class="main">
   <div class="section">
    <div class="header">
     <h5>Introduction</h5>
    </div>
    <div class="content">
                  Hello World! <button onclick='clickMe()'>Click Me</button>
    </div>
   </div>
  </div>
    </div>

    <script src="pagebus.js" type="text/javascript"></script>
    <script src="framework.js" type="text/javascript"></script>
    <script src="hello_world_example.js" type="text/javascript"></script>
</body>
</html>
```

3. Manifest.xml - the description of what your App interacts with in the universe.
4. framework.js - a framework script that takes care of all the plumbing between Apps and Containers. You shouldn't need to edit or review this file.

Forms Example - Sample App

This code sample demonstrates how to build forms and make use of theme injection to style the form and handle responsive design. The most important aspect of this sample is the HTML mark-up structure so we will take a close look at how the forms is put together.

The Hello World Form code sample contains:

1. Default.htm - Mark-up required to render the form
2. Manifest.xml - The description of what your App interacts with in the universe.
3. framework.js - A framework script that takes care of all the plumbing between Apps and Containers. You shouldn't need to edit or review this file.

> ⚠ If a form navigates away to a different page, the theme is lost. This is fixed in v1.6 of the code sample.

General layout

To build a form which makes the best use of theming, you need to have some boiler plate mark-up set up.The following code snippet shows the minimum set of elements you should put on the page to get the initial layout working. Detailed diagrams of how the mark-up should be laid out can be found here (todo).

Boiler Plate Mark-Up

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Hello World Form</title>
</head>
<body>
    <div class="background"></div>
    <div class="app">
  <div class="header">
   <h1>Hello World Form</h1>
        </div>

  <div class="main">
   <div class="section">

   </div>
  </div>
 </div>
</body>
</html>
```

Declaring a form

The next set of mark-up will add a form to the page and should be placed within the declared as a **"section"**.

Form mark-up

```
<form class="form" action="Default.htm" method="post">
 ...
</form>
```

A form consists of two main structural sections:

- A header - title area for a form
- A content area - an area to hold form fields and controls

You can nest as many header-content pairs as you like within a form or just use content when grouping form elements.

Within the content area you should place <fieldset> and <legend> tags as required to build a good semantic HTML form.

Form mark-up

```
<form class="form" action="Default.htm" method="post">
 <div class="header">
  <h5>Details</h5>
 </div>
 <div class="content">
  <fieldset>
   <legend></legend>
  </fieldset>
  <fieldset>
   <legend></legend>
  </fieldset>
 </div>
</form>
```

Populating a form

We are now ready to fill in the form with controls and action buttons.

Input fields

Every control that you add to a form should be wrapped in a div marked with the class **"input"**. This will handle grouping a set of related elements which can include: labels, inputs, buttons, error messages, help text and descriptions.

Basic input

The most basic example is as follows:

A simple input field

```
<div class="input">
 <label for="firstName">First Name</label>
 <input type="text" name="firstName"/>
</div>
```

Input with help/description messages

You can decorate input fields with additional pieces of information including descriptions and help links to better describe what your input fields do. The mark-up below shows two new elements. The description element will appear below input fields in small font. The help element should act as an anchor to use as either a pop up window with help text or some form of tooltip on mouseover.

Adding descriptions/help

```
<div class="input">
 <label>First Name</label>
 <div class="help"><a href="javascript:void(0)">What is this?</a></div>
 <input type="text" name="firstName"/>
    <span class="description">(A unique code used when you submit your entry to the
appstore)</span>
</div>
```

Disabled input

To declare an input field as disabled add the class "disabled" to the input block.

A disabled input field

```
<div class="input disabled">
 <label for="firstName">First Name</label>
 <input type="text" name="firstName"/>
</div>
```

Postpended input

If an input field requires some form of additional element such as an icon, button or link it is possible to declare an input field as "postpended". This will add the item on the same line as the input field. This is useful for things such as search fields with inline buttons/icons.

A postpended input field

```
<div class="input">
 <label>Postpended input field</label>
 <div class="postpended">
        <input id="Text1" value="" placeholder="+ add another"/>
     <div><a id="a1" href="javascript:void(0)">+ add</a></div>
    </div>
</div>
```

Two column input

Some input fields are best laid out in pairs side by side occupying equal parts of the width of the form. This is often seen on login screens. To achieve this layout declare the input fields as children of the "two-column" class. There is no change in how you declare the input fields.

Split column fields

```
<div class="two-column">
 <div class="input">
     <label for="field1">Field 1</label>
  <input type="text" name="field1"/>
    </div>
    <div class="input disabled">
     <label for="field2">Field 2</label>
  <input readonly="readonly" type="text" name="field2"/>
    </div>
</div>
```

Radiobuttons and checkboxes

Declare a set of radio buttons as follows:

Radio controls

```
<div class="input">
 <label>Likeability</label>
    <div class="radiogroup">
     <span><input type="radio" name="likeable" checked="checked"/><label>I
do</label></span>
        <span><input type="radio" name="likeable"/><label>I don't</label></span>
        <span><input type="radio" name="likeable"/><label>Meh</label></span>
    </div>
</div>
```

Declare a set of check boxes as follows:

Checkboxes

```
<div class="input">
 <label>Checkboxes</label>
    <div class="checkgroup">
     <span><input type="checkbox" name="options" checked="checked"/><label>option
1</label></span>
        <span><input type="checkbox" name="options"/><label>option 2</label></span>
    </div>
</div>
```

Validation

Validation can be added to input elements in two steps.

Step 1 is to add the "validatable" class to an input group. This will handle highlighting the field when an error occurs as well as toggling the visibility of an error message which should be to your input block as shown below:

Adding validation

```
<div class="input validatable">
 <label for="firstName">First Name</label>
 <input type="text" name="firstName"/>
 <div class="errormessage">Name greater than 200 chars</div>
</div>
```

Step 2 is to add the "error" class to the input group when an error occurs in order to trigger the validation styles. The can be done in various ways such as through Javascript.

Showing validation errors

```
<div class="input validatable error">
 <label for="firstName">First Name</label>
 <input type="text" name="firstName"/>
 <div class="errormessage">Name greater than 200 chars</div>
</div>
```

Actions

Action buttons will typically appear at the bottom of the form. The types of buttons in this area are classified in two groups:

- Primary - buttons that submit the form, move to the next screen in a wizard or generally have the most important impact on the data
- Secondary - cancel, back and clear buttons

Each group is given a different visual treatment. To add actions declare the following mark-up in the content area of your form below any input fields:

Adding action buttons

```
<div class="actions">
 <div class="secondary">
     <button>Cancel</button>
     <button>Back</button>
 </div>
    <div class="primary">
        <button>Save</button>
    </div>
</div>
```

The **"actions"** area accepts buttons, href's or input elements.

Context Example App

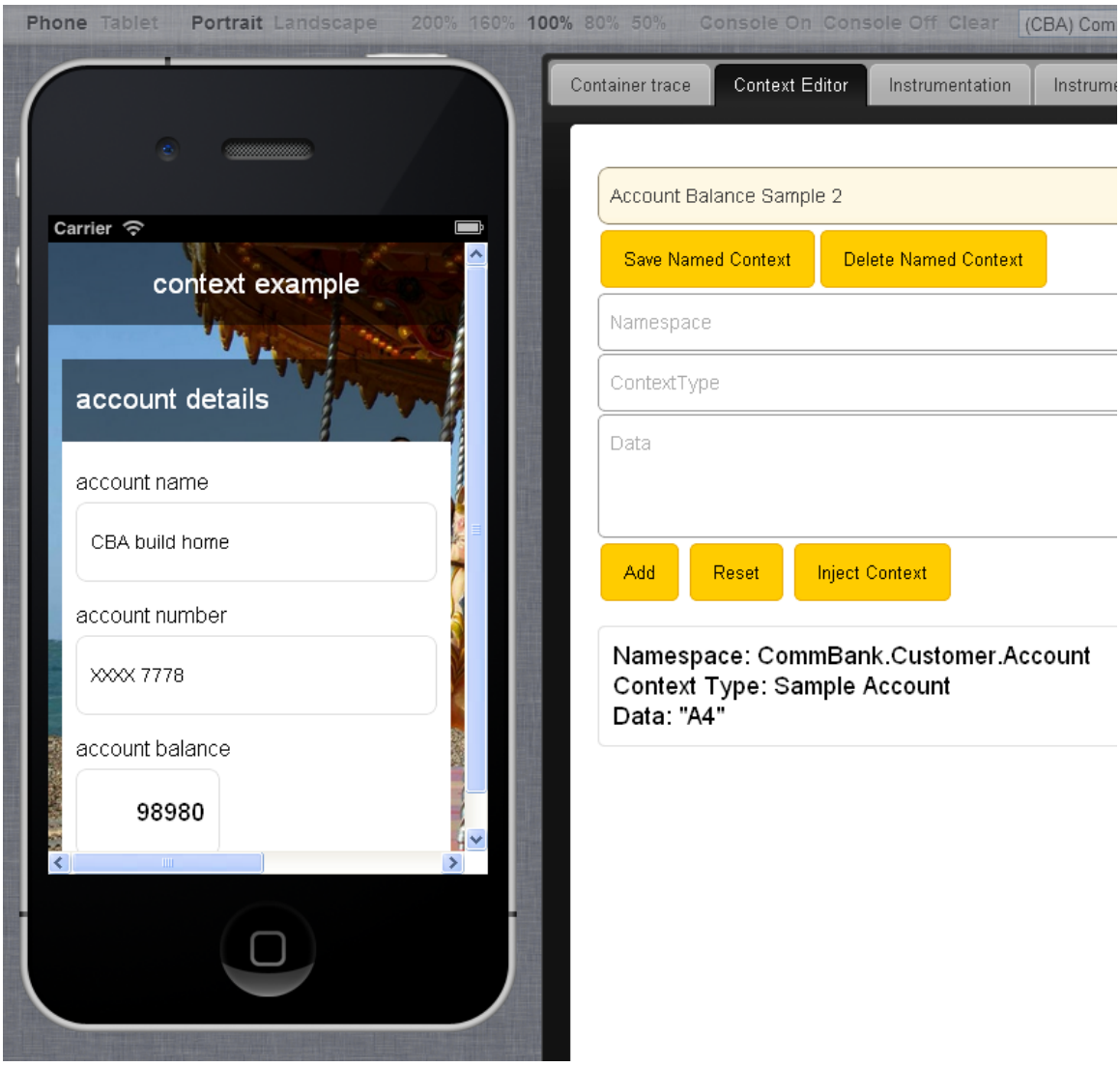
Concepts illustrated

The Context Example App was built to illustrate a use of the Context as well as how to use MockJAX to provide a mock implementation of a service call when developing an application.

How does it work?

The Context Example App subscribes to the AppStore.App.EVENTS.ON_CONTEXT_CHANGE event and will thus get notified when the context is changed. It will also inspect the context when it starts up. If there is an item in the context that has the namespace set to "CommBank.Customer.Account", then the data will be used to lookup the account detail through a postJSON call. The account detail will then be displayed accordingly.

MockJAX was used to mock (or stub out) the implementation of the service that will return the account data. The account data is served from a JSON object that is stored in the Scripts/AllData.js file.



Navigation Example - Sample App

Navigation within an app

When a container launches an app it appends some fields to the app's URL query string so that the app API can communicate back to the container.

If the app is not a Single Page Application then it's important that this information is present in the query string of all the pages in the app. The container is aware when the content of an iframe changes because of a navigation event, and if the new page fails to connect to the container it will delete the iframe in order to prevent phishing attacks.

Links

For the *href* attribute for any <a> tags that don't reference JavaScript or local # tags e.g.

```
<a href='#section1'>Go to section 1</a>
```

the app must ensure that the correct query strings are added.

Forms

The same rules apply even if a form within a page is posting back to the same page. If the *action* attribute on a

<form> tag is not set then by default the browser uses the current URL, which will have the correct query string data. However if the *action* attribute is explicitly set e.g.

```
<form id='form1' runat='server' action='Default.aspx'>
 <input type='text' name='firstName'/>
</form>
```

then the browser will use the exact URL in the *action* attribute.

The solution

The app API has a helper method checkQueryString(url) that returns a url with the correct query string data. If you are constructing urls in code then you need to call this method e.g.

```
var theURL = AppStore.App.checkQueryString("default.aspx?foo=bar");
```

If your app is using jQuery then the app API adds a plugin, fixAppURL(), that will fix the urls for elements that already exist on the page. The Navigation Example uses this method to fix all urls on the page with one statement:

NavigationSample.js

```
$(function() {
 $("form, a").fixAppURL();
});
```

Navigating to a different app

Apps declare which other apps they may navigate to in the *invokes* section of their manifest. At runtime one or more of those apps might not be available within a given container, whether due to container configuration, user permissions, device capabilities or several other factors. The app can check which of its outgoing endpoints have been resolved by calling the canNavigate() method, which takes the endpoint interface id as its parameter.

To invoke another app the calling app calls the navigate() method, passing in the endpoint id and an optional parameters object, which can be any JavaScript object, which presumably the app being launched will understand. These data contracts are not specified in either app's manifest, it is up to app developers to make sure that when they call another app they pass the correct parameters.

The Navigation Example contains references to two apps in its manifest, one which is present in the container and one which isn't. There are links on Page 1 to the two apps, which are initially hidden. It uses the following code to check all the app links on the page and only enable those which have been resolved by the container:

```
$(".appLink").click(
  function() {
   AppStore.App.navigate($(this).data("endpoint"), { colour: $(this).data("colour"),
flavour: $(this).data("flavour") });

   return false;
  }
 ).toggleClass(
  function(i, c, u) {
   return AppStore.App.canNavigate($(this).data("endpoint")) ? "hidden" :
'canNavigate';
  }
 );
```

It also uses data attributes on the links to store the various parameters for each app.

Building your App

To build your App, the minimum needed is to supply a Manifest.xml file. Additionally you'll want to make your App align with the styles and themes of the Future Bank experience. The easiest way to do is to copy the Hello World - Sample App that is the zip file. This will give a manifest and the basic page structure for the Container's style and theming logic to be injected into your App.

Style and theming

You don't have to implement our css style rules, you can choose to adhere to the Design Guidelines with a different implementation, but may be less work to use the css classes defined in the Doing it with style guide.

Manifest

These instructions detail the changes needed to make your App load up correctly in the Container.

> For this instruction, all paths are relative to the root folder you created in Setting up your machine

1. Start by copying the *Apps/HelloWorld* folder to a new folder with the name of *YourApp*
2. In your *Apps/YourApp*'s folder, edit the Manifest.xml file to
    a. change the App name
    b. change the App Id
    c. change the Publisher Id- must be the same as the name on your competition registration
    d. change Publisher Name
3. Edit the container's (root folder) Environment.xml
    a. Inside the <apps> node, add a new *app* node pointing to your apps *manifest.xml*

```
<apps>
 ....
 <add host="Apps/YourApp/manifest.xml"/>
 ....
</apps>
```

4. Save the Environment.xml file, then refresh (shortcut-F5) your Developer Container.

5. Confirm that you can navigate to your new App in the Developer Container. You can change the App that is loaded in the container in the dropdown on the top right
6. You can then move onto implementing the other aspects of your App as part of the other developer info.

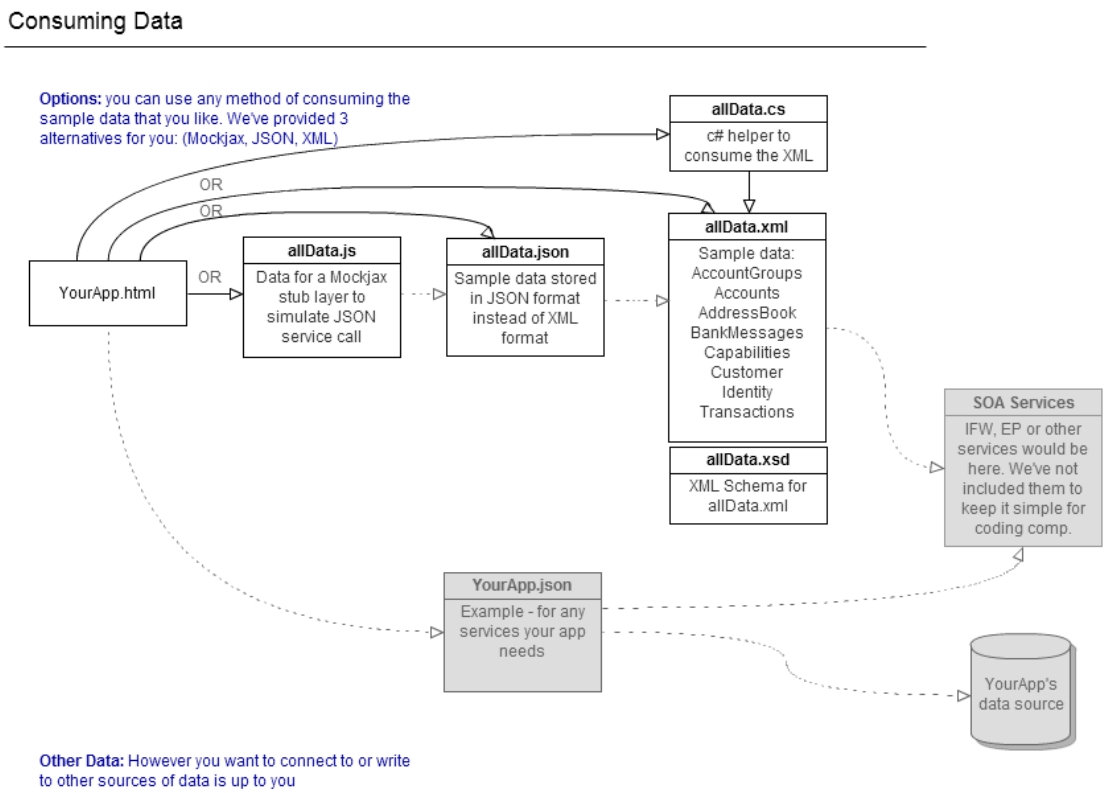
Coding Comp Sample Dataset

We have collapsed all of our data services into a single, very rich data API for the purpose of this competition. This has a very wide range of sample data for one customer. There are 2 files that describe it:

1. The Sample Data XML file - contains a fictitious customer's complete view of all their information with the Bank
2. The XSD schema definition - provides an overall view of the structure in which all data nodes are arranged.

Consuming the data from your app could be done a number of ways depending on whether you want to consume the XML serverside or just deal with JSON objects client side:

1. Using mockjax and the mockjax file to simulate a JSON call within the browser. This would be helpful if you only want to use clientside code and no server side technologies. The best way to see this is with the Context Example App
2. hosting the JSON data file on your webserver and using JQuery tools to interact with it via JSON service requests. This would be the simplest way of accessing the data if you don't plan on segmenting it or augmenting it with any other data.
3. using your favourite serverside technology, manipulate the XML to retrieve the data you want to return to your client side.
    a. We've provided a couple of classes to help consume this more easily if you are using C#



You can use this data as the basis for any kind of application. Each of the main nodes within XML schema is

described in further detail separately in the child pages (Address Book, Bank Messages, Accounts, etc..) below. Each child page provides a view on usage of each attribute along with a sample XML data extract from within XML schema.

1. Account Groups names for groupings that make it easier for customers to view their accounts.
2. Accounts and Account Totals the list of accounts and balances for each.
3. Address Book data and display a unique view of people that the customer pays.
4. Bank Messages the list of the latest messages for the customer.
5. Capabilities these define what kind of functions the customer can perform on each of the accounts.
6. Customer and Identity samples provide all details related to a particular individual.
7. Completed Transactions and Scheduled Transactions the list of transactions with details for transaction history and future dated transactions.

Account Groups

Account Groups provides ability to group certain accounts in one group and view them together. Account Groups consists of name, totals and accounts list. Name refers to name of an account group provided by user. Totals refer to signed & unsigned totals with appropriate sign for the value. In addition, it also consists of totals assets, total liabilities, and total available funds for the specific account group. Finally, it consists of list of accounts that forma the specific account group.

Attributes			Description
isDefaultGroup			Boolean value to specify whether the account group is default.
name			Friendly name of account group
totals			
	netTotals		
		signed	Shows positive / negative value of totals in decimals in AUD
		unsigned	Shows absolute value of the totals in decimals in AUD currency
		sign	CR- Credit DR- Debit If totals are 0, then it is always shown as CR.
	totalAssets		
	totalLiabilities		
	totalAvailable		
	accounts		
		ID	

The following sample shows a snippet with above mentioned attributes for an account group

```
<accountGroup isDefaultGroup="true">
      <name>Credit card accounts</name>
      <totals>
        <netTotal currency='AUD'>
          <signed>0</signed>
          <unsigned>0</unsigned>
          <sign>CR</sign>
        </netTotal>
        <totalAssets currency='AUD'>1000</totalAssets>
        <totalLiabilities currency='AUD'>1000</totalLiabilities>
        <totalAvailable currency='AUD'>0</totalAvailable>
      </totals>
      <accounts>
        <account id='A2'/>
        <account id='A3'/>
      </accounts>
</accountGroup>
```

Accounts

Accounts consist of details about an account. Name, number, product code, balance, available funds, limit & capabilities are high level attributes available under an account.
Name refers to the account name. Number refers to bank identifier code, BSB, account number attached to the specific account. Bank Identifier code is alpha numeric and is also called as SWIFT code which is used in international transfers. BSB refers to bank state branch id. Account number refers to unique identifier of an account. In addition, it also consists of capabilities which are applicable to the account.

Attributes		Description
Id		Unique account identifier for each account
name		
	preferredName	User preferred name of the account
	legalName	Legal name of the account as decided by bank
	typeName	Type name of account resembling to a specific type
	nickName	Account Nickname
Number		
	bic	Boolean value to show whether bic (Bank Identifier code/ SWIFT code) is applicable to the account or not. If applicable, then shows value of bic.

	bsb	Boolean value to show whether BSB is applicable to the account. If applicable, then shows value of bsb Length – Max 6
	accountNumberFormatted	Boolean value to show whether account number in formatted type is masked or not. If yes, then shows masked account number. Format for mask number – XXXX 5678 Length – Max 16
	accountNumberUnformatted	Boolean value to show whether account number in unformatted type is masked or not. If yes, then shows masked account number. Format for mask number – XXXX 8765 Length – Max 16
	preferred	Shows BSB (if applicable, last four digits only) & account number for the account
productCode		Product code for the account Possible values are – • DDA • MCD • HLN
balance		
	signed	Positive / negative value of balance in decimals in AUD currency
	unsigned	Absolute value of the balance in decimals in AUD currency

	sign	CR- Credit DR- Debit If balance is 0, then it is always shown as CR.
availableFunds		Available funds for the account in AUD currency
limit		Limit applicable to the account
capabilities		Capabilities applicable to the account

```
<accounts>
    <account id='A1'>
      <name>
        <preferredName using='typeName'>Gold Saver</preferredName>
        <legalName>Netbank Saver</legalName>
        <typeName>Savings</typeName>
        <nickname hasNickName='false'></nickname>
      </name>
      <number>
        <bic hasBic="false"></bic>
        <bsb hasBsb="false"></bsb>
        <accountNumberFormatted isMasked="true">XXXXX XXXX XXXX
7373</accountNumberFormatted>
        <accountNumberUnformatted isMasked="true">71717 1717 3737
7373</accountNumberUnformatted>
        <preferred>27272 27 272 77272</preferred>
      </number>
      <productCode>DDA</productCode>
      <balance currency='AUD' isSupported='true' isLive='false'
asAt='2011-01-01T12:00:00'>
        <sign>CR</sign>
        <signed>10000</signed>
        <unsigned>10000</unsigned>
      </balance>
      <availableFunds currency='AUD' isSupported='true' isLive='true' asAt='2011-01-
01T12:00:00'>10000</availableFunds>
      <limit currency='AUD' isSupported='true' isLive='true'
asAt='2011-01-01T12:00:00'>10</limit>
      <capabilities>
        <transferFrom isSupported='true'/>
        <transferTo isSupported='true'/>
        <bpayFrom isSupported='false'/>
        <completedTransactions isSupported="false"/>
        <pendingTransactions isSupported="false"/>
        <scheduledTransactions isSupported="false"/>
      </capabilities>
     </account>
</accounts>
```

Account Totals

Account Totals refers to total value for the account. It includes net Totals, total Assets, total Liabilities & total Available funds.
Net Totals refers to total value of the amount left in the accounts after all debits and credits. Total Assets refers to amount equivalent to total assets in customer's account whereas total liabilities refer to amount equivalent to total liabilities in customer's account. Total Available refers to total available amount in user's account.

Attributes		Description
netTotals		
	signed	Positive / negative value of Net Totals in decimals in AUD currency
	unsigned	Absolute value of the Net Totals in decimals in AUD currency
	sign	CR- Credit DR- Debit If balance is 0, then it is always shown as CR
totalAssets		Totals assets for the account in AUD currency Displayed as positive amount
totalLiabilities		Total liabilities for the account in AUD currency Displayed as negative amount
totalAvailable		Total available funds for the account in AUD currency

```
<totals>
   <netTotal currency='AUD'>
    <signed>18321.50</signed>
    <unsigned>18321.50</unsigned>
    <sign>CR</sign>
   </netTotal>
   <totalAssets currency='AUD'>18321.50</totalAssets>
   <totalLiabilities currency='AUD'>0</totalLiabilities>
   <totalAvailable currency='AUD'>18321.50</totalAvailable>
</totals>
```

Address Book

The Address Book is used for paying funds to other people. It contains a comprehensive list of all Payees for BPay, transfers to Australian & international accounts and Social funds transfer (via email & facebook ids). Each payee has a name, globally unique Id, a list of the recent payments made to this payee and a list of multiple payment methods. Each payment method has an Id and a status code. The status code indicates if the payment method is new, has been used before, or has been deleted by the customer. Use of new payment methods may be subject to additional security considerations. Each payment method has attributes specific to the payment method type.
All Attributes are mandatory. Id's are alpha-numeric. Each collection details how many items are valid (for that collection).

Attributes						Description
mostRecentlyUsed						Shows the most recently used payee
Payees						
	Payee					
		Id				Globally unique identifier of Payee. Referenced from scheduled and completed transactions.
		name				Friendly name of Payee. Set by the customer. Not blank
			bPayCodes			0-unbounded Bpay Billers
				bPayCode		
					Id	Globally unique identified for payment method
					billerName	Official name of biller. Cannot be edited by customer.
					billerCode	
					customer Reference Number	

					methodStatus	Options: NEW: New biller has not yet been paid to. VALID: Biller successfully been paid to in the past DELETED: Customer has removed the payment method
			australianAccounts			
				australianAccountId		Globally unique identified for payment method.
				bsb		6 Digit BSB. Contains no space characters. Mandatory
				accountNumber		7-10 digit number. Contains no space characters
				methodStatus		Options: NEW: New biller has not yet been paid to. VALID: Biller successfully been paid to in the past DELETED: Customer has removed the payment method
			internationalAccounts			

				internationalAccountId		Globally unique identified for payment method
				bic		Bank Identifier Code for the international account. Needed to do an IMT. Also called swift code.
				Iban		International Bank Account Number for international account.
				country		
					code	ISO3166 – 3 Character Country Code
					{value}	Friendly Country Name
				methodStatus		Options: NEW: New biller has not yet been paid to. VALID: Biller successfully been paid to in the past DELETED: Customer has removed the payment method
			emailAddresses			
				emailAddresses		

					Id	Globally unique identified for payment method
					email	Actual email address in format – “x@x.com”
					methodStatus	Options: NEW: New biller has not yet been paid to. VALID: Biller successfully been paid to in the past DELETED: Customer has removed the payment method
			facebookIds			
				facebookId		
					Id	
					facebookId	Actual facebook ID for the payee
					methodStatus	Shows the status of VALID / INVALID/ NEW for facebook ID
			recentTransaction			
				Id		Transaction Identifier for the most recent transaction done to payee.

```
This sample has two payees, only one of which has payed before – and appears in the
Most Recently Used list.

<addressBook>
    <mostRecentlyUsed>
      <payee id='P1'></payee>
    </mostRecentlyUsed>
</addressBook>

The first payee can be paid via a linked Australian account number or social payment
(via an email address).

<payees>
      <payee id='P1'>
        <name>John Smith</name>
        <australianAccounts>
          <australianAccount id='PM1'>
            <bsb>062978</bsb>
            <accountNumber>6256288</accountNumber>
            <methodStatus>VALID</methodStatus>
          </australianAccount>
        </australianAccounts>
        <bPayCodes/>
        <emailAddresses>
          <emailAddress id='PM2'>
            <email>John.Smith@hotmil.com</email>
            <methodStatus>VALID</methodStatus>
          </emailAddress>
        </emailAddresses>
        <facebookIds/>
        <internationalAccounts/>
        <recentTransactions>
          <recentTransaction id='TC10'/>
          <recentTransaction id='TC12'/>
        </recentTransactions>
      </payee>
</payees>

The second payee can be paid via BPay – but has never yet been paid.

     <payee id='P2'>
        <name>Telstra</name>
        <bPayCodes>
          <bPayCode id='P2M1'>
            <billerName>Telstra Ltd</billerName>
            <billerCode>87765</billerCode>
            <customerReferenceNumber>90909090</customerReferenceNumber>
            <methodStatus>NEW</methodStatus>
          </bPayCode>
        </bPayCodes>
        <australianAccounts/>
        <emailAddresses/>
        <facebookIds/>
        <internationalAccounts/>
        <recentTransactions/>
      </payee>
```

Bank Messages

Bank Messages are used by bank to communicate with the users. Bank messages are sent by bank which is then made available to user via their inbox within banking platform.

Bank Messages consists of Message Format, requires acknowledgement, is Read, date Sent, type, is broadcast, date Read attributes.

Attributes	Description
messageFormat	Describes the format of message. Possible values are - 1. HTML 2. PlainText
requiresAcknowledgement	Boolean value to show whether user needs to acknowledge the receipt of message
isRead	Boolean value to show whether user has read the message
dateSent	Date when the message was sent to the user
type	Describes the type of the message Possible values are – 1. SECURITY 2. MARKETING 3. OFFER 4. DISCOUNT 5. INFORMATION
isBroadcast	Boolean value to show whether the message needs to be broadcasted to all users
dateRead	Shows the date when the message was read Date Read = available, nillable = false

```
<bankMessages>
    <bankMessage id='M1'>
      <message format='HTML'><![CDATA[]]></message>
      <requiresAcknowledgement>true</requiresAcknowledgement>
      <isRead>true</isRead>
      <dateSent>2010-01-01</dateSent>
      <type code='SECURITY'>Security Announcement</type>
      <isBroadcast>false</isBroadcast>
      <dateRead xsi:nil='false'>2010-01-02</dateRead>
    </bankMessage>
</bankMessages>
```

Capabilities

Capabilities provide a view what a particular user can do with their account. It contains "transferToSelf", "transferToAustralianAccount", "transferToInternationalAccount" and "bPAY"
Each capability provides maximum transfer allowed, remaining daily limit available after transfer and default account from which transfer will happen

Attributes		Description
transferToSelf		
	maximumSingleItem	Whether unlimited transfer to one of user's account is allowed. If not allowed, then provide actual limit amount for a single transfer
	remainingDailyAvailable	Shows remaining daily limit available to transfer to one of user's own accounts.
	defaultFromAccount	Provides identifier for default account to be used for such kind of transfer
transferToAustralianAccount		
	maximumSingleItem	Whether unlimited transfer to Australian account is allowed. If not allowed, then provide actual limit amount for a single transfer
	remainingDailyAvailable	Shows remaining daily limit available to transfer to other Australian accounts.
	defaultFromAccount	Provides identifier for default account to be used for such kind of transfer
transfertoInternationalAccount		
	maximumSingleItem	Whether unlimited transfer to international account is allowed. If not allowed, then provide actual limit amount for a single transfer
	remainingDailyAvailable	Shows remaining daily limit available to transfer to international accounts.

	defaultFromAccount	Provides identifier for default account to be used for such kind of transfer
bPay		
	maximumSingleItem	Whether unlimited bPAY to any account is allowed. If not allowed, then provide actual limit amount for a single transfer
	remainingDailyAvailable	Shows remaining daily limit available to bPAY to any other accounts.
	defaultFromAccount	Provides identifier for default account to be used for bPAY

```
<capabilities>
    <transferToSelf isSupported='true'>
      <maximumSingleItem isUnlimited='true'>1000</maximumSingleItem>
      <remainingDailyAvailable isUnlimited='true'>1000</remainingDailyAvailable>
      <defaultFromAccount id='A1'/>
    </transferToSelf>
    <transferToAustralianAccount isSupported='true'>
      <maximumSingleItem isUnlimited='false'>0</maximumSingleItem>
      <remainingDailyAvailable isUnlimited='false'>20000</remainingDailyAvailable>
      <defaultFromAccount id='A1'/>
    </transferToAustralianAccount>
    <transferToInternationalAccount isSupported='false'>
      <defaultFromAccount id='A1'/>
      <maximumSingleItem isUnlimited='false'>0</maximumSingleItem>
      <remainingDailyAvailable isUnlimited='false'>0</remainingDailyAvailable>
    </transferToInternationalAccount>
    <bPay isSupported='false'>
      <maximumSingleItem isUnlimited='false'>0</maximumSingleItem>
      <remainingDailyAvailable isUnlimited='false'>20000</remainingDailyAvailable>
      <defaultFromAccount id='A1'/>
    </bPay>
</capabilities>
```

Customer

Customer tag consists of information related to type, name, optional person data and optional non personal data. Type consists of a code which shows whether the customer is a Person or Non Person. Name consists of attributes such as legal name & preferred name. Optional person data consists of title, first name, middle name, last name and suffix.

Attributes			Description
type			

	code		Customer type code Possible values are – • PERSON • NONPERSON
name			
	legalName		Legal name of the customer
	preferredName		Preferred name of the customer
	optionalNonPersonData		Provides non personal data about customer
	optionalPersonData		
		title	
		firstName	
		middleName	
		lastName	
		suffix	

```
<customer>
    <type>
      <code>PERSON</code>
    </type>
    <name>
      <legalName>Mr John Smith</legalName>
      <preferredName overriden='false'>Mr John Smith</preferredName>
      <optionalNonPersonData></optionalNonPersonData>
      <optionalPersonData>
        <title>Mr</title>
        <firstName>John</firstName>
        <middleName>Bell</middleName>
        <lastName>Smith</lastName>
        <suffix>2</suffix>
      </optionalPersonData>
    </name>
</customer>
```

Identity

The identity consists of important information about the user. It contains Netbank Client ID, friendly name of the user and date when user logged in successfully last time. Netbank client ID is numeric only.

Attributes	Description
netbankID	Unique login ID for Netbank platform Format: 8 digit Identifier
friendlyName	Friendly Name of user

lastSuccessfulLogin	Last successful login date of user in Netbank platform

```
<identity>
    <netbankId>12345678</netbankId>
    <friendlyName>Mr John Smith</friendlyName>
    <lastSuccessfulLogin>2002-09-24</lastSuccessfulLogin>
</identity>
```

Transactions (Completed Transactions)

Completed transactions refer to information about customer's initiated transaction. Completed transaction consist of description, date, tags, location, transaction type, amount, balance after, receipt number, debit account details and payee information
Description refers to raw and cleaned description for the transaction entered by user. Date refers to value and posting date/time for the transaction. Tags refer to the categories in which the transaction is kept by the user. Location refers to the information about the actual geographic location where the transaction occurred. Transaction type refers to the source from where transaction was executed. Amount refers to the transaction amount in AUD currency with an appropriate sign of debit or credit. Balance after refers to the balance left in account after completion of the transaction. Receipt number refers to the unique identifier of the transaction receipt. Debit account details provide the account identifier from which the transaction was initiated whereas payee information provides the payee identifier to whom the payment was done.

Attributes			**Description**
Id			Unique Transaction identifier for every transaction
Description			
	cleaned		Description shown after cleaning. This refers to l ength of description and truncating any other characters beyond decided length.
	raw		This refers to raw description entered by user
date			
	isPending		Boolean value to show whether the transaction is pending or not
	postingDate		Date on which the transaction was posted to core banking system

	valueDate		Date on which the transaction actually happened at EFTPOS
	postingDateTime		Date/Time for posting transaction
	valueDateTime		Date/Time for actual transaction occurrence
	preferred		User preferred date to be shown
tags			Shows the tags under which the transaction is categorised. Following are the tags available for selection – • FOOD • CLOTHES • PETROL
location			
	suburb		Suburb where the transaction occurred
	state		State in which transaction occurred Possible values – • NSW • WA • QLD • VIC • ACT • SA • TAS • NT
	country		Country name in which transaction happened
	geoLocation		
		laitude	Latitude geographic location where transaction occurred
		longitude	Longitude geographic location where transaction occurred
		radius	Radius in metres of the area where transaction occurred

	transactionType		Type of the payment used to do transaction.Possible Option – EFTPOS
	amount		
		signed	Positive / negative value of transaction amount in decimals in AUD currency
		unsigned	Absolute value of the transaction amount in decimals in AUD currency
		sign	CR- Credit DR- Debit If balance is 0, then it is always shown as CR.
	balanceAfter		
		signed	Positive / negative value of pending balance post-transfer in decimals in AUD currency
		unsigned	Absolute value of the pending balance post-transfer in decimals in AUD currency
		sign	CR- Credit DR- Debit If balance is 0, then it is always shown as CR.
	receiptNumber		Receipt number generated by transaction platform for the transaction
	fromAccount		
		Id	The account from which the transaction was initiated & was debited
	toPayee		
		Id	Payee identifier to whom transaction is made

The sample shows all the required attributes for the completed transaction. Following sample shows description and date information for a transaction which is not pending –

```
<description>
   <cleaned>Rent for previous month</cleaned>
   <raw>Rent for my previous month pending</raw>
 </description>
 <date>
   <isPending>false</isPending>
   <postingDate>2010-10-01</postingDate>
   <valueDate>2010-10-01</valueDate>
   <postingDateTime>2010-10-01T01:00:00</postingDateTime>
   <valueDateTime>2010-10-01T01:00:00</valueDateTime>
   <preferred>2010-10-10</preferred>
 </date>
```

A pending transaction will have a date on which the transaction will be executed. Following sample shows date information for a pending transaction –

```
 <date>
   <isPending>true</isPending>
   <postingDate>2012-08-03</postingDate>
   <valueDate>2012-08-01</valueDate>
   <postingDateTime>2012-08-03T11:00:00</postingDateTime>
   <valueDateTime>2012-08-01T12:00:00</valueDateTime>
   <preferred>2012-08-04</preferred>
 </date>
```

Following sample shows a snippet about tags, location and transaction type for a completed transaction –

```
<tags isSupported='true'>FOOD</tags>
  <location known='true'>
    <suburb>Crows Nest</suburb>
    <state code='NSW'>NSW</state>
    <country code='AUS'>Australia</country>
    <geoLocation isSupported='false'>
      <latitude>0</latitude>
      <longitude>0</longitude>
      <radius unit='m'>1000</radius>
    </geoLocation>
  </location>
  <transactionType code='EFTPOS'>Eftpos</transactionType>
```

Following sample shows a snippet on amount, balance after, receipt number, debit account and payee details for a completed transaction -

```
 <amount currency='AUD'>
   <signed>1</signed>
   <unsigned>1</unsigned>
   <sign>CR</sign>
 </amount>
 <balanceAfter currency='AUD'>
   <signed>1</signed>
   <unsigned>1</unsigned>
   <sign>CR</sign>
 </balanceAfter>
 <receiptNumber>1010101</receiptNumber>
 <fromAccount id='A1'/>
 <toPayee id='P1'/>
```

Transactions (Scheduled Transactions)

Scheduled transaction refers to any upcoming transaction scheduled by the customer to be executed on a particular date in future. Scheduled transactions consist of attributes like transaction identifier, amount, description, next scheduled date, frequency, low funds alert, from account details, to account details.
ID refers to a unique transaction identifier. Amount refers to the transaction amount for scheduled transaction. Description refers to the description of the scheduled transaction. Next scheduled date refers to the future date on which the transaction is meant to be executed. Frequency refers to the repetitive pattern for a transaction to be executed. Low funds alert refers to the flag indicator on the account which notifies the customer if the account from which the scheduled transaction is meant to go out has low funds near to scheduled date. From account details shows the account identifier from which the amount for transaction will be debited. To account details consists of payee information to whom the transaction is done.

Attributes			Description
Id			Unique scheduled transaction identifier specific to transaction
amount			Amount for scheduled transaction to be debited
description			Description of scheduled transaction that will appear on user's statement
nextScheduledDate			Future date on which the transaction is scheduled to happen
frequency			Frequency for scheduled transaction. Possible values – • ONCE • DAY • WEEK • MONTH
lowFundsAlert			Boolean value to show whether indicator (flag) for low funds alert is set for the specific scheduled transaction
fromAccount			The account from which the scheduled transaction is meant to be executed
to			
	friendlyDescription		Description of the transaction appearing on payee's side

	optionalPayee		
		methodID	Method identifier for payment type
		payeeID	Payee identifier to whom payment is going
	type		Type of payee (BILLER OR PAYEE) to whom the payment is going

```
<scheduledTransaction id="9998887">
        <amount currency='AUD'>130</amount>
        <description>Home and Contents Insurance</description>
        <nextScheduledDate>2012-10-11</nextScheduledDate>
        <frequency unit='DAY'>4</frequency>
        <lowFundsAlert>true</lowFundsAlert>
        <fromAccount id='A5' />
        <to>
          <friendlyDescription>111090 CBA repayment</friendlyDescription>
          <optionalPayee>
            <method id='P5M1'/>
            <payee id='P5'/>
          </optionalPayee>
          <type>PAYEE</type>
        </to>
</scheduledTransaction>
```

Doing it with style

Making an App look good takes a lot of work if you have to do everything. Making it not look out of place with other Apps in a Container is a little harder. Making it work with multiple corporate brands and themes is even harder still.

We've built a set of css style components to help your App look good by default, so it doesn't have to be hard to achieve the Design Guidelines. These css rules are magically made available to your App with a little framework code that injects the CSS files from the container into your App's iFrame.

Style Specification

Visual structure of a UI
A picture of the major components of the UI and the CSS classes that define them

Foundations
Information on implementing UI foundations like grid/columns, navigation and UI building blocks that you can use to help develop your own user interfaces designs

Building Blocks
Provides bare bone structure for building an app

- Typography shows information on font sizes depending on themes
- Colour provides a view on colour pattern that can be used in app
- Button and Links defines the buttons and links structure that can be used in app

- Iconography provides a set of icons that can be used in app

Style
In Code

Implementing themes
Step by Step theming A guide to add themes one step at a time

Declaring theme injection
How to use the Manifest.xml to declare the themes being used.

UI Component Structure
Visual layouts to show how the the divs + classes are used to create different layouts of subsections of a UI

Examples
These show where the classes are used on Future Bank Apps

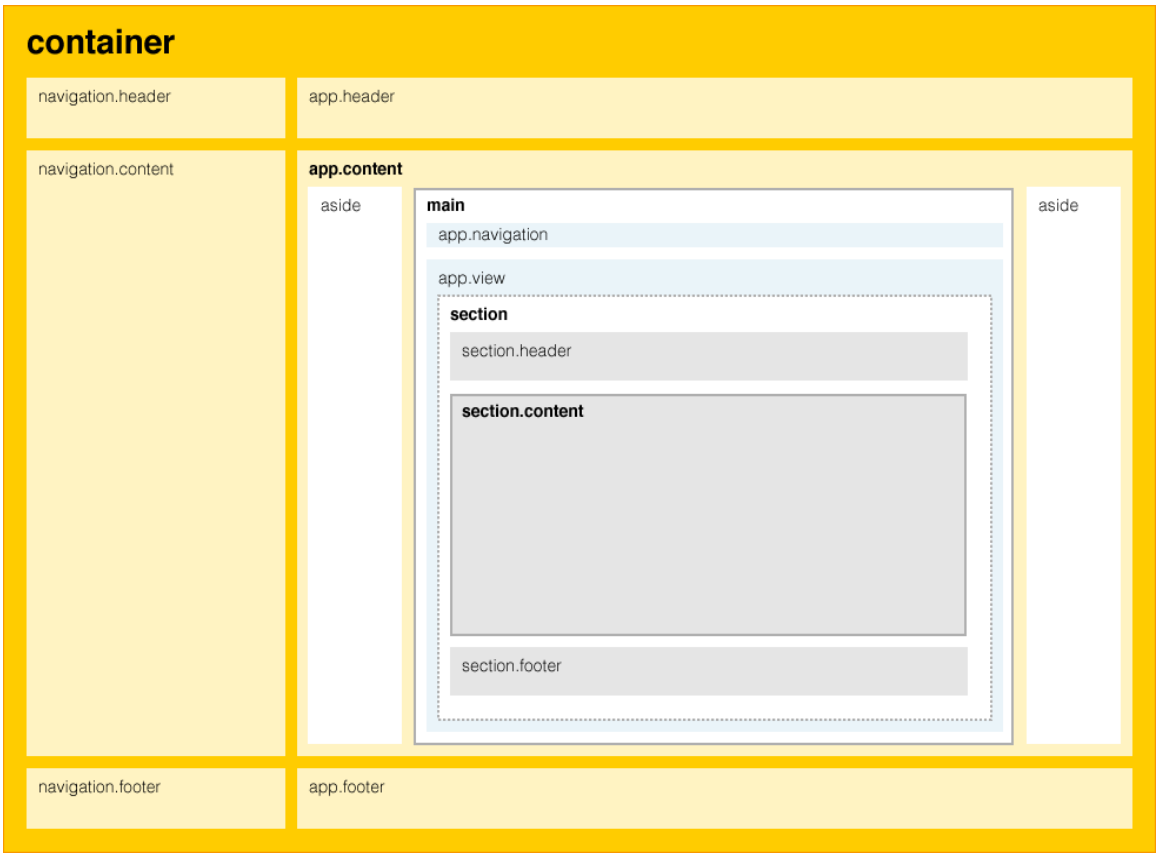
- Selectable Content
- Scrollable Content
- Filter Content
- Collapsible Section
- Subdividing List
- Static List
- UI Control (Radio)
- Form
- Featured Content

Visual structure of a UI

Below is a diagram to introduce the top-level taxonomy for **container**, **navigation** and **app content**

From a layout perspective, other positions of content are possible (eg navigation above or on right etc). This diagram illustrates the names and the nesting relationship between them more than the positioning.

The App is responsible for anything inside app.header, app.footer or app.content



Foundations

Baseline

Baseline values as a best practices example, and how it scales responsively

Foundation Details

It is not necessary to implement foundation guidelines into an app for coding comp as the UI is not developed enough to cater for the rules associated with grids, columns, baselines and navigation. The Foundations are something that will be developed post coding comp.

We will be developing the foundations in the following areas:

Grid/Columns

Outline a universal grid and highlight its differences across different resolutions.

Baseline

Baseline values as a best practices example, and how it scales responsively.

Navigation

Illustrate behaviours of the container navigation and the utilities menu.

*Note** Please do not assume navigation will always be on the left*

Navigation0

Is this something we need to define in terms of our sample apps or more generically that that?
Illustrate behaviours of the container navigation and the utilities menu
Note* don't assume navigation will always be on the left

Building Blocks

Buttons and Links

Colour

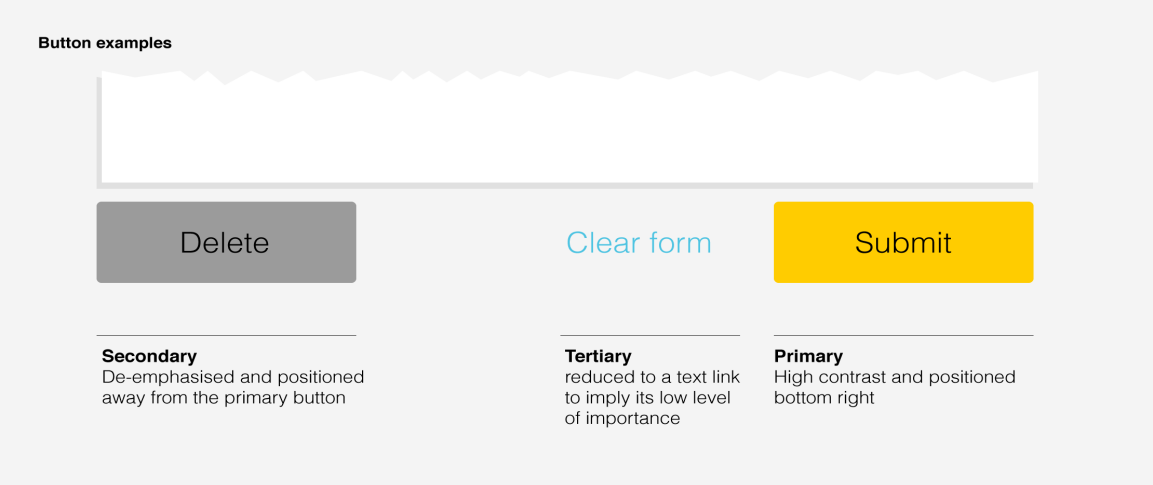
Iconography

Typography

Buttons and Links

Best practices
It's important to establish some visual language with buttons by establishing primary, secondary, tertiary etc styles.
Keep button text on a separate layer. Use web text for button text, rather than image buttons.
Reserve the brightest primary or support colour for primary and the de-emphasise the colours and treatment for secondary and tertiary buttons.
For even greater differentiation between buttons, use a text link alongside a button.
Styling a button in a way that is obvious it's a selectable object is important, eg. if every other object on an interface has square corners, give the buttons rounded corners to help identify it as a button.

Submit button (primary)
High visibility and primary compared to other buttons. When used to 'continue' to next step position to the right of the screen, which help it imply movement to the right. If its the final step in a process position it centred to imply it's the final step.
Delete button (secondary)
Try not to position too close to a submit button to avoid accidentally deleting something you want to submit.
Clear button (tertiary)
A good example of a tertiary action as it has low implication if accidentally selected.



Button sizes
One of the guiding principles of touch-first design is larger touch targets. All buttons should be a minimum of 44px x 44px. Touch targets should be larger than 9mm regardless of a display's pixel density (refer to Pixel Density for more)

Be especially considerate if: the result of a touch error is severe or really frustrating; the UI element is located toward edge of the screen or difficult to hit; or when the UI element is part of a sequential task – like using a numerical dial pad.

Lists

Make sure all link lists have touch target sizes equal to buttons, or at least an equal amount of space between them. Visually this will give us a lot of room to read the links, as well as plenty of room to click them on touch devices.

Colour

Primary Colours

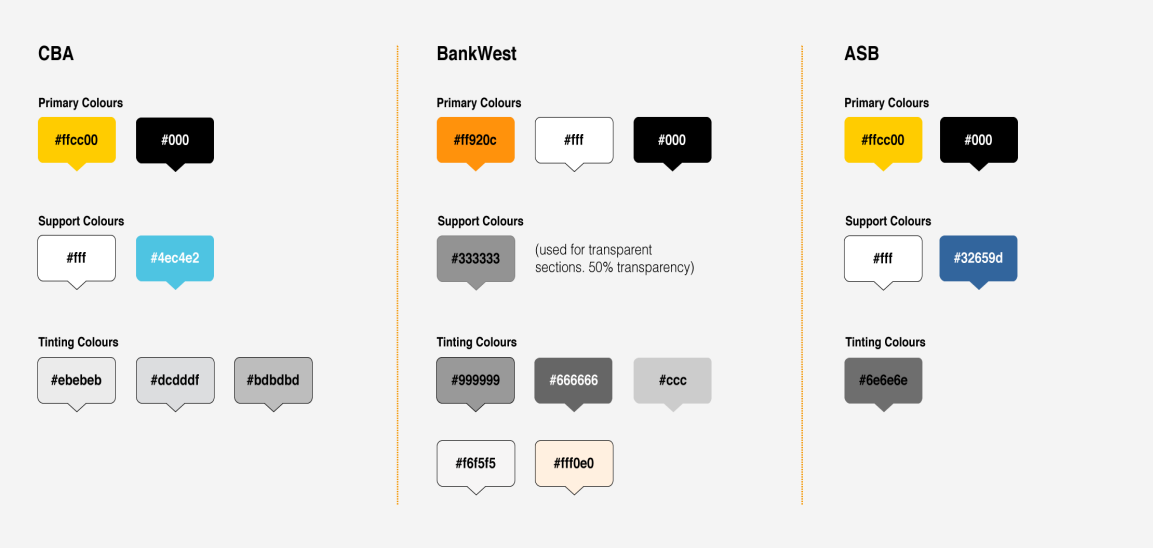
Primary colours are the main colours that define your brand; the ones that people associate your brand with. For instance, when you think of the Commonwealth Bank of Australia you think of yellow and black. Typically two primary colours are used when branding. Note that red and green may be indistinguishable to colour-blind users.

Support colours

These colours should not feature extensively throughout the app pages. The most appropriate use of support colours are for highlights or buttons/links. Also useful for a bright backgrounds which have important content that aims to grab the user's attention.

Tint colours

These colours should not be used for text due to issues with low contrast which affects accessibility. Useful for subtle messaging, sub-headings or backgrounds where subtle differentiation from white. Apply it when you do not want to take focus away from other elements but want something to be found if the user wants or needs it.



.colorPrimary1
.colorPrimary2
.colorSupport1
.colorSupport2
do these need to be listed out in detail?

Iconography

We are providing a basic suite of common icons for developers to use. For now these are supplied in black and white versions for placement on light or dark backgrounds and large and small for high and low pixel density displays

Download icons.zip from here

If other icons are needed they could be found at

1. istockphoto.com
2. thenounproject.com
3. symbolset.com - icon based fonts (our long term preferred method of dealing with icons)

Typography

User interfaces often benefit from contrast in type sizes to create clarity within a document. An example of this is the use of a large size font for headings and a smaller size font for body text. However, using too many different sizes can often create confusion. Consistency is very important and to help that we recommend that a theme contain no more than 4 different font sizes.

Create a hierarchy within grouped information
Highlight the most important text in a scenario by making it the largest, bold or a support colour, or a combination of all three. Equally to take focus away from a certain piece of text you can reduce its size slightly, use a lighter weight or use a tinting colour, being mindful of contrast in regards to accessibility.
The following typography scale can been applied:

Small	0.75em	12px
Medium	1em	16px
Large	1.5em	24px
Oversize	2em	32px

http://v1.jontangerine.com/silo/css/pixels-to-ems/
Accessibility
Default font size for browsers is 16px. If you specify smaller font sizes, it is best not to use values below about 75% or 80% of the original font size.

UI component structure

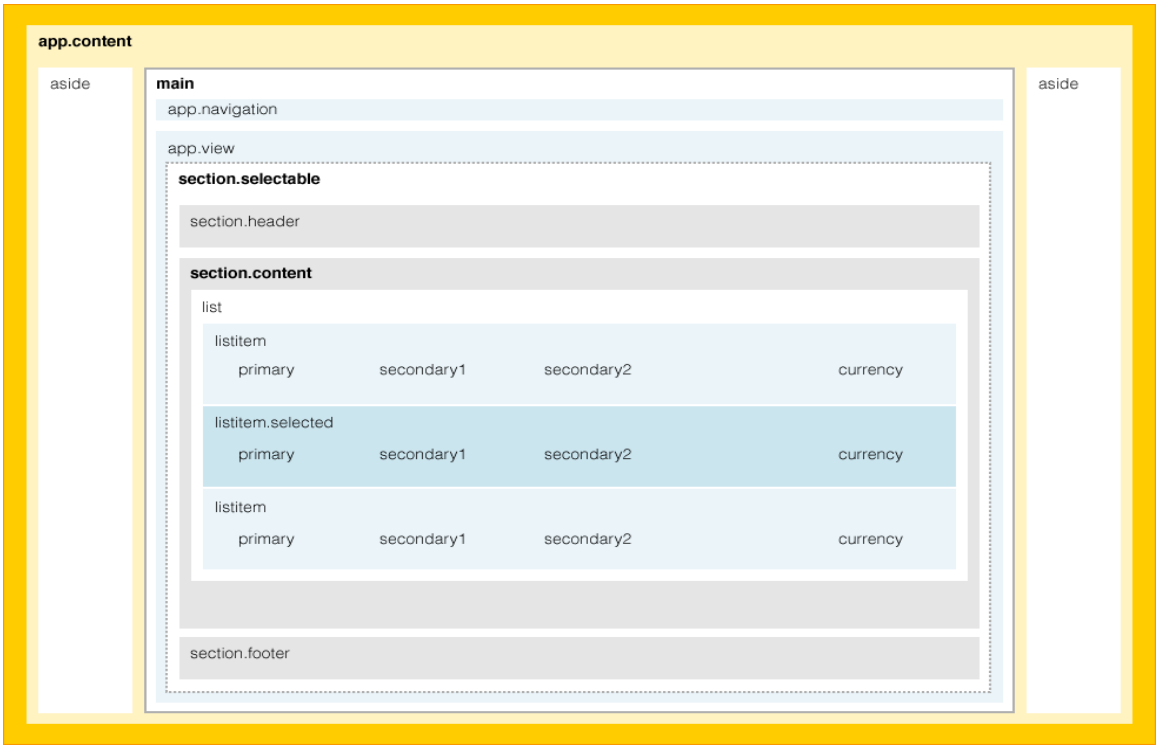
These are examples of different kinds of page layouts and what the css style rules are that are responsible for achieving the layout. To use the css classes defined, you just need to add class to your DIV with the name of the component you want. Make sure you take note of nesting of classes to get the right outcome.

Section.selectable
Section.collapsible
Section.filter
Section.form

Section.feature

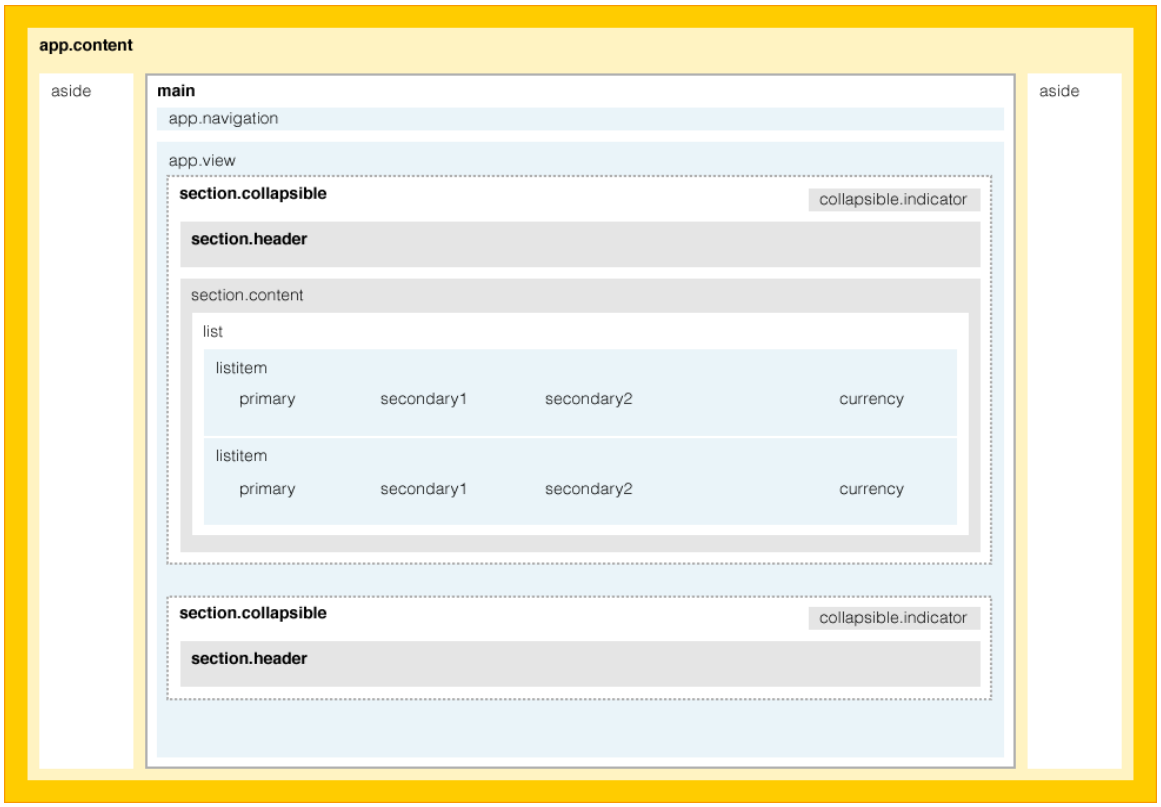
Section.selectable

Comprises of a list containing selectable listitems. It is used to display islands of content that have a call to action Eg in the All Accounts App Example



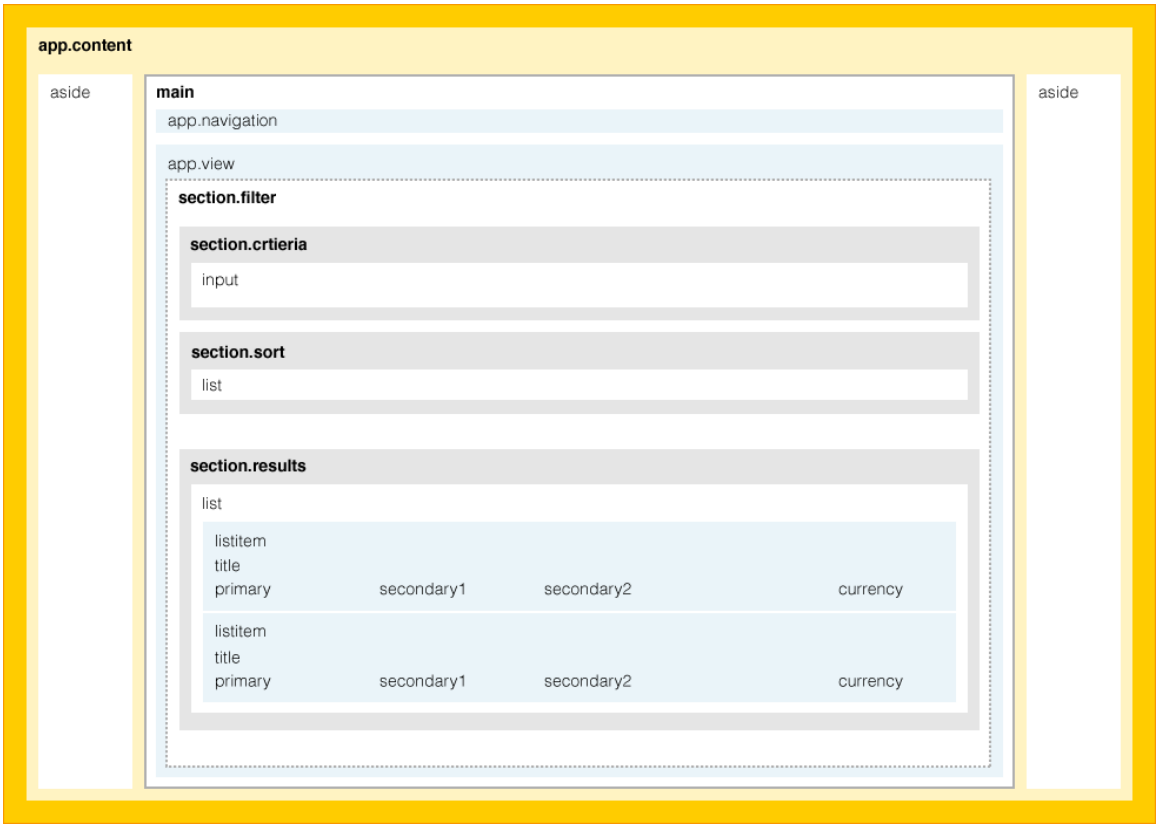
Section.collapsible

Comprises of collapsible sections which use section's header to hide/reveal the section's content within it. Handy for forms that have sub states or data that can be summarised to make the UI more readable. Its be used in the Transfer Money Example App



Section.filter

Filter section consist of criteria, sort and results which are displayed as part of filter search. This pattern is good for pages that list repeating content. You can see how this is marked up in the Transaction History App

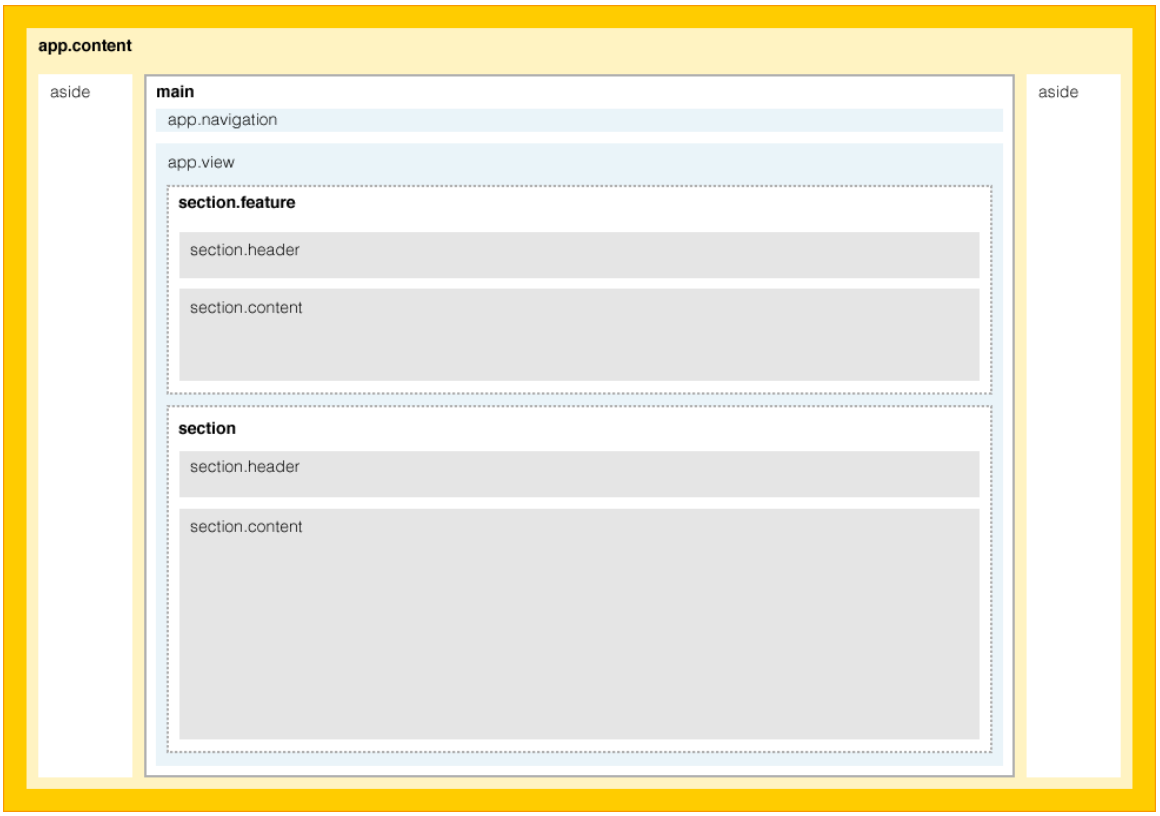


Section.form

Forms typically have a number of layout requirements to aid usability for data entry. These style classes consist of input (label, input fields, validation), validated error message, media upload, appendable input and actions. These can been seen being used in the Form Example

Section.feature

Feature section consist of header and content specific to the feature. Its been used in the Whats New App



Implementing themes

Using default styles provided by the container

The container which houses various applications has the capacity to pass on theming resources to its child applications. This is controlled through declarations made in an applications Manifest file.

It is best practice to declare all style resources in the Manifest.xml file. This should be done regardless of whether you are making use of container resources or not. Declaring your custom style resources in the manifest will ensure all style declarations are centralised keeping your app clean and well structured. In addition, it will ensure that the order in which styles declaration appear in your HTML files is always correct. i.e with your custom resources always being injected after any container resources. In future, it will also ensure that JavaScript files obey the same rules and work properly with your custom JavaScript files.

Declaring resources to use in your app

Applications can declaratively point to style sheets which will be transferred to the application at runtime. To make use of these style sheets you should declare in your applications Manifest.xml file which resources you support as follows:

Add the container element to the manifest as a child of the themes element:

```
<themes>
  <container></container>
</themes>
```

Then within the container element declare the resources your application would like to make use. The <supports> element is used to indicate the name of the container resource you wish to be applied to your application:

```
<themes>
  <container>
     <supports name="Look"/>
     <supports name="Typography"/>
     <supports name="Sections"/>
  </container>
</themes>
```

The container element is not mandatory and you can choose to omit this element all together should you not wish to make use of the container style resources.

Resources provided by the container

The container provides four types of resources.

Typography

Styles defined in *typography* will define everything related to the font in your application including elements such as heading, paragraphs, labels and links. If you use this style resource you can assume you will receive a fully styled set of typographic elements.

Look

Styles defined in l*ook* will define everything related to the colours that make up a theme. This resource provides you with a set of colours organised within categories of selectors including primary, secondary and tertiary colours.

Layout

Defines a number of structural elements for page layout. Common elements such as heading, footers and information sections are included.

Forms

The forms resource will completely theme any elements within a form. You can define a vertically oriented or horizontally oriented layout.

Precedence Rules

Adding custom styles to your application

While themes provide a broad set of styles for your app to utilise there may be times when you need more than what the default styles are giving you. If this is the case then you will need to add additional styles to supplement the theme by creating custom CSS style sheets in your application.

When doing so it is important to consider what selectors you introduce into the theming namespace. We recommend that you do not override the existing selectors, unless you really need to but instead create separate names for your selectors. This can easily be done by prefixing your classes with something such as custom-<yourselector> or myapp-<selector>. This will ensure no clashes with the existing container theme styles will occur and that when themes are updated you will be less likely to encounter strange layout issues.

To add custom style sheets to your application you must add the following to your applications manifest:

1. Declare a the custom tag in your manifest giving it the name of a “fallback” CSS file if the app is running in

a container theme your application does not support.

```
<custom fallback="basic">
```

2. Declare the theme you wish to support/extend and add additional styles to.

```
<theme name="pony">
```

3. Define the style sheet declarations you are adding. Note that the style sheets will be added to your page in the order that you declare them.

```
<link rel="stylesheet" href="~/Themes/pony/MyCustom.css"/>
```

A complete example may look like this:

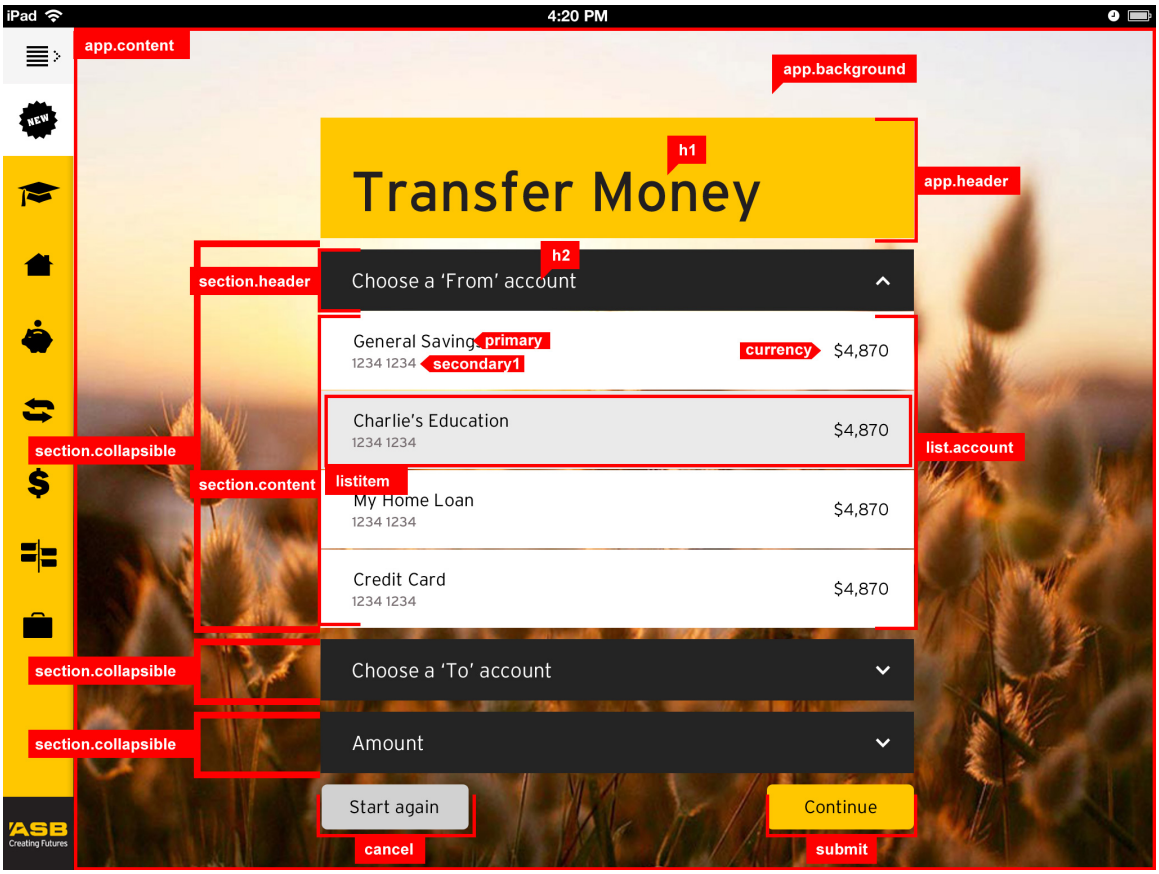
```
<themes>
  <container>
    <supports name="Typography"/>
  </container>
  <custom fallback="basic">
    <theme name="pony">
      <link rel="stylesheet" href="~/Themes/pony/AppSearch.css"/>
    </theme>
    <theme name="basic">
      <link rel="stylesheet" href="~/Themes/basic/Simple.css"/>
    </theme>
  </custom>
</themes>
```

Find information about how to extend a theme Here

Style & Theme Implementation Samples

Collapsible Section Example

"Collapsible Section" refers to the section which can be expanded and collapsed by the user as required. User can click on the "up" arrow to collapse the section & can click on "down" arrow to expand the section. Collapsing and expansion can also be achieved by clicking the sections instead of arrows.

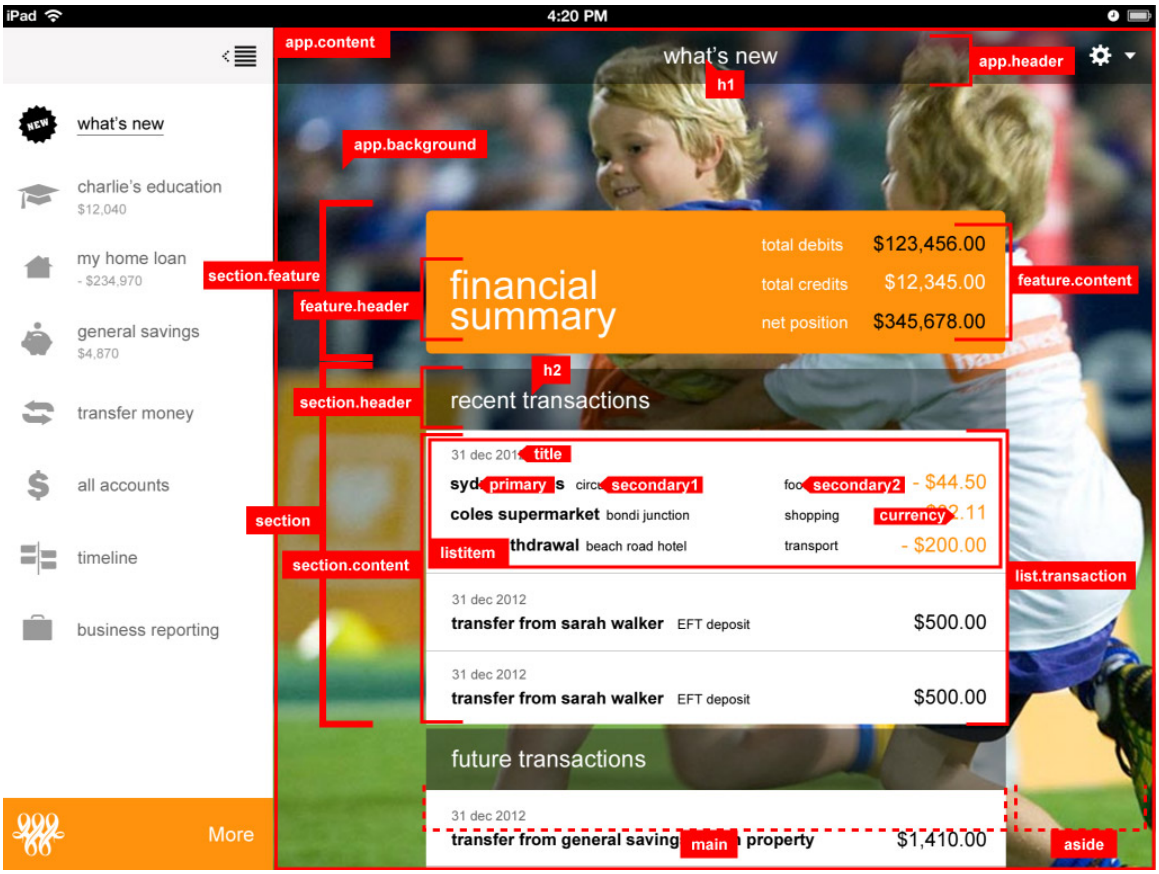


Collapsible Section – Annotation
All the core annotations are provided in "Selectable Content" section. Below section provides a view on annotations specific to collapsible section only.

Annotation Name	Annotation – Usage
Section.collapsible	Refers to collapsible section which can be collapsed and expanded by user
List.account	Shows account list for the user

Featured Content Example

"Featured Content" refers to the providing details on specific feature. For e.g. below screen provides details on financial summary feature.



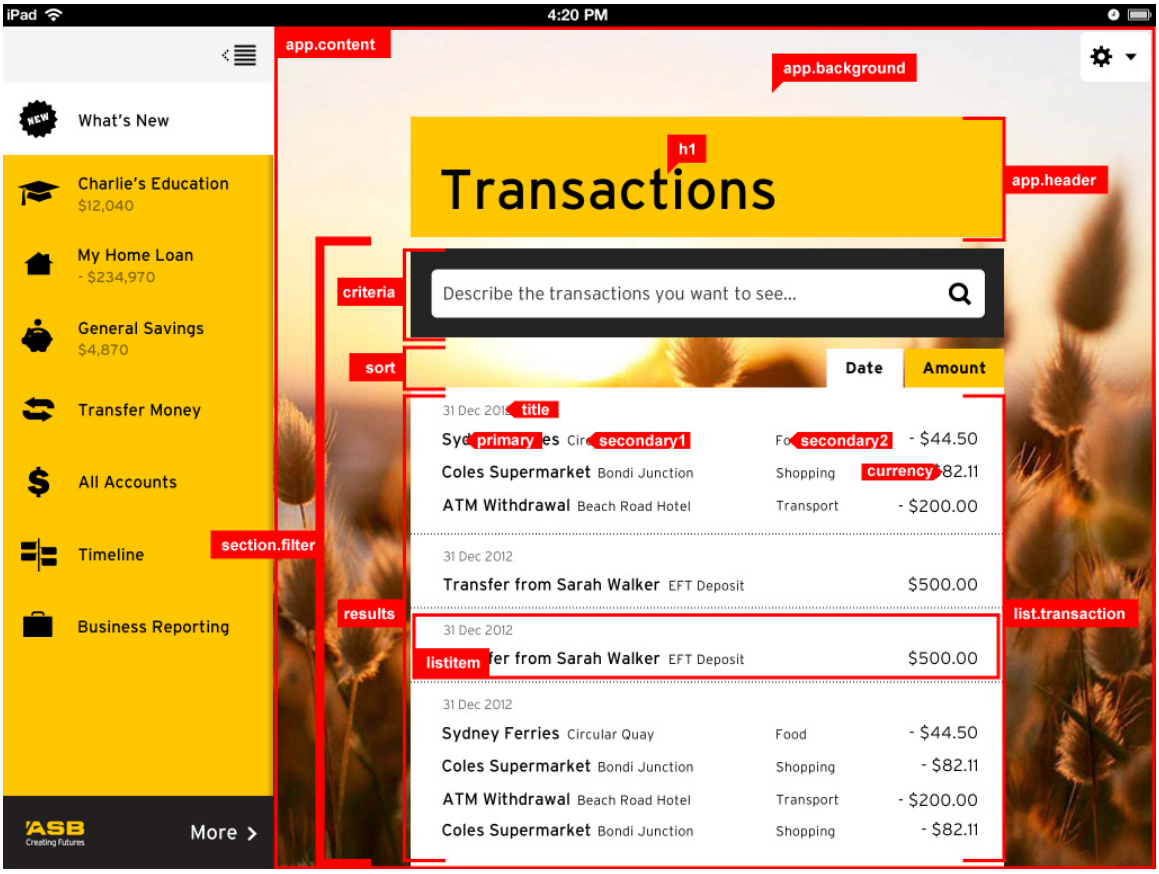
Featured Content – Annotations
All annotations specific to "Featured Content" are explained below. Rest annotations are similar to those explained in "Filter Content" section.

Annotation Name	Annotation – Usage
Section.feature	Refers to the section where any feature details are shown
Feature.header	Refers to the header of the feature within feature section
Feature.content	Refers to the feature content for specific feature

Filter Content Example

"Filter Content" refers to searching for a particular text based on some search criteria and then filtering the result set by using some sort criteria. For e.g. in below screen, user searches for all transactions and then sort the

result set on basis of "date" filter.



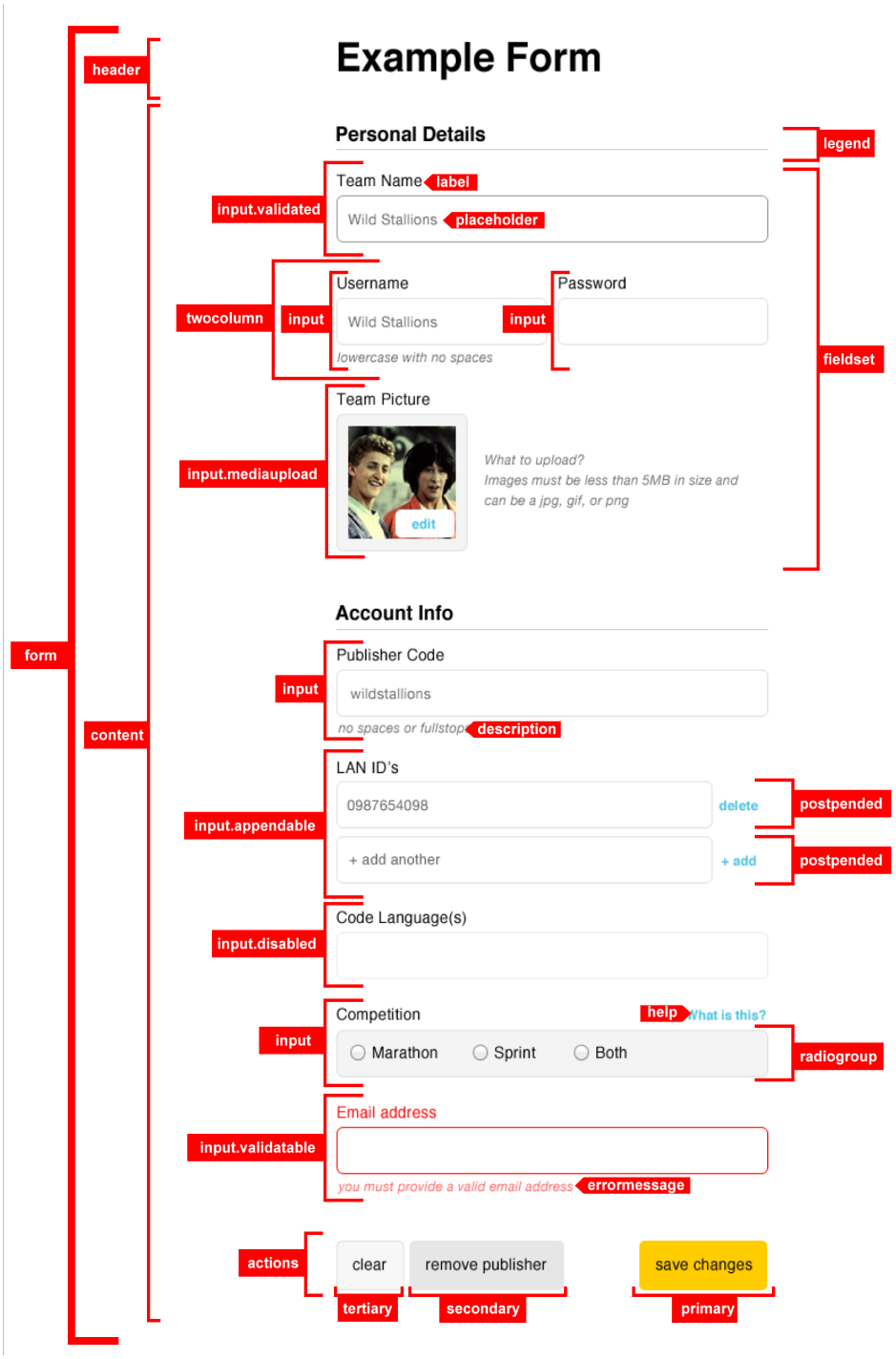
Filter Content – Annotation
Specific annotations related to "Filter Content" are explained below

Annotation Name	**Annotation – Usage**
Criteria	Refers to search criteria provided by user for search purpose
sort	Provides filters which can be used for sorting purpose
title	Shows the title of the transaction. For e.g. date of transaction
primary	Applies to description of the transaction
Secondary1	Applies to the secondary information for a particular transaction. For e.g. location where the transaction occurred.
Secondary2	Applies to the transaction tag
Section.filter	Filter section which shows required information for filtering results

results	Refers to result set which is displayed as part of search
list.transaction	Shows the transaction list consisting of transactions

Form Example

Form refers to a structure where user enters required information in order to carry out some action. Below example shows a sample form for capturing details on coding competition registration.



Form – Annotations

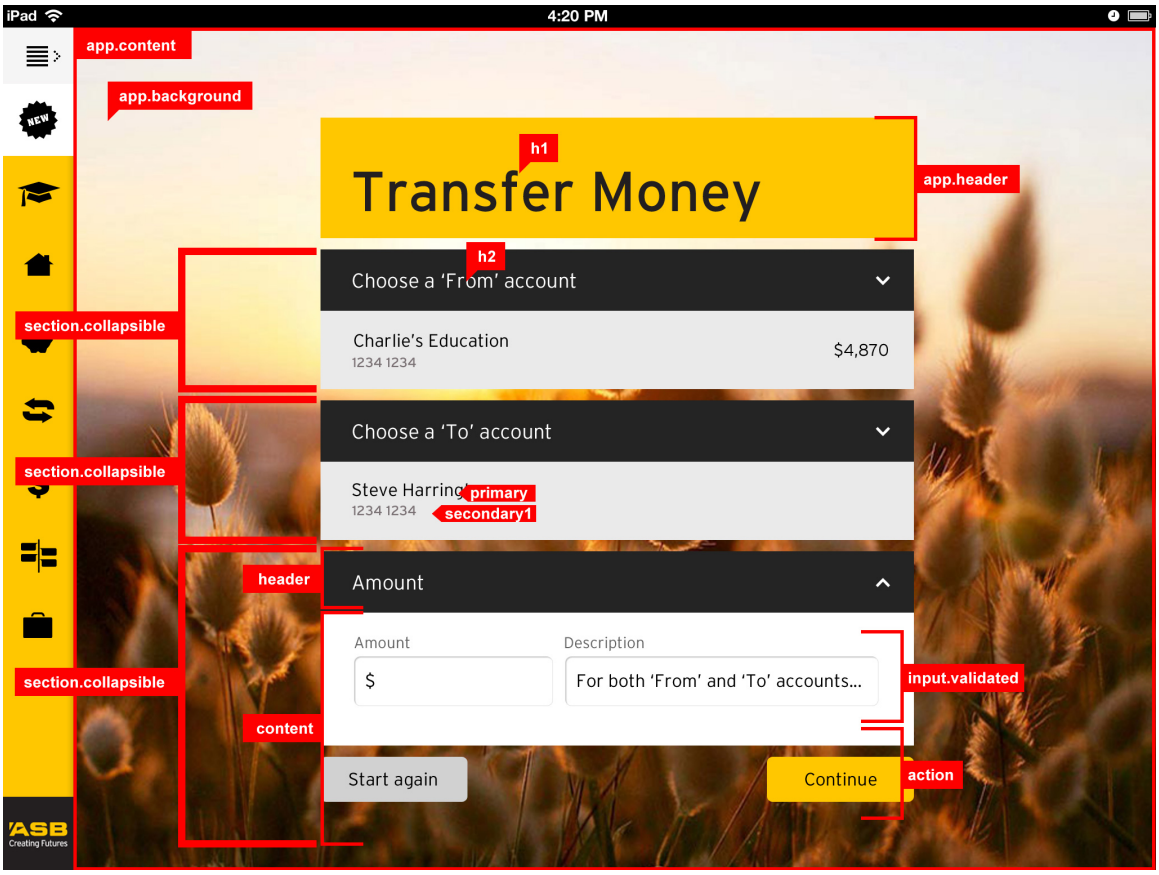
All the annotations specific to a form are explained below

Annotation Name	Annotation – Usage
header	Header of the form. Consist of title for the form
legend	A legend shown with a title and a underline
label	Label name placed in content
Input.validated	Input that need to be validated on completion of any action
placeholder	Refers to the text location within input field
twocolumn	Shows two columns consisting of fields
input	Refers to the input field
fieldset	Refers to a set of fields that can be grouped together
Input.mediaupload	Any media that can be uploaded
form	Complete form structure
content	Content within form structure
description	Descriptive text for a particular input field.
Input.appendable	A control that allows addition of more value-fields
postpended	Particular control that takes an action (for e.g. delete, add)
Input.disabled	Field value which is not allowed to be entered.
help	Tool tip providing help text around the particular section
radiogroup	A group of radio buttons representing different values
Input.validatable	Any field which fails validation will be shown in this format
errormessage	Error message in red shown for validation error
actions	Area where all action items can be populated
tertiary	Third level of action items can be shown in this format
secondary	Second level of action items can be shown in this format

primary	All main actions can be shown in this format

Input & Action Example

"Input & Action" refers to the input text field which need to be validated on a particular action. Amount & Description fields are good examples of validated field in below screen. In addition, it also refers to the particular action that is taken by the user by clicking a specific control. Transfer now button is good example of action in below screen.



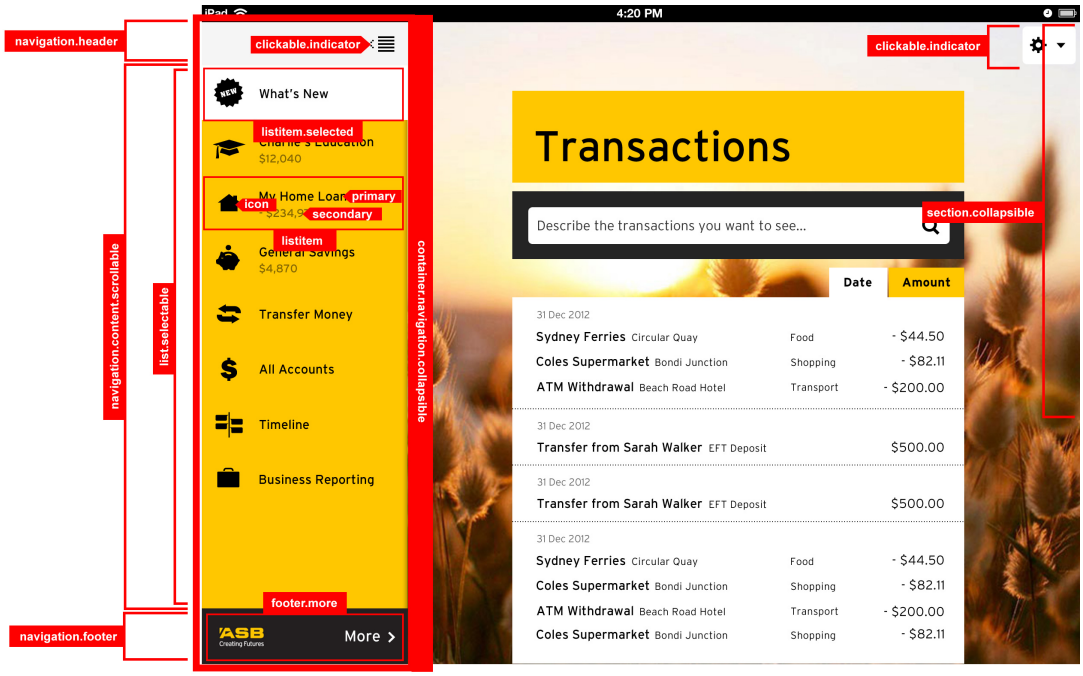
Input & Action – Annotation

Annotations specific to input fields are shown below. Core annotations are explained in "Selectable Content" example

Annotation Name	Annotation – Usage
Header	Refers to header portion of a collapsible section
Input.validated	Refers to input fields in which user can provide input
Content	Refers to content portion of a collapsible section
action	Shows the action control which can be used by user to take a particular action

Scrollable Content Example

"Scrollable Content" refers to the scrollable section of the navigation list. User can look at list of menu items in left side panel by scrolling up and down.



Scrollable Content – Annotation

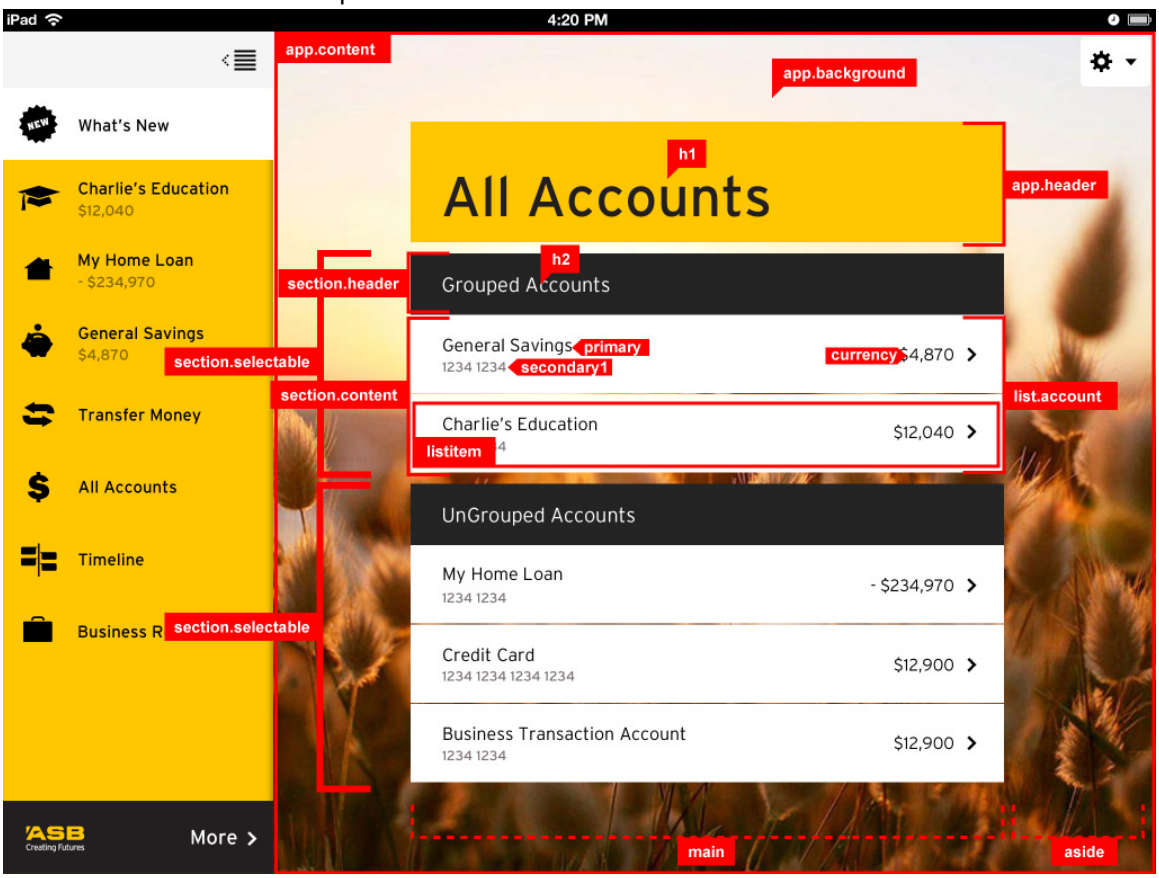
Core annotations and their usage are mentioned in "Selectable Content" example. In addition, there are some more annotations that are specific to "Scrollable Content" which are as below

Annotation Name	Annotation – Usage
navigation.header	Displays header which includes navigation components
clickable.indicator	Indicates settings for selecting themes
listitem.selected	Selected list item highlighted in different colour within a list
primary	Title of the list item in side panel
icon	Icon representing the list item in list
secondary	Secondary annotation refers to amount displayed in dollars for each list item within a list
navigation.content-scrollable	Scrollable list of navigation items
list.selectable	Shows a list of items which can be selected by user. On selection, an action is carried out by system
navigation.footer	Footer of navigation screen

footer.more	Clicking "more>" provides more information about footer

Selectable Content Example

"Selectable Content" refers to the selection action that a user can do for a list item within list. For e.g. user can select "General Savings" account by clicking on the account record. The arrow shown at the end of account record tells the user that the particular section is selectable.



Selectable Content - Annotation

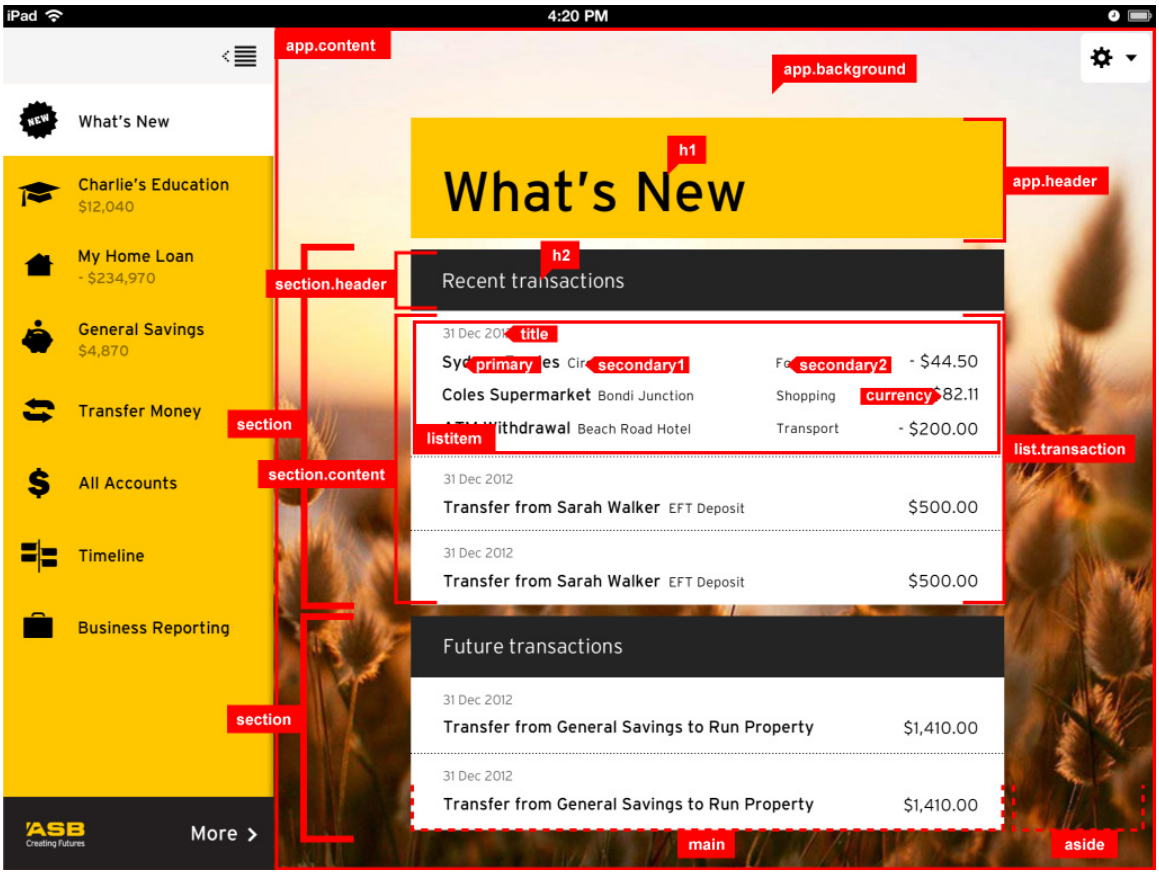
Annotations specific to "Selectable Content" are described below

Annotation Name	Annotation Usage
app.content	Application content space where all the data in specific controls is displayed for viewing purpose
app.background	Application background view. Changes on basis of themes selected
h1	Heading 1. Used to show the title for the screen within application header
app.header	Header section of the screen. This is used for displaying the header and title of the screen.

h2	Heading 2. Used to show the title of each section
section.header	Shows the section header where title of the particular section is displayed
section.selectable	Selectable portion of the section which can consist of any kind of action For e.g. The user can click on account within account list and go to account details
primary	Title for each list item within section content
currency	Reflects currency in '$X' against a particular account within account list
secondary1	This pattern is used to show BSB + Account number for an account within account list
section.content	Refers to content displayed within a particular section of the screen
list	Shows list of accounts within a particular section
listitem	Refers to each individual item within the list displayed in section content
main	Main section where all required data / information is displayed
aside	Section other than main where no data is displayed

Static List Example

"Static List" refers to the display of transactions in a list with related information about each of the transaction.



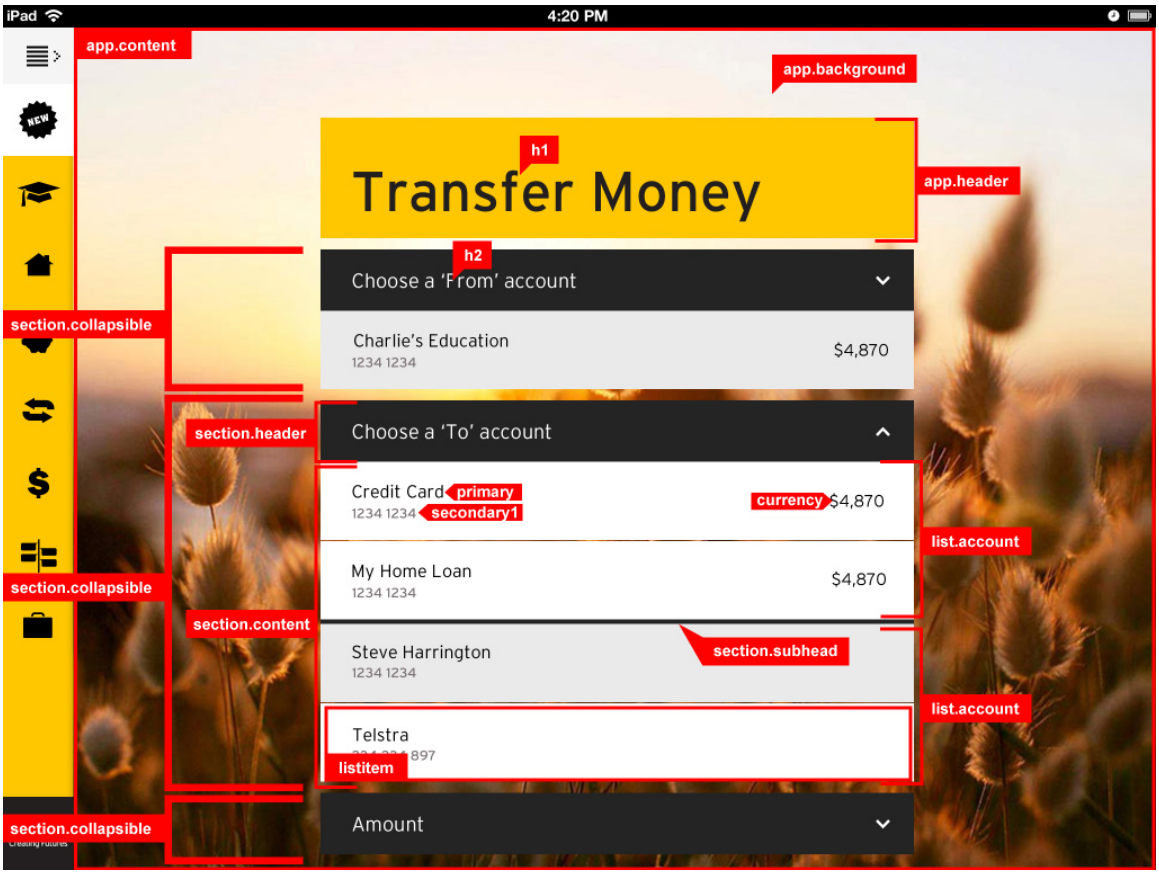
Static List – Annotation

This section focuses on showing annotation usage specific to the "Static List" annotation. Core annotations are explained in "Selectable Content" section.

Annotation Name	Annotation – Usage
Title	Shows the title of the transaction. For e.g. date of transaction
List.transaction	Shows a static list of transactions

Subdividing List Example

"Subdividing List" refers to the sub heading type of divider. Couple of items in list are shown in one category followed by another one separated by a subhead divider.



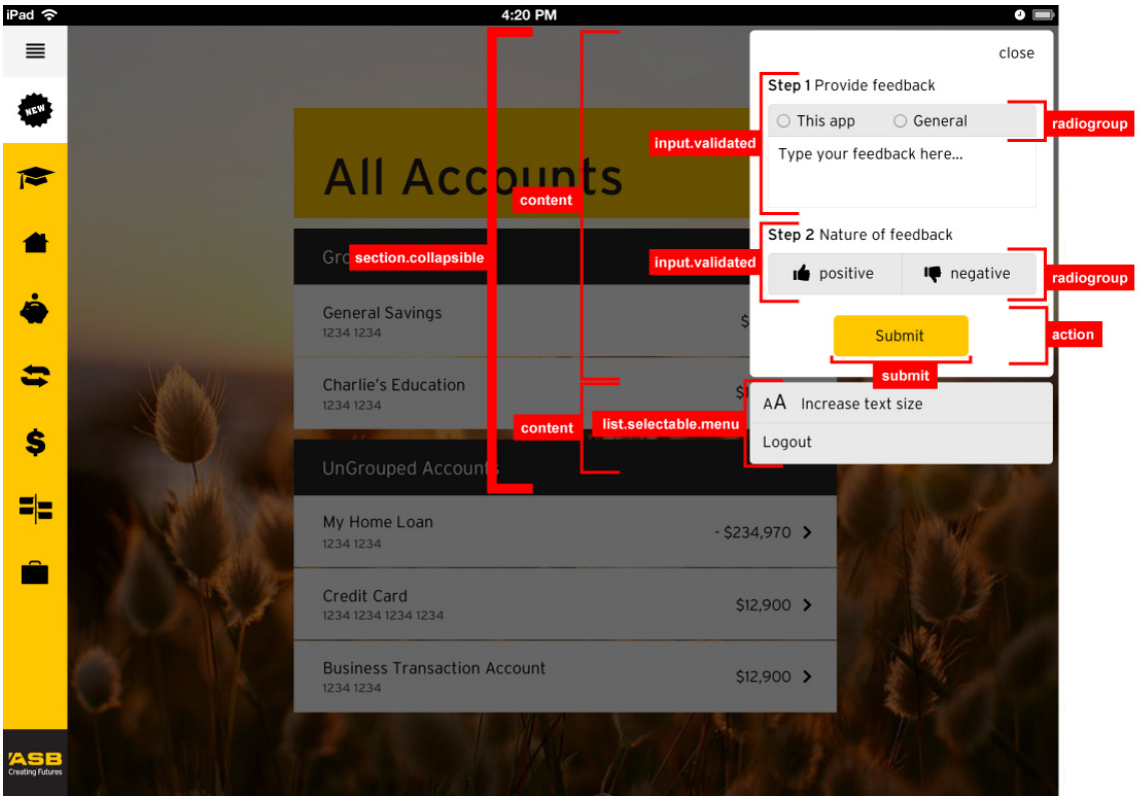
Subdividing List – Annotation

"Collapsible Section" & "Selectable Content" provides a view on core annotations. This section shows annotations specific to "Subdividing List".

Annotation Name	Annotation – Usage
Section.content	Shows contents within a section
Section.subhead	Shows a thick line divider between various categories of accounts. For e.g. all of your own accounts will be shown followed by your external accounts from other banks separated by a thick line divider.

UI Control (Radio) Example

UI Radio controls refers to grouping couple of radio buttons and showing them as a group on the screen.



UI Radio Controls – Annotations

Annotations specific to Radiogroup control in relation to above screen are shown below. Rest annotations are explained in "Input & Action" & "Collapsible Section" sections.

Annotation Name	Annotation – Usage
radiogroup	Refers to group of radio buttons available for user selection
Content	Refers to the actual content which is visible to the user
action	Refers to area where user can take any action
List.selectable.menu	Refers to menu items which displays various functionalities for user

Step by Step theming

This page will guide you through a step by step process you can follow to completely theme your app using concepts discussed in the Doing it with style section. The example used as a reference for this tutorial is the HellOWorld App.

Step 1: Base mark-up

To make use of theming each app you build requires some boiler plate mark-up to get you going. The following code snippet shows the minimum set of elements you should put on the page to get the initial layout working. This mark-up corresponds to the diagram shown in Visual structure of a UI.

Boiler Plate Mark-Up

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Title here</title>
</head>
<body>
    <div class="background"></div>
    <div class="app">
  <div class="header">
   <h1>Some header</h1>
        </div>

  <div class="main">
   <div class="section">
    ...
   </div>
  </div>
 </div>
</body>
</html>
```

Step 2: Declare you want to use theming

You now need to edit your applications manifest file to enable theme injection. Insert the following XML into your Manifest.xml. You can read a detailed description about what this means in the Implementing themes section.

Declaring Container Theme Resources

```
<themes>
  <container>
     <supports name="Look"/>
     <supports name="Typography"/>
     <supports name="Sections"/>
  </container>
</themes>
```

Step 3: Build your application

You should now add your applications components including the HTML and Javascript you require for your app to work. Make sure to structure your app with the correct mark-up described by the diagrams in the UI Component Structure section. For the HelloWorld App you would want to add a heading and a content section.

Step 4: Do you require custom styles?

If the styles injected with theming do not cover all your visual needs then you should add some custom CSS to your app.

1. Create a new CSS file with whatever name you wish to give it. Do not directly link to this CSS file in your aspx file. Instead you should declare this stylesheet in your applications Manifest.xml.
2. Add the following to your manifest and provide the relative path to your custom CSS file. You can add as many custom stylesheets as you require. You can read a detailed description about what this means in the Implementing themes section.

Declaring Custom Style Resources

```
<themes>
   <container>
      ...
   </container>
 <custom fallback="custom">
     <theme name="custom">
        <link rel="stylesheet" href="~/path/to/custom.css"/>
     </theme>
    </custom>
</themes>
```

Guiding principles

1. Isolation between Apps is the goal. Undeclared and ungoverned coupling between Apps will eliminate the ability to deploy and manage Apps independently.
2. Reusable programming components is not the goal.
3. The architecture allows for implementation flexibility: The integration between components is done using web standards, so any technology can be used to build any part of the ecosystem. The right technology will be the most efficient and cost effective overall. By allowing individual Apps to use different technologies, we can experiment with new options until finding a winner without having to build out large parts of the ecosystem or being unable to upgrade from a suboptimal technology.
4. The business model for the bank is to standardise on a few key technologies to allow greater long term support of assets. This will naturally temper the amount of technology variance over time.

FAQs, common issues & errors

Running

Using Internet Explorer...

Although Apps and Containers can run in a variety of browsers, the Developer Container currently only supports Chrome, Firefox and Safari. Internet Explorer < 10 do not support CSS 3 which the developer container makes use of. You can still run your App in IE by just loading it's page URL directly in the browser.

Theme is lost when navigating to a new page..

If a form navigates away to a different page, the theme is lost. There is a Javascript file in v1.6 of the framework samples that shows how to fix this problem. It is due Sheldon class Apps not expecting to navigate away from the current page.

Http 404.3 errors with IIS7.5 and .woff files

This happens because IIS7.5 needs the MIME type association set for this font file. See

http://www.localwisdom.com/blog/2011/10/iis-7-web-config-change-for-html5-and-css3-mime-types/

The MIME type association can be done in IIS as well (if you don't want to include a web.config file) - see below

http://www.iis.net/ConfigReference/system.webServer/staticContent/mimeMap
where filename extension is set to woff and MIME type to font/x-woff

Debugging

Using incorrectly cached javascript files...

One of the most common problems developers run into is not being able to see the latest version of the javascript file when running in The Developer Container. Sometimes browsers use cached versions of files for longer than they should. This happens more often because we use iFrames. To ensure that you are using the latest version try one of the following:

1. Right click on the screen over the App, hold the Control key and select '*Refresh*' - this forces the iFrame to refresh
2. Refresh the page by holding down the Control key when you refresh the browser (F5).
3. If this still doesn't work, clear your browsing history before trying again.

Nothing is displayed where the App should be...

To work out what is wrong when nothing is displayed, use the Browser's debugger or extension eg Firebug Firefox extension. In Chrome, the keyboard shortcut is *Ctrl + Shift + I*

There is an Error Console that will display the output of any javascript or page load error. The other tab to check in the debugger is the Network tab, it shows the HTTP response codes and error responses for any missing requests.

Incorrect Style being used...

The browser's debugger will make this easy to trace the cause of style problems. In Chrome, open the debugger with Ctrl + Shift + I and click the magnifying glass icon in the lower left corner. Now use your mouse to select and click on the HTML element that has the wrong style. On the right side of the debugger will display all style rules that are in effect on that element and which .css file they are defined in.

Dealing with data

One of the freedoms and complexities of The Competition Brief is that you can use any technology.To keep the playing field level we've not included the Helix App framework's JSON service components so as not to favour .net developers.

We've provided a number of file based ways of consuming the Coding Comp Sample Dataset that will work on your machine at home.

Additionally, if your App requires you to create your own server side services and data sources, you are free to use any pattern and technology to connect to these. Ensure that you include code for all tiers of your App.

API Documentation

The relationship between Apps and Containers requires a few API contracts to be met. These are specified below.

App API	Container	Environment.-	Manifest.xml	Navigation

Classes of Apps

Rationale

There are 3 classes of Apps depending on the technology you want to use to build it with and how aware of the Helix framework your App is. Each of the classes has a different level of understanding of the framework and a different way of implementing the APIs. Its a way of describing Apps based on their capability instead of their technology.

Understanding this is important if you want to build a multi page App and want it to respond to changes in Themes.

API Access

The major APIs that define the class of App are based how many of the Container features they are aware of:

1. **Participating in session management**
    a. Distributed session kill
    b. Keep alive
    c. Authentication before calling container services
2. **1 way messaging**
    a. Accessing the environment config
    b. Authentication
    c. Theme injection
3. **2 way messaging**
    a. Container services eg
        i. App Search
        ii. Device capabilities
    b. Setting Context
    c. Invalidation events
    d. Navigation
        i. Modal
        ii. To another App
    e. Instrumentation
    f. Security
        i. Step up
        ii. reauthenticate

These API's are either taken care of by framework code or dealt with by the App developer.

Definition

Sheldon	*Scarlett*	*Schultz*

		
• The smartest App • Is given rich tools to help program effectively • Thinks he's smarter than you (so needs some controls) • Has access to full gamut of APIs • Primary focus: speed and quality • Is hosted by the app store	• **Frankly, we don't give a damn** how this is coded • Lives in the sandbox of APIs – less rich access to bank assets • Can use any technology to build anything she wants • Primary focus: low cost and quality • May or may not be hosted by the app store • Primary focus: speed and quality	• It Knows nothing • Is an app that doesn't know it's participating in the ecosystem • Can be wrapped relatively simply • Doesn't use APIs, but a shim is supplied to ensure a minimum level of compliance • Primary focus: low cost and speed • Probably not hosted by the app store
It is a single web page that uses DOM manipulation to show different content.	It could be a single page or multiple pages using form posting or hyperlinks to show each view. eg (JSP, ASP, ASP.Net, ASPNet MVC, PHP, HTML) It is aware of all	It is either a single page or, more likely, a multi page site. It may have a single page wrapper iFrame around a multi page site. Third party web apps typically fit into this class.
It is aware of all API's	It may be aware of all or most of the API's. It can use 2 way messaging.	It is not aware of APIs that relies on 2 way messaging or involve session management.
Can receive 1 way messaging events based via framework javascript. Always able to receive events because it never leaves the page.	Can receive 1 way messaging events based via framework javascript if on the same page. Needs to wire up subsequent pages to OpenAjax bus to receive events manually.	Can receive 1 way messaging events based via framework javascript. App developer needs to implement

Can send 2 way messaging events based using framework javascript.	Needs to request a token on subsequent pages to participate in 2 way messaging. The container needs to manage the security boundary by enforcing that only declared pages can still communicate with the container.	Can not participate in 2 way messaging as it may breach the security boundary

App API

Overview

Terminology

- *App API* - the runtime Application API currently contained in AppStore.App.js
- *App program* - the business application that is using the App API to be a part of the Helix ecosystem.
- *Container API* - the runtime API for the container, currently contained in AppStore.Container.js.
- *Container program* - the application that utilises the Container API to host app programs.
- *App manifest* - the XML file provided by the app progrhe am that describes its properties. This is used by the container to know how to deal with the app program. Also refers to the JSON version used by the JavaScript APIs at runtime.

App Manifest

Every app has a *manifest* that contains information about the app. Manifests are used by containers to determine the app's capabilities, requirements and dependencies. The manifest is discussed in more detail on the Manifest.xml page.

App API

The App API is exposed through a single object which can be referenced by window.AppStore.App. The app program requires 3 includes:

Core App include files

```
~/Scripts/pagebus/full/pagebus.js
~/Scripts/AppStore.Common.js
~/Scripts/AppStore.App.js
```

The PageBus file must be included before the other files, and if jQuery is used it should be before AppStore.App.js

Apps that wish to make use of theming also need to include AppStore.Theme.js.

There is also a jQuery Ajax extension that passes some additional environment details as out of band data, and utilises HTTP POST rather than GET. This extension is in AppStore.Ajax.js.

> **Coding Comp Note**
>
> For the CTO coding comp sample apps the AppStore Common, App and Theme files are combined into a single file called Framework.js.

Initialisation

Apps are loaded into iframes controlled by the container. The initialisation process is managed by the Container and App APIs and is asynchronous.

If the app program uses jQuery then no explicit initialisation is necessary so long as the jQuery library is loaded before AppStore.App.js.

The window.AppStore.App object will delay the firing of the jQuery.ready event until the App API has been initialised and all relevant context data has been received from the container.

For app programs that don't use jQuery the App API ready() method should be called, passing in a callback method that will be called when initialisation has completed. Currently the callback function has no parameters.

Notifying the container that the app has loaded

The App API will automatically notify the container that the app has loaded as soon as the initial handshaking and data transfer has taken place. For containers that display a "Loading..." message or icon (aka spinner) this notification occurs by sending the container a specific message - AppStore.Constants.TOPIC.FROM_APP.APP_READY - which the container relies on to know when to fully display the app and remove the "Waiting..." overlay. If an app can take some time to render, for example when it calls some JSON services after loading, the user experience may be better if the wait message is displayed until after the app has completed its initial rendering. An app can stop the automatic APP_READY message by calling AppStore.App.holdReadyMessage() as early as possible (ie NOT in a jQuery.ready handler), and then calling AppStore.App.sendReadyMessage() when rendering is completed.

Properties

AppStore.App.user

This object holds information about the current user. This field is for information only, authentication and authorisation do NOT use this object.

AppStore.App.context

This object holds the application context passed from the container. There are some *well-known* contexts such as app, user, customer and account, but app publishers are free to define their own contexts.

The AppStore.Context object is a container object that can hold one or more AppStore.ContextElement objects. A ContextElement may represent some entity such as a customer, an account, or even an app.

A Context object has a *signature* that is constructed from the types of ContextElements it contains (but not the data). This is to make it easier to check whether a Context object contains the right combination of elements for a given purpose.

> **Coding Comp info**
>
> The HelloWorld3 sample demonstrates how to read the initial context and how to respond to context changes by binding to the context changed event.

AppStore.App.bindings

This object holds the resolved bindings for the endpoints specified in the app program's manifest. Apps don't generally need to access this directly, it is used by the App API to determine which dependencies have been resolved in the current container and environment.

AppStore.App.params

This object holds any arbitrary parameters that have been passed from the container API, which may have originated in another app program. Parameters are passed as data-only JavaScript objects, and can have any arbitrary structure, so long as the calling app and the receiving app both understand the structure.

A word about navigation

From the point of view of the container there is a major difference between an app navigating to other pages that are within the same app and an app requesting that the container launch another different app. The methods canNavigate() and navigate() (described below) refer to navigation to a *different* app. Apps must declare any other apps they intend to launch in the *invokes* section of their manifest.

Conversely, an app consisting of multiple pages does not need to declare all its pages, it only needs to declare its entry point. However all navigation to pages within an app, including form posts, needs to preserve certain query string values which are used by OpenAjax to connect with the container after each page loads (this connection is handled by the App API and is transparent to the app developer). If a page doesn't connect with the container after it has loaded the container will *delete the entire iframe* in order to prevent frame phishing attacks.

There are two methods provided to help preserving the query string values, AppStore.App.checkQueryString() and $().fixAppURL(), described below.

Methods

string checkQueryString(url), $().fixAppURL()

In order for apps to communicate with the container they need some data that is passed via query string. The container is aware when the contents of the app iframe change, whether this is because of a navigation event (to another page within the same app) or a round trip to the server caused by a form post. It's vital that any pages in the app retain this connection data in their query strings, so any href attributes for <a> links or action attributes for <form> elements need to be corrected at runtime before any navigation.

If your app uses jQuery, $().fixAppURL() is a simple way to fix up any <a> tags or forms so that they have the correct query strings. A typical usage would look like

```
$(function() {
 $("form, a").fixAppURL();
});
```

fixAppURL() also fixes forms that use GET rather than POST, by inserting the relevant values as hidden input fields.

If your app is not using jQuery, or you need to construct URLs dynamically in code, checkQueryString(url) takes any url string and adjusts or adds the query strings values and returns the final url.

```
var newURL = AppStore.App.checkQueryString("Food.aspx?c=" + escape(colour) + "&f=" +
escape(flavour));
```

bool canNavigate(endPointId)

Apps declare their dependencies on other apps in the *invokes* section of their manifest.

canNavigate returns true if the container has resolved at least one binding for the given endPointId. Useful for showing/hiding links to functionality based on whether the container contains the appropriate app programs.

void navigate(endPointId, launchMode, flags)

Sends a request to the container API to launch the app that is bound to the given endPointId.

The parameters launchMode and flags are not currently used, however in a future release the usage may be

- **launchMode**: Could be used to give the container more specific details about how to handle the launch. For example the launched app might need to be modal, or the calling app might want the new app to replace it visually i.e. display in the same position on the screen, or to appear in a specific position so as to appear to be a part of the original app.
- **flags**: Could be used to describe more behaviour, such as to give the new app a fresh (i.e. blank) context, or to prevent context updates from being inherited.

This example shows how an app might only display links to other apps that are resolved in the current container:

canNavigate() and navigate() sample

```
<!-- html -->
<a href="#" title="App 1" data-endpoint="App1" data-colour="blue"
data-flavour="chocolate" class="appLink hidden">Go to App 1</a>

<a href="#" title="Another app" data-endpoint="SomethingElse" data-colour="amber"
data-flavour="pilsener" class="appLink hidden">Another app</a>



// javascript

$(function() {
 // show valid links, hook up app navigation to links
 $(".appLink").click(
  function() {
   AppStore.App.navigate($(this).data("endpoint"), { colour: $(this).data("colour"),
flavour:           $(this).data("flavour") });


   return false;
  }
 ).toggleClass(
  function(i, c, u) {
   return AppStore.App.canNavigate($(this).data("endpoint")) ? "hidden" :
'canNavigate';
  }
 );
});
```

void close(returnValue)

Tells the container to close the app. The returnValue (any JavaScript object) is passed back to the container.

void setContext(newContext)

Notifies the container that the context for this app has changed. A common use case is where the user has selected a different account. Note that it is up to the container to determine what to do with the new context. Some of the things a container might do on receiving a context update are:

- pass the new context to other applications that are sharing context between them
- update its navigation menus to reflect options that are relevant to the new context (e.g. account, customer)
- launch an app that is configured to handle the new context. The app may be configured by the container or it the container may even allow it to be set as a user preference.

Events

App programs can handle events by binding callback functions using a similar syntax to jQuery

```
bind("event_name", callbackFunction, callbackScope, callbackData)
```

Event names are specified using pseudo-constants in ApStore.App.EVENTS e.g. *AppStore.App.EVENTS.ON_CONTEXT_CHANGE*.

callbackFunction is the function to be called when the event is received

callbackScope is the scope that the callback function will be called in i.e. the "this"

callbackData is an optional data object that will be passed to the callback function

ON_CONTEXT_CHANGE

Raised when the App API has received notification from the container that some part of the context has changed, for example as when another app has updated the customer profile, or a transaction has changed an account balance.

Parameters:

- newContext - the new context
- prevContext - the previous context i.e. prior to the notification, allows the callback function to tell what has changed.

ON_CONTAINER_ERROR

This message is sent by the container when an error has occurred while processing a container service request. The payload consists of an AppStore.Error object (located in AppStore.Common.js).

A number of well known errors are defined in AppStore.Constants.ERROR.

ON_APP_SEARCH_RESULTS

This message needs to be handled by apps that call the AppStore.App.appSearch() method to retrieve information about other apps. Note that this must be declared as an invoked container service in the app's manifest and the container must grant access to the calling app in its own manifest.

This API is generally limited to apps that are used to manage the AppStore ecosystem, it is not intended for general consumption.

ON_LAUNCH_DATA_RECEIVED

This is a special case, this is an alias for the onInitComplete function that is passed as an argument to the ready() function. This event occurs when intialisation is complete and the App API has received the inititial data from the Container API, including user, context, bindings and parameters.

If your app uses jQuery there is no need to handle this event or call the ready() method, just use the normal jQuery document ready function.

Parameters:

None.

Container

Overview

The Container API is a helper class that wraps the common functions required by an HTML AppStore container. It is part of framework.js

Container responsibilities

At runtime the container is responsible for:

- Initialising the container application, including reading configuration
- Logon
- Initial display of the top level window
- Management of screen real estate e.g. determining where to display an app
- Management of app context
- Enforcing declared dependencies e.g. not loading apps with unresolved dependencies

It does this by using the Manifest declared in each App.

Navigation, context and parameters

One of the key design principals of project Pony is to reduce coupling of components, and to manage any dependencies through explicit declaration and policy enforcement. Apps do not navigate directly to other apps, but they may request that the container navigate to an endpoint , which is resolved by the container at runtime.

There are two variations on navigation, context-based and point-to-point. In context-based navigation an app may set some context, such as a particular customer or account, and the container may then launch a predefined app or simply change the displayed navigation menus to reflect the possible actions so as to let the user decide.

It's possible that an app may wish to navigate to a specific app, or to one out of a list of apps. This may be because the calling app relies on the specific details of the called app, such as side-effects (e.g. a new customer record is created that has an associated NetBank account). The calling app developer may want to specify a number of possible apps to let the container choose the most appropriate app based on form factor or branding, or because any particular container might not have a given app so declaring multiple possibilities increases the chance that the app can run successfully in .

Data services

The container provides a few data services that App's can consume: eg AppSearch and Publisher and Host search. These allow App's to discover other elements in the Environments and deployment universe. For coding comp, these have been disabled to allow the container to work on your home computer.

Environment.xml

This file declares the Apps and Containers that exist in an environment. Environments and deployment universe describes the conceptual function of this file.

The nodes of interest are:

Manifest.xml

The Manifest is the heart of how Apps are managed and released independently of other assets. It also allows an App to be built using any technology.

All relevant runtime information is declared in the app manifest file. This is an XML file that contains information such as the app name and version, the publisher, an icon location and screenshot locations. The manifest also declares all the app's dependencies, including data services and ui services (including other apps).

The manifest also declares the functionality that the app exposes, if any. This may include one or more entry points that may take parameters etc.

Apps may have runtime dependencies on certain container functionality or on other apps. Apps may give hints to the container about device capabilities such as preferred screen resolutions, hardware requirements (e,g, GPS, smart card etc).

The app manifest is located in the root of your app folder, under the file *manifest.xml*

For the purpose of the coding competition, we will focus on the following aspects:

Basic App Attributes

```
<app>
    <appId>CommBank.PublisherSearch</appId>
    <appName>Entrants</appName>
    <version>1.0.0</version>
    <publisherId>CBA</publisherId>
    <publisherName>Commonwealth Bank of Australia</publisherName>
    <description>App description.</description>
    <versionNotes></versionNotes>
    <appIcon>~/Images/icon.png</appIcon>
    <screenshots>
        <screenshot>~/images/screenshot1.jpg</screenshot>
    </screenshots>
```

<appId>{unique string}</appId>

The appId allows the helix eco-system to identify your app via a unique identifier. In the real world, this will be based on some form of namespacing, but you can use something like *cba.comp.myapp*

<appName>{name of your app}</appName>

This is a print friendly name that may be displayed within various Helix control systems. It is suggested you just make your app name the same as your appId defined above.

<version>1.0.0</version>

In the real world, Helix apps will be immutable, and variants of the app will be identifiable by their unique version number, allowing for such capabilities as swap in/swap out, piloting and rollback. For the coding competition, just leave this as default, 1.0.0.0

<publisherId>CBA</publisherId>
<publisherName>Commonwealth Bank of Australia</publisherName>

A design principle behind Helix is to allow for federated development across the CBA group. The publisherId and

publisherName allows for tracking of the development team responsible for producing the app. For the coding competition, you can edit these fields as you see fit. The text is free-form. E.g. <publisherId>BWA</publisherId> <publisherName>BankWest ESSD</publisherName>

<description>App description</description>

Coding competition entrants are encouraged to write a rich description of their app. After you app is submitted to Helix's Mission Control - your app will be browsable and selectable by the coding comp judges, and a good description will certainly help set the tone of what your app actually does.

In a production environment, app descriptions will aid testers, builders and deployment coordinators work with new apps as they flow through the various manufacturing phases.

<versionNotes></versionNotes>

You can leave this element blank. This will be useful in production where apps have multiple versions, and version notes can contain such concepts as bug fixing errata, what's new lists etc.

<appIcon>~/images/icon.png</appIcon>

The appIcon provides a visual glyph to accompany your app. Coding Comp entrants are encouraged to provide a suitable icon when your app is submitted. The value to provide is a pointer to a web viewable image file (e.g. JPEG or PNG) - the file path is *relative to* the root of your app.

```
<screenshots>
    <screenshot>~/images/screenshot1.jpg</screenshot>
</screenshots>
```

<screenShot>~/images/icon.png</screenShot>

You can provide a list of screenshots or other graphic representations of your app. When your app is viewed in the Helix Mission Control application, screenshots can be viewed. Like the appIcon element above, links to screenshots are to web viewable graphics files, and are *relative* to your app's root folder. You can provide as many screenshots as you like by adding more <screenshot> elements within the <screenshots> parent element.

Theming

```
<themes>
  <container>
    <supports name="Sections" />
    <supports name="Typography" />
  </container>
  <custom fallback="default">
    <theme name="bankwest">
      <link rel="stylesheet" href="~/Themes/bankwest/Home.css"/>
    </theme>
    <theme name="asb">
      <link rel="stylesheet" href="~/Themes/ASB/Home.css"/>
    </theme>
    <theme name="commbank">
      <link rel="stylesheet" href="~/Themes/CommBank/Home.css"/>
    </theme>
  </custom>
</themes>
```

Need someone to fill out the themes bit...

Server-Side App Entry Points

```
<exposes>
      <ui interfaceId="ContainerLaunch" interfaceVersion="1" resolvesTo="~/Default.htm">
          <parameters></parameters>
          <permittedConsumers>
              <container xsi:type="AnyContainer"/>
              <user xsi:type="UserAnonymous"/>
          </permittedConsumers>
      </ui>
  </exposes>
```

Full production apps will be able to describe a rich set of interfaces - both in what they <expose(s)> and what they can <invoke> (more on <invoke> later). To keep the coding competition simple, you only need to concern yourself with the primary launch point for you app - i.e. the HTML file or ASP, ASPX, JSP etc. that renders a base page to your browser.

In the example above, our sample app has a file called Default.htm - this is executed when your app is launched (i.e. the container loads your app's designated URL into the IFRAME).

Thus, if you are building a PHP page (e.g. main.php), you will need to change the above resolvesTo= XML attribute to the location of your page (*relative* to the root of you app). E.g. this might be:

<ui interfaceId="ContainerLaunch" interfaceVersion="1" resolvesTo="~/main.php">

This is all you need to do to get your app set up for launching.

Just Out of Interest

... You don't need to concern yourself with the following to get your App built and submit capable to the Coding Comp App Store. But if you're interested, some of the concepts currently being explored in the Helix App model are:

```
<invokes>
    <ui interfaceId="publisherdetails">
        <bindings>
            <app publisherId="CBA" appId="CommBank.PublisherDetails" interfaceId="Launch" interfaceVersion="1">
                <parameters>
                    <wellKnownParameter xsi:type="dt:Publisher"/>
                </parameters>
            </app>
        </bindings>
    </ui>
</invokes>
```

In a production environment, each app will live within a rich ecosystem of other apps - and how context can be passed to one another needs to be strictly controlled. To support this, <exposes> and <invokes> is definable by an app. It enables:

- The definition of what apps an app can call, and what apps can call it.
  - Definitions include the concept of interface versions and pre-defined parameters
- Does not mandate pre-knowledge or dependencies between apps
  - Apps interoperate based on matching interfaces - not actual binaries or types.
- Assists security controls
- Assists deployment control of apps
  - Apps can declare what container they can work in (and by definition what they can not)
  - Apps can define what user types are supported - e.g. NetBank customers, CBA staff members, ASB retail customers

Navigation

Explain how navigation:

- between Apps works.
- between views inside an App works.
- from an external website works (deep linking)
- to an external website works

Instrumentation & error handling

For Coding Comp, the functionality of standardised Instrumentation has been removed to enable Apps to run on you home machine.

JavaScript best practices

The following components can help do menial or tedious tasks

JQuery - forms the base of the heavy lifting

JQueryUI

jsRender - is a jQuery plugin that provides templating and one-way data binding for JavaScript/HTML.

jsViews adds two-way binding and live updating.

UI controls

For the Coding Competition we've not prescribed a particular UI Control library. This gives you the freedom to create UI's however you see fit.

In the long term Helix Sheldon Apps will use a best practice JQuery UI based library. This is still evolving as different libraries are investigated for team and skill fit.

Given that any library can be used, the Design Guidelines will need to be followed to make Apps look cohesive with each other in the Future Bank Container.

Validation

Validation of data is something that has a number of aspects in Helix Apps:

1. Where to do it
2. How to display it
3. Using the correct style attributes to have it displayed in a consistent way

To keep coding competition simple, Apps are free to implement validation as needed using whichever technology and approach make sense to the developer. Adherence with the Design Guidelines and theming implementing would be beneficial.

Including external resources

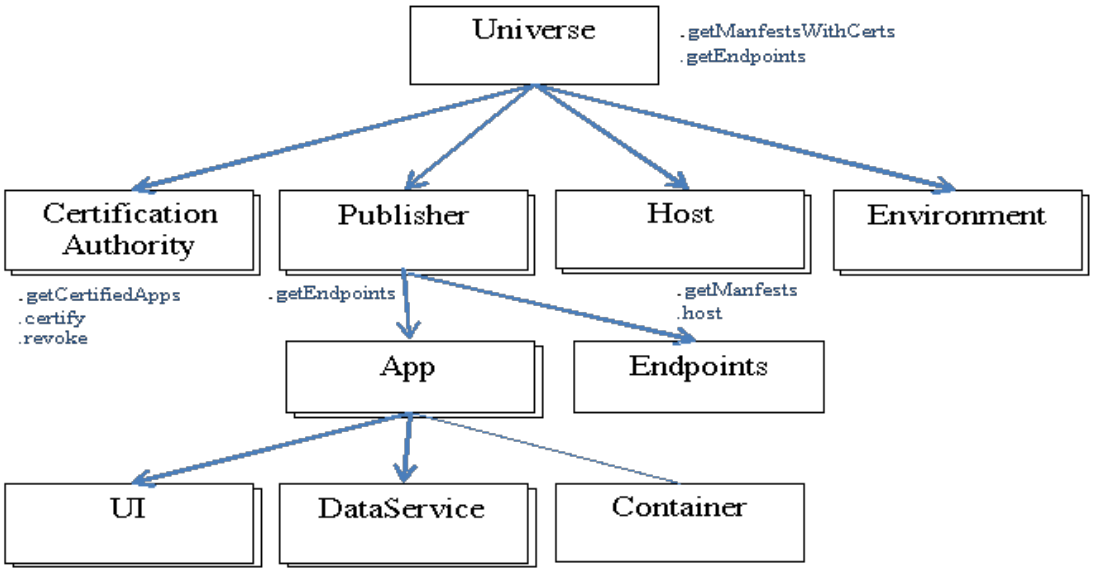
Apps are free to include third party resources. They can be libraries that are included with the App or referenced from other URLs. For Helix Apps, all external dependencies will have to be declared in the manifest file, but to make it easier for coding comp, you just need to declare them with the App submission.

Some things you could use external resources for are:

1. developer efficiency libraries to do heavy lifting - eg JQuery libraries etc
2. Mashups with other web content eg social media sites etc
3. integrating other products into the Helix App ecosystem

Environments and deployment universe

Actors



Hosting an App

An app can be hosted by submitting it to a Host. An app may be hosted on multiple hosts. Hosts may be shared – or single purpose (eg a specific host for hosing production apps?). The ability to submit an app for hosting may be restricted to certain persons. The host may expose a number of methods to submit an app (web service, command line, file share (for integration with CI environment) web page etc).

Certifying an App

When an app is hosted – a checksum is generated for the app. That checksum can then be certified by a certification authority (if said believe the app meets its guidelines)

Environment Definition

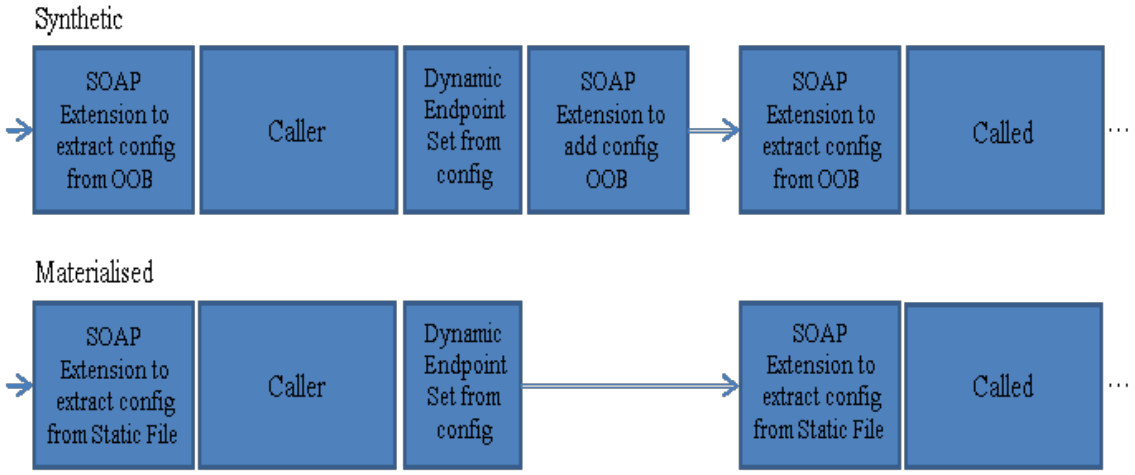
An environment definition is a set of applications, a set of data sources and an entry point (which resolves to a URL). This environment definition can potentially be stored & shared.

```
<?xml version="1.0" encoding="utf-8"?>
<environment name="Womble55">
  <uiEntryPoint publisher="CBA" appId="Prometheus" interfaceId="Launch"
interfaceVersion="1" />
  <apps>
    <add havingCertifications="UX Penetration Perf UAT" host="test1"/>
    <add publisher="CBA" host="test1"/>
  </apps>
  <dataSources>
    <add dataSourceSet="T1"/>
    <add publisher="CBA" dataSource="Transact" endpoint="HappyPathStub"/>
  </dataSources>
</environment>
```

When specifying which apps should be in an environment – the environment author can select apps which have been certified by specific authorities, exist on a certain host, belong to a certain publisher etc. Apps can be containers, user interfaces and containers.
The environment author can select a pre-configured cohesive dataSourceSet and/or specific dataSource endpoints.

Synthetic vs Materialised views

Given the environment definition above, there are two means to make this environment available for use.
Materialized: the configuration file would be submitted to ACDC. ACDC would be responsible for copying all the files for all the apps (dataservice & ui apps) to the appropriate servers, configuring vDirs and configuring static configuration files for each of the apps to point to the correct endpoints.
Synthetic: Apps do not need to be physically moved around to create the environment. The apps reside on one/many host servers – and the configuration for their endpoints is passed (to the apps out of band) when they are invoked.
A combination of the two. Would give the benefits of the dynamism within Test, and the determinism (& increased security within Production). This dual behaviour could be implemented via SOAP extensions.



CallerSOAP Extension to add config OOBSOAP Extension to extract config from OOBDynamic Endpoint Set from config
CalledSOAP Extension to extract config from OOB…CallerSOAP Extension to extract config from Static FileDynamic Endpoint
Set from config
CalledSOAP Extension to extract config from Static File
…SyntheticMaterialised

DataSource Set

Sets of cohesive data sources can be grouped into a datasourceset. Which can (for convenience) be added directly into an environment.

```
<dataSourceSet name="T1">
  <dataSource publisher="CBA" dataSource="CommSeeDB" endpoint="CSDB"/>
  <dataSource publisher="CBA" dataSource="CreditCard" endpoint="Test1"/>
  <dataSource publisher="CBA" dataSource="Transact" endpoint="Transact1"/>
  <dataSource publisher="CBA" dataSource="IFW" endpoint="S2"/>
</dataSourceSet>
```

DataSources

Datasources are owned by publishers. Each data source has a name and multiple endpoints. The details of the endpoint (stored in a CDATA section) are interpreted by the dataSourceFactory.

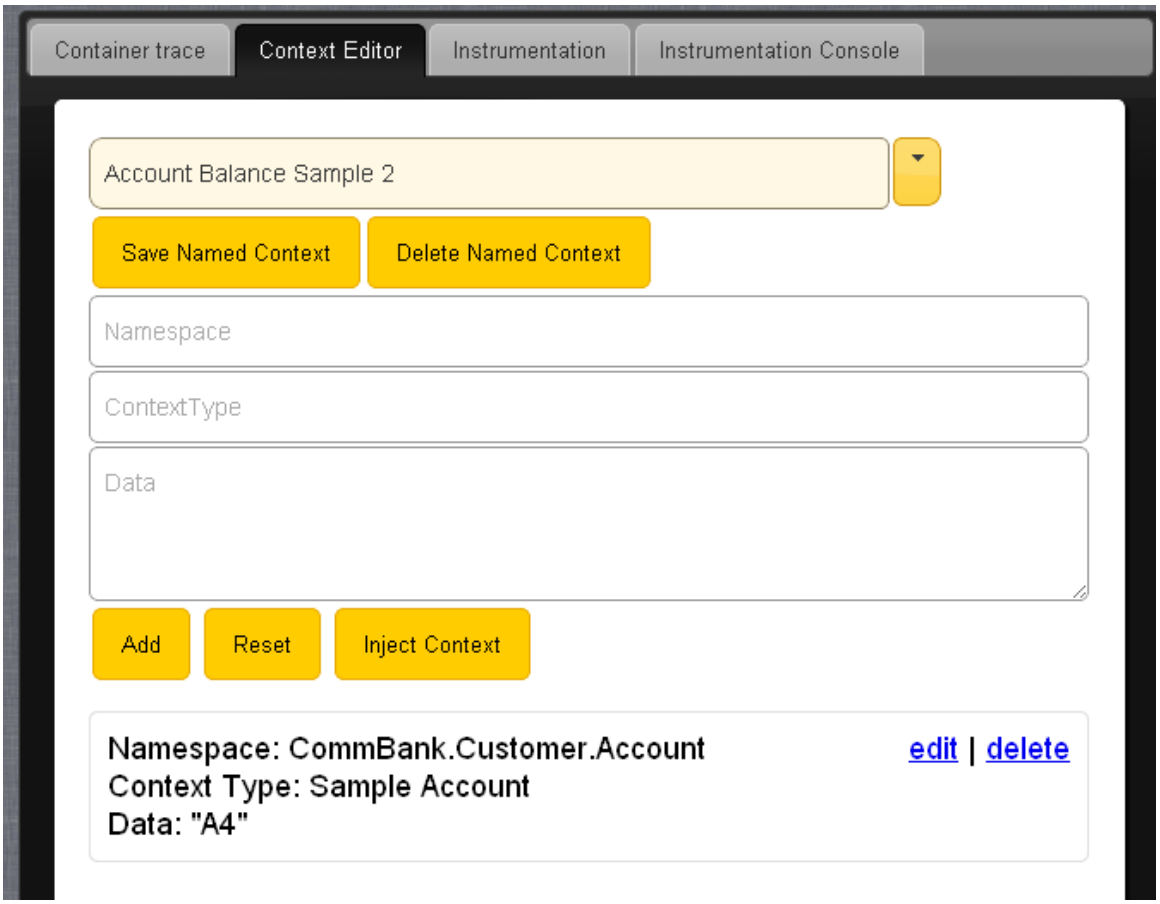
```
<dataSources publisher="CBA">
  <dataSource name="CommSeeDB">
    <endPoint name="T1">
      <![CDATA[Data
Source=username/password@myserver/myservice:dedicated/instancename;]]>
    </endPoint>
    <endPoint name="T2">
      <![CDATA[Data
Source=username/password@myserver/myservice:dedicated/instancename;]]>
    </endPoint>
    <endPoint name="T3">
      <![CDATA[Data
Source=username/password@myserver/myservice:dedicated/instancename;]]>
    </endPoint>
    <endPoint name="BARD">
      <![CDATA[Data
Source=username/password@myserver/myservice:dedicated/instancename;]]>
    </endPoint>
  </dataSource>
  <dataSource name="IFW">
    <endPoint name="S1">
      <![CDATA[10.17.28.27]]>
    </endPoint>
    <endPoint name="S2">
      <![CDATA[10.17.28.27]]>
    </endPoint>
  </dataSource>
</dataSources>
```

Universe

The universe contains a register of all Publishers, Certification Authorities, Hosts & Environments. It exposes a service 'getManifestsWithCerts' which returns all Manifests decorated with earned Certifications.

Context Editor App

What is it used for?



Context can be used to enable the exchange of data between applications and other applications and or the container. An example of the use of context could be if one application sets the selected Account. Another application which subscribes to this context can then use this fact to display detail for the selected Account.

The context editor application was built as a test harness to enable the App developer to inject context in order to test the application being developed.

Using the application

The application allows for capturing one ore more context items. While capturing the context items, they will only exist in the scope of the Context Editor Application. By clicking the "Inject Context" button, it will be set at the container level and will then be available for other applications.

The captured context can also be saved to local storage to enable a load a load of a saved "Named Context".

1) Add context item(s)

In order to add a new context item the Namespace, ContextType and Data fields need to be populated. An example of the format of the data can been seen in the screenshot above. The Data field can have the following formats:

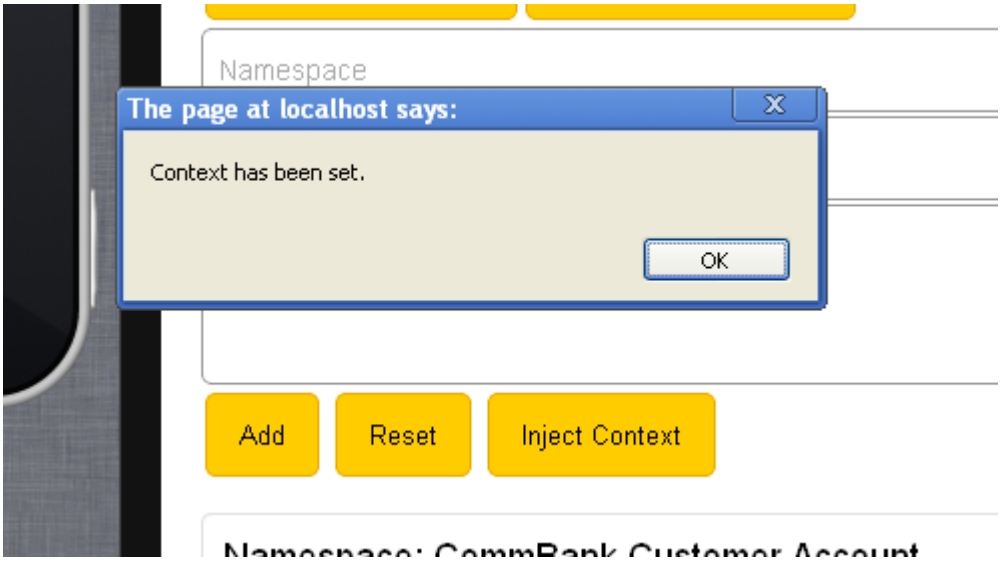
- Numeric
  e.g. 1200
- Text (enclose in double quotes)
  e.g. "Enclose text in double quotes"
- JSON
  e.g. {AccountNumber: "1234567", AccountName: "Holiday saver"}

If the add button is clicked, the App will check that all the fields have been populated and then display the newly added item in the list below. A context item from the list can be edited by clicking on the "edit" link.

The changes are applied by clicking on the "Apply Change" button or the edit mode is cancelled by clicking on "Cancel". As suggested, the "delete" link will remove an item from the list.

2) Injecting the context

In order to make use of the context items, the captured context items (displayed in the list) must be injected to the AppStore.App.Context. This is done by clicking the "Inject Context" button. This action will cause the context items captured to be published.



3) Save the captured context items for future re-use

The items captured can be saved to local storage by typing a description in the "Enter a name for Context Set…" box and clicking on the "Save Named Context" button. This will persist the list to local storage and can the be loaded by selecting it from the menu that appears when the "arrow down" button is clicked. As suggested the named context set can be deleted by clicking on the "Delete Named Context" set button.

Guidance for extending the capabilities

The Helix framework as provided for Coding Comp is quite limited to deliberately lower the amount of effort and learning curve required to enter the comp. You are free to extend and expand on the framework in any way that you see fit.

In doing so, remember the Architectural Themes and Framework component overview to ensure that your extension fits with the design goals and business benefits that the architecture is aiming to achieve.

Naming Conventions

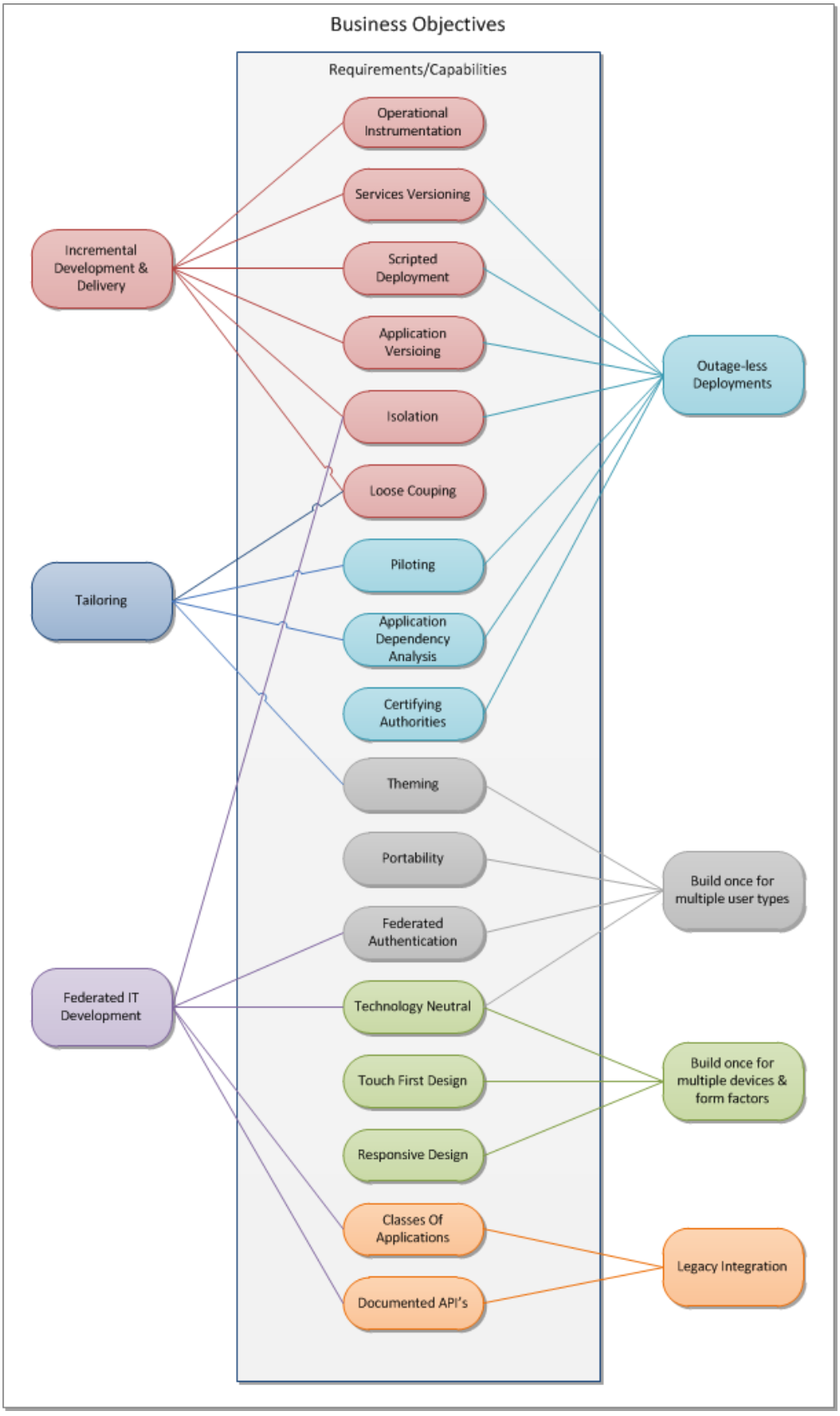
For Coding competition Apps there are some minimal naming conventions to be aware of, but nothing prescriptive in terms of code readability styles.

1. Ensure that your javascript functions do not clash with framework functions.
2. Familiarise yourself with the CSS class names in Doing it with style

Architectural Themes

These are the key themes of the architecture. They represent the business benefits that can be achieved if the application is built following the requirements or capabilities. Not implementing a capability

means that the business benefit associated with that capability can not be meet.



30 second overview

The Helix Project has been tasked with the goal of establishing a new, modern architecture that will enable:

1. ***Delivery Agility***
    - An **open ecosystem** for development removes bottlenecks
    - **Unlock world class core banking capability** by leveraging 3rd party development
    - Segregation of consumer apps to **compartmentalise & ultimately lower risk**
2. ***New User Interfaces***
    - Interaction-oriented' user interfaces can be **built once and target any device**
    - Ensure Group is positioned well to **exploit next-wave consumer devices internally**
3. ***Portability***
    - Introduce portability and customisation: **any app can be used in to any channel**
    - All functions open to re-use: **access by anyone, anywhere**

Project Helix will deliver:

- **Tools** - development and test tools & environment support
- **Processes** – build, test, review, deployment & activation
- **Patterns** – legacy integration, isolation, responsive design, theming, outage-less deployments, security
- **Platform API's** – documentation & code libraries, open framework
- **Real demo's** - working code examples for NetBank touch

Helix is a conceptual architecture to manage functions in the channel domain (online, branch, call centre etc).

Its goals are to all alow rapid, incremental development of individual business functions.

Apps built using the architecture are device and user agnostic.

The Helix architecture allows for integration with legacy assets so this architecture can be progressively rolled out without large scale change being required.

Motivations for the architecture

The Helix Project initiative has been started with the goal of resolving these issues by establishing a new, modern architecture that will enable a greater degree of rapid development and a simpler means of supporting multiple platforms and form factors.

The current suite of business applications being used by CBA's staff and customers, including CommSee, NetBank and CommBiz have been serving the business well for a number of years. Each of these solutions are based on a similar technology and architecture, which has resulted in a number of issues.

Lessons from the past

- A recognition that the existing process of synchronised monolithic deployments of end to end bank functionality has worked well to support programs like Core Banking Replacement, but doesn't suit many of the kind of challenges we see in the future. The appetite & rate of change is greater than these platforms can meet.
- There are number of functions that are fundamentally the same, regardless of who's performing it and what device they are using. These don't warrant being built multiple times.
- There are legitimate reasons for multiple subtly different user interfaces for the same business function to exist. Previously, these were sometimes coupled together with other assets which caused undue inflexibility. It's not always appropriate to solve a problem once and for all.

Challenges we see coming in the future

- Business desire for small scale experimentation with banking products, customer experiences and tools without having to commit large amounts of funding and waiting for long periods of time.
- Fragmentation of consumer devices resulting in the bank needing to offer the same business function on

multiple devices and websites. This is true both for customers and for staff.
- Touch centric user interfaces are becoming available in all form factors, so the separation of an experience by Mobile, Tablet & Desktop is becoming less clear. Existing online capabilities will need modification to make best use of touch.

It is proposed that a new architecture be evolved that address these pain points called the Composite Application Framework. This project will define and prove this new architecture pattern for building online applications.

Design Guidelines

'Simple, consistent, useful and intuitive'

Using a combination of both touch-first and mobile-first design principles, aids us in achieving one of the key goals of Project Helix. That is to create simple, consistent, useful and intuitive interfaces which work across an ever-increasing variety of resolutions and devices.

UI Design Principles

UI Design Principles provides an overview of information around various design principles that can help in application development.

Touch first design
helps to provide information on design pertaining to touch screen

Mobile first design
helps to understand design for mobile devices

Customer centric processes
shows approach to design by focusing on customer rather than technology

Responsive design
helps to understand changing layout based on size of device display as well as designing app in different views

Device Considerations

Device Considerations which consists information on varioud device components like browser requirements, orientation, etc.

Browser Requirements
provides information on supporting of various browsers

Orientation
helps to understand about screen orientation for mobile and tablet devices

Resolution
talks about resolution for various components of screen

Pixel Density
provides info on display sizes for various themes

Once design guidelines are understood, you are ready to implement style and themes so that your App is Doing it with style

Device Considerations

Browser requirements

For the purposes of coding comp, the developer container's browser requirement is Google Chrome. We have not tested performance with the other modern browsers extensively at this stage.

Multiple views

What are our guidelines for touch vs PC, explain the possibilities and design philosophy for app navigation. Refer to this: http://www.lukew.com/ff/entry.asp?1569

Orientation

This comes into play with mobile and tablet devices where the user can orientate the screen both portrait as well as landscape, changing width and height of viewport instantly. The container and app content and corresponding content within needs to respond dynamically to these immediate changes to orientation

Further details can be obtained from section 'Responsive Navigation' explained under Responsive Design

Pixel Density

Resolution-independent measure

The pixel isn't a resolution-independent measure and will render differently across devices with different pixel densities.

To illustrate the variety of pixel density values to contend with, here is a sample list of current PPI values across a range of popular mobile and tablet devices:

Device	Pixel Density
iPhone 4S	326 PPI
Samsung Galaxy Note	285 PPI
iPad 3	264 PPI
Amazon Kindle Fire	170 PPI
Galaxy Tab 10.1	149 PPI
iPad 2	132 PPI

As more devices and displays are released with higher pixel densities it becomes increasingly important to cater for variations in order to consistent UI's. (http://en.wikipedia.org/wiki/List_of_displays_by_pixel_density

Physical display size

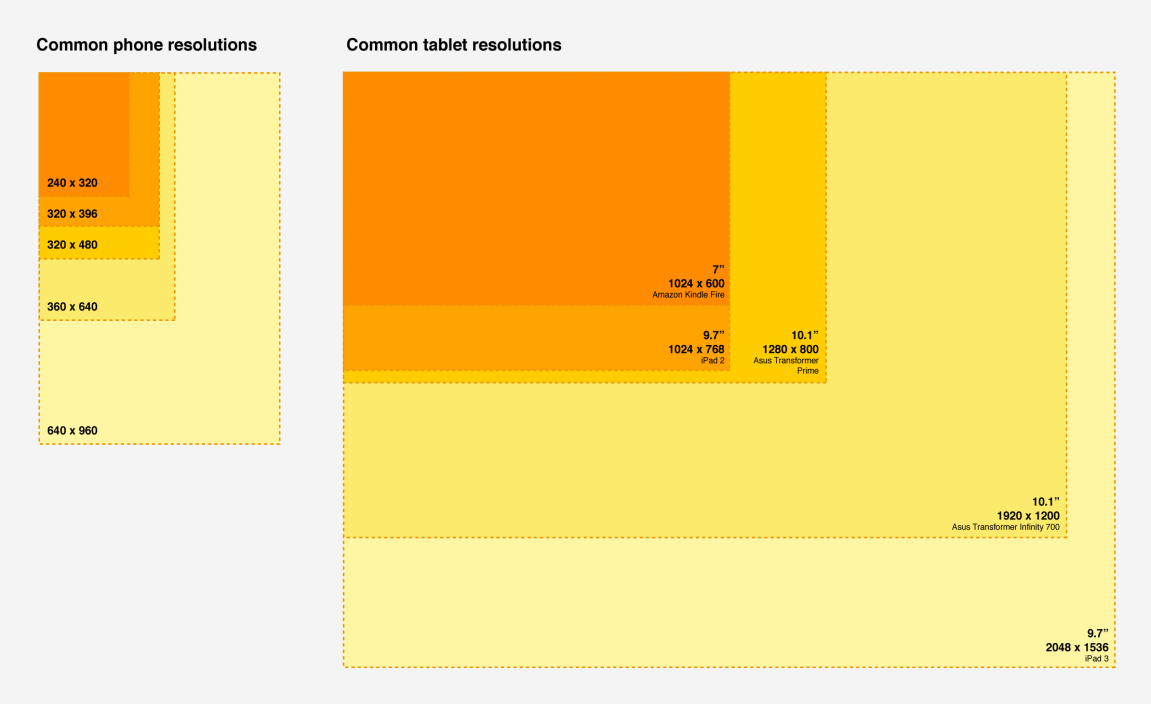
The following formula is used to calculate the required pixels for devices with different pixel densities so that you can control of the physical size an element will display:

Pixels = Device Pixel Density (PPI) × Measurement (mm)

The following example uses this formula to calculate the pixel size of a 9mm target on a 135 PPI display:
Pixels = 135 PPI × 9 mm
Pixels = 135 PPI × (0.03937 inches per mm × 9 mm)
Pixels = 135 PPI × 0.35433 inches
Pixels = 48 px

Resolution

For desktop screen resolutions we use a **minimum** viewable pixel area of 980px x 580px based on the 1024px x 768px screen. Currently resolutions increase to as much as 2560px x 1440px.
(http://www.w3schools.com/browsers/browsers_resolution_higher.asp)
Below Illustrates the various resolutions across a sample of common mobile devices and tablets:



UI Design Principles

'Simple, consistent, useful and intuitive'
Using a combination of both touch-first *and* mobile-first design principles, aids us in achieving one of the key goals of Project Helix. That is to create simple, consistent, useful and intuitive interfaces which work across an ever-increasing variety of resolutions and devices.
References
Digital styleguide guide
http://www.elyseholladay.com/2012/07/16/style-guide-guide.html

Customer centric processes

Anticipate a user's needs

Where possible cut down the amount of actions required by the user, allow the UI to do the work
Anticipate a user's needs, but don't hide anything. An example would be 3 separate dropdowns for day/month/year vs 1 text-entry field.
Immediate feedback
Provide feedback immediately, particularly in form interactions. Don't wait to verify until the user submits.

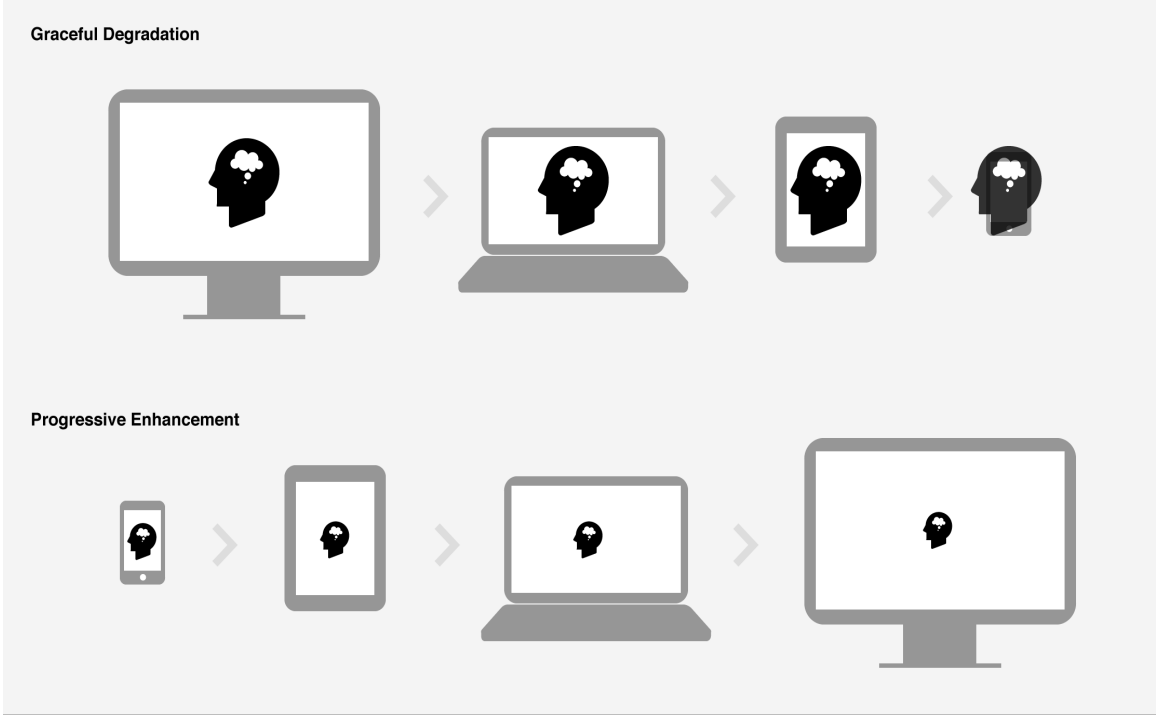
Performance
Keep in mind where you might be bulking up download size with graphics, whether they are CSS gradients or large images, with more complicated interactions or content, or unoptimized code.
*Note** If your app will be too constrained by a particular resolution or device, then you should go ahead and design a device-specific app*

Mobile-first design

Mobile provides a great opportunity to reevaluate what content/functionality is necessary and gives us an opportunity to strip away the cruft across the board (and not just for mobile users either). These constraints also encourage ease-of-use, intuitiveness and speed as essential ingredients to a good user experience.
"When a team designs mobile first, the end result is an experience focused on the key tasks users want to accomplish, without the extraneous detours and general interface debris that litter today's desktop-accessed websites. That's good for the user experience and good for business."
Luke Wroblewski http://www.lukew.com/ff/entry.asp?933

Graceful Degradation vs Progressive Enhancements

Scaling up from the mobile context (progressive enhancements) versus scaling down from the desktop context (graceful degradation) ensures that your message, content and functionality remain intact as the screen real estate and connection speed increase. Scaling down runs the risk of your core message and functionality getting lost by the time you squish it all the way down to the mobile context. Once content is structured, the first context to pipe the content into is the mobile web. Why start here and not the desktop? The mobile web is far more restrictive, eclectic and unstable than other contexts.
Mobile users will do anything and everything a desktop user will do, provided it's presented in a usable way, and so try not to cut out critical content on mobile just because it's difficult to place. If you have to cut something on smaller form factors, determine what is most crucial first, but keep in mind we don't know what our users want to do on our app on the tablet or mobile device so we should not decide for them. It's a myth that mobile users don't want/need certain information.



Single-Use Screens

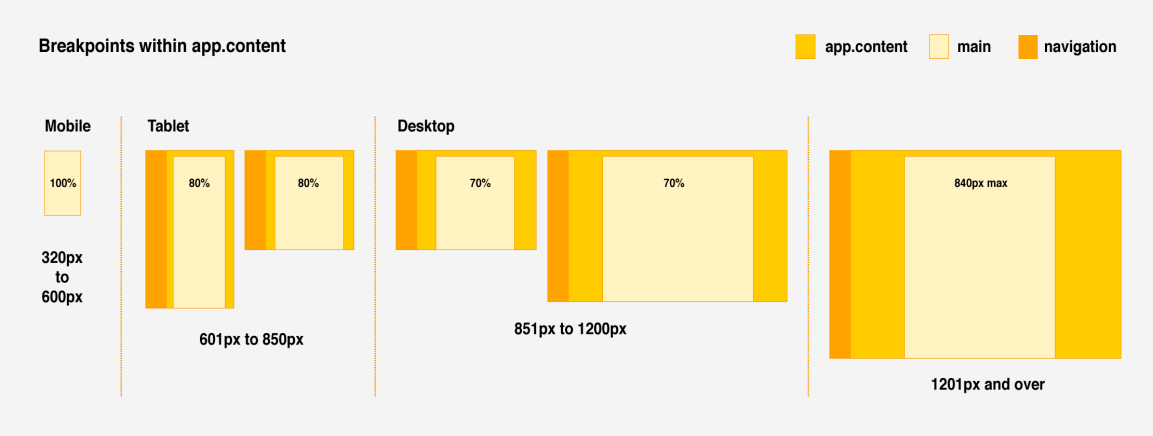
Break down tasks into their simplest form. Avoid dashboard-style cluttering of the interface purely because the space is available. If the screen size is large enough that multiple views/tasks can fit, it doesn't mean the extra views should be present to fill space. Extra content often adds clutter and overwhelms the user.

References
Mobile-first responsive web design
http://bradfrostweb.com/blog/web/mobile-first-responsive-web-design/

Responsive Design principles

Responsive design is a combination of fluid grids and images with media queries to change layout based on the size of a device display. It normally uses client-side feature detection to determine available screen capabilities and *respond* accordingly. It's most useful for layout but can also be extended to interactive elements as well (although this often requires Javascript).

Defining Breakpoints
Within **app.content** the **main** column contains the actual content scales width at various screen sizes. The container navigation in our current sample containers once expanded does not scale but opens to a fixed width of 280px.

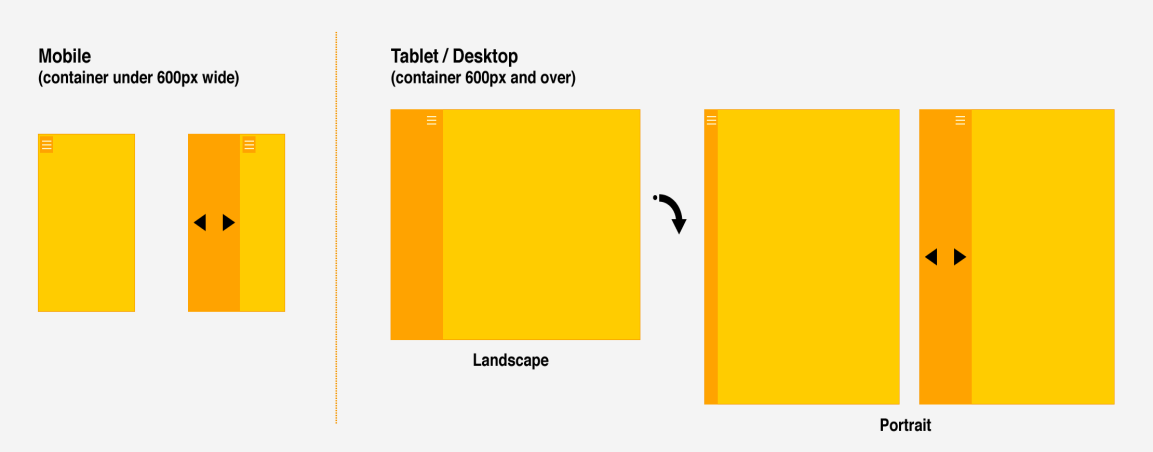


For our sample apps and themes to date, we have established the following breakpoints:

- app.content equal to and under 600px main width = 100%
- app.content between 601px and 850px main width = 80%
- app.content between 851px and 1200px main width = 70%
- app.content 1201px and over main width = 840px max

We determined that the **main** area should not scale wider than 840 pixels for reasons such as:
i) utilising our 'singular task' per app concept, 840px provides ample space compared with most websites for page components such as lists, tables, forms and media etc. Any wider and these assets become too heavy.
ii) Keeping a decent percentage of the background viewable as it often contains full-screen imagery.

Responsive Navigation
Below illustrates how the navigation behaves at different resolutions and orientations.



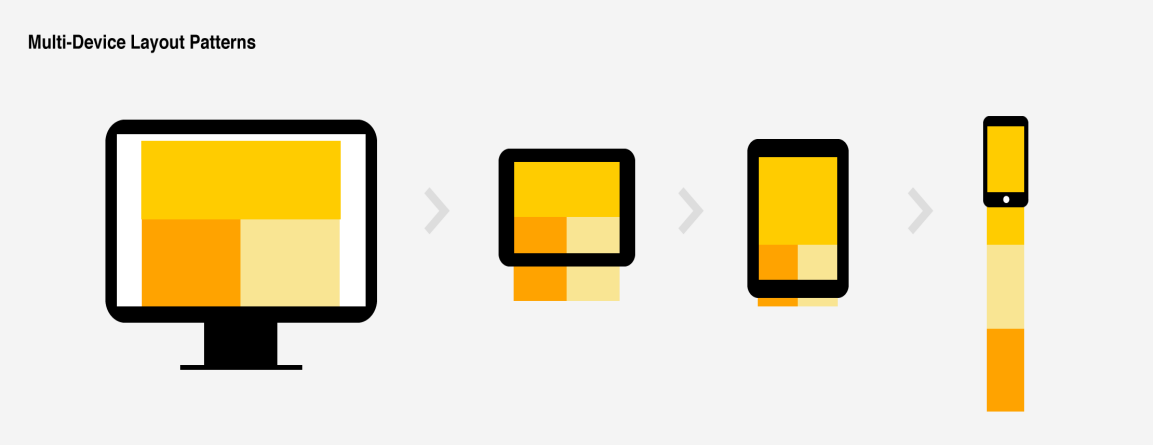
Collapsing navigation
*Maybe we can show an example here with regards to our **filter.sort** solution for mobile??*

Multiple views

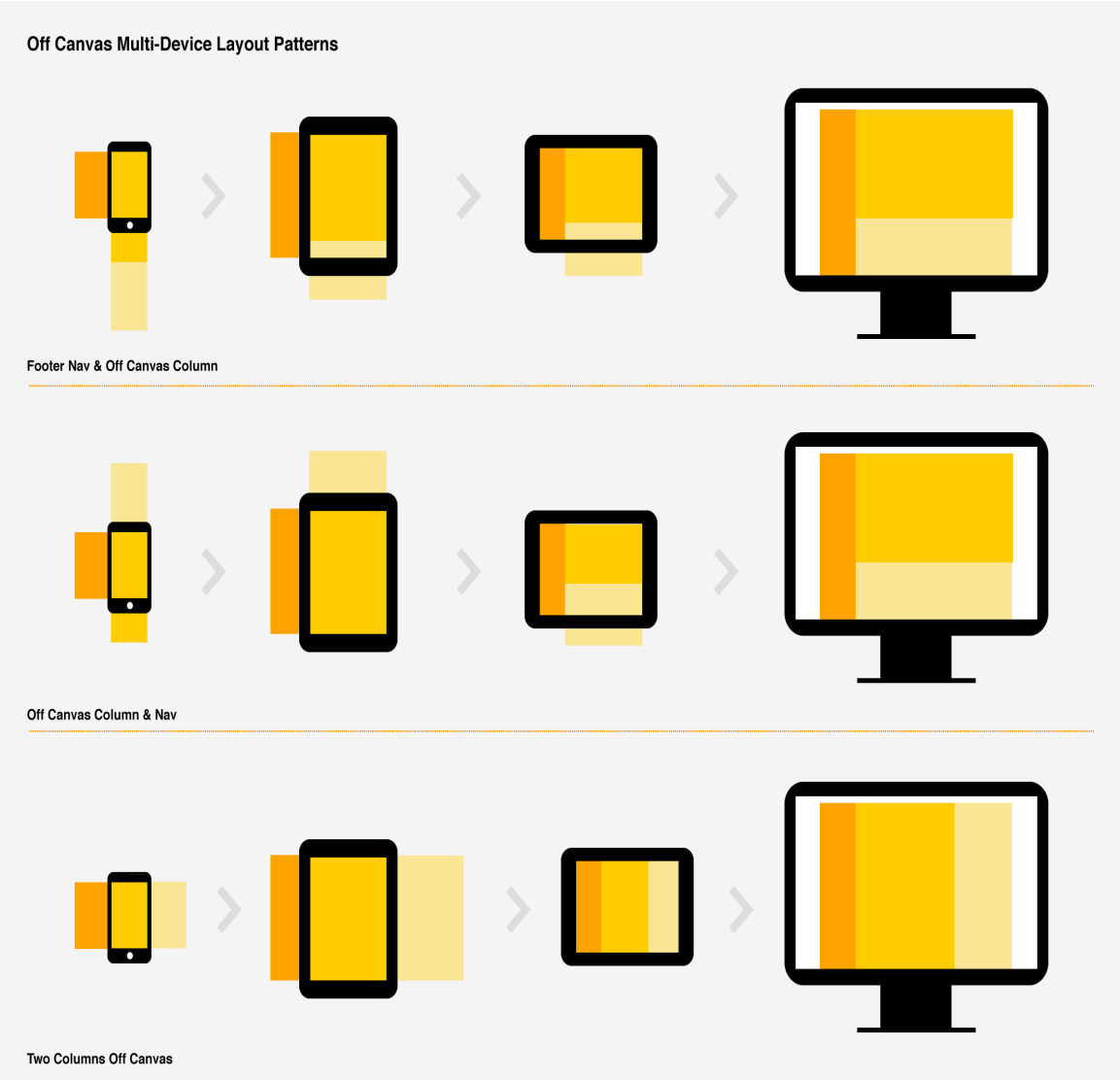
One of the classes listed in our UI overview is **app.navigation** and its function is to navigate between app views.

It is likely that there will be many scenarios where you have more content within an app than can fit inside the viewable area, or times where you feel a better user experience would be to divide a process into the separate steps, especially where screen size is limited. Although our sample apps do not illustrate this yet, this is an important aspect of responsive design which requires investing time to develop during the next phase post-coding comp.

Responsive design solutions through fluid grids and media queries can handle content when resolutions scale. Below is an example of a common responsive design pattern for handling multiple content areas. In the first example below, by the time the layout is displayed at mobile size, the content areas are simply stacked one on top of another making the user scroll a long way down the canvas to get to later content which in many cases is a weak UX solution.

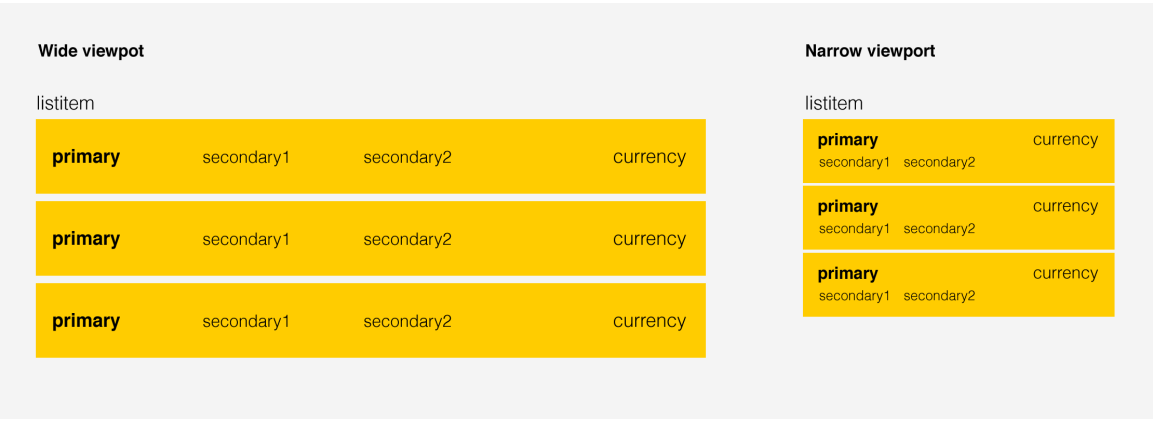


An approach to improve on this experience is what is termed **Off Canvas Multi-Device Layout Patterns**. This u ses the space outside a browser's viewport to hide secondary elements until people need them. It's another example of **mobile-first design** principles using the mobile form factor as a start point rather than an afterthought. Below illustrates a number of potential applications of this thinking.



We have utilised a version of **Two Column Off Canvas** for the container navigation on smaller resolutions

Responsive Page Components
Within lists we have 'fixed' and 'fluid' elements. These have been allocated a hierarchy and when scaling down secondary items fall down below primary (fixed) items to help reduce overall widths of rows.



Scaling

This has been mentioned under Defining Breakpoints regarding the scaling of the **main** column within **app.content**. Padding, margins and font sizes all increase when scaling up from mobile, and will need to be defined in at those breakpoints also.

References
An introduction to responsive design
http://www.lukew.com/ff/entry.asp?1436

Touch-first design

What does this mean and how has it influenced our design?
The principle of touch-first is to focus design primarily with touch devices as a starting point when considering interaction and UI then extend towards desktop next, rather than the older approach of solving desktop challenges first and then adapting them for touch interfaces (often as an afterthought).
As a pointing device, a finger, being much larger than a mouse, requires a larger touch target than what's required by a mouse cursor. So, to ensure usability, interactive elements need to be bigger. As interactive elements increase in size, other things need to increase in size to maintain balance. This leads to an aesthetic characterized by generous margins and padding.
We allow for a slight deviation from the norm in one experience in order to better support all possible experiences. What we end up with is a design that might afford more than necessary space for a mouse but an appropriate amount of space for a finger. Making a UI touch-friendly in this way also results in a UI that might be more useable for mouse-and-desktop users. A button that's easier to touch is often easier to click. By erring in the direction of usability, we get the bonus of improved performance of the design in its original desktop context.
Avoid hover states. Hover states are great, but do not work on touch devices. Come up with non-hover ways to reveal state, content, or functionality. Add hover states as enhancements only.

Try to avoid:

- Hyperlinks that aren't 100% obvious
- Javascript tooltips that show important information or metadata
- Displaying action items on hover, typically edit / delete items

References
Touch-First design
http://uxdesign.smashingmagazine.com/2012/01/17/designing-well-tempered-web/

Apps are managed and released independently

Current State

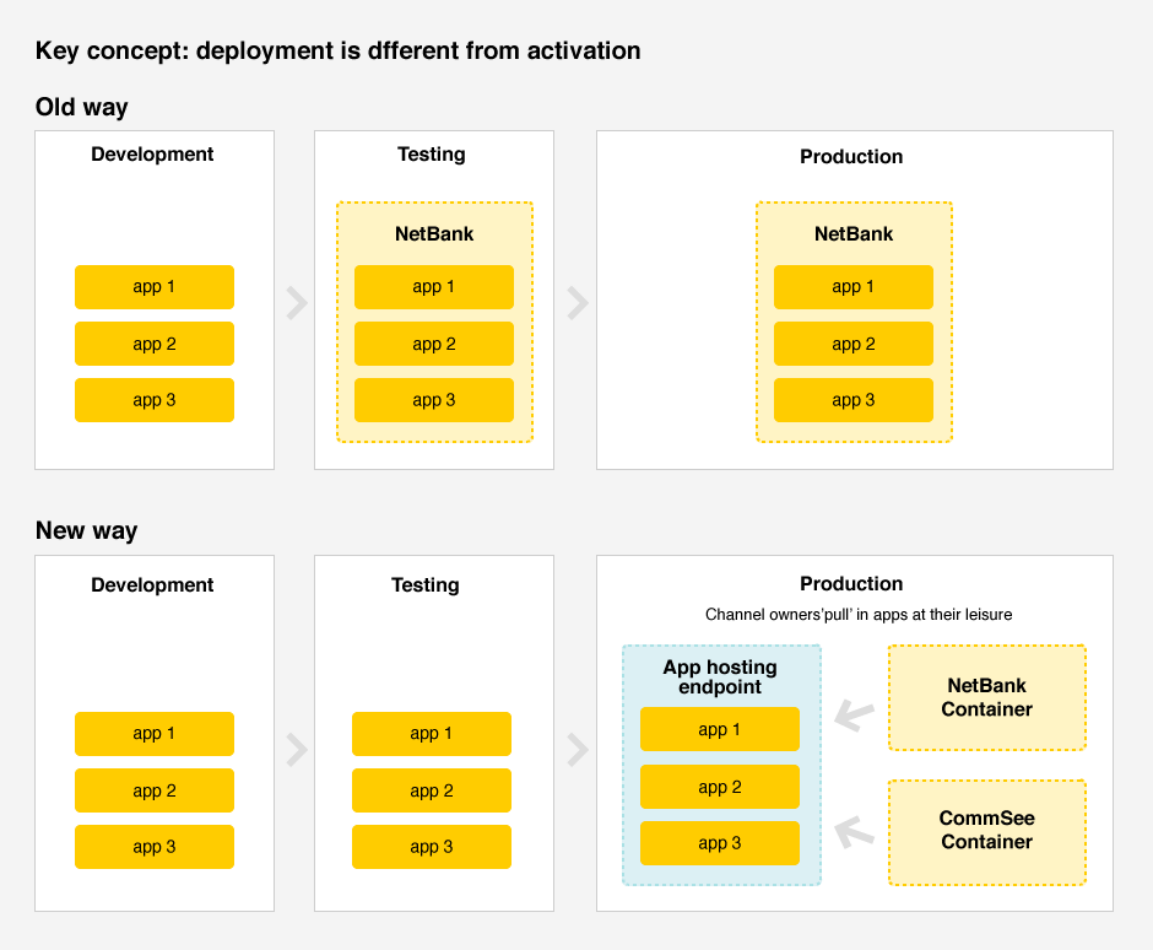
Major business function changes are deployed twice or three times a year in bank wide deployments. Some changes are able to be deployed as patches or emergency fix which occur monthly or weekly.

Each business function is generally written for project team isolation during the SDLC process. However all changes across the project teams are packaged and deployed together.

Future State

Business functions are written for isolation from concept, development, test, deployment and operation in production. Business functions are deployed to production when ready, independently of other business functions.

Deployment of a business function or publishing of an application is separate process to enabling or consuming a business function to end users.



Solution Requirements

1. *Isolation:* Development, Testing, Deployment and Operations of business functions are isolated from other business functions. Business functions are coded such that all reuse is done via centralised API's and SOA services and other strict contracts which can are manage and interrogated.
2. *Scripted Deployment*: Publishing of code to production is separated from consumption. Application code deployments need to be 100% scripted with no human intervention.
3. *Application Versioning*: All applications must support versioning, backwards compatibility and side by side deployment.
4. *Services Versioning:* All API's and SOA services must support versioning, backwards compatibility and side by side deployment.
5. *Loose Coupling*: Business functions must be coded so that they are not directly dependent on other applications.
6. *Operational Management*: All business functionality must have an operational view for production management and rolling out incremental changes to applications.
7. *Service Consumption:* The business & performance logic which exists in the channel platforms needs to be implemented in the services layer in order for the services to be re-usable by the new applications.

Business Benefits

1. Reduced project delivery timeframes and overhead costs.
2. Faster delivery of solutions to production.
3. More frequent changes to production with less risk.
4. Independence from unrelated business functions. Each BU or project can move at the pace that makes sense to them, and change the speed.

Scalable UIs using Responsive design

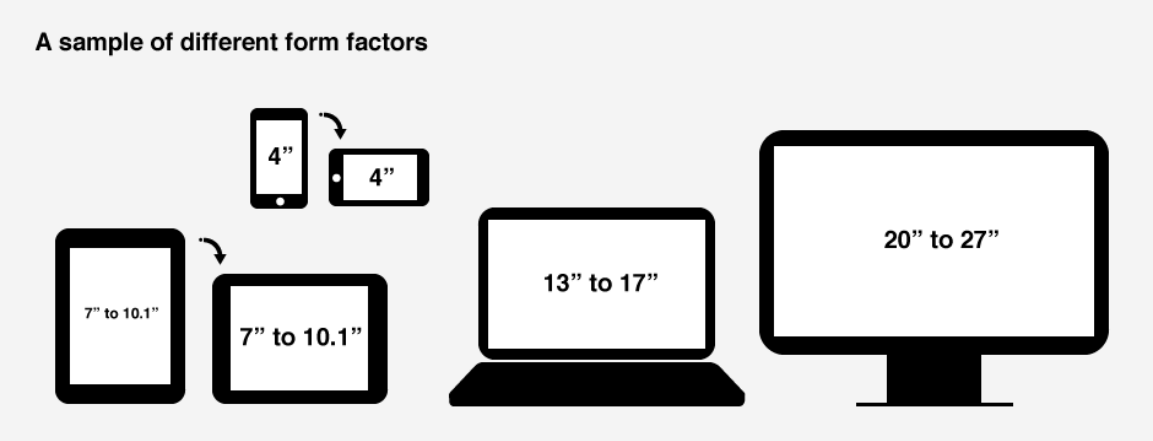
To support the same App being used on multiple devices, the App needs to employ responsive design principles

+

When we say *Responsive Design*, we really mean the following concepts:

1. Touch first design
2. Responsive - scalable (UIs http://en.wikipedia.org/wiki/Responsive_Web_Design)
3. MVC (model view controller) view swapping to target a specific device when css responsive doesn't work
4. Performance optimisations for Mobile first vs tablet first vs touch centric? Benefits / issues
5. Retina graphics - pixel density + network latency
6. Graceful degradation of secondary content - to make use of desktop real estate
7. Designs that work well when a keyboard is on screen or not
8. Whether gestures are supported vs required

Apps are based on Web technology, and therefore Containers & Apps should be designed & built using Responsive Design principles, so that they can be written once & then run in a variety of different form factors.



Responsive Design indicates that a Container and App can dynamically adapt its layout to the viewing environment to best use the available space. Practicality – Responsive Design could be great for low use, low value processes, e.g. account closure - to present a common UI across all form factors, but may not be appropriate for high use, high value processes, e.g. customer UX for payments, which may require specialised UIs for different form factors.This is where the ability to swap out one view for another fits in. More information on Responsive Design can be obtained here.

Building once for multiple user types

Historically, the same business function may have multiple channel implementations (a staff version, an existing customer version, a prospect customer version, a business customer version etc).

Current State

Each business function is currently written per channel; CommSee: Internal Bank Staff, CommBiz: Business Customers and NetBank: Retail Customers.

Target State

Where appropriate, business functions can be designed as Apps that are written once, and deployed to a range of user types and channels. This is suited to business functions with low value being written multiple times because the user experience is essentially the same.

There are a some business functions that we want to have tailored versions unique to subsets of users or particular containers, but there are a large amount that do not warrant being built multiple times. When the user experience is the same for all user types, using the one App in multiple containers would be more efficient. This is typically suited to servicing functions which need to be included in a container to offer breadth of bank capability, or business functions that are less frequently used.

This doesn't mean that the look and feel of the App must be the same in all containers. The same App can appear consistent with other Apps by leveraging the following architectural components:

- Scalable UIs using Responsive design
- the ability to inject the container's visual theming elements
- Security considerations being done at the container or via cross cutting concerns

Solution Implications

1. Portability**:** Business functions need to be built so that they can be consumed by different user types.
2. Theming: The application cannot assume look or feel as the channel should dictate the brand or theme.
3. Technology Neutral: Applications need to be written in a technology and platform neutral manner.
4. Federated Authentication: A common authentication provider model is required for different user types.

Business Benefits

1. Increase reuse of existing code assets
2. Reduce duplication of code (and cost of development and maintenance)

Tailoring to a subset of users and devices

There are times when the business want to provide a more tailored experience to subsets of users, either by who they are, what channel they are on or what device they are using. Examples of this for a personal loan may be:

1. Staff in branches having a different set of questions to fill in for a product origination than specialist call centre staff
2. New to bank prospect customers being asked a different set of questions than existing customers
3. Identified low risk customers being offered preapproved credit products with minimal information being required.

Current State

Channel platforms are designed and build as one large application platform that is comprised of smaller tightly coupled business functions. Business functions are designed with a ‘lowest common denominator’ approach to user experience.

Future State

Channel platforms are assembled or configured from a set of loosely coupled business function packages as apps. Different versions of apps can fulfil the same task enabling tailoring of the business functionality.

The Composite Apps architecture allows multiple Apps to be configured in a container that could fulfil the same business function. Based on logic in the container, the best App to display for each user can be dynamically determined.

Each of the Apps must declare the business services they consume, the other Apps they depend on and where they have been tested to work in the ecosystem. This is done via contracts. If there are two Apps that can fulfil the contract, one App could be substituted for the other.

Substituting one App for another allows A-B testing and experimentation to occur.

Another example scenario is to have a single version of an App for all mobile devices, but create a second tailored version of that App just for Android Tablets.

Solution Requirements

1. Loose Coupling: Apps must be coded so that they are not directly dependent on other applications.
2. Application Dependency Analysis: Business functions that depend on other business functions or services will need to be identified when assembling applications. They also declare which user types, devices or other factors are used to determine the tailoring logic.
3. Theming: The application can elect to only implement parts of a container's theme to allow for a more fine grained tailored experience.
4. Piloting: Multiple versions of a business function can enabled in parallel, with some users running a different version of an application based on business rules.

Business Benefits

1. Reduced project costs to meet business requirements.
2. Improved user experience by:
    a. better serving of customers that currently straddle multiple platforms, eg small business customers, high net worth individuals
    b. Removing cluttered functions that are not appropriate to a given customer

Federated IT capability

Current State

Business functions are developed by a set of centralised teams using a standardised set of technologies.

Future State

Business functions can be built or bought using a range of technologies and be made accessible within the common containers. IT teams can live in any organisation and still create / deploy IT value.

Solution Requirements

1. Classes of Applications: Business function integration can support different levels of compliance to exist within the composite application framework.
2. Documented API's: API's of the composite application framework must be documented so that they can implement or consumed by 3rd party vendors. This will enable interoperability of different applications and containers.

Business Benefits

1. Increased project delivery capacity
2. Business functionality can built using the technology most appropriate
3. Business functionality can be bought and integrated more easily

Outage-less deployments & changes

Current State

The majority of production changes, regardless of the functionality impacted or degree of change requires a full channel outage for the code deployment and business verification of that code change.

Future State

Most production changes will be a 2 step process:

1. Code deployment of business functionality to production, and
2. Activation or enabling the business functionality to a sub set or all end users of the channel.

Changes to back end systems which are not able to support versioning or side by side deployment will require some outage. Partially outages will be possible as business functions which are dependent on those back end systems will be able to be turned off or disabled.

Solution Requirements

1. Application Versioning: All applications must support versioning, backwards compatibility and side by side deployment.
2. Services Versioning: All API's and SOA services must support versioning, backwards compatibility and side by side deployment.
3. Scripted Deployment: Publishing of code to production is separated from consumption. Application code deployments need to be 100% scripted with no human intervention.
4. Application Dependency Analysis: Business functions impacted by changes to SOA services which cannot be versioned will result in business functionality that needs to be regression tested and disabled during the deployment.
5. Piloting: Multiple versions of a business function can enabled in parallel, with some users running a different version of an application based on business rules.
6. Certifying Authorities: Apps can be certified for different types of quality assurance, either before being deployed or after deployment as part of piloting criteria.

Business Benefits

1. High availability of the online channels of key functionality during planned outages.
2. More frequent changes to production with less risk.

Legacy integration

Current State

Business functionality is built targeting only one channel and/or form factor.

Future State

New business functionality is built using the new framework and is able to work side by side with the legacy channels, first by augmenting the existing platform with Apps and then by replacing it with a new container that can call legacy apps.

Solution Requirements

1. Application Classes: Legacy business functions shouldn't need to change to operate within the composite application framework, however some of the key business objectives won't be realised until those business applications are re-designed and implemented with a high level of compliance with the composite application framework.
2. Documented Standardised API's: Legacy channels will need to build integration to the new framework following the API's of the composite application framework to support hosting the new applications inside those channels.

Business Benefits

1. Cost avoidance as there is no need to invest in rebuilding existing assets to work with new framework.
2. Faster time to market with new framework.

Building once for multiple devices and form factors

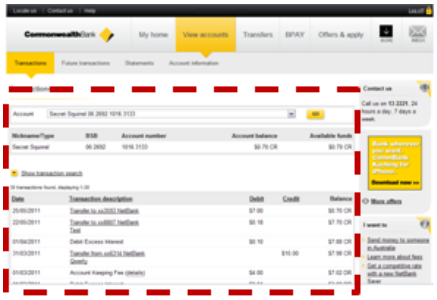
The industry has a trend of an increasing set of mobile devices and form factors. We need to be in a position to exploit the next-wave of consumer devices without writing our code.

Current State

Each business function is currently written to target a particular device (web, desktop or mobile) or screen size or resolution.

Target State

Each business function is designed to be written once and consumed on a range of devices with different form factors, including the devices which haven't been invented yet.



Solution Implications

1. Touch-first design: Business functions which required the use of a keyboard can't easily be used on mobile or touch devices. Business functions which are designed for touch can easily be used on devices which have a keyboard and mouse.
2. Responsive Design principles: Business functions need to designed and written in a way that is able to respond to the size and orientation of the screen.
3. Technology Neutral: Applications need to be written in a technology and platform neutral manner.

Business Benefits

1. Reduced project costs by minimising duplication of code (waste)
2. Maintaining breadth of bank on multiple devices.
3. Better UI design for improved user experience

Editing or commenting on this Wiki

Please feel free to edit this wiki, or leave comments on any page. This site will become a living resource for development within the new Helix framework - for the Coding Comp and far beyond.

To edit a page or add a comment, please sign up to Confluence.

We will be monitoring comments on all pages & where appropriate we will attempt to reply back as soon as possible.