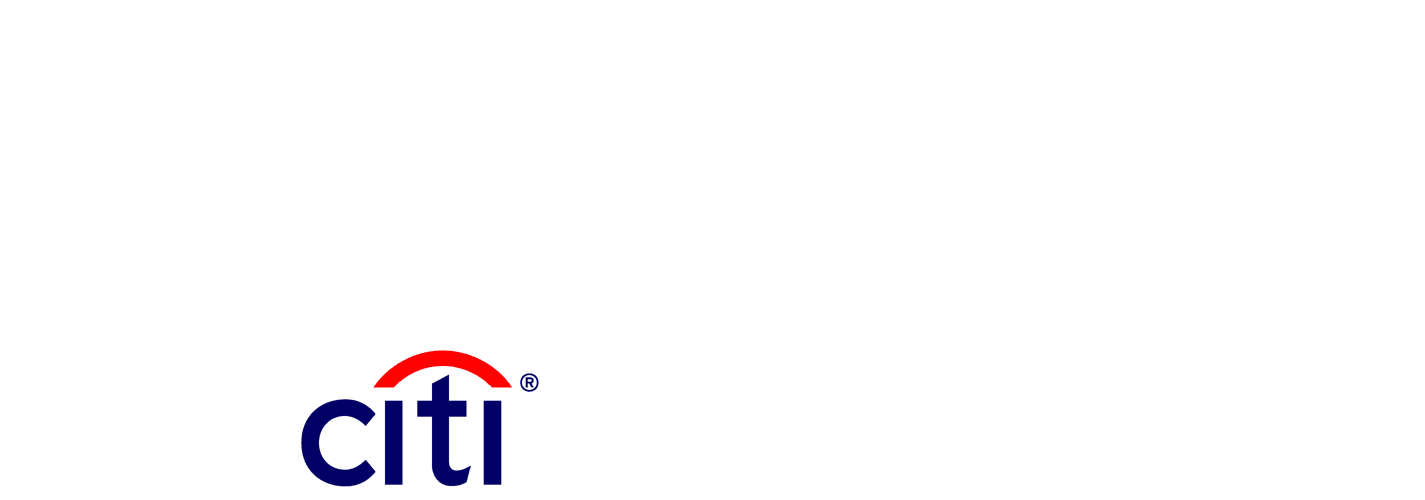
Volume

1

Cross CHANNEL CONVERGENCE ARCHITECTURE

Platform

C3 Cookbook Reference Guide

Cross channel CONVERGENCE architecture

C3 Cookbook Reference Guide

**Change History**

| **Date of Update** | **Document Version** | **Update Description/ High-Level Change** | **Author** |
| --- | --- | --- | --- |
| March 23 2015 | 0.1 | Initial version | Raj Gurupackiam |

© Citicorp USA

Table of Contents

[Table of Contents i](#_Toc424856780)

[Part 1 Introduction and Overview 1](#_Toc424856781)

[Chapter 1 Introduction 1](#_Toc424856782)

[1.1 Document Conventions 2](#_Toc424856783)

[1.2 C3 Architecture Overview and Design Goals 2](#_Toc424856784)

[Chapter 2 **Setting up a Developer Runtime Environment** 3](#_Toc424856785)

[2.1 Webstorm IDE 3](#_Toc424856786)

[2.2 **NodeJS with NPM Installer** 4](#_Toc424856787)

[2.3 **Web Server** 4](#_Toc424856788)

[2.4 Grunt JS 4](#_Toc424856789)

[2.5 **Web Browser** 4](#_Toc424856790)

[2.6 Debugging 4](#_Toc424856791)

[Chapter 3 Single Page Application Overview 5](#_Toc424856792)

[3.1 Overview of Single Page Application 5](#_Toc424856793)

[3.2 Key requirements addressed in C3 Single Page Application 8](#_Toc424856794)

[Part 2 WRITING C3 APPLICATIONS 9](#_Toc424856795)

[Chapter 4 Where to start with C3 applications 9](#_Toc424856796)

[4.1 Copying an Existing C3 Applications 9](#_Toc424856797)

[4.2 C3 Starter Kits 10](#_Toc424856798)

[4.3 Create Your Own C3 Applications without using generator tool 10](#_Toc424856799)

[4.4. Create Directory Structure for C3 Applications 11](#_Toc424856800)

[4.5 Starting point for C3 Applications 12](#_Toc424856801)

[Chapter 5 C3 Components 13](#_Toc424856802)

[5.1 C3.Bootstrap 13](#_Toc424856803)

[5.2 C3.Application 15](#_Toc424856804)

[5.4 Router 15](#_Toc424856805)

[5.4.1 C3.AppRouter 16](#_Toc424856806)

[5.4.2 C3.SubRouter 18](#_Toc424856807)

[5.5 Controller 18](#_Toc424856808)

[5.5.1 C3.Controller 19](#_Toc424856809)

[5.5.2 C3.BaseController 21](#_Toc424856810)

[5.5.3 C3.ModuleController 21](#_Toc424856811)

[5.5.4 C3.SubAppController 24](#_Toc424856812)

[5.5.5 C3.PageController 28](#_Toc424856813)

[5.6 C3.SubApp 31](#_Toc424856814)

[5.7 C3.Component 32](#_Toc424856815)

[5.8 C3.Region 33](#_Toc424856816)

[5.9 C3.Layout 34](#_Toc424856817)

[5.10 C3.Error 37](#_Toc424856818)

[5.11 C3.Model 38](#_Toc424856819)

[5.12 C3.SessionModel 39](#_Toc424856820)

[5.13 C3.Collection 41](#_Toc424856821)

[5.14 C3.View 42](#_Toc424856822)

[5.15 C3.FlowBaseView 44](#_Toc424856823)

[5.16 C3.Helpers 54](#_Toc424856824)

[5.16.1 Text Helper 55](#_Toc424856825)

[5.16.2 Hidden Helper 57](#_Toc424856826)

[5.16.3 Password Helper 58](#_Toc424856827)

[5.16.4 Button Helper 59](#_Toc424856828)

[5.16.5 Radio Helper 60](#_Toc424856829)

[5.16.6 Checkbox Helper 62](#_Toc424856830)

[5.16.7 Date Helper 64](#_Toc424856831)

[Chapter 6 Require vs Define 66](#_Toc424856832)

[6.1 RequireJS Module Loader Syntax 66](#_Toc424856833)

[Chapter 7 SEO RENDERING 71](#_Toc424856834)

[7.1 Pre Login Pages 71](#_Toc424856835)

[Chapter 8 Definition vs. Instantiation of Components 72](#_Toc424856836)

[8.1 Define vs Instantiation 72](#_Toc424856837)

[Part 3 Design Patterns 75](#_Toc424856838)

[Chapter 9 C3 Design Patterns 76](#_Toc424856839)

[9.1 C3 Design Patterns 76](#_Toc424856840)

[9.2 Simple Layout Design Pattern - Approach 76](#_Toc424856841)

[9.3 Mashup(Composite) Design Patterns – Approach 79](#_Toc424856842)

[9.4 Transaction Flow Design Patterns – Approach 80](#_Toc424856843)

[9.5 Singleton View Design Patterns – Approach 82](#_Toc424856844)

[9.6 Module Design Patterns – Approach 84](#_Toc424856845)

[Chapter 10 Navigation Related Tasks 91](#_Toc424856846)

[10.1 **Navigation Related Task** 91](#_Toc424856847)

[10.2 How to think about Routing 91](#_Toc424856848)

[10.3 Push State - Approach 92](#_Toc424856849)

[10.4 Push State - Approach 93](#_Toc424856850)

[10.5 Navigate using Events - Approach 94](#_Toc424856851)

[10.6 Handling Anchor Tags 94](#_Toc424856852)

[10.7 Back/Cancel Link – Handling 95](#_Toc424856853)

[10.8 Sequence – Handling 96](#_Toc424856854)

[10.9 Transactional Locking 96](#_Toc424856855)

[Chapter 11 MFA and Unauthorized Features 97](#_Toc424856856)

[11.1 MFA and Unauthorized Page – Handling 97](#_Toc424856857)

[Chapter 12 Site Catalyst and Anti Malware 98](#_Toc424856858)

[12.1 Site Catalyst Tags - Defined 98](#_Toc424856859)

[12.2 Anti Malware Tags – Requirements and Handling 100](#_Toc424856860)

[Chapter 13 C3 Content Service 104](#_Toc424856861)

[13.1 C3 Content Service - Enablement 104](#_Toc424856862)

[13.2 C3 Content Service – Content to Return 105](#_Toc424856863)

[13.3 C3 Content Service – Debugging Approach 105](#_Toc424856864)

[13.4 C3 Content Service – Configurations 107](#_Toc424856865)

[13.5 C3 Content Service – Top Navigation Menu 108](#_Toc424856866)

[13.6 C3 Content Service – Left Navigation Menu 110](#_Toc424856867)

[13.7 C3 Content Service – Branding 112](#_Toc424856868)

[13.8 C3 Content Helpers 112](#_Toc424856869)

[Chapter 14 C3 Display Related Tasks 116](#_Toc424856870)

[14.1 HandleBar – Template 116](#_Toc424856871)

[14.2 Handlerbars – C3 117](#_Toc424856872)

[14.3 Handlebars and Partial - Intro 127](#_Toc424856873)

[14.4 Regions vs Layouts 131](#_Toc424856874)

[14.5 Masking 132](#_Toc424856875)

[14.6 Form Validation 135](#_Toc424856876)

[Chapter 15 C3 Widgets 140](#_Toc424856877)

[15.1 C3 Accordian Widgets 140](#_Toc424856878)

[15.2 C3 Tooltip Widget 143](#_Toc424856879)

[15.3 C3 Progress Bar Widget 151](#_Toc424856880)

[15.4 C3 OveryLay Widget 156](#_Toc424856881)

[15.5 C3 DatePicker Widget 162](#_Toc424856882)

[Chapter 16 Events and Messaging 178](#_Toc424856883)

[16.1 C3 Events 180](#_Toc424856884)

[16.2 C3 Model Events 181](#_Toc424856885)

[16.3 C3 Router Events 181](#_Toc424856886)

[16.4 C3 View Events 181](#_Toc424856887)

[16.5 C3 Model Events – Usage 184](#_Toc424856888)

[16.6 C3 View Events – Usage 192](#_Toc424856889)

[16.8 Event Aggregator 203](#_Toc424856890)

[16.9 C3 Radio 206](#_Toc424856891)

[16.10 C3 Command Object 207](#_Toc424856892)

[16.11 C3 Request/Response Object 208](#_Toc424856893)

[16.12 Decision Tree 210](#_Toc424856894)

[Chapter 17 Models and Collections 212](#_Toc424856895)

[17.1 C3.Model Initialize 212](#_Toc424856896)

[17.2 C3 Model Defaults 213](#_Toc424856897)

[17.3 C3 Model Attribute Values 213](#_Toc424856898)

[17.4 C3 Model Changing Attribute Values 214](#_Toc424856899)

[17.5 C3 Model Deleting Attribute Values 215](#_Toc424856900)

[17.6 C3 Cloning Models 216](#_Toc424856901)

[17.7 C3 Model – Handling CRUD 217](#_Toc424856902)

[17.8 C3 Model – Validation 218](#_Toc424856903)

[17.9 C3 Model – Saving 222](#_Toc424856904)

[17.10 C3 Model – Reset Data 223](#_Toc424856905)

[17.11 C3 Model – Delete 224](#_Toc424856906)

[17.12 C3 Model – Parsing Server Response 224](#_Toc424856907)

[17.13 C3 Collection 225](#_Toc424856908)

[17.14 Handling Data Latency 227](#_Toc424856909)

[18.1 C3 tool to generate css using less 229](#_Toc424856910)

Part

1

Part 1 Introduction and Overview

Chapter 1 Introduction

This Chapter provides an introduction to the C3 Platform (JFP). The chapter provides an overview of C3s initial business purpose, design goals, problem space, development and reuse approach, as well as a summary of the major requirements.

t

his document act as a reference guide for explaining the use and implementation strategies for the design patterns for Cross Channel Convergence Architecture (C3).The focus of this document is an architecture overview of the JFP framework and a description of the requirements and approach for each of the major component

This document is intended to assist C3 application developers by documenting how some common tasks are performed. This document is not a ‘from-scratch’ step-by-step tutorial for someone who knows nothing of Marionette or the basic technology stack behind the C3 Architecture. It is a topic-by-topic description of how to do things in a C3 application for a developer who basically understands Marionette and C3 Tech Stack concepts but wants details on how to accomplish a specific task.

This document does not address the ‘whys’ and ‘wherefores’ but tries to stick to the ‘how’s’. The References section is intended to help you find the documents which do describe the ‘whys’ and ‘wherefores’ and detailed design of the different C3 mechanisms which have been implemented to help accomplish various tasks.

This document is a snapshot in time. Changes/refinements are still being made to the infrastructure and new ‘standards’ are being established. Over time, these should be incorporated in this document, but we are not documenting things ahead of time and the maintenance of this document is a part-time effort. Therefore, do not be surprised if you find that some tasks descriptions here have been super ceded by a new approach.

We have tried to review each of the chapters with multiple developers.

icon key

* Requirement
* Best Practice
* Design Pattern

1. Definition

* Approach
* Key Concept
* Instancing
* Reuse - Leverage

Throughout this document certain key concepts and information will be highlighted using the Icon Key described at the left..

# 1.1 Document Conventions

This cookbook contains recipes that guide a beginning programmer to a deeper knowledge of the subject by answering specific, "how-to" style questions.

These Cookbook recipes address more topics than API documentation for a class, but are smaller in scope than a topic-based guide.

Each recipe has been given its own chapter following the same format as below:

**Title**

Broadly explains the topic of the recipe

**Problem**

Outlines the recipe's goals

**Solution**

Summarizes the correct approach to addressing the problem

**Discussion -** Explores the solution in detail

# 1.2 C3 Architecture Overview and Design Goals

<https://globalconsumer.collaborationtools.consumer.citigroup.net/sites/PlatformsDevelopment/Portal/HTML5%20Cross%20Channel%20Convergence/Ref%20App%20Dev/Cookbook/Cross%20Channel%20Convergence%20Application%20Cookbook%20v1.5.docx>

Chapter 2 **Setting up a Developer Runtime Environment**

This chapter describes Setting up developer runtime environment.

t

his chapter describes how to set up the developer runtime environment to develop a C3 Applications. C3 requires and recommend the following Developer Runtime Environment.

1. Web storm IDE – CATE Approved and available at Market Place
2. NodeJS with NPM Installer
3. Web Server
4. Grunt JS
5. Web Browser

# 2.1 Webstorm IDE

C3 recommends making use of Web storm IDE for javascript development. Webstorm IDE is CATE approved and can be installed from Market Place.

<https://globalconsumer.collaborationtools.consumer.citigroup.net/sites/PlatformsDevelopment/Portal/Platform%20Rearchitecture/NGA/Technical%20Speaker%20Series%202015/JFP_NGA_IDE_WebStorm_Eclipse_v1.0.docx>

Please notice that we recommend to use WebStorm in local machine to do all C3 related development:

* JSHint: Javascript static analysis
* YUIDoc: generate Javascript document
* Mocha/Chai: Javascript unit testing
* Istanbul: calculate Javascript unit test coverage

This sample project shows how to use JSHint/YUIDoc/Mocha/Chai/Istanbul in WebStorm:

<https://globalconsumer.collaborationtools.consumer.citigroup.net/sites/PlatformsDevelopment/Portal/Platform%20Rearchitecture/NGA/Technical%20Speaker%20Series%202015/WebStorm_SampleProject.zip>

# 2.2 **NodeJS with NPM Installer**

All the required NodeJS modules can be installed via NPM installer. NPM Installer comes with NodeJS installation.

Note: NodeJS Modules can be shared across projects. It’s not required to install NodeJS in every project.

# 2.3 **Web Server**

C3 recommends using node/express built inside the Webstorm IDE for development.

For Testing and Production – Developers can use the web server of their choice

# 2.4 Grunt JS

C3 requires and recommend installing GruntJS in local for doing build and automation activities.

.

# 2.5 **Web Browser**

C3 recommends Citi supported web browser for testing.

# 2.6 Debugging

For Debugging in local, C3 recommend to make use of Webstorm supported debugging. Please follow the below link to understand the same.

https://www.jetbrains.com/webstorm/help/running-and-debugging-javascript.html

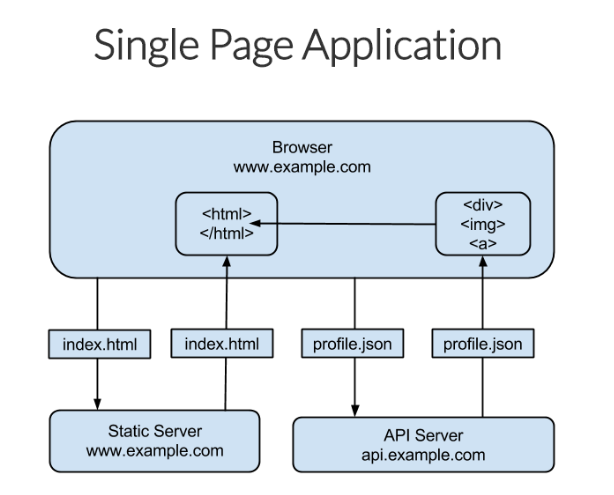
Chapter 3 Single Page Application Overview

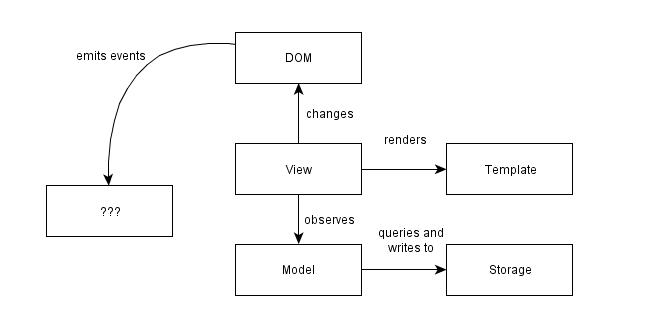
This chapter gives an overview of the Single Page Application pattern

t

his chapter provides a “big picture” view of the Single Page Application Pattern.

# 3.1 Overview of Single Page Application





Dispatcher

**Write-only DOM**: No state / data are read from the DOM. The application outputs HTML and operations on elements, but nothing is ever read from the DOM. Storing state in the DOM gets hard to manage very quickly: it is much better to have one place where the data lives and to render the UI from the data, particularly when the same data has to be shown in multiple places in the UI.

**Models as the single source of truth**: Instead of storing data in the DOM or in random objects, there is a set of in-memory models which represent all of the state/data in the application.

**Views observe model changes**. We want the views to reflect the content of the models. When multiple views depend on a single model (e.g. when a model changes, redraw these views), we don't want to manually keep track of each dependent view. Instead of manually tracking things, there is change events system through which views receive change notifications from models and handle redrawing themselves.

**Decoupled modules that expose small external surfaces**: Instead of making things global, we should try to create small subsystems that are not interdependent. Dependencies make code hard to set up for testing. Small external surfaces make refactoring internals easy, since most things can changes as long as the external interface remains the same.

**Minimizing DOM dependent-code**. Why? Any code that depends on the DOM needs to be tested for cross-browser compatibility. By writing code in a way that isolates those nasty parts, a much more limited surface area needs to be tested for cross-browser compatibility. Cross-browser incompatibilities are a lot more manageable this way. Incompatibilities are in the DOM implementations, not in the Javascript implementations, so it makes sense to minimize and isolate DOM -dependent code.

**Controllers is not typical server side controllers**

Single page have more complex state transitions than a server-side app:

* there are DOM events that cause small state changes in views
* there are model events when model values are changed
* there are application state changes that cause views to be swapped
* there are global state changes, like going offline in a real time app
* there are delayed results from AJAX that get returned at some point from backend operations

These are all things that need to be glued together somehow, and the word "Controller" is not same as Server side controller to act as coordinator for all these things.

We clearly need a model to hold data and a view to deal with UI changes, but the glue layer consists of several independent problems.

**Routing and Navigation happens at Client Side:**

Non-SPA relies heavily on server side for the next page display and navigation. Non-SPA mostly makes the client as dumb and handles most of the processing and business logic at server side.

SPA minimize the roundtrip to server by offloading the data to client side and allow us to build processing and to some extend business knowledge at client side to handle the next view display. Customers will feel the fluidity of page rendering since all the navigation and routing is handled at the client side.

# 3.2 Key requirements addressed in C3 Single Page Application

1. Rendering
2. Routing
3. Browser History
4. State Management
5. Search Engine Optimization
6. Browser Compatibility
7. API Integration
8. Configuration

Each of the requirements mentioned above will be discussed in detail in below sections

Part

2

Part 2 WRITING C3 APPLICATIONS

Chapter 4 Where to start with C3 applications

This chapter gives an idea about the initial start to write C3 Applications

There are 4 ways to start working on a C3 Applications:

* Copy an existing C3 Applications and modify
* Copy one of the starter kit examples and modify
* Create a C3 Applications using the Generator tool
* Create one from scratch without using the tool.

# 4.1 Copying an Existing C3 Applications

This implies you have all the source code and configuration files for applications. If so, you have a very rich base to work with. Review the existing app and when you are ready to start modifying it, read through the steps in *Create Your Own C3 Application* section.

# 4.2 C3 Starter Kits

There exist several ‘starter kits’ which provide the basic skeleton of a C3 application.

Each one demonstrates a different design pattern for a C3 applications ranging from simple (simple transitions, welcome page) to complex (involves view, model and controller helpers as well as external communications, sequence protection ).

The kits can be found at the following locations:

Refer to RTC examples from the platform handoff

Whether or not you use any of these as your base, we recommend you try installing/running them and inspecting the files provided to understand how they work.

Once you understand the samples, read through the following section for how to start modifying these to make your own applications.

# 4.3 Create Your Own C3 Applications without using generator tool

Another alternative is to create your own app from scratch without using the tool. We assume you have a Functional Spec or APS (Application Product Specification) for the app you want to create.

Below sections will discuss in detail about the same.

# 4.4. Create Directory Structure for C3 Applications

The below directory structure is advised for the development. Please double click the below Visio diagram to get the full view.



**Application** – root directory for your application to reside.

**build** – a directory to hold all your precompiled files like Template.

**test** – a directory holds mocha/chai test suites

**Gruntfile.js** – the tasks to compile/run the application.

The **Application** directory has these subdirectories and files:

**src** – a directory holds the application level resources

**modules** – a directory holds any modules the application needs to have. Each module can have controllers, layouts, models, routes, templates, views, and 0 to many sub apps. The module entry point is defined in the <module\_name>-module.js script. Again, each sub app can have its own controllers, layouts, routes, models, templates, views.

* controllers – a directory holds application level controllers, normally we name it as app-controller.js
* layouts – a directory holds application level layout.
* models – a directory holds application level model objects or collection objects.
* routes – a directory holds application level routers, normally name it as app-router.js.
* templates – a directory holds app/top level template (such as header/footer)
* views – a directory holds top/app level views
* utils – a directory is optionally to hold any mixins or application level utilities

**images** – a directory holds all images the application will use

**styles** – a directory holds all CSS style sheets the application will use.

**index.html** – this is the static HTML DOM definition that will serve for the single page application.

# 4.5 Starting point for C3 Applications

It’s recommended to develop C3 Application as a single entry point application and avoid multiple entry points.

Each entry points is defined by App.start() present inside main.js(note: name can be anything for this file, we are following a convention to ensure it reflect the purpose)

App – is the object that extends from C3.Application

This main.js file will be the first file loaded by index.html or any starter html file.

Apart from doing the App.Start(), main.js file also sets the necessary platform and application specific configuration and libraries that will be used by the application.

Main.js file should require C3.Bootstrap.js file to do the above.

Chapter 5 C3 Components

This chapter gives a detailed overview of components that make up the C3 Applications.

t

his chapter gives a detailed overview of components that make up the C3 Applications.

# 5.1 C3.Bootstrap

* C3.Bootstrap is the first platform component that comes into play for any C3 based applications.
* C3.Bootstrap will help to configure platform and application specific configuration and libraries
* Following are API’s available on c3.bootstrap object.
  + API to initialize the application:
    - setupInDevMode/setupInTestMode/setupInProdMode() : informing bootstrap about the mode(DEV/TEST/PROD), before initializing, so that it can fetch corresponding data and config. By default PROD mode is enabled.
    - initialize() : It fetches platform specific configuration and data, according to the mode, the bootstrap is in.
  + Application specific setter methods:
    - setApplicationConfig(): populate application specific configuration. These are the require js configurations for the application modules.
    - setApplicationData(): populate application specific data. These are the data, which are injected to the applications, before starting it up.
    - setApplicationRule(): populate application specific rule. These rules determine, if we should go ahead to start the application or not.
    - setStartModuleName(): Name of the application module to start up.
  + Application specific getter methods:
    - getApplicationConfig(): get application specific configurations, as contained in bootstrap.
    - getApplicationData(): get application specific data, as contained in bootstrap.
    - getApplicationRule(): get application specific rule, as contained in bootstrap.
    - getStartModuleName(): get application module name, as contained in bootstrap.
  + Platform specific getter methods:
    - getPlatformConfig(): get platform specific configurations, as contained in bootstrap.
    - getPlatformData(): get platform specific data, as contained in bootstrap.
    - getPlatformRule():get platform specific rule, as contained in bootstrap.
  + API to start the application:
    - startApplication():
    - It will
    - set up require js according, to the configurations,
    - injects data,
    - checks rules, if that permits the startup of the application(as defined by the application name), and
    - starts the application.
    - Here configurations, data and rules mean both, platform and applications configurations, data and rules, available to bootstrap.
  + Application teams are expected to call the API’s in the following sequence,
    - Create an instance on c3.bootstrap
    - Call setupInDevMode/setupInTestMode/setupInProdMode(), this is optional, default mode is PROD
    - Call initialize
    - Call application specific setter methods
    - Call startApplication

# 5.2 C3.Application

* The Application is a container for the rest of your code.
* Every C3 Single page app should have one and only one instance of Application.
* Platform Create the instance from C3.Application
* C3.Application is developed on top of Marionette.Application
* Other Components can retrieve the Application instance when needed via controller’s getApplication()

# 5.4 Router

* Reduce the boilerplate code of handling route events and then calling a single method on another object. Have your routers configured to call the method on your object, directly.
* C3 Comes with two different flavors of routers based on where its getting used
  + C3.AppRouter – **Extend directly from Marionette.AppRouter**
  + C3.SubRouter

# 5.4.1 C3.AppRouter

* Used to configure Application Level Routes
* In your application you extend from C3.AppRouter to act as application level router.(Pls see the sample code below)
* Configure a C3.AppRouter with appRoutes. The route definition is passed on to Backbone's standard routing handlers. This means that you define routes like you normally would. However, instead of providing a callback method that exists on the router, you provide a callback method that exists on the controller, which you specify for the router instance (see below.)
* **Dynamic Routing** – C3 allow you to define routes that contain a mix of static and dynamic route parameters.
  + For example you might want to retrieve a marketing page with a variable pageId with a friendly URL string. Such that your URL would look like "http://online.citibank.com/#/marketing/12". Once this route was activated you would want to access the id given in the URL string. This example is implemented below.
* Dynamic Routing Cont. **":params" and "\*splats"** - C3 uses two styles of variables when implementing routes. First there are ":params" which match any URL components between slashes. Then there are "\*splats" which match any number of URL components. Note that due to the nature of a "\*splat" it will always be the last variable in your URL as it will match any and all components.
* Any "\*splats" or ":params" in route definitions are passed as arguments (in respective order) to the associated function. A route defined as "/:route/:action" will pass 2 variables (“route” and “action”) to the callback function
* C3.AppRouter provides you PreRoute and PostRoute function to act like a route Filter.

define([

'c3'

], function(C3) {

'use strict';

return C3.AppRouter.extend({

appRoutes: {

//Landing page

'home': 'landing',

'home?:params': 'landing',

'': 'landing',

//Dashboard

'dashboard/:path': 'dashboard',

//Contact-us page

'contact': 'contactUs',

//pnt/tboa

'pnt': 'pnt',

'pnt/\*subapp': 'pnt',

//Marketing page

'marketing': 'marketingPage',

'marketing/:pageId': 'marketingPage',

//Logout action

'logout': 'logout',

// All Legacy URLs

///////////////////////////////////////////////////////////////////

// Landing page

'CBOL/mkt/welcome/flow.action?:params': 'legacyUrls',

'CBOL/mkt/welcome/flow.action': 'legacyUrls',

// Dashboard

'CBOL/ain/dashboard/flow.action?:params': 'legacyUrls',

// TBOA

'CBOL/pnt/loctfr/flow.action?:params': 'legacyUrls',

// Marketing page

'JRS/pands/detail.do?:params': 'legacyUrls',

// Contact us

'CBOL/contactus': 'legacyUrls'

}

});

});

# 5.4.2 C3.SubRouter

* Used to configure SubApp Level Routes
* In your application you extend from C3.SubAppRouter to act as SubApp level router.(Pls see the sample code below)
* C3.SubAppRouter accepts subRouterctx, variable to define the subapp context name to be used in the URL. This will be used when forming the full qualified routes. Pls see the example below
* C3.SubRouter extends from C3.AppRouter so the Concept of Dynamic routing and Route filters mentioned above holds good for the SubRouter also.

define([

'c3'

], function (C3) {

'use strict';

return C3.SubRouter.extend({

subRouterCtx: 'tboa',

appRoutes: {

}

});

});

# 5.5 Controller

* + - A Controller is an object used in the C3 Router. Controllers are where you store your Router's callbacks.
    - C3 Comes with different flavors of controllers based on where its getting used
      * C3.Controller
      * C3.BaseController
      * C3.ModuleController
      * C3.SubAppController
      * C3.PageController

# 5.5.1 C3.Controller

* Extend directly from Marionette.Controller
* Plain Vanilla Controller, it’s used when we want to use controller as just a router’s callback.
* It comes with two methods preExecute and postExecute
* preExecute - plug point for processing logic prior to executing the actual controller logic
* postExecute - plug point for processing logic after executing the actual controller logic

define([

'c3',

'msgbus',

'modules/common/models/session-model'

],

function(C3, MsgBus, SessionModel) {

'use strict';

var AppController = C3.Controller.extend({

landing: function(params) {

if (SessionModel.get('authenticated')) {

//C3.navigate('dashboard/home');

C3.navigate('pnt/tboa', true);

return;

}

// Load offers section in landing page async.

require(['modules/landing/landing-module']);

},

pnt: function(subapp) {

if (!SessionModel.get('authenticated')) {

C3.navigate('logout');

} else {

require(['modules/pnt/pnt-module']);

}

},

dashboard: function() {

if (!SessionModel.get('authenticated')) {

C3.navigate('logout');

} else {

require(['modules/dashboard/dashboard-module']);

}

},

marketingPage: function(pageId) {

require([

'modules/marketing/marketing-module'

], function() {

MsgBus.reqres.setHandler('get:marketing:pageid', function() {

return pageId;

});

//MsgBus.commands.execute('start:marketing:module');

});

},

contactUs: function() {

},

logout: function() {

require([ 'modules/landing/module/layout/landing-layout'], function(LandingView){

var pageLayout = MsgBus.reqres.request('application:page');

var layout = new LandingView({showSignOff:true});

pageLayout.c3Body.show(layout);

requirejs.undef('modules');

require(['modules/components/offers/offers-comp'], function(OfferModule){

OfferModule(layout);

});

});

},

legacyUrls: function(params) {

// Landing Page

var route = C3.getCurrentRoute();

if (route === '' || route.indexOf('CBOL/mkt/welcome/flow.action') > -1) {

C3.navigate('home?render', {

trigger: true,

replace: true

});

// Dashboard

} else if (route.indexOf('CBOL/ain/dashboard/flow.action') > -1) {

C3.navigate('dashboard/home', {

trigger: true,

replace: true

});

// PnT - TBOA

} else if (route.indexOf('CBOL/pnt/loctfr/flow.action') > -1) {

C3.navigate('pnt/tboa', {

trigger: true,

replace: true

});

// Marketing page

} else if (route.indexOf('JRS/pands/detail.do') > -1) {

C3.navigate(route.replace(/^(JRS\/pands\/detail\.do\?ID=)(\w.+)/g, 'marketing/$2'), {

trigger: true,

replace: true

});

// Contact us

} else if(route.indexOf('CBOL/contactus') > -1) {

C3.navigate('contactus', {

trigger: true,

replace: true

});

}

}

});

// IMPORTANT NOTE:

// As RequireJS maintains the context of all loaded scripts, `new AppController()` is only executed once

// This in fact returns same instance of the AppController every time this script is required (Singleton)

return new AppController();

});

# 5.5.2 C3.BaseController

* Extends from C3 Controller.
* Provide APIs to deal with layouts.
* API’s provided are
  + getLayout
  + showLayout

# 5.5.3 C3.ModuleController

* Used for modules.
* Extend from C3.BaseController.
* A Layout class is required.
* A name variable is required
* Container region is required for Module/SubAppController that shows the layout. It's set to main region by default. Can be overridden using region property or options.region.
* Each Module Controller comes with showLayout Method which can be overridden to handle special use cases eg: show layout as well as default views such as the leftnav.
* It has Module level Radio Channel baked Inside.
* Expose API’s to create Subapp, Model and View to be used for the module. When using this API’s Module Controller create Subapp, Model and view respectively passing its Radio channel.
* It also facilitates to invoke MFA/Unauthorized page based on the error response code of a request.

define([  
 'c3',  
 'modules/radio/module/layout/radio-module-body-layout'  
], function (C3,RadioModuleLayout) {  
 'use strict';  
  
 return C3.ModuleController.extend({  
  
  
 name: 'radio.module.RadioController',  
  
 layoutClass: RadioModuleLayout,  
  
  
 // Initialize function that will be called after instantiating the controller object  
 initialize: function (options) {  
 console.log("Radio controller initialized");  
  
 },  
  
  
 startRadioModule: function () {  
  
 var self = this;  
  
 require(['modules/radio/radiosubapp1/radio-subapp1'], function (RadioSubApp1) {  
  
  
 var radioSubApp1 = self.createSubApp(RadioSubApp1,{region: self.getLayout().RadioSubApp1Information });  
  
 radioSubApp1.start();  
 });  
  
  
  
 require(['modules/radio/radiosubapp2/radio-subapp2'], function (RadioSubApp2) {  
  
  
 var radioSubApp2 = self.createSubApp(RadioSubApp2,{region: self.getLayout().RadioSubApp2Information });  
  
 radioSubApp2.start();  
 });  
  
 }  
  
  
  
 });  
  
});

# 5.5.4 C3.SubAppController

* Used for SubApp
* A Layout class is required.
* A name variable is required.
* Container region is required for Module/SubAppController that shows the layout. It's set to main region by default. Can be overridden using region property or options.region.
* Each SubApp Controller comes with context variable to store data between views if necessary.
* It has subapp level Radio channel baked inside. Pls refer Radio Section to understand more about Radio and it’s channel.
* Expose API’s to create Model and View to be used for the subapp. When using this API’s Subapp Controller create Model and view respectively passing its Radio channel.
* Expose API’s to get module and subapp channel.

define([

'underscore',

'c3',

'modules/radio/radiosubapp1/layout/radiosubapp1-body-layout'

], function (\_, C3, RadioSubApp1FlowLayout) {

'use strict';

return C3.SubAppController.extend({

// Mandantory fields.

name: 'radiosubapp1.radiosubapp1Controller',

layoutClass: RadioSubApp1FlowLayout,

initialize: function () {

console.log('RadioSubapp1 controller initialized');

},

// Defines the entry point for subapp

start: function () {

this.\_radioSubApp1Start();

},

\_radioSubApp1Start: function () {

var self = this;

console.log('Radiosubapp1 controller:show');

// Instantiate the view

require([

'modules/radio/radiosubapp1/views/radiosubapp1-success-view1',

'modules/radio/radiosubapp1/views/radiosubapp1-success-view2',

'modules/radio/radiosubapp1/models/radiosubapp1-model1'

], function (RadioSubApp1View1, RadioSubApp1View2, RadioSubApp1Model1) {

RadioSubApp1Model1.enableChannel(self.channel);

var RadioSubApp1View1 = self.createView(RadioSubApp1View1,{model: RadioSubApp1Model1} );

var region1 = self.getLayout().RadioSubApp1View1Information;

region1.show(RadioSubApp1View1);

self.channel.on("SubApp1:View1:Event2", function()

{

var moduleChannel = self.getModuleChannel();

moduleChannel.trigger("SubApp1:View1:Event2");

}

);

self.channel.comply("SubApp1:View1:Command2", function()

{

var moduleChannel = self.getModuleChannel();

moduleChannel.command("SubApp1:View1:Command2");

}

);

self.channel.reply("SubApp1:View1:Request2", function()

{

var moduleChannel = self.getModuleChannel();

return moduleChannel.request("SubApp1:View1:Request2");

}

);

self.getModuleChannel().on("SubApp2:View1:Event2", function()

{

self.channel.trigger("SubApp2:View1:Event2");

}

);

self.getModuleChannel().comply("SubApp2:View1:Command2", function()

{

self.channel.command("SubApp2:View1:Command2");

}

);

self.getModuleChannel().reply("SubApp2:View1:Request2", function()

{

return self.channel.request("SubApp2:View1:Request2");

}

);

var RadioSubApp1View2 = self.createView(RadioSubApp1View2,{model: RadioSubApp1Model1} );

var region2 = self.getLayout().RadioSubApp1View2Information;

region2.show(RadioSubApp1View2);

self.getModuleChannel().on("SubApp2:View2:Event2", function()

{

self.channel.trigger("SubApp2:View2:Event2");

}

);

self.getModuleChannel().comply("SubApp2:View2:Command2", function()

{

self.channel.command("SubApp2:View2:Command2");

}

);

self.getModuleChannel().reply("SubApp2:View2:Request2", function()

{

return self.channel.request("SubApp2:View2:Request2");

}

);

self.channel.on("SubApp1:View2:Event2", function()

{

var moduleChannel = self.getModuleChannel();

moduleChannel.trigger("SubApp1:View2:Event2");

}

);

self.channel.comply("SubApp1:View2:Command2", function()

{

var moduleChannel = self.getModuleChannel();

moduleChannel.command("SubApp1:View2:Command2");

}

);

self.channel.reply("SubApp1:View2:Request2", function()

{

var moduleChannel = self.getModuleChannel();

return moduleChannel.request("SubApp1:View2:Request2");

}

);

self.listenTo(RadioSubApp1View1, 'radiosubapp1:customer:data', self.\_dashCustomerData);

});

},

\_dashCustomerData: function(data) {

}

});

});

# 5.5.5 C3.PageController

* Used for page level controller, when we want to pre determine the subapps to be loaded once the module route is being called. This controller is helper when we want to configure subapps which can be shifted in/out in tander with business needs with minimal changes in code. Just change the config( shown below) and your change is being picked up at runtime.
* The config specifies which subapp/view needs to be a loaded to which region.
* The application team needs to specify the region where they want to pre populate the subapp or a view.
* payeeManagement:{"moduleMap":[
* {
* "region":"row\_1\_col\_2",
* "subapp":"modules/payeeManagement/addPayee/addPayee-subapp"
* },
* {
* "region":"row\_1\_col\_1",
* "view":"modules/payeeManagement/payeeOffers/views/payee-offers"

This specifies that first column of first row, will get populated with payee-offers view and second column of first row gets populated with addPayee-subapp.

The same regions can be populated with different view/subapp for some other business id,

This allows the flexibility to configure a page based on brands, when some modules needs not to be loaded for some pages.

* A Layout class is required.
* A name variable is required.
* Each Page Controller comes with context variable to store data between views if necessary.
* It has controller level Radio channel baked inside. Pls refer Radio Section to understand more about Radio and it’s channel.

Sample page controller

define([

'c3'

],function(C3) {

'use strict';

return C3.PageController.extend({

// A name is not required, however, it's highly recommend for debugging purpose

// As javascript doesn't have classname similar to other language such as java

pageName: 'payeeManagement',

name : 'payeeManagement',

// Mandantory fields.

// They can be set either 1. here in class definition or

// 2. in constructor -> new ModuleController({layoutClass: PnTLayout, region: MsgBus.reqres.request('main:region')})

layoutClass: {

"className": C3.LayoutStore.BaseLayout,

"params" : {

"subHeader":true,

"subFooter":false,

"tableMap":[{"rowVal":[{"size":4,"SM":false,"XS":false},{"size":8,"SM":true,"XS":true}]}]

}

},

region: C3.Utils.MsgChannel.getApplicationChannel().request('application:page').c3Body,

// Initialize function that will be called after instantiating the controller object

initialize: function(options) {

this.region = this.getApplicationChannel().request('application:page').c3Body;

this.region.show(this.getLayout());

},

pageStart: function(){

console.log('hello am in page start');

},

startManagePayee: function(){

//insert code here for page controller

},

startAddPayee: function(){

this.loadStart();

console.log('hello');

}

});

});

Sample configuration for app config based on busid

busMap:{

BUSID:{

contentVersion:'v1',

default\_lang:'en\_GB',

supported\_langs:['en\_GB','es\_GB'],

pageConfigs : {

payeeManagement:{"moduleMap":[

{

"region":"row\_1\_col\_2",

"subapp":"modules/payeeManagement/addPayee/addPayee-subapp"

},

{

"region":"row\_1\_col\_1",

"view":"modules/payeeManagement/payeeOffers/views/payee-offers"

}

]

}

}

},

AUGCB:{//insert code for another locale

}

}

# 5.6 C3.SubApp

To enable a Modular design and development for Citi apps, an approach has been prescribed that allows for developers / teams to create their modules independently and integrate with the main app when ready.

This allows for functionalities to be added to the main app in a highly decoupled manner.

C3 provides modularization via SubApps and modules

C3.SubApp comes with the following features

* C3.SubApp inherits from [Marionette.Object](http://marionettejs.com/docs/v2.1.0/marionette.object.html), so all of its methods are available too
* Maintain the lifecycle of the subapp
* Provide sanity check like getting the controller and associating the controller for the subapp.
* Every SubApp Initialization happen from module controller.
* Module controller does this via one of it’s route invocation methods.
* Therefore every SubApp will have a route associated with it.
* Provide API’s like onBeforeStart and onStart to associate logic before and on starting of the subapp.

Pls refer to the section “How to write Modules and Subapps” to know more in detail about modularizing the C3 Code.

define([

'modules/pnt/tboa/controllers/tboa-controller'

], function(TboaController) {

return C3.SubApp.extend({

name: 'pnt.tboa.TboaSubApp',

// Mandantory field

// SubAppController class need to be defined

controllerClass: TboaController,

// By default, start() function[platform code] calls the controller.start() and start the subapp.

// however the start function can be overridden for specific usecases

// optional function some prework is required onBeforeStart/onStart

onBeforeStart: function() {

console.log('~~~~~~~~~Before Strating TBOA subapp~~~~~~~~');

},

onStart: function() {

console.log('~~~~~~~~~Strating TBOA subapp~~~~~~~~');

}

});

})

# 5.7 C3.Component

* Components are similar to Subapp which holds its own set of controllers, models and views except that it don’t have any routes associated with it.
* A Component can be initialized and used anywhere, where as a subapp can be initialized only via its route’s methods present in the controller.
* A component like a subapp, require its own controller to do its job of mediation between model and views.
* A component can make use of a subapp controller when it need of a controller.
* C3 provides two types of components
  + C3.BaseComponent
  + C3.UIComponent
* BaseComponent can be used as reusable libraries that don’t need itself to be associated with any regions.
* UIComponent can be used as reusable libraries that need a region to associate itself.
* Pls check below on more information about regions.

define([  
 'platform/apps/components/error-component/controller/error-controller'], function(ErrorController) {  
 'use strict';  
  
 return C3.UIComponent.extend({  
 name: 'error',  
 controllerClass: ErrorController,  
 onBeforeStart: function() {  
 },  
 onStart: function() {  
 }  
   
 });  
});

# 5.8 C3.Region

* C3 Regions provide consistent methods to manage show and destroy views in your applications and layouts.
* They use a jQuery selector to show your views in the correct place.
* Using the C3.Layout (mentioned below) class you can create nested regions.
* Developed on top of Marionette.Region, it provides all the functionality Marionette.Region has to offer <http://marionettejs.com/docs/v2.4.1/marionette.region.html>

define([

'jquery',

'underscore',

'c3',

'templates',

'msgbus'

], function ($, \_, C3, JST, MsgBus) {

'use strict';

return C3.View.extend({

template: JST['modules/landing/templates/landing.hbs'],

events:{

'click #citi\_apps\_landing\_heroMessage\_hero1signOn': 'openMenu'

},

showSignOff: false,

initialize: function (options) {

var OffersRegion = new C3.Region({

el: '#cbolui\_apps\_landing\_offers'

});

MsgBus.reqres.setHandler('offers:landing:region', function(){

return OffersRegion;

});

if(options && options.showSignOff){

this.showSignOff = true;

}

$('body').on('keyup', function(event) {

if (event && event.which === 27 && $(this).hasClass('citi-layouts-menuActive')) {

$('#cbolui\_branding\_menuBtn').trigger('click');

}

});

},

render: function () {

this.$el.html(this.template({showSignOff: this.showSignOff}));

},

openMenu: function(event){

event.preventDefault();

$('#cbolui\_branding\_menuBtn').trigger('click');

}

});

});

# 5.9 C3.Layout

* A Layout is a hybrid of a View and a collection of Region objects.
* They are ideal for rendering application layouts with multiple sub-regions  
  managed by specified region managers.
* A layout can also act as a composite-view to aggregate multiple  
  views and sub-application areas of the screen allowing applications to  
  attach multiple region managers to dynamically rendered HTML.
* You can create complex views by nesting layout managers within Regions.
* C3 Provides Layout store, from which you can get two distinct Layouts BaseLayout and PageLayout

define([

'c3',

'msgbus',

'layouts',

'modules/common/routes/app-router',

'modules/common/controllers/app-controller',

'modules/common/views/footer-view',

'platform/apps/app-config',

'handlebars',

'c3helpers'

],function (C3, MsgBus, Layouts,AppRouter, AppController, FooterView,AppConfig,Handlebars,C3Helpers){

'use strict';

var app = new C3.Application({

root: AppConfig.rootCotext

});

//Add regions

app.addRegions({

'parentContainer':'#C3Container'

});

var pageLayout = new C3.LayoutStore.PageLayoutWithNav(); //-🡪 retrieve Layout

app.parentContainer.show(pageLayout);

pageLayout.c3Footer.show(new FooterView());

MsgBus.setApplication(app);

MsgBus.reqres.setHandler('application:page', function() {

return pageLayout;

});

if(C3Helpers && Handlebars){

C3Helpers.registerHelpers(Handlebars);

}

//Initialize App

app.on("before:start",function() {

//Setup and initialize router

app.router = new AppRouter({

controller: AppController

});

});

//After initialize

app.on('start', function() {

// Start Backbone history

C3.history.start({ pushState: true, root: app.root });

// TODO: session expiration will be handled at platform level

//Handle redirection for server session expiration

$.ajaxSetup({

statusCode: {

204: function() { //204: No Content is returned by the services when out of session

C3.history.navigate('home?expired', true);

}

}

});

// take over links that has data-override-events='backbone' attribute and convert the links to SPA navigation

this.overrideLinks();

//Navigate to deep link URLs if an internal URL is bookmarked and accessed directly

// this.processDeepLink();

});

require(['modules/components/Signon/signon-comp'], function(SignonModule){

SignonModule(pageLayout.c3Navigation\_signon);

});

require(['modules/components/branding/branding-comp'], function(BrandingModule){

BrandingModule(pageLayout.c3Navigation\_branding);

});

// export C3 and MsgBus in browser env for debugging purpose

// TODO: Do we need to make this available in PROD env???

if (window) {

window.C3 = C3;

window.MsgBus = C3.MsgBus = MsgBus;

}

return app;

});

Define Layout

define([

'c3',

'templates',

'modules/pnt/module/views/links-view'

], function(C3, JST, LinksView) {

'use strict';

return C3.Layout.extend({

name: 'pnt.module.pntLayout',

template: JST['modules/pnt/module/templates/layout.hbs'],

tagName: 'main',

attributes: {

'role': 'main',

'data-src': 'pnt-layout'

},

regions: {

header: '#citi\_pnt\_header',

leftNav: '#citi\_pnt\_quickTasks',

appBody: '#citi\_pnt\_accountsPanel'

},

// defaultViews will be displayed automatically after layout is displayed

defaultViews: {

leftNav: LinksView

}

});

});

# 5.10 C3.Error

* Extend from Marionette.Error
* Used to capture and throw business friendly Errors for logging and display
* Accept two parameters Name and Message

throw new C3.Error({

name: 'ViewDestroyedError',

message: 'View (cid: "' + view.cid + '") has already been destroyed and cannot be used.'

});

# 5.11 C3.Model

* Every application begins with the definition of a model. It’s the data that you need to manipulate and represent in the app.
* The model is a layer in which the data resides and will likely contain many objects.
* As a rule of thumb, you should break your problem domain into individual elements and describe each of these as a model object.
* C3.Model extend from Backbone.Model and provide every application a model layer to store the data.

Pls refer to detail section “How to use C3.Models” to know more in detail about different use case revolving around Models.

define([

'underscore',

'c3'

], function (\_, C3) {

'use strict';

return C3.Model.extend({

url: '/c3/sb/accounts/payment/transfer/tboa1',

//url: 'NGAMA/GBGCB/pnt/tboa/SourceAndDestinationAccounts',

defaults: {

id: 'tboa-input-model',

selectedFromAccount: '',

selectedToAccount: ''

},

validation: {

selectedFromAccount: {

required: true

},

selectedToAccount: {

required: true

}

},

sync: function(method, model, options) {

if(method==='update' || method==='patch') {

model.url='/c3/sb/accounts/payment/transfer/tboa2';

//model.url = 'NGAMA/GBGCB/pnt/tboa/AmountAndDate?sourceAccountIndex='+model.get('sourceAccountIndex') +'&destinationAccountIndex='+model.get('destinationAccountIndex');

}

return C3.Model.prototype.sync.apply(this, arguments);

},

initialize: function () {

}

});

});

# 5.12 C3.SessionModel

* Provide Session Storage.
* It can use either Windows.SessionStorage or C3’s in-memory cache.
* If Windows.sessionStorage needs to be used, the browser should support it. C3 check if browser supports the storage else it will use the in-memory cache.
* The sessionStorage property allows you to access a session [Storage](https://developer.mozilla.org/en-US/docs/Web/API/Storage) object. sessionStorage is similar to [localStorage](https://developer.mozilla.org/en-US/docs/Web/API/Window.localStorage), the only difference is while data stored in localStorage has no expiration set, data stored in sessionStorage gets cleared when the page session ends.
* A page session lasts for as long as the browser is open and survives over page reloads and restores. Opening a page in a new tab or window will cause a new session to be initiated.
* Data stored in in-memory cache will be cleared when either user session logs out, or reloading the page or closing the browser tab/window.

Pls refer to section “How to use Session Storage” to more in detail about best practices and guidelines.

define([

'c3'

], function (C3) {

'use strict';

/\*\*

\* This model is used to sign up an user or sign on an user or sign off an user

\* so the url will be assumed be /auth/login or /auth/signon or /auth/logout

\*/

var AuthenticationModel = C3.Model.extend({

defaults: {

id: 'AuthenticationModel',

username: '',

password: '',

grant\_type: 'password',

scope: 'account',

authenticated: false

},

url: '',

postAuth: function (opts, callback, args) {

var self = this;

//self.url = "NGAMA/GBGCB/JSO/signon";

self.url = "/auth/" + opts.method;

self.fetch({

type: 'POST',

async: false,

cache:true,

emulateHTTP:true,

data: {

username: this.get('username'),

password: this.get('password'),

grant\_type: 'password',

scope: '/accounts'

},

//data: JSON.stringify(\_.omit(opts, 'method')),

success: function (model, response, options) {

if (response.access\_token !== null) {

self.set('authenticated', true);

//try {

// var tokendata = JSON.parse(response);

C3.SessionModel.set("access\_token", response.access\_token);

C3.SessionModel.set("authenticated", true);

/\*} catch (error) {

C3.SessionModel.set("access\_token", response.access\_token);

C3.SessionModel.set("authenticated", true);

}\*/

} else {

self.set('authenticated', false);

}

},

error: function (error) {g

console.log("model fetch failed" + error);

self.set('authenticated', false);

if (callback && 'error' in callback) callback.error(error);

}

});

},

logOutCall: function (opts, callback, args) {

var self = this;

//self.url = "NGAMA/GBGCB/JSO/logoff";

self.url = "/auth/" + opts.method;

self.fetch({

type: 'POST',

async: false,

//data: JSON.stringify(\_.omit(opts, 'method')),

success: function (collection, response, options) {

console.log("log out successfull");

},

error: function (error) {

console.log("model fetch failed" + error);

self.set('authenticated', false);

if (callback && 'error' in callback) callback.error(error);

}

});

},

login: function (opts, callback, args) {

this.postAuth(\_.extend(opts, {method: 'login'}), callback);

},

logout: function (opts, callback, args) {

this.logOutCall(\_.extend(opts, {method: 'logout'}), callback);

},

signup: function (opts, callback, args) {

this.postAuth(\_.extend(opts, {method: 'signup'}), callback);

}

});

return new AuthenticationModel();

});

# 5.13 C3.Collection

* In the previous section we focused on the single models, but usually applications use C3.Collection to provide ordered sets of models.
* This has some useful side effects, such as being able to fetch an entire collection from a back-end server and listening for events across any of the models in a collection.
* When defining a collection, you will always pass through the model that is being contained

define([

'underscore',

'c3'

], function (\_, C3) {

'use strict';

return C3.Collection.extend({

url: '/GBGCB/REST/offers/getOffers.jws?ttc=742&isFlashSupported=true&offerType=CONTEXTUAL,MARKETING,DIGITAL\_ONBOARDING',

parse: function(response) {

return response.offersViewObj.offerViewBeanList;

},

initialize: function(options){

if(options && options.screenId) {

this.url = this.url + '&screenId=' + options.screenId;

}

}

});

});

# 5.14 C3.View

* When speaking of C3.View, we are referring to the JavaScript objects that are created in a C3 application to organize the code into logical views.
* This means that although a view can update the HTML that is present in a view, usually with a templating library, the view will listen for changes in the model and render the changes on a designated section of your HTML page.
* C3.View provides way to retrieve data from model to be set in the view.
* C3.View provides way to validate form elements

define([

'c3',

'templates',

'modules/common/models/session-model',

'modules/common/models/authentication-model',

], function ($, C3, JST, SessionModel, AuthModel) {

'use strict';

return C3.View.extend({

template: JST['modules/components/signon/templates/login-screen.hbs'],

model : AuthModel,

events: {

'mousedown #citi\_branding\_signon\_signonBtn': function (e) {

e.preventDefault();

this.signon();

},

'click #citi\_branding\_signon\_signoffBtn' : function(e){

e.preventDefault();

this.signoff();

}

},

initialize: function () {

},

render: function () {

this.$el.empty().html(this.template({authenticated: C3.SessionModel.get('authenticated'),validMap:this.validMap}));

},

signon: function (event){

this.model.set({

username: $('#username').val(),

password: $('#pwd').val()

});

if(this.executeAllValiationRules()){

$('#citi\_branding\_signonForm\_errorContainer').toggleClass('cbolui-hidden');

$('form :input').prop('disabled', true);

this.model.login(this.model);

if(this.model.get('authenticated')){

$('body').removeClass('citi-layouts-menuActive');

SessionModel.set('authenticated',true);

if(this.model.get('access\_token')){

SessionModel.set('access\_token',this.model.get('access\_token'));

}

//C3.history.navigate('dashboard/home', true);

this.render();

C3.navigate('pnt/tboa',true);

}else{

$('form :input').prop('disabled', false);

$('#citi\_branding\_signonForm\_errorContainer').removeClass('cbolui-hidden').find('[role="alert"]').attr({'aria-hidden': 'false'}).focus();

}

}

},

signoff: function(event){

this.$el.empty().html(this.template({authenticated: false,validMap:this.validMap}));

this.model.logout(this.model);

C3.SessionModel.clear();

this.model.clear();

$('body').removeClass('citi-layouts-menuActive');

C3.navigate('logout');

}

});

});

# 5.15 C3.FlowBaseView

* Flow based view is used for Transactional Flow Design, where
* On completion of one page the date from the page is passed to next page to maintain the flow.
* The data is passed via the Submit: Success Event
* The data is passed to the listening method for the event in the controller.

define([

'jquery',

'underscore',

'c3',

'templates',

'c3helpers',

'msgbus',

'backbone-validation',

'bootstrap'

], function ($, \_, C3, JST, C3Helpers, MsgBus) {

'use strict';

return C3.FlowBaseView.extend({

template: JST['modules/pnt/tboa/templates/tboa-input.hbs'],

selectTemplate: JST['modules/pnt/tboa/templates/select-account-input.hbs'],

areyousureModal: JST['modules/pnt/tboa/templates/partials/are-you-sure-model.hbs'],

// model: new TboaInputModel(),

// UI hash for the entire view. This prevents jQuery selectors from spreading in the view.

// this.bindUIElements() can be called after rendering, so that ui.property returns a jQuery object instead of selector string

// UI elements are not bind in this view because we need to re-bind whenever partial templates are render

ui: {

nextButton: '#tboa\_input\_next',

cancelButton:'#InternalTransfer\_Cancel',

fromAccount: '#fromAccount',

fromAccountSelect: '#fromAccountSelect',

toAccount: '#toAccount',

toAccountSelect: '#toAccountSelect'

},

// Delegates events for the view

events: {

"click @ui.nextButton": "submit",

"change @ui.fromAccountSelect": "\_fromListChanged",

"change @ui.toAccountSelect": "\_toListChanged"

},

modelEvents: {

"change:selectedFromAccount": "\_rePopulateToAccount",

"change:selectedToAccount": "\_rePopulateFromAccount"

},

initialize: function () {

console.log("tboa-input-view:initialize");

var self = this;

C3Helpers.Handlebars.registerPartial("selectAccountInput", this.selectTemplate);

C3Helpers.Handlebars.registerPartial("areyousureModal", this.areyousureModal);

$('#tboamodalyes').off();

$("#tboamodalno").off();

$("#tboamodalyes").on('click', function () {

$('#myModal').modal('hide');

self.model.set({"selectedFromAccount": self.$(self.ui.fromAccountSelect).val()});

self.submit();

});

$("#tboamodalno").on('click', function () {

self.$(self.ui.fromAccountSelect).val(self.model.get("selectedFromAccount"));

});

},

// preprocess data from rendering

// returned data will be used to render against template and show

preprocessData: function () {

return this.\_preprocess();

},

// prepare data for submission

// returned data will be set into model and submit to server

serializeForm: function () {

var srcAcctIndex = '';

var destAcctIndex = '';

var a = $(":input").serializeArray();

$.each(a, function () {

$.each(a, function () {

if ('fromAccountSelect' === this.name) {

srcAcctIndex = this.value;

} else if ('toAccountSelect' === this.name) {

destAcctIndex = this.value;

}

});

});

return {

sourceAccountIndex: srcAcctIndex,

destinationAccountIndex: destAcctIndex,

amount: '',

currencyIndicator: '',

\_flowExecutionKey: this.model.attributes.\_flowExecutionKey,

\_eventId: "interSubmit",

"jfp.layout": "Raw"

}

},

/////////////////////////////////////////////////////////////////////////////////////

// Varies optional event handlers for this view tied directly to UI

/////////////////////////////////////////////////////////////////////////////////////

onSubmit: function () {

this.ui.nextButton.attr("disabled", "true");

this.ui.cancelButton.attr("disabled", "true");

},

onSubmitSuccess: function () {

this.ui.nextButton.remove();

this.ui.cancelButton.remove();

},

onSubmitError: function () {

this.ui.nextButton.removeAttr("disabled");

},

// TODO:

onValidationError: function () {

$nextButton.removeAttr("disabled");

},

\_fromListChanged: function () {

//first check if next button exist or not. If it doesnt exist then it means bottom screen is rendered

if ($('#tboa\_input\_next').length) {

if ($("#fromAccountSelect").val()==="") {

this.ui.nextButton.attr("enabled", "false");

$("#tboa\_input\_next").prop("disabled", true);

} else if ($("#toAccountSelect").val()!=="") {

this.ui.nextButton.attr("enabled", "true");

$("#tboa\_input\_next").prop("disabled", false);

$("#cbolui-iconDomID-pageErrorTBOA-iconText").html("");

}

else {

$("#tboa\_input\_next").prop("disabled", true);

}

this.model.set({"selectedFromAccount": this.$(this.ui.fromAccountSelect).val()});

}

else {

//show the confirmation Model.

// $("#mybutton").trigger('click');

$('#myModal').modal('show');

}

},

\_toListChanged: function () {

if ($('#tboa\_input\_next').length) {

if ($("#toAccountSelect").val()==="") {

this.ui.nextButton.attr("enabled", "false");

$("#tboa\_input\_next").prop("disabled", true);

} else if ($("#fromAccountSelect").val()!=="") {

this.ui.nextButton.attr("enabled", "true");

$("#tboa\_input\_next").prop("disabled", false);

$("#cbolui-iconDomID-pageErrorTBOA-iconText").html("");

}

else {

$("#tboa\_input\_next").prop("disabled", true);

}

this.model.set({"selectedToAccount": this.$(this.ui.toAccountSelect).val()});

}

else {

//show the confirmation Model.

// $("#mybutton").trigger('click');

$('#myModal').modal('show');

}

},

\_rePopulateFromAccount: function () {

console.log("\_rePopulateFromAccount:");

var beforeRefreshFrom = $(this.ui.fromAccountSelect).val();

console.log("Before Selection:" + beforeRefreshFrom);

var fromAccounts = this.model.attributes.destSrcMap[this.model.attributes.selectedToAccount];

console.log("Selected TO Account ::" + this.model.attributes.selectedToAccount);

$(this.ui.fromAccount).html(this.selectTemplate(this.\_formatAccountList(fromAccounts, this.model.attributes.defaultSelectionText, 'fromAccountSelect', 'FROM:')));

$(this.ui.fromAccountSelect).val(beforeRefreshFrom);

},

\_rePopulateToAccount: function () {

console.log("\_rePopulateToAccount:");

var beforeRefreshTo = $(this.ui.toAccountSelect).val();

console.log("Before Selection:" + beforeRefreshTo);

var toAccounts = this.model.attributes.srcDestMap[this.model.attributes.selectedFromAccount];

$(this.ui.toAccount).html(this.selectTemplate(this.\_formatAccountList(toAccounts, this.model.attributes.defaultSelectionText, 'toAccountSelect', 'TO:')));

$(this.ui.toAccountSelect).val(beforeRefreshTo);

},

\_checkToRender: function () {

var errorMsg = this.model.attributes.errorMsg;

if (errorMsg != null) {

this.render();

}

},

\_preprocess: function () {

console.log("tboa-view-input.js:preprocess");

var data = this.model.attributes;

var output = {

fromElement: {

displaylabel: '',

id: '',

element: []

},

toElement: {

displaylabel: '',

id: '',

element: []

}

};

output.fromElement = this.\_formatAccountList(data.sourceAccountVBList, data.defaultSelectionText, 'fromAccountSelect', 'FROM:');

output.toElement = this.\_formatAccountList(data.destinationAccountVBList, data.defaultSelectionText, 'toAccountSelect', 'TO:');

\_.extend(output, data);

// console.log("preprocess returns:"+JSON.stringify(output))

var errorObj = this.model.get('error');

if (errorObj.status) {

return \_.extend(output, {error: errorObj});

}

return output;

},

\_formatAccountList: function (accountList, defaultSelectionText, id, displaylabel) {

console.log("\_formatAccountList:");

var output = {

displaylabel: '',

id: '',

element: []

};

output.displaylabel = displaylabel;

output.id = id;

output.element = \_.map(accountList, function (account) {

console.log("tboa-input-view:map");

var element = {};

element.itemDescription = account.accountIdentifierText.split('{')[0];

if (account.accountResolverVO[0]) {

if (account.accountResolverVO[0].negativeHandler1 && (account.accountResolverVO[0].negativeHandler1 == 'DISPLAY\_ZERO' || account.accountResolverVO[0].negativeHandler1 == 'DISPLAY\_WITH\_MINUS')) {

element.itemDescription += " -";

}

if (account.accountResolverVO[0].resolverBalanceText1 && account.accountResolverVO[0].resolverBalanceText1 === 'Field Label') {

element.itemDescription += " " + account.accountResolverVO[0].resolverBalance1;

}

if (account.accountResolverVO[0].resolverBalanceText2 && account.accountResolverVO[0].resolverBalanceText2 === 'Field Label') {

element.itemDescription += account.accountResolverVO[0].resolverBalanceText2;

}

}

element.instanceID = account.instanceID;

return element;

});

var defaultElement = {};

defaultElement.itemDescription = defaultSelectionText;

defaultElement.instanceID = ""; //hope no account is named default... might need to clean this

output.element.splice(0, 0, defaultElement);

// console.log("tboa-input-view:output:"+JSON.stringify(output));

return output;

}

});

});

define([

'underscore',

'c3',

'msgbus',

'modules/pnt/tboa/layouts/flow-layout',

'c3helpers'

], function (\_, C3, MsgBus, TboaFlowlayout, C3Helpers) {

'use strict';

return C3.SubAppController.extend({

// A name is not required, however, it's highly recommend for debugging purpose

// As javascript doesn't have classname similar to other language such as java

name: 'pnt.tboa.TboaController',

// Mandantory fields.

layoutClass: TboaFlowlayout,

initialize: function () {

console.log('TBOA controller initialized');

},

// Defines the entry point for subapp

start: function () {

this.\_tboaInput();

},

\_tboaInput: function () {

var self = this;

require(['modules/pnt/tboa/views/tboa-input-view', 'modules/pnt/tboa/models/tboa-input-model'],

function (TboaInputView, TboaInputModel) {

self.model = new TboaInputModel();

self.model.fetch({

type: 'POST',

success: function (data, response, options) {

self.tboaInputView = new TboaInputView({model: self.model});

var region = self.getLayout().region1;

region.show(self.tboaInputView);

var errorObj = self.model.get("error");

if (errorObj && errorObj.status) {

region.$el.find("#cbolui-iconDomID-pageErrorTBOA-iconText").html(

"<span class='cbolui-icon-globalSpriteBase cbolui-icon-redError' id='cbolui-iconDomID-pageErrorTBOA-iconChild'></span><Strong>" +

errorObj.statusText + "</Strong>");

region.$el.find("#tboa\_input\_next").attr("disabled", "true");

} else {

self.listenTo(self.tboaInputView, 'submit:success', self.\_tboaAdditionalInput);

}

},

error: function (data, jqXHR,options) {

console.log("\_tboaInput model fetch error failed...")

region.$el.find("#cbolui-iconDomID-pageErrorTBOA-iconText").

html(" <span class='cbolui-icon-globalSpriteBase cbolui-icon-redError' id='cbolui-iconDomID-pageErrorTBOA-iconChild'></span><Strong>" +

jqXHR.statusText + "</Strong>");

}

});

}

);

},

\_tboaCombinedInput: function () {

var region3 = this.getLayout().region3;

region3.reset();

var region1 = this.getLayout().region1;

region1.$el.show();

var region2 = this.getLayout().region2;

region2.$el.show();

},

\_tboaAdditionalInput: function (data) {

console.log('tboaAdditionalInput');

var self = this;

// save model from previous step in context

self.context.tboaInputModel = data.model;

// extract JSON response

var JSONResponse = data.response;

var region1 = this.getLayout().region1;

var errorMsg = JSONResponse.errorMsg;

if (errorMsg != null) {

region1.$el.find("#cbolui-iconDomID-pageErrorTBOA-iconText").html(" <span class='cbolui-icon-globalSpriteBase cbolui-icon-redError' id='cbolui-iconDomID-pageErrorTBOA-iconChild'></span><Strong>" + errorMsg + "</Strong>");

} else {

region1.$el.find("#tboa\_input\_next").remove();

require(['modules/pnt/tboa/views/tboa-additional-input-view','modules/pnt/tboa/models/tboa-additional-input-model'],

function (TboaAdditionalInputView,TboaAdditionalInputModel) {

// Instantiate the view

self.model = new TboaAdditionalInputModel();

self.model.set({data:JSONResponse});

self.\_\_calculateModelAttributes(self.model);

self.tboaAdditionalInputView = new TboaAdditionalInputView({model:self.model});

var region = self.getLayout().region2;

region.show(self.tboaAdditionalInputView);

// listen to submit event on view

self.listenTo(self.tboaAdditionalInputView, 'submit:success', self.\_tboaRecap);

}

);

}

},

\_tboaRecap: function (data) {

console.log('tboaRecap');

var self = this;

// save model from previous step in context

self.context.tboaAdditionalInputModel = data.model;

// extract JSON response

var JSONResponse = data.response;

var region1 = self.getLayout().region1;

var region2 = self.getLayout().region2;

var errorMsg = JSONResponse.errorMsg;

if (errorMsg != null) {

region1.$el.find("#cbolui-iconDomID-pageErrorTBOA-iconText").html(" <span class='cbolui-icon-globalSpriteBase cbolui-icon-redError' id='cbolui-iconDomID-pageErrorTBOA-iconChild'></span><Strong>" + errorMsg + "</Strong>");

$("#tboaPrimaryNav").prop("disabled", true);

}

else {

$("#tboaPrimaryNav").prop("disabled", false);

require(['modules/pnt/tboa/views/tboa-recap-view','modules/pnt/tboa/models/tboa-recap-model'],

function (TboaRecapView,TboaRecapModel) {

self.recapModel = new TboaRecapModel();

self.recapModel.set({data:JSONResponse});

var tboaRecapView = new TboaRecapView({model:self.recapModel}, {previousModelReference:{

tboaInputModel: self.context.tboaInputModel.attributes,

tboaAdditionalInputModel: self.context.tboaAdditionalInputModel.attributes

}});

var layout = self.getLayout();

var region1 = self.getLayout().region1;

var region2 = self.getLayout().region2;

var region3 = self.getLayout().region3;

region3.show(tboaRecapView);

region1.$el.hide();

region2.$el.hide();

// listen to submit event on view

self.listenTo(tboaRecapView, 'submit:success', self.\_tboaConfirmation);

self.listenTo(tboaRecapView, 'changeButton:clicked', self.\_tboaCombinedInput);

}

);

}

},

\_tboaConfirmation: function (data) {

console.log('\_tboaConfirmation');

var self = this;

require(['modules/pnt/tboa/views/tboa-confirmation-view','modules/pnt/tboa/models/tboa-confirmation-model'],

function (TboaConfirmantionView,TboaConfirmantionModel) {

// extract JSON response

var JSONResponse = data.response;

var confirmationModel = new TboaConfirmantionModel();

confirmationModel.set({data:JSONResponse});

self.tboaConfirmantionView = new TboaConfirmantionView({model:confirmationModel});

var region1 = self.getLayout().region1;

var region2 = self.getLayout().region2;

var region3 = self.getLayout().region3;

region1.reset();

region2.reset();

region3.reset();

region3.show(self.tboaConfirmantionView);

self.listenTo(self.tboaConfirmantionView, 'doAnotherButton:clicked', self.\_tboaInput);

}

);

},

/\*\*

\* calculate necessary model attributes

\* @param model

\* @private

\*/

\_\_calculateModelAttributes:function(model){

var amountOptionLength = 0;

var pymntOptions = model.attributes.data.paymentOptionsVB;

if (pymntOptions && pymntOptions.otherAmtObj&&pymntOptions.otherAmtObj.otherAmountVBList) {

amountOptionLength += pymntOptions.otherAmtObj.otherAmountVBList.length;

}

var currencyLeft = true; //default to left

if(model.attributes.data.inputOptions !== null && model.attributes.data.inputOptions.ccyDescriptorPosition !== null){

if(model.attributes.data.inputOptions.ccyDescriptorPosition === 'prefix'){

currencyLeft = true;

}else{

currencyLeft = false;

}

}

var paymentLen = {amountOptionLength:amountOptionLength};

var currencyPosition = {currencyLeft:currencyLeft};

model.set({data:paymentLen});

model.set({data:currencyPosition});

}

});

});

# 5.16 C3.Helpers

C3 client must utilize handlebar helpers to generate html markup. Using helpers will enforce standard way of creating html elements and eliminate bad syntax. Also helpers will provide easier way of introducing or removing attributes from html elements.

All helpers have built in mechanisms to retrieve label, placeholder, and error messages from content management system through content service. Helpers utilize **mappingKey** which is linked to a specific content id / content type / appid.

To help generation of mapping file for a module, every template will define **mappingKey** information between **@mappingsource@** annotation. Below is mapping definition example.

**@mappingsource@**

**firstname=content\_id\_1/config/cbol\_common**

**lastname=content\_id\_2/config/cbol\_common**

**password=content\_id\_3/config/cbol\_common**

**token=content\_id\_4/config/cbol\_common**

**@mappingsource@**

# 5.16.1 Text Helper

Text helper outputs html input element with type text. Text helpers supports following **highlighted** attributes. Any server side error for this field will be automatically displayed on the view when error message key is in format id + "\_error".

{{text 'id' 'value' class='css style class goes here' style='inline css goes here' minlength='3' maxlength='20' pattern='[a-zA-Z.,]' errorStyle="tooltip|inline"

placeholder='{

"message" : { "mappingKey" : "firstname", "attribute" : "node id of the place holder message for input field" }

}'

rules='{

"ruleName1": { "event": "focusout", "message" : "message shown when user omits entering data into input field"},

"ruleName2": { "event": "focusout", "message" : { "mappingKey" : "firstname", "attribute" : "node id of the validation error message for input field" }}

}'

maskRule='{

"rule" : {"event" : "keydown", "func" : "functionName"}

}'

attributes='{

"attributeName1" : "attributeValue1",

"attributeName2" : "attributeValue2"

}'

|  |  |
| --- | --- |
| **Attribute Name** | **Attribute Description** |
| Id | id and name value for the input field |
| Value | value for the input field |
| Class | One or more class names for the input field |
| Style | Specifies inline css for the element |
| Minlength | Min length for the input field |
| Maxlength | Max length for the input field |
| Pattern | Valid character set for the field |
| errorStyle | tooltip or inline. Errors for the field will be shown as a tooltip or as a inline message |
| Placeholder | Message shown on the input field to aid the user. This message will be removed when the user focus on the input element |
| Rules | Provide validation rules for input field |
| ruleName1, ruleName2 to ruleNameN | Rule to be applied. Possible values are required, minlength, maxlength, pattern |
| event | When to trigger the ruleName. Possible values are focusout, blur. Default event: focusout. |
| Message | Message to be shown when rule returns false |
| mappingKey | mappingKey of the content record to retrieve validation error message or placeholder message |
| Attribute | node id of the content record to retrieve validation error message or placeholder message |
| maskRule | Provide masking rule for input field |
| event | Event to trigger masking rule for input field. Default event: keydown |
| Func | Function to invoke for masking input field |
| Attributes | Provide additional attributes for input field |
| attributeName1, attributeName2 to attributeNameN | Provide attribute names to be available in html markup |
| attributeValue1, attributeValue2 to attributeValueN | Provide attribute values to be available in html markup |

}}{{/text}}

**Example Usage:**

For rules attribute (validation) if event is not provided then it defaults to focusout, maskRule attribute (masking) if event is not provided then it defaults to keydown. Server error for this field must be present in "fname\_error" JSON key.

{{text 'fname' class='inputfield' minlength='3' maxlength='20' pattern='[a-zA-Z.,]' errorStyle="tooltip|inline"

placeholder='{ "message" : { "mappingKey" : "firstname", "attribute" : "placeholder" }'

rules='{

"required": { "event": "focusout", "message" : "First Name is required"},

"minlength": { "message" : { "mappingKey" : "firstname", "attribute" : "minlength" }},

"maxlength": { "message" : { "mappingKey" : "firstname", "attribute" : "maxlength" }},

"pattern": { "event": "focusout", "message" : "Valid characters allowed in this field a-z,A-Z,.,-"}

}'

maskRule='{

"rule" : {"event" : "keydown", "func" : "c3.masking.firstname"}

}'

attributes='{

"readonly" : "readonly",

"tabindex" : "1",

"data-input-state" : "focus"

}'

}}{{/text}}

**Output:**

<input type="text" name="fname" id="fname" class='inputfield' size="25" placeholder="Enter first name" readonly tabindex="1" data-input-state="focus"

/>

Below rules will be automatically applied on above html element when it loses focus

(a) Required – First Name is required

(b) minlength 3 – First name minimum length is 3

(c) maxlength 20 – First name maximum length is 20

(d) valid character set [a-zA-Z.-] – Valid characters allowed in this field a-z, A-Z, ., -

(e) Masking rules – Mask first three characters on keydown

# 5.16.2 Hidden Helper

Hidden helper outputs html hidden element

{{hidden 'id' 'value' class='css class goes here' style='inline css goes here'

attributes='{

"attributeName1" : "attributeValue1",

"attributeName2" : "attributeValue2"

}'

}}{{/hidden}}

|  |  |
| --- | --- |
| **Attribute Name** | **Attribute Description** |
| Id | id and name value for the hidden input field |
| Value | Value of the hidden input field |
| Class | One or more class names for the input field |
| Style | Specifies inline css for the element |
| Attributes | Provide additional attributes for input field |
| attributeName1, attributeName2 to attributeNameN | Provide attribute names to be available in html markup |
| attributeValue1, attributeValue2 to attributeValueN | Provide attribute values to be available in html markup |

**Example Usage:**

{{hidden 'token' '837891274831784'}}{{/hidden}}

**Output:**

<input type="hidden" name="token" id="token" value="837891274831784"/>

# 5.16.3 Password Helper

Password helper which outputs html password element

{{password 'id' 'value' class='css class goes here' style='inline css goes here' minlength='3' maxlength='20' pattern='[a-zA-Z.,]' errorStyle="tooltip|inline"

placeholder='{

"message" : { "mappingKey" : "password", "attribute" : "node id of the place holder message for password field" }

}'

rules='{

"ruleName1": { "event": "focusout", "message" : "message shown when user omits entering data into input field"},

"ruleName2": { "event": "focusout", "message" : { "mappingKey" : "password", "attribute" : "node id of the validation error message for password field" }}

}'

attributes='{

"attributeName1" : "attributeValue1",

"attributeName2" : "attributeValue2"

}'

}}{{/password}}

|  |  |
| --- | --- |
| **Attribute Name** | **Attribute Description** |
| Id | id and name value for the input field |
| Value | value for the input field |
| Class | One or more class names for the input field |
| Style | Specifies inline css for the element |
| Minlength | Min length for the input field |
| Maxlength | Max length for the input field |
| Pattern | Valid character set for the field |
| errorStyle | tooltip or inline. Errors for the field will be shown as a tooltip or as a inline message |
| Placeholder | Message shown on the input field to aid the user. This message will be removed when the user focus on the input element |
| Rules | Provide validation rules for input field |
| ruleName1, ruleName2 to ruleNameN | Rule to be applied. Possible values are required, minlength, maxlength, pattern |
| Event | When to trigger the ruleName. Possible values are focusout, blur. Default event: focusout. |
| Message | Message to be shown when rule returns false |
| mappingKey | mappingKey of the content record to retrieve validation error message or placeholder message |
| Attribute | node id of the content record to retrieve validation error message or placeholder message |
| Attributes | Provide additional attributes for input field |
| attributeName1, attributeName2 to attributeNameN | Provide attribute names to be available in html markup |
| attributeValue1, attributeValue2 to attributeValueN | Provide attribute values to be available in html markup |

**Example Usage:**

{{ password 'password' minlength='8' maxlength='20' pattern='[a-zA-Z0-9$%#@!.,]' errorStyle="tooltip|inline"

placeholder='{ "message" : { "mappingKey" : "password", "attribute" : "password.placeholder" }'

rules='{

"required": { "event": "focusout", "message" : "Password is required"},

"minlength": { "message" : { "mappingKey" : "password", "attribute" : "minlength" }},

"maxlength": { "message" : { "mappingKey" : "password", "attribute" : "maxlength" }},

"pattern": { "event": "focusout", "message" : "Valid characters allowed in this field a-zA-Z0-9$%#@!.,"}

}'

}}{{/password}}

**Output:**

<input type="password" name="password" id="password"/>

# 5.16.4 Button Helper

Creates input element of type button

{{button 'id' value='{

"message" : { "mappingKey" : "content id of the message holding button value", "attribute" : "node id inside content" }

}' class='css class goes here' style='inline css goes here'

attributes='{

"attributeName1" : "attributeValue1",

"attributeName2" : "attributeValue2"

}'

}}{{/button}}

|  |  |
| --- | --- |
| **Attribute Name** | **Attribute Description** |
| Id | id and name value for the input field |
| Value | value for the input field can be JSON object or string |
| message | Placeholder for mappingKey and attribute nodes |
| mappingKey | mappingKey of the content record to retrieve button value |
| Attribute | node id of the content record to retrieve button value |
| Class | One or more class names for the input field |
| Style | Specifies inline css for the element |
| Attributes | Provide additional attributes for input field |
| attributeName1, attributeName2 to attributeNameN | Provide attribute names to be available in html markup |
| attributeValue1, attributeValue2 to attributeValueN | Provide attribute values to be available in html markup |

**Example Usage:**

{{button 'prev' value='{

"message" : { "mappingKey" : "buttons", "attribute" : "prev" }

}' class='internal' attributes='{

"tabindex" : "2"

}'

}}{{/button}}

**Output:**

<input type="button" name="prev" id="prev" value="previous" class="internal" tabindex="2" />

# 5.16.5 Radio Helper

Creates input element of type radio

{{radio name='name of the radio field' group='{

"radio1": {

"id": "id of first radio element ", "value": {

"message": { "mappingKey": "radio", "attribute": " node id of the message for radiofield "

}}, "class": "css class goes here", "style": "inline css goes here",

"attributes": {

"attributeName1": "attributeValue1",

"attributeName2": "attributeValue2"

}

},

"radio2": {

"id": "id of second radio element ", "value": {

"message": { "mappingKey": "radio", "attribute": "node id of the message for radiofield"

}}, "class": "inputfield", "style": "inline css goes here",

"attributes": {

"attributeName1": "attributeValue1",

"attributeName2": "attributeValue2"

}

},

"radio3": {

"id": "id of third radio element ", "value": {

"message": { "mappingKey": "radio", "attribute": "node id of the message for radiofield"

}}, "class": "inputfield", "style": "inline css goes here",

"attributes": {

"attributeName1": "attributeValue1",

"attributeName2": "attributeValue2"

}

}

}'

rules='{

"ruleName1": { "event": "focusout", "message" : "message shown when user omits selecting radio field"},

"ruleName2": { "event": "focusout", "message" : { "mappingKey" : "radio", "attribute" : "node id of the validation error message for radio field" }}

}'

}}{{/radio}}

|  |  |
| --- | --- |
| **Attribute Name** | **Attribute Description** |
| name | name for the radio field |
| group | fields holds radio elements |
| radio1 | Holds first radio element of the group |
| Id | id for the radio field |
| value | value for the input field can be JSON object or string |
| message | Placeholder for mappingKey and attribute nodes |
| mappingKey | mappingKey of the content record to retrieve button value |
| attribute | node id of the content record to retrieve button value |
| class | One or more class names for the radio field |
| style | Specifies inline css for the element |
| attributes | Provide additional attributes for radio field |
| attributeName1, attributeName2 to attributeNameN | Provide attribute names to be available in html markup |
| attributeValue1, attributeValue2 to attributeValueN | Provide attribute values to be available in html markup |
| radio2 | Holds second radio element of the group (Optional) |
| radio3 | Holds third radio element of the group (Optional) |
| Rules | Provide validation rules for radio field |
| ruleName1, ruleName2 to ruleNameN | Rule to be applied. Possible values are required and others |

**Example Usage:**

{{radio name='gender' group='{

"radio1": {"id": "radio1", "value": {"message": {"mappingKey": "radio", "attribute": "value1"}},

"class": "radiofield", "attributes": {"checked": "true"}},

"radio2": {"id": "radio2", "value": {"message": {"mappingKey": "radio", "attribute": "value2"}},

"class": "radiofield"},

}', rules='{"required": {"event": "submit", "message":"selection must be made"}

}}{{/radio}}

**Output:**

<input type="radio" name="gender" id="radio1" value="male" class="radiofield" checked/>

<input type="radio" name="gender" id="radio2" value="female" class="radiofield"/>

# 5.16.6 Checkbox Helper

Creates input element of type checkbox

{{checkbox name='name of the checkbox field' min='no of minimum selections' max='no of maximum selections' group='{

"checkbox1": {

"id": "id of first checkbox element ", "value": {

"message": { "mappingKey": "checkbox", "attribute": " node id of the message for checkbox field "

}}, "class": "css class goes here", "style": "inline css goes here",

"attributes": {

"attributeName1": "attributeValue1",

"attributeName2": "attributeValue2"

}

},

"checkbox2": {

"id": "id of second checkbox element ", "value": {

"message": { "mappingKey": "checkbox", "attribute": "node id of the message for checkbox field"

}}, "class": "inputfield", "style": "inline css goes here",

"attributes": {

"attributeName1": "attributeValue1",

"attributeName2": "attributeValue2"

}

},

"checkbox3": {

"id": "id of third checkbox element ", "value": {

"message": { "mappingKey": "checkbox", "attribute": "node id of the message for checkbox field"

}}, "class": "inputfield", "style": "inline css goes here",

"attributes": {

"attributeName1": "attributeValue1",

"attributeName2": "attributeValue2"

}

},

……

}'

rules='{

"ruleName1": { "event": "focusout", "message" : "message shown when user omits selecting checkbox field"},

"ruleName2": { "event": "focusout", "message" : { "mappingKey" : "checkbox", "attribute" : "node id of the validation error message for checkbox field" }}

}'

}}{{/checkbox}}

|  |  |
| --- | --- |
| **Attribute Name** | **Attribute Description** |
| Name | name for the checkbox field |
| Min | No of minimum checkboxes to be selected |
| Max | No of maximum checkboxes to be selected |
| Group | fields holds checkbox elements |
| checkbox1 | Holds first checkbox element of the group |
| Id | id for the checkbox field |
| Value | value for the input field can be JSON object or string |
| Message | Placeholder for mappingKey and attribute nodes |
| mappingKey | mappingKey of the content record to retrieve button value |
| Attribute | node id of the content record to retrieve button value |
| Class | One or more class names for the checkbox field |
| Style | Specifies inline css for the element |
| Attributes | Provide additional attributes for checkbox field |
| attributeName1, attributeName2 to attributeNameN | Provide attribute names to be available in html markup |
| attributeValue1, attributeValue2 to attributeValueN | Provide attribute values to be available in html markup |
| checkbox2 | Holds second checkbox element of the group (Optional) |
| Rules | Provide validation rules for checkbox field |
| ruleName1, ruleName2 to ruleNameN | Rule to be applied. Possible values are required, minCheckbox (uses value from min attribute), maxCheckbox (uses value from max attribute) and others |

**Example Usage:**

{{checkbox name='preferences' group='{

"checkbox1": {"id": "checkbox1", "value": {"message": {"mappingKey": "checkbox", "attribute": "value1"}}, "class": "checkboxfield", "attributes": {"checked": "true"}},

"checkbox2": {"id": "checkbox2", "value": {"message": {"mappingKey": "checkbox", "attribute": "value2"}}, "class": "checkboxfield", "attributes": {"checked": "true"}},

"checkbox3": {"id": "checkbox3", "value": {"message": {"mappingKey": "checkbox", "attribute": "value3"}}, "class": "checkboxfield"}

}' rules='{"required": { "event": "submit", "message" : "selection must be made"}

}}{{/checkbox}}

**Output:**

<input type="checkbox" name="preferences" id="checkbox1" value="email" class="checkboxfield" checked/>

<input type="checkbox" name="preferences" id="checkbox2" value="txt" class="checkboxfield" checked/>

<input type="checkbox" name="preferences" id="checkbox3" value="Call" class="checkboxfield"/>

# 5.16.7 Date Helper

Creates input element of type text, this text field will accept date in a specified format. It can apply rules like required, pattern. Also the date helper can either show the field as input text box or input text box with date widget allowing the selection.

{{date 'id' value=Value of the date field' class='css style class goes here' style='inline css goes here' errorStyle="tooltip|inline" widget="true" dateFormat="dd/mm/yyyy"

minDate='provide earliest date to be shown/accepted in format specified in dateFormat attribute'

maxDate='provide a date after minimum date to be shown/accepted in format specified in dateFormat attribute'

holidays='{"list": [list of comma separated dates in format specified in dateFormat attribute]}'

disableWeekend='true'

placeholder='{"message": { "mappingKey" : "firstname", "attribute" : "node id of the place holder message for input field" }}'

rules='{"ruleName1": {"event": "focusout","message": "message shown when user omits entering date field"},"ruleName2": {"event": "focusout","message": {"mappingKey":"date", "attribute":"node id of the validation error message for checkbox field"}}}'

attributes='{"attributeName1" : "attributeValue1","attributeName2" : "attributeValue2"}'}}{{/date}}

|  |  |
| --- | --- |
| **Attribute Name** | **Attribute Description** |
| Id | Id and name for the date field |
| value | Value of the date field |
| class | One or more class names for the date field |
| style | Specifies inline css for the element |
| errorStyle | tooltip or inline. Errors for the field will be shown as a tooltip or as a inline message |
| widget | date field as a widget |
| dateFormat | Format of the date in which it should be captured |
| minDate | Earliest date which should be shown or accepted. If minDate is not populated then past dates are not allowed and it starts from today’s date. Possible options are [+-][\d]+[dmy] |
| maxDate | Provide a date after minDate which should be shown or accepted. If maxDate is not populated then next 1 year from today’s date is allowed. Possible options are [+-][\d]+[dmy] |
| holidays | Holidays is JSON object which holds the list of holidays which should not be allowed as user input |
| disableWeekend | By default weekends are always shown or accepted. If weekend should not be shown then disableWeekend must be set to false |
| placeholder | Message shown on the input field to aid the user. This message will be removed when the user focus on the input element |
| rules | Provide validation rules for input field |
| ruleName1, ruleName2 to ruleNameN | Rule to be applied. Possible values are required, pattern, range (based on minDate and maxDate) |
| event | When to trigger the ruleName. Possible values are focusout, blur. Default event: focusout. |
| message | Message to be shown when rule returns false |
| mappingKey | mappingKey of the content record to retrieve validation error message or placeholder message |
| attribute | node id of the content record to retrieve validation error message or placeholder message |
| attributes | Provide additional attributes for input field |
| attributeName1, attributeName2 to attributeNameN | Provide attribute names to be available in html markup |
| attributeValue1, attributeValue2 to attributeValueN | Provide attribute values to be available in html markup |

**Example Usage:**

{{date 'dob' class='dateField' errorStyle='inline' widget='true' placeholder='{ "message" : { "mappingKey" : "dob", "attribute" : "placeholder" }' rules='{"required": { "event": "focusout", "message" : "selection must be made", "range": { "event": "focusout", "message" : {"mappingKey": "dob", "attribute": "rangemsg"}} minDate='-100y' minDate='-18y'}}{{/date}}

**Output:**

<input type="text" name="dob" id="dob" value="dd/mm/yyyy" class="datefield" checked/>

Chapter 6 Require vs Define

This chapter gives a detailed overview of Require vs Define syntax

t

his chapter gives a detailed overview of Require vs define syntax being used in the Asynchronous Module loader.

# 6.1 RequireJS Module Loader Syntax

RequireJS as a module loader provides you two different syntax to load the dependency modules. You want to know the difference between the two syntaxes and when to use what.

* Define: If you want to declare a module other parts of your application will depend on.
* Require: If you just want to load and use stuff.

The **define()** function accepts two optional parameters (a string that represent a module ID and an array of required modules) and one required parameter (a factory method).

The return of the factory method **MUST** return the implementation for your module (in the same way that the [Module Pattern](http://addyosmani.com/resources/essentialjsdesignpatterns/book/#modulepatternjavascript) does).

The “**require()”** function doesn't have to return the implementation of a new module.

Using **define()** you are asking something like *"run the function that I am passing as a parameter and assign whatever returns to the ID that I am passing but, before, check that these dependencies are loaded"*.

Using **require()** you are saying something like *"the function that I pass has the following dependencies, check that these dependencies are loaded before running it"*.

The “**require()”** function is where you use your defined modules, in order to be sure that the modules are defined, but you are not defining new modules there.

define([

'c3',

'msgbus',

'layouts',

'modules/common/routes/app-router',

'modules/common/controllers/app-controller',

'modules/common/views/footer-view',

'platform/apps/app-config',

'handlebars',

'c3helpers'

],function (C3, MsgBus, Layouts,AppRouter, AppController, FooterView,AppConfig,Handlebars,C3Helpers){

'use strict';

var app = new C3.Application({

root: AppConfig.rootCotext

});

//Add regions

app.addRegions({

'parentContainer':'#C3Container'

});

var pageLayout = new C3.LayoutStore.PageLayoutWithNav1();

app.parentContainer.show(pageLayout);

pageLayout.c3Footer.show(new FooterView());

MsgBus.setApplication(app);

MsgBus.reqres.setHandler('application:page', function() {

return pageLayout;

});

if(C3Helpers && Handlebars){

C3Helpers.registerHelpers(Handlebars);

}

//Initialize App

app.on("before:start",function() {

//Setup and initialize router

app.router = new AppRouter({

controller: AppController

});

});

//After initialize

app.on('start', function() {

// Start Backbone history

C3.history.start({ pushState: true, root: app.root });

// TODO: session expiration will be handled at platform level

//Handle redirection for server session expiration

$.ajaxSetup({

statusCode: {

204: function() { //204: No Content is returned by the services when out of session

C3.history.navigate('home?expired', true);

}

}

});

// take over links that has data-override-events='backbone' attribute and convert the links to SPA navigation

this.overrideLinks();

//Navigate to deep link URLs if an internal URL is bookmarked and accessed directly

// this.processDeepLink();

});

require(['modules/components/Signon/signon-comp'], function(SignonModule){

SignonModule(pageLayout.c3Navigation\_signon);

});

require(['modules/components/branding/branding-comp'], function(BrandingModule){

BrandingModule(pageLayout.c3Navigation\_branding);

});

// export C3 and MsgBus in browser env for debugging purpose

// TODO: Do we need to make this available in PROD env???

if (window) {

window.C3 = C3;

window.MsgBus = C3.MsgBus = MsgBus;

}

return app;

});

The following overview description references the steps depicted in the above diagram:

1. When an ActionForward is returned to the ActionServlet, by a JFP Process Action, the ActionServlet searches for a match. The ActionServlet searches its various tables in a fixed order. It will first search the Tiles definitions, if any for this module, if none are found it will search the Struts ActionForwards for this module.
2. In the example above the ActionForward returned, references a Tiles definition with the name “defConfirmPayeeName”. Because it is a Tiles definition, the Tiles processor for this module will be instantiated and given control.
3. The Tiles definition, “defConfirmPayeeName” extends a Tiles definition named “citiMainLayout”. The Tiles definition “citiMainLayout” references “citiMainLayout.jsp” as the realization of this layout. The Tiles processor forwards the response to “citiMainLayout.jsp”
4. Within “citiMainLayout.jsp” are several “Tiles tags” of the form “tiles:insert”, which when encountered in the processing of the JSP page, will cause the Tiles code to be called. In this example there are 3 “tiles:insert” tags, which will result in the include processing for 3 separate tiles.
5. The tiles tag <tiles:insert attribute=’header’/> will ultimately cause the header.jsp page to be called via an include. The “header.jsp” page will in turn, in this example, call the JFP navigation service to return information to be used to build the HTML fragment for the header tile.
6. Next, the tiles tag “tiles:insert attribute=’body’”, for the body tile is overwritten by the <put name=”body” value=’addapayee/pages/ConfirmPayeeName.jsp’/> in the “defConfirmPayeeName” Tiles definition. This will cause “ConfirmPayeeName.jsp” to be called via an include. ConfirmPayeeName.jsp will build the HTML fragment for the body tile in concert with dynamic customer data in the ActionForm and dynamic phrase resolution via the Message Resource classes.
7. Dynamic Model content (i.e. customer data) is extracted from the ActionForm via Struts Tags, such as <html:text property =”payee”/>. Tags of this sort will, by default, look in the ActionForm for a matching property getter, such as “getPayee”. Alternatively the name of a specific bean, as source for model data, can be named such as the following <html:text property=”payee” name=”payeeBean” />, which instructs the tag to look for the property “payee” in a bean named “payeeBean”, and then place its value in the HTML stream.
8. Phrase content is obtained from Message Resource classes via Struts Tags (e.g. <bean:message bundle=”addapayee” key=”title.payeename” />).
9. The tiles tag <tiles:insert attribute=’footer’/> will ultimately cause the header.jsp page to be called via an include. The “footer.jsp” page will in turn, in this example, call the JFP navigation service to return information to be used to build the HTML fragment for the footer tile.
10. After all tiles tags have been called and processing is complete for “citiMainLayout.jsp”, the entire HTML response is returned to the browser for rendering and display.

Chapter 7 SEO RENDERING

This chapter gives an overview of how to handle the SEO Rendering

t

his chapter provide an overview of how to handle SEO Rendering and make the application Search Friendly.

# 7.1 Pre Login Pages

All Pre Login Pages Need to be rendered via Server side HTML for better SEO Visibility.

Chapter 8 Definition vs. Instantiation of Components

This chapter gives an overview of define vs. instantiation of C3 Components.

t

his chapter gives an overview of how to define a component and how to instantiate the component.

# 8.1 Define vs Instantiation

* Any of the C3 Components can be extended using .extend Function.
* Once extended we have the definition available for the new class we created inheriting from the parent one.

define([

'c3',

'templates',

'modules/pnt/module/views/links-view'

], function(C3, JST, LinksView) {

'use strict';

return C3.Layout.extend({

name: 'pnt.module.pntLayout',

template: JST['modules/pnt/module/templates/layout.hbs'],

tagName: 'main',

attributes: {

'role': 'main',

'data-src': 'pnt-layout'

},

regions: {

header: '#citi\_pnt\_header',

leftNav: '#citi\_pnt\_quickTasks',

appBody: '#citi\_pnt\_accountsPanel'

},

// defaultViews will be displayed automatically after layout is displayed

defaultViews: {

leftNav: LinksView

}

});

});

Here C3.Layout is defined but its not instantiated.

To instantiate:

* Require the newly created C3 component or define as dependency
* Once required , instantiate the component using the new operator(regular javascript instantiation of objects) or you can leverage one of the controller exposed API’s viz., createSubapp, createModel or createView to instantiate the object

define([

'c3',

'msgbus',

'modules/radio/module/layout/radio-module-body-layout'

], function (C3,MsgBus,RadioModuleLayout) {

'use strict';

return C3.ModuleController.extend({

name: 'radio.module.RadioController',

layoutClass: RadioModuleLayout,

region: MsgBus.reqres.request('application:page').c3Body,

// Initialize function that will be called after instantiating the controller object

initialize: function (options) {

console.log("Radio controller initialized");

},

startRadioModule: function () {

var self = this;

require(['modules/radio/radiosubapp1/radio-subapp1'], function (RadioSubApp1) {

var radioSubApp1 = self.createSubApp(RadioSubApp1,{region: self.getLayout().RadioSubApp1Information });

radioSubApp1.start();

});

require(['modules/radio/radiosubapp2/radio-subapp2'], function (RadioSubApp2) {

var radioSubApp2 = self.createSubApp(RadioSubApp2,{region: self.getLayout().RadioSubApp2Information });

radioSubApp2.start();

});

}

});

});

Part

3

Part 3 Design Patterns

Chapter 9 C3 Design Patterns

This chapter gives an overview of the C3 design Patterns

t

his chapter gives an overview of C3 Design patterns and how to make use of it adhering to business requirements.

# 9.1 C3 Design Patterns

🗐C3 understand not every application address the business in same way and each has its own needs to address the requirements.

To address and standardize the business logic needs, C3 visualize individual business needs as set of repeatable patterns viz.,

* Simple Layout Design Pattern
* Mashup (Composite) design Pattern
* Transactional Flow Design Pattern
* Singleton View Pattern
* Module Design Pattern

# 9.2 Simple Layout Design Pattern - Approach

* This is the most common pattern. Display every view via Layout.
* Layout is encouraged over direct View display as it provide great flexibility in managing the views
* Even if it’s a simple View prefer to be displayed via Layout.
* The view itself is a simple C3.Layout object which renders itself after the layout initialized.
* Then the module itself returns this rendered layout, whoever uses this module, just instantiate the layout object.
* Welcome module in the RTC ref\_app is an example (welcome-layout.js). The model will be initialized when the view is instantiated and once the model is loaded

**Components Involved**

|  |  |  |  |
| --- | --- | --- | --- |
| S.No | Component Name | Required/Optional | Comments |
| 1 | C3.Application | Required |  |
| 2 | C3.AppRouter | Required |  |
| 3 | C3.AppController | Required |  |
| 4 | C3.Layout | Required |  |
| 5 | C3.View | Required |  |
| 6 | C3.Model | Optional | Needed only if data is pulled from the backend |
| 7 | Module, C3.Subpp, C3.SubAppRouter, C3.SubAppController. | Optional | Needed if it’s a separate module/subapp |



# 9.3 Mashup(Composite) Design Patterns – Approach

* This is similar to a tile view – which has a layout that has predefined regions.
* Each of the regions will display the predefined modules.

Our dashboard example is the good example of this design pattern

|  |  |  |  |
| --- | --- | --- | --- |
| S.No | Component Name | Required/Optional | Comments |
| 1 | C3.Application | Required |  |
| 2 | C3.AppRouter | Required |  |
| 3 | C3.AppController | Required |  |
| 4 | C3.Layout | Required |  |
| 5 | C3.View | Required |  |
| 6 | C3.Model | Optional | Needed only if data is pulled from the backend |
| 7 | Module, C3.Subpp, C3.SubAppRouter, C3.SubAppController. | Required | Needed since Mashup is all about display each region with separate module |



# 9.4 Transaction Flow Design Patterns – Approach

* The TBOA example is a good example for this design pattern –
* the SubApp controller type is used to make transaction when the event happens,
* FlowBaseView pass the returned application state (model data) to the next page.

|  |  |  |  |
| --- | --- | --- | --- |
| S.No | Component Name | Required/Optional | Comments |
| 1 | C3.Application | Required |  |
| 2 | C3.AppRouter | Required |  |
| 3 | C3.AppController | Required |  |
| 4 | C3.Layout | Required |  |
| 5 | C3.View | Required |  |
| 6 | C3.Model | Required | Needed since we exclusively gonna deal with passing data from one page to another. |
| 7 | Module, C3.Subpp, C3.SubAppRouter, C3.SubAppController. | Optional |  |
| 8 | C3.FlowBaseView | Required | This View is exclusively deals with passing data after submit:success event |



# 9.5 Singleton View Design Patterns – Approach

In web applications there are often objects in the user interface that only exist once. For example, a navigation bar at the top of the page, or a search box, or information about the user that is currently logged in. When we have these objects in a normal program we would use a global variable, or more eloquently a singleton pattern. A singleton is a nice way of auto-instantiating an object on demand and maintaining a single instance under what is more or less a global variable.

The Singleton View pattern is a simple pattern that changes the way a view is instantiated when it only needs one instantiation throughout the entire application.

Module Controller and Subapp controller can return a singleton instance of view/model using createViewSingleton and createModelSingleton methods respectively.

Developers can also bring up their own singleton pattern while instantiating the objects.

define([

'underscore',

'c3',

'modules/radio/radiosubapp1/layout/radiosubapp1-body-layout'

], function (\_, C3, RadioSubApp1FlowLayout) {

'use strict';

return C3.SubAppController.extend({

// A name is not required, however, it's highly recommend for debugging purpose

// As javascript doesn't have classname similar to other language such as java

name: 'radiosubapp1.radiosubapp1Controller',

// Mandantory fields.

layoutClass: RadioSubApp1FlowLayout,

initialize: function () {

console.log('RadioSubapp1 controller initialized');

},

// Defines the entry point for subapp

start: function () {

this.\_radioSubApp1Start();

},

\_radioSubApp1Start: function () {

var self = this;

console.log('Radiosubapp1 controller:show');

// Instantiate the view

require([

'modules/radio/radiosubapp1/views/radiosubapp1-success-view1',

'modules/radio/radiosubapp1/views/radiosubapp1-success-view2',

'modules/radio/radiosubapp1/models/radiosubapp1-model1'

], function (RadioSubApp1View1, RadioSubApp1View2, RadioSubApp1Model1) {

RadioSubApp1Model1.enableChannel(self.channel);

var RadioSubApp1View1 = self.createViewSingleton(RadioSubApp1View1,{model: RadioSubApp1Model1} );

var region1 = self.getLayout().RadioSubApp1View1Information;

# 9.6 Module Design Patterns – Approach

* To enable a Modular design and development for Citi apps, an approach has been prescribed that allows for developers / teams to create their modules independently and integrate with the main app when ready.
* This allows for functionalities to be added to the main app in a highly decoupled manner.
* There is plenty of scope for additional improvements to this approach by way of tooling and writing additional framework components. What is described here is the foundational structure that allows modules to be created and easily integrated into the main app.
* The mechanics of this structure are demonstrated and can be seen in action in the Reference App. With this consideration, let’s look at how this is achieved.

Every module is composed of:

* Module (definition): In reference app, these following the naming convention <moduleName>-module.js and are the top most elements in the modules / <moduleName> folder.
* Module Router: In reference app, these following the naming convention <moduleName>-router.js and are placed in the modules / <moduleName> /routers folder
* Module Controller: In reference app, these following the naming convention <moduleName>- controller.js and are placed in the modules / <moduleName> /controllers folder
* Module level functions: If there are any modular functions that don’t live in a dedicated sub-app, then their corresponding Views are placed in modules/<moduleName>/Views folder. Same applies to the Models, templates etc. components as well.
* Sub-apps: Sub-Apps are more granular implementations of a functionality that add up to make a complete module’s features. All subapps share the module’s definition, router and controller.

<moduleName>-module.js: This is the module initialization file. This acts as the entry point to the module.

Using the standard Javascript modularization mechanisms (be sure to check out Additional References section for a great write up on standard Javascript modularization techniques), any module specific can be exposed from this file since this acts as the “face” of the module.

The minimum that needs to be achieved here is basic initialization for the module – setting up the module router and associating it with the module’s controller file.

To initialize the module router, the first route is triggered to it at the module initialization step.

<moduleName>-router.js: The Module level router is similar to the app-router except that the module router is restricted to its specific module only.

Hence, it contains all the routes to be handled by the module. Since these are routes for a single module, they should follow the convention of having the same root for the module (eg: /dashboard for the dashboard module).

As with the app-router, the route handles will be encapsulated in the corresponding <moduleName>-controller.js.

<moduleName>-controller.js: Similar to the app-controller, this module level controller also extends from C3.Controller.

The key difference between app-controller and module controller is the scope (app vs module). The main job of the controllers (in general) that is clearly visible in the module-controller is:

* Provide initialization and / or cleanup tasks for all sub-apps
* Provide a handler for all module routes
* Load dependency using require for each router / handler as needed
* Create the Layout object for the module. Note that a module may have several different layout objects as well. In that case, you can define the different layout objects and use them accordingly. For most use cases, though, a single layout object is enough.
* Instantiate the View and orchestrate the functionality (typical controller function)

Sub Apps: A Sub-app will typically contain at least:

* View: Views can be extended from C3.View. A View is closely related to a model or a Collection as its main job is to ‘manage’ the model and render it.
* Template: At least one template is associated with a view. The template’s responsibility is to provide the structure of the html along with the model and content placeholders. These placeholders will get populated by the model and contentMap objects when the view generates the DOM from the template, usually in its render() function. One view 🡨🡪 One template relationship is usually enough but if required, you can use multiple templates for aggregating a single view’s UI.
* Model: Models (and their plural – Collections) have the key responsibility of providing the data to a view to update its UI. The data in the models can be retrieved from form fields or various datastores or service calls.

***Module Example***

The payment and transfer module (pnt) is a transactional module that handles all payment and transfer related functionalities. The current functionality has been divided into smaller functions, such as make a payment, manage payee, add a drawee, see a drawee, transfer between own accounts (TBOA), local inter Citi transfer, and related activities. Each of these function areas can be defined as a sub app.

So the module itself, we let it drive the module display layout through templates, and workflow through the module level application router and application controller.

The pnt-layout.js defines the regions: header, leftNav, and appBody.

define([

'c3',

'templates',

'modules/pnt/module/views/links-view'

], function(C3, JST, LinksView) {

'use strict';

return C3.Layout.extend({

name: 'pnt.module.pntLayout',

template: JST['modules/pnt/module/templates/layout.hbs'],

tagName: 'main',

attributes: {

'role': 'main',

'data-src': 'pnt-layout'

},

regions: {

header: '#citi\_pnt\_header',

leftNav: '#citi\_pnt\_quickTasks',

appBody: '#citi\_pnt\_accountsPanel'

},

// defaultViews will be displayed automatically after layout is displayed

defaultViews: {

leftNav: LinksView

}

});

});

The app router defines the routes and its relations with the app controller through the handler (startTboa())

define([

'c3'

], function (C3) {

'use strict';

return C3.ModuleRouter.extend({

subRouterCtx: 'pnt',

appRoutes: {

'': 'startTboa',

'tboa': 'startTboa',

'tboa/start': 'startTboa'

}

});

});

The module controller (pnt-controller.js) is the mediator that defines the module’s actual workflow.

define([

'c3',

'msgbus',

'modules/pnt/module/layouts/pnt-layout'

],function(C3, MsgBus, PntLayout) {

'use strict';

return C3.ModuleController.extend({

// A name is not required, however, it's highly recommend for debugging purpose

// As javascript doesn't have classname similar to other language such as java

name: 'pnt.module.PntController',

// Mandantory fields.

// They can be set either 1. here in class definition or

// 2. in constructor -> new ModuleController({layoutClass: PnTLayout, region: MsgBus.reqres.request('main:region')})

layoutClass: PntLayout,

region: MsgBus.reqres.request('main:region'),

// Initialize function that will be called after instantiating the controller object

initialize: function(options) {

console.log("pnt controller initialized");

},

// Matching function for tboa route in routers

startTboa: function() {

console.log("Start TBOA subapp");

// get layout instance, this is a platform function

var tboaContainerRegion = this.getLayout().appBody;

require(['modules/pnt/tboa/tboa-subapp'], function(TboaSubApp) {

// Create Tboa subapp and call start() function to start it.

var tboaSubapp = new TboaSubApp({region: tboaContainerRegion});

tboaSubapp.start();

});

}

});

});

The module entry point (pnt-module.js) actually initializes the application router with the controller:

define([

'c3',

'modules/pnt/module/routes/pnt-module-router',

'modules/pnt/module/controllers/pnt-module-controller'

], function(C3, PntModuleAppRouter, PntModuleController) {

'use strict';

// Instantiate Subrouter so that proper handler can be called for all module level routes

// Singleton: this will only be executed once, the reference will be managed by require.js

return new PntModuleAppRouter({

subRouterCtx: 'pnt',

controller: new PntModuleController()

});

});

***sub apps Example***

* In this example, we take the reference app’s TBOA sub app, and come up a dummy activity sub app for demonstration purpose.
* The sub app itself has a transactional flow: input->recap->confirmation.
* The flow layout is defined through handlebars template(flow-layout.hbs) and the layout object (flow-layout.js). The flow workflow is defined by the controller object (tboa-controller.js).
* The input view object (tboa-input-view.js) defines the list of from account list and to account list through the handlebars template (tboa-input.hbs).
* A partial helper (“selectAccountInput”) registered with the select-account-input.hbs template. The view events are defined and the view is also listening to the model change events (that is, whenever the from account changed, the two account list has to be populated accordingly).
* Additional information will be appended to the input view through the tboa-additional-input-view.js view object.
* Then submit event handler will lead the flow to recap view; finally to the confirmation view. The controller is also mediating the data model from one view to the other through a context object defined in the controller.

Chapter 10 Navigation Related Tasks

This Chapter provides an overview of all the possible Navigate related tasks that can be achieved using C3.

t

his chapter provides an overview of how to do navigation in C3 and how to handle the specific use case for navigations.

# 10.1 **Navigation Related Task**

* Use C3 Router and route helpers (Application, Module, SubApp) to define the navigation routes.
* Handle the HREF present in HTML controls.
* Handle back, cancel – Rollback Links
* Handle Sequence Protection
* Handle Do Another – Repeat Functionality Links
* Handle Transactional Locking

.

# 10.2 How to think about Routing

It’s important that we define the router’s role, in order to design our app properly.

All a router does is

• execute controller actions corresponding to the URL with which the user first “entered” our C3 app. It’s important to note that the route-handling code should get fired only when a user enters the application by a URL, not each time the URL changes. Put another way, once a user is within our C3 app, the route-handling shouldn’t be executed again, even when the user navigates around;

• update the URL in the address bar as the user navigates within the app (i.e. keep the displayed URL in sync with the application state). That way, a user could potentially use the same URL (by bookmarking it, emailing it to a friend, etc.) to “restore” the app’s current configuration (i.e. which views are displayed, etc.). Keeping the URL up to date also enables the browser’s “back” and “forward” buttons to function properly.

It’s very important to differentiate triggering routing events from updating the URL. In traditional web frameworks, actions are triggered by hitting their corresponding URLs. This isn’t true for javascript web applications

.And now that we have a basic app functioning as we want it to, we’ll add in a router to manage the URL-related functionality. Our router will only get triggered by the first URL it recognizes, resulting in our app getting “initialized” to the correct state (i.e. showing the proper data in the proper views).

After that initialization step has fired once, the router only keeps the URL up to date as the user navigates our app: changing the displayed content will be handled by our controllers, as it has been up to now

# 10.3 Push State - Approach

C3 allows you to leverage HTML5’s pushState functionality by changing your

history starting code to C3.history.start({pushState: true});

When using pushState, URL fragments look like the usual “/contacts/3” instead of “#contacts/3”.

This allows you to serve an enhanced, javascript-heavy version of the page to

users with javascript-enabled browsers, while serving the basic HTML experience to clients without javascript (e.g. search engine crawlers).

Be aware, however, that to use pushState in your application your server has to respond to that URL. This is a frequent error when trying out pushState.

You’re free to have your server systematically respond with your index.html page

regardless of the requested URL, but something needs to be sent to the client when the

URL is requested (e.g. when loading a bookmark). When sending index.html to all client

requests, you’re bascially delegating the URL resolution to your C3 app:

when the browser will load index.html, the app will start along with the route-handling code, which will load the correct application state (since the route corresponding to the URL requested by the client will get triggered).

.

# 10.4 Push State - Approach

We can make use of C3.Navigate to navigate to particular URL or Fragments.

Since checking the current URL fragment and keeping it up to date are things we’ll be doing quite frequently as we develop our app. C3.Navigate comes real handy for that.

We check the URL and URL Fragments and redirect the user.

“Redirecting” is a bit of a misnomer: we’re not redirecting anything (as we would be with a server), we are just

• updating the URL with the proper fragment

• executing the proper controller action, which will display the desired views

Note that C3.Navigate doesn’t just change the URL fragment; it also adds the new URL to the browser’s history. This, in turn, makes the browser’s “back” and “forward” buttons behave as expected.

# 10.5 Navigate using Events - Approach

When we call C3.Navigate, it manually changes URL and calling a controller action if the URL contains no fragment. But that isn’t very DRY, we’ll end up setting route fragments and calling controller methods everywhere, and it will be a nightmare to maintain.

if(this.getCurrentRoute() === ""){

App.trigger("tboa:home");

}

App.on("tboa:home", function(){

App.navigate("home");

});

# 10.6 Handling Anchor Tags

There are three options to handle the anchor tags.

* always return false in the handler or call preventDefault on the event
* use data-event-override=’backbone’ this will allow C3 to override the behavior of anchor
* Use <button> whenever you are using <a> with a dummy href.

# 10.7 Back/Cancel Link – Handling

C3 got you covered providing two ways to handle the requirement

* Hide and display Regions
* Memento Pattern - provides a stack to push and pop changes to models and collections

***Hide and Display Regions***

One trick to mimic Make changes and preserve the old data entered by the Customer is to hide the previous Display regions instead to close it.

Note: Closing a region/view will remove all the data associated with it.

Once user navigates to next page, we hide the region and show new region. When customer clicks on make changes, we bring back the old region.

\_tboaCombinedInput: function () {

var region3 = this.getLayout().region3;

region3.reset();

var region1 = this.getLayout().region1;

region1.$el.show();

var region2 = this.getLayout().region2;

region2.$el.show();

},

# 10.8 Sequence – Handling

* Page Sequence is maintained from server side via a Unique Sequence Key.
* Client ensure the key is passed back as part of the PRG.
* Flow Base View suits best for this.

# 10.9 Transactional Locking

* When we have decided we are going to submit the form, we call **sfLock()** to activate the locking. The locking will automatically be cleared when the next .jsp is displayed.
* The method ConfirmGoLock() checks to see if the framework is already locked and displays a popup to the customer that he is not allowed to click away at this time because a transaction is in process.
* The appropriate phrase must exist in your phrase bundle. The phrase used is in the jba\_common phrases, so it is in your bundle if you included the appID jba\_common in your list of appIDs in your JFPLangConfig-{subapp}.xml file. {Refer to *Adding Phrases* section for details}.

Chapter 11 MFA and Unauthorized Features

This chapter gives an overview of the requirements and approach for MFA and Unauthorized Features

t

his chapter provides an overview of the MFA and UnAuthorized and how to integrate with C3. C3 provides support for both MFA and UnAuth features integration in the application.

# 11.1 MFA and Unauthorized Page – Handling

* C3 Provides support to integrate MFA(Multifactor authentication)
* If Any server call(via model invocation) returns response code as 400 or 403, C3 invoke the MFA or UnAuth component based on the response code and response name present in the response.
* C3 expects MFA/UnAuth component extending from C3 Components to be configured for this purpose.
* Development team can write their own MFA/UnAuth component of their choice and configure in the C3 bootstrap application configuration data for the C3 platform to invoke, when in need.
* C3 expects the MFA/UnAuth Component to invoke C3 Radio event “mfa:validated” or “unauth:validated” respectively to indicate the component have completed its job of validation.
* C3 Platform listen onto these events described above and resubmit the original request fired before the MFA/UnAuth invocation.

Chapter 12 Site Catalyst and Anti Malware

This chapter provides an overview of the requirements and approach of the Site catalyst and Anti Malware Features in C3.

t

his chapter provides an overview of the site catalyst and anti malware features and how to integrate in C3.

.

# 12.1 Site Catalyst Tags - Defined

Normally sitecatalyst is tracked in two events in a page – Onload and OnClick

***Onload Tracking:***

This tracking happens when the page is loaded. The sitecatalyst variables that need to be tracked for a particular page are proposed in the pageDef content.



Inside the cmwa helper, the pageDefs are processed one by one and the values are formatted.

There can be 2 types of siteCatalyst cmNodes present in the pageDefs.

Static variables and Dynamic variables.

**Static Variables**

The cmNode itself will have the value.

Example:

<cmnode id="prop1" type="string" hasParseData="false">**Public**</cmnode>

<cmnode id="prop2" type="string" hasParseData="false">**SignOn**</cmnode>

<cmnode id="prop3" type="string" hasParseData="false">**SignOn**</cmnode>

<cmnode id="prop4" type="string" hasParseData="false">**Details**</cmnode>

<cmnode id="pagename" type="string" hasParseData="true">**Challenge Question Input**</cmnode>

In this case, the values are assigned to Sitecatalyst variables without any processing like

s.prop1 = “Public”

s.prop2 = “SignOn”

s.pageName = “Challenge Question Input”

**Dynamic Variables**

The cmNode will have # as its value

Example:

<cmnode id="eVar10" type="string" hasParseData="false">#</cmnode>

<cmnode id="eVar40" type="string" hasParseData="false">#</cmnode>

<cmnode id="eVar41" type="string" hasParseData="false">#</cmnode>

<cmnode id="eVar42" type="string" hasParseData="false">#</cmnode>

In this case, these sitecatalyst variables get the values from as options passed in the cmwa helpers.

Developer Perspective:

When a new page is added, the cmNodes will be provided by the CMS team. And the existing implementation will populate all the sitecatalyst variables that are already handled. So CBOL developer may not have to make changes.

But if the requirement needs a new Sitecatalyst dynamic variable to be added, the developer needs to write the Implementation as from where the variable value can be obtained.

Syntax:

{{#**cmwa input**=**'{"evar40": "hi", "evar41": "hi2" }'** }}{{/**cmwa**}}

Dynamic Variables

# 12.2 Anti Malware Tags – Requirements and Handling

🗐 **IBM Trusteer will provide Pin Point anti-malware product which will be integrated in the Citibank Online Applications of Citibank NA consumers**: The product will offer Active and Passive malware detection in the client side of Citibank online applications. The malware will be detected in the client side browser by Pinpoint product. Trusteer will share the malware detection information with Citi Fraud and SOC teams.

As part of Pinpoint products, Trusteer will provide four JavaScripts which will be integrated in various functional modules in the application's UI layer. The functional modules will be referred to as touch points in the design document and Trusteer provided JavaScripts will be referred as Trusteer Snippets.

Passive Detection Pinpoint Snippet: - The passive code snippet will be integrated in all the pages in the application. The required information to be passed from online application to Trusteer cloud servers are Session Identifier, User Agent, and IP address. The Session Identifier will be provided by Citi application while User Agent and IP address will be retrieved by Trusteer from User Session. The Session Identifier will be existing JFPSessionId which is unique random number for each session.

UID API Collectors snippets: - The User identifier will be used for identification of the user by Citi Fraud. The user identifier will be returned at the page immediately after login. The corresponding function to retrieve the user identifier will be implemented in amwCommon. However, in the application level the function will be overridden in application level Java Script.

Malware Detection Snippet: -The malware detection snippet will be integrated in in login and alternative login pages where user can authenticate to the landing pages.

Pinpoint Session Reset Code: - The session reset will be integrated only in log off module.

Platform team will implement three different tags which will integrate Trusteer provided three type of snippets (malware detection, User Data Collector, session reset code).

Platform will provide following handlebar helpers

1. amwSignOn
2. amwPostSignOn
3. amwSignOffTag
4. amwCommon

The details of the tags are as follows:-

***amwSignOn*:-** This tag will be used to include Trusteer’s active malware detection snippet and the callback function for session Identifier collection. This tag will be integrated in the login and alternative-login pages.

***amwPostSignOn*:-** This tag will be used to include Trusteer’s User Data Collector snippet. As also the callback functions for session Identifier and User Identifier will be implemented here. This tag will be integrated in the landing pages.

***amwSignOff:-*** This tag will be used to includeSession Reset Code Snippet and the callback function for session Identifier. This tag will be integrated in the log off.

***amwCommon:*** This tag provide the JFPSessionID

Configuration:

In the code snippets there are three variables pertaining to application would be required to feed. The dynamic variables are SNIPPET\_ID, HOST and PAGE\_NAME.

Examples:

SignOn:

{{#amwSignOn jfp\_amw\_host\_sign\_on =" xxxx" jfp\_amw\_snippet\_id=”xxx”

jfp\_amw\_page\_name\_sign\_on=”xxxx”

}}{{/ amwSignOn }}

PostSignon:

{{#amwPostSignOn jfp\_amw\_host\_post\_sign\_on=" xxxx" jfp\_amw\_snippet\_id=”xxx”

jfp\_amw\_page\_name\_post\_sign\_on=”xxxx”}}{{/ amwPostSignOn }}

SignOff:

{{#amwSignOff jfp\_amw\_domain=" xxxx" jfp\_amw\_snippet\_id=”xxx”

}}{{/ amwSignOff}}

amwCommon:

{{# amwCommon jfp\_amw\_host\_common =" xxxx" jfp\_amw\_path\_foldername =”xxx”

jfp\_amw\_page\_name\_common=”xxxx” }}{{/ amwSignOff}}

Chapter 13 C3 Content Service

This chapter provides an overview of the requirements and approach of the C3 Content Service.

t

his chapter provides an overview of the C3 Content Service. C3 Client need the following attributes to be given in the model appid, screened.

# 13.1 C3 Content Service - Enablement

C3 content service is disabled by default, how to enable it when needed.

Update the static cache configurations for JSONCMCache and JSONBrandingCache by following the chain resource loader overwriting mechanism. Create file 150-JFP-Content-Service.xml with below configuration and copy it to \WEB-INF\config\jfp\portal\cache\JPSDBCacheConfig.xml.d folder.

At Server side.

<!-- All dependent caches which are part of Content service must be defined in this file, these caches are added to end so that main caches are loaded -->

<cache-config>

<static-cache name="JSONCMCache" enabled="true">

</static-cache>

<static-cache name="JSONBrandingCache" enabled="true">

</static-cache>

</cache-config>

Please ensure that following files are present in the application for JFP content service to work properly.

\WEB-INF\classes\resources\CacheConfig\jfp-collectioncache-ehcache.xml

\WEB-INF\config\jfp\cache\spring-application-context-cltehcache.xml

\WEB-INF\config\jfp\portal\cache\JPSDBCacheConfig.xml.d\100-JFP-Content-Service.xml

# 13.2 C3 Content Service – Content to Return

🗐 **Inform JFP Content Service on what content it needs to return for a particular request**

As html generation happens on C3 client, JFP content service is created to retrieve content. Content retrieval depends on mapping configuration. Mapping configuration is unique for every view. Each view should provide screenID and appid as part of C3.View definition.

Below configuration should be made available for content service under \WEB-INF\config\jfp\content. This configuration gets loaded through chainresourceloader and can be easily overwritten through .d mechanism

Default filename: ContentMapping.xml

<content-mappings> <!--Parent element only one allowed per file -->

/\*\*

\* Below element is allowed one per view

\*/

<content-mapping screenid=”value” appid=”value”>

/\*\*

\* comma separated list of appid/contenttype/contentid

\*/

</content-mapping>

</content-mappings>

Overwriting folder: ContentMappings.xml.d

# 13.3 C3 Content Service – Debugging Approach

Change below loggers to COMMONS\_DEBUG to capture additional information for debugging content service.

ObjectLogger.ContentService.LogLevel=COMMONS\_DEBUG

ObjectLogger.ContentService.Owner=ContentService

ObjectLogger.ContentService.LogListeners=ContentServiceLog

ObjectLogger.JSONCMCache.LogLevel=COMMONS\_DEBUG

ObjectLogger.JSONCMCache.Owner=CMCache

ObjectLogger.JSONCMCache.LogListeners=JFPCacheControllerLogFile

ObjectLogger.JSONBrandingCache.LogLevel=COMMON\_DEBUG

ObjectLogger.JSONBrandingCache.Owner=CMCache

ObjectLogger.JSONBrandingCache.LogListeners=JFPCacheControllerLogFile

ContentService logger will log into contentservice/cs.log

JSONCMCache, JSONBrandingCache will log into cache/JFPCacheController.log. Check for [JSONCMCache] and [JSONBrandingCache] token in the log file if they are present as shown below then caches are created successfully. Also ensure [CMCache] and [BrandingCache] tokens are present above [JSONCMCache] and [JSONBrandingCache] in the log file.

DONE refresh cache [JSONCMCache] in 0 ms.

DONE refresh cache [JSONBrandingCache] in 0 ms.

To see pagemapping entries present in JSONCMCache copy the below code to a jsp.

<%@ page import="com.citigroup.cdcla.jfp.cache.JFPCacheController, com.citigroup.cdcla.jfp.cache.JSONDependentCache, java.util.Iterator, net.sf.ehcache.Element" %>

<%

JFPCacheController controller = JFPCacheController.getInstance();

// get the specific cache reference

JSONDependentCache jsoncache = (JSONDependentCache)controller.getCache("JSONCMCache");

if (jsoncache == null){

out.println("<H2> No registered jsoncache found !!!");

return;

}

else{

out.println(jsoncache.getCltEHCache().getSize());

out.println("<TABLE border=1 cellpadding=10><TR>");

out.println("<TH>KeyName</TH><TH>Size</TH><TH>Count</TH><TH>Value</TH></TR>");

Iterator iter = jsoncache.getCltEHCache().getKeys().iterator();

while ( iter.hasNext() ) {

String key = (String)iter.next();

/\* To see all keys present in jsoncache, remove the key.contains(“pagemapping”), also to see specific content type in jsoncache change the pagemapping to copy or page\_def or config\*/

if(null != key && key.contains("pagemapping")){

Element element = jsoncache.getCltEHCache().get(key);

out.println("<TR><TD>" + key + "</TD>");

out.println("<TD>" + element.getValue() + "</TD>");

out.println("</TR>");

}

}//end while

}%>

# 13.4 C3 Content Service – Configurations

The below file will having the Content service configurations.

File name: WEB-INF/config/jfp/portal/JPS.properties.d/050-JFP-content-service.properties

# This file holds all the configuration needed for Content Service

# Content ID which holds the mapping of screen ID to app id

citigroup.jfp.contentservice.mappingtoappid.contentid=MarketingPageAppID

# APP ID of content ID record which is holding mapping of screen ID to app id

citigroup.jfp.contentservice.mappingtoappid.appid=cbol\_common

# Content Type of content ID record which is holding mapping of screen ID to app id

citigroup.jfp.contentservice.mappingtoappid.contenttype=config

# Default App ID to be used when the config record is not found

citigroup.jfp.contentservice.defaultappid=JRSPRODSERV

# Default page\_def content ID to be used when page\_def content ID is not calculated

citigroup.jfp.contentservice.pagedef.defaultcontentid=jJRSPRODSERV\_ProdServDetail

# Default branding content ID to be used when branding not found in page\_def

citigroup.jfp.contentservice.branding.defaultcontentid=DefaultBranding

citigroup.jfp.branding.content.appId=JPSCBOL

jfp.contentmap.config.file=/resources/JPSConfig/ContentMapping.xml

# topmenu rule contentID

citigroup.jfp.contentservice.topmenu.rule.contentid=@citigroup.jfp.contentservice.topmenu.rule.contentid@

# topmenu rule AppID

citigroup.jfp.contentservice.topmenu.rule.appid=@citigroup.jfp.contentservice.topmenu.rule.appid@

#Left menu rule contentid

citigroup.jfp.contentservice.leftmenu.rule.contentid=@citigroup.jfp.contentservice.leftmenu.rule.contentid@

#Left menu rule appID

citigroup.jfp.contentservice.leftmenu.rule.appid=@citigroup.jfp.contentservice.leftmenu.rule.appid@

# topmenu mark up contentID

citigroup.jfp.contentservice.topmenu.markup.contentid=@citigroup.jfp.contentservice.topmenu.markup.contentid@

# topmenu mark up AppId

citigroup.jfp.contentservice.topmenu.markup.appid=@citigroup.jfp.contentservice.topmenu.markup.appid@

#Left menu markup contentid

citigroup.jfp.contentservice.leftmenu.markup.contentid=@citigroup.jfp.contentservice.leftmenu.markup.contentid@

#Left menu markup appid

citigroup.jfp.contentservice.leftmenu.markup.appid=@citigroup.jfp.contentservice.leftmenu.markup.appid@

# 13.5 C3 Content Service – Top Navigation Menu

C3 Content Service does not know how to retrieve Top Navigation menu unless we write special rule.

To generate html markup for the top navigation menu we need a rule (CITI\_PERSONALIZATION\_RULE Record – Content\_Type as ‘phrase’, Rule\_Engine\_Type as ‘JFP’) and a template (CITI\_CM\_CONTENT Record – Content\_Type as ‘copy’).

Rule specifies which menu content id to utilize based on page def content id. By default platform expects the rule content\_id as “topNavigationMenuRule” and app\_id as “JPSNAVIGATION”

Below is sample rule content and PAGE\_MAPPING key will have page\_def content id

var returnVal = new Packages.java.util.Vector();

var mapping = inParams.get("PAGE\_MAPPING");

var branding = inParams.get("branding\_id");

var pageDef = inParams.get("pageDef");

if(mapping == "jaddPayee\_JPSCBOL"){

/\*\*

\* Must only add values in the format contentID.appID

\*/

returnVal.add("Menu\_TopNav.JPSCBOL");

}

return returnVal;

To inform JFP content service to use different rule content\_id and app\_id configure it through JFPGlobals.properties

citigroup.jfp.contentservice.topmenu.rule.contentid= topNavigationMenuRule

citigroup.jfp.contentservice.topmenu.rule.appid=JPSNAVIGATION

Template provides the format of html structure on which menu widgets can work and beautify. By default platform expects the template content\_id as “topNavigationMenuStyle”, app\_id as “JPSNAVIGATION” and content\_type as “COPY”

Below is sample menu template markup. Highlighted markup containing dynamic variables are needed to pass data from the above rule to generate html markup

<cmroot id="topNavigationMenuStyle">

<cattributes>

<cmnode id="CONTENT\_ID" type="string" hasParseData="false">topNavigationMenuStyle</cmnode>

<cmnode id="NAME" type="string" hasParseData="false">Top Navigation Menu Markup</cmnode>

<cmnode id="TEXT" type="string" hasParseData="true">

&lt;mset var=&#034;tree&#034; appId=&#034;${top\_nav\_app\_id}&#034; contentId=&#034;${top\_nav\_content\_id}&#034; cacheFlag=&#034;false&#034;/&gt;

&lt;div class=&#034;jfpw-menu-accordion-container&#034;&gt;

&lt;ul id=&#034;side-left-nav-menu&#034; class=&#034;jfpw-lvl-0&#034;&gt;

&lt;jfpforeach var=&#034;menuItemL1&#034; items=&#034;tree.item.menuItems&#034;&gt;

&lt;jfpif test=&#034;${menuItemL1.isMenu == false}&#034;&gt; …………………………….</cmnode>

</cattributes>

</cmroot>

# 13.6 C3 Content Service – Left Navigation Menu

C3 Content Service does not know how to retrieve Left Navigation menu unless we write special rule.

To generate html markup for the left navigation menu we need a rule (CITI\_PERSONALIZATION\_RULE Record – Content\_Type as ‘phrase’, Rule\_Engine\_Type as ‘JFP’) and a template (CITI\_CM\_CONTENT Record – Content\_Type as ‘copy’).

Rule specifies which menu content id to utilize based on page def content id. By default platform expects the rule content\_id as “leftNavigationMenuRule” and app\_id as “JPSNAVIGATION”

Below is sample rule content and PAGE\_MAPPING key will have page\_def content id

var returnVal = new Packages.java.util.Vector();

var mapping = inParams.get("PAGE\_MAPPING");

var branding = inParams.get("branding\_id");

var pageDef = inParams.get("pageDef");

if(mapping == "jaddPayee\_JPSCBOL"){

/\*\*

\* Must only add values in the format contentID.appID

\*/

returnVal.add("Menu\_LeftNav.JPSCBOL");

}

return returnVal;

To inform JFP content service to use different rule content\_id and app\_id configure it through JFPGlobals.properties

citigroup.jfp.contentservice.topmenu.rule.contentid= leftNavigationMenuRule

citigroup.jfp.contentservice.topmenu.rule.appid=JPSNAVIGATION

Template provides the format of html structure on which menu widgets can work and beautify. By default platform expects the template content\_id as “leftNavigationMenuStyle”, app\_id as “JPSNAVIGATION” and content\_type as “COPY”

Below is sample menu template markup. Highlighted markup containing dynamic variables are needed to pass data from the above rule to generate html markup

<cmroot id="leftNavigationMenuStyle">

<cattributes>

<cmnode id="CONTENT\_ID" type="string" hasParseData="false">leftNavigationMenuStyle</cmnode>

<cmnode id="NAME" type="string" hasParseData="false">Left Navigation Menu Markup</cmnode>

<cmnode id="TEXT" type="string" hasParseData="true">

&lt;mset var=&#034;tree&#034; appId=&#034;${left\_nav\_app\_id}&#034; contentId=&#034;${left\_nav\_content\_id}&#034; cacheFlag=&#034;false&#034;/&gt;

&lt;div class=&#034;jfpw-menu-accordion-container&#034;&gt;

&lt;ul id=&#034;side-left-nav-menu&#034; class=&#034;jfpw-lvl-0&#034;&gt;

&lt;jfpforeach var=&#034;menuItemL1&#034; items=&#034;tree.item.menuItems&#034;&gt;

&lt;jfpif test=&#034;${menuItemL1.isMenu == false}&#034;&gt;………………………………………………………….

</cmnode>

</cattributes>

</cmroot>

# 13.7 C3 Content Service – Branding

JFP Content Service attempts to retrieve branding content using page definition of the page. Page definition content id is built using screenID and appid request parameters. All configurations applicable to Branding Framework are utilized to retrieve the branding

Branding can be retrieved using {{cmbranding}} helper in C3 client. More information on the helpers can be found in [{{cmbranding}} helper](#_{{cmbranding}}_helper)

# 13.8 C3 Content Helpers

Once the content is available in C3 model, platform provides following content helpers to use inside view templates. Content helpers are responsible to retrieve content from C3 model and aid in generation of the view.

**{{cmout}} helper**

cmout helper is used to retrieve content from C3 model and supports all content types except rule, cmmenu

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Required** | **Description** |
| contentid | true | Content ID holding the content in the DB |
| type | true | Content type holding the content in the DB |
| appid | true | App ID holding the content in the DB |
| attribute | true | XPATH expression of the node which we want to retrieve. No run time variables are allowed |

Example usage:

{{cmout contentid=”value” type=”copy” appid=”cbolcommon” attribute=”CONTAINER.REPLICANT[0].TEXT”}}{{/cmout}}

**{{cmlink}} helper**

Cmlink helper is used to generate anchor <a/> element on the c3 client. Cmlink helper only works with cmlink content type records

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Required** | **Description** |
| contentid | true | Content ID holding the content in the DB |
| appid | true | App ID holding the content in the DB |
| class | false | If passed will have precedence over content STYLE\_CLASS node while generating “class” attribute value of <a /> |
| target | false | If passed will have precedence over content TARGET node while generating “target” attribute value of <a /> |
| bodyContent | false | If passed will have precedence over content DESCRIPTION node while generating “link text” of <a /> |
| title | false | If passed will have precedence over content TOOLTIP node while generating “title” attribute value of <a /> |
| id | false | If passed will have precedence over “id” attribute of <a/> generated based on content ID |
| urlParameters | false | If passed will be appended to “href” attribute of <a/> element |
| anchorAttributes | false | If passed will have precedence over content ANCHOR\_PARAMS node in appending attributes to <a/> element |

Example usage:

{{#cmlink contentid="liPromotionsDet" id="ANCHOR\_ID" urlParameters="ttc=140&token=1234" class="test" target="\_top" title="title" anchorAttributes="tabindex=’-1’ data-contains-tooltip=’true’"}}{{/cmlink}}

**{{cmbranding}} helper**

Cmbranding helper is utilized to populate header and footer areas of the page

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Required** | **Description** |
| component | true | Value “header” means retrieves HEADER\_MARKUP node from branding content  Value “footer” means retrieves FOOTER\_MARKUP node from branding content  Value “head” means retrieves JAVASCRIPT\_REPLICANTS and CSS\_REPLICANTS and loads them through require |
| appid | true | APP ID of the content containing branding in DB |

**{{cmmenu}} helper**

Cmmenu helper is utilized to populate left and top navigation menu areas of the page

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Required** | **Description** |
| menuType | true | Value “top” means retrieves TOP Navigation menu  Value “left” means retrieves LEFT Navigation menu |
| appid | true | APP ID of the content which generates html containing menu in DB. In other words app\_id of template referenced in [How to retrieve top navigation menu through JFP Content Service](#_How_to_retrieve) or [How to retrieve left navigation menu through JFP Content Service](#_How_to_retrieve_1) |

Chapter 14 C3 Display Related Tasks

This chapter gives an overview of the Display related tasks.

t he purpose of chapter is to provide an overview of the display related tasks

# 14.1 HandleBar – Template

* Handlebars (www.handlebarsjs.org) is one of the most popular templating libraries in the JavaScript community.
* The library follows the Mustache.js style, where the philosophy encourages logic-less templates.
* Note that this is a different approach than Underscore. As a result of this, Handlebars templates will not accept arbitrary JavaScript within the templates, forcing you to use built-in constructs. However, there is the ability to add your own helpers
* Like all templating libraries, Handlebars accepts the template structure and compiles it to a JavaScript function that accepts JSON data for presentation. One of the advantages of Handlebars is that, in following the same style as Mustache templates, the libraries are interchangeable, while you can keep the same templates.
* Also, with its template precompilation options, Handlebars is considered one of the most advanced libraries available.

# 14.2 Handlerbars – C3

Handlebar templates are created inside the folder “template” by convention and each file comes with the extension “.hbs” by convention

GruntFile (which’s going to do various task of JS file handling, like build, minify, precompile etc) ------ will have the context path set for the .hbs file to search for

Eg From Reference App

// configurable paths

var yeomanConfig = {

app: 'src',

dist: 'dist'

};

handlebars: {

compile: {

options: {

namespace: 'JST',

amd: true,

processName: function(filePath) {

return filePath.replace(yeomanConfig.app + '/', '');

}

},

files: {

'build/ref\_app/templates.js': [

'<%= yeoman.app %>/\*\*/\*.hbs'

]

}

}

},

Note: The above configuration comes as reference from reference app. Regional team can have their own path and they can come up with their own gruntfile

The above configuration states to use \*.hbs present under the src and its subfolder and compile the files to template.js and place it in build/ref\_app

Now we can mention which .hbs file to load inside the C3.View as follows

template: JST['modules/pnt/tboa/templates/tboa-additional-input.hbs'],

Left side hash refers to the key to refer to the template file present in the right side.

Only one template file can be loaded for a C3 View.

We can split the template into multiple partials.

Partials are same as template file; it’s just used for modularization of Template files

More than one partials can be registered inside C3.View

To Register Partials use C3Helpers.Handlebars.registerPartial

Eg: C3Helpers.Handlebars.registerPartial("paymentTemplate", this.paymentTemplate);

Define([

'jquery',

'underscore',

'c3',

'templates',

'handlebars',

'msgbus',

'c3widgets'

], function ($, \_, C3, JST, HandleBars, MsgBus,C3Widgets) {

'use strict';

return C3.FlowBaseView.extend({

template: JST['modules/pnt/tboa/templates/tboa-additional-input.hbs'],

paymentTemplate: JST['modules/pnt/tboa/templates/partials/tboa-payment-options.hbs'],

recurringTemplate: JST['modules/pnt/tboa/templates/partials/tboa-recurring-input.hbs'],

personalNoteAndFav: JST['modules/pnt/tboa/templates/partials/personal-Note-Favourite-tboa.hbs'],

transferTime: JST['modules/pnt/tboa/templates/partials/tboa-transfer-time.hbs'],

dateAndTime: JST['modules/pnt/tboa/templates/partials/tboa-date-and-time.hbs'],

areyousureModal: JST['modules/pnt/tboa/templates/partials/are-you-sure-model.hbs'],

inputDateAmount: JST['modules/pnt/tboa/templates/partials/input-date-amount.hbs'],

ui: {

nextButton: '#tboaPrimaryNav',

favCheck: '#favouriteTransactionChecked',

recurringCheck: '#recurringchkbox'

},

events: {

"click @ui.nextButton": "submit",

"click @ui.favCheck": "openFavContainer",

"click @ui.recurringCheck": "openRecurringContainer"

},

initialize: function () {

C3Helpers.Handlebars.registerPartial("paymentTemplate", this.paymentTemplate);

C3Helpers.Handlebars.registerPartial("recurringTemplate", this.recurringTemplate);

C3Helpers.Handlebars.registerPartial("personalNoteAndFav", this.personalNoteAndFav);

C3Helpers.Handlebars.registerPartial("transferTime", this.transferTime);

C3Helpers.Handlebars.registerPartial("dateAndTime", this.dateAndTime);

C3Helpers.Handlebars.registerPartial("areyousureModal", this.areyousureModal);

C3Helpers.Handlebars.registerPartial("paymentOptions", this.inputPaymentOptions);

C3Helpers.Handlebars.registerPartial("inputDateAmount", this.inputDateAmount);

\_.extend(this.events, this.createEvent());

},

preprocessData: function () {

//console.log("TBOA Recap:::" + JSON.stringify(this.model.attributes.data));

return {model: this.model.attributes.data};

},

createEvent: function () {

var data = this.model.attributes.data,

eventMap = {};

if (data.amountOptionLength > 0) {

for (var i = 0; i < data.paymentOptionsVB.otherAmtObj.otherAmountVBList.length; i++) {

eventMap["click #other-amount-" + i + "-withtext"] = "amountBoxClicked";

}

}

return eventMap;

},

amountBoxClicked: function (event) {

$('#' + event.target.id.substring('other-amount-'.length, event.target.id.length)).attr("checked", true);

},

serializeForm: function () {

var formFieldValues = {};

var fromFXpopUp = "true";

var pymntOptionType;

var selectedRadioID;

var amtOption;

var currencyCode;

var amount;

var txnDate;

var hours;

var minutes;

var recurringchkbox;

var selectedFrequency;

var terminationOption;

var terminationOptionValue;

var paymentEnabled = $('#paymentEnabled').val();

var modeOfTransfer = $('#modeOfTransfer').val();

var paymentRadio = $("input[name='paymentRadio']:checked").attr("id");

var paymentRadio1 = $("input[name='paymentRadio1']:checked").attr("id");

var favouriteName;

var personalNote = "";

var isImmediate;

var amountId;

var terminationOptionId;

var isWithoutAmt;

personalNote = $('#personalNote').val();

if (paymentRadio1 != null && paymentRadio1 != "") {

pymntOptionType = 'otherAmount';

selectedRadioID = paymentRadio1;

if (paymentRadio1.match(/Credit/g)) {

amtOption = "Credit";

currencyCode = $('#toCurrency').attr('value');

amount = $('#Credit-withtext').val();

amountId = 'Credit-withtext';

} else if (paymentRadio1.match(/Debit/g)) {

amtOption = "Debit";

currencyCode = $('#fromCurrency').attr('value');

amount = $('#Debit-withtext').val();

amountId = 'Debit-withtext';

}

} else if (paymentRadio != null && paymentRadio != "") {

selectedRadioID = paymentRadio;

isWithoutAmt = paymentRadio.indexOf("-withouttext");

if (isWithoutAmt != -1) {

amountId = paymentRadio;

} else {

amountId = paymentRadio.substr(0, paymentRadio.length - 6);

}

amountId = paymentRadio;

amtOption = $('#amountOption-' + amountId).val();

currencyCode = $('#currency-' + amountId).attr('value');

pymntOptionType = $('#paymentOptionType-' + amountId).val();

// amount = $('#' + "other-amount-" + amountId).val();

amount = $('#' + amountId).val();

}

txnDate = $('#txnDate').val();

var timeSupported = $('#timeoftransfers').css("display");

if ($('#timeoftransfers').is(':visible')) {

if ($('#hours').val() == "DefaultHour") {

hours = "DefaultHour"

} else {

if ($('#hours').val() == "Immediate") {

isImmediate = true;

} else {

hours = $('#hours').val().substring(0, 2);

minutes = $('#hours').val().substring(2, 5);

isImmediate = false;

}

}

} else {

isImmediate = false;

}

if ($('#recurringchkbox-control').prop('checked')) {

recurringchkbox = "true";

if ($('#Frequency\_single').length == 0) {

selectedFrequency = $('#Frequency').val();

} else {

selectedFrequency = "singleOption";

}

terminationOption = $("input[name='terminationOption']:checked").attr("value");

terminationOptionValue = "UntilCancelled";

if (terminationOption == "UntilAmount") {

terminationOptionValue = $('#Recurring\_transferamount').val();

terminationOptionId = 'Recurring\_transferamount';

} else if (terminationOption == "UntilDate") {

terminationOptionValue = $('#Recurring-datepicker-input').val();

terminationOptionId = 'Recurring-datepicker-input';

} else if (terminationOption == "NumberOfTransactions") {

terminationOptionValue = $('#repeat\_amount').val();

terminationOptionId = 'repeat\_amount';

}

}

var val1 = $('#flowExcutionKey').val();

var val2 = "submit";

var favouriteTransactionChecked = "false";

var favouriteTransaction = "";

if ($('#favouriteTransactionChecked').is(":visible")) {

if ($('#favouriteTransactionChecked').prop('checked')) {

favouriteTransactionChecked = "true";

favouriteTransaction = $('#favouriteName').val();

}

}

var finalFormFieldValues = {

'sourceAccountIndex': $('#fromAccountSelect').val(),

'destinationAccountIndex': $('#toAccountSelect').val(),

'jfp.layout': 'Raw',

'\_flowExecutionKey': val1,

'\_eventId': val2,

'paymentOptionType': pymntOptionType,

'paymentRadio': selectedRadioID,

'amountOption': amtOption,

'currencyCode': currencyCode,

'amnt': amount,

'txnDate': txnDate,

'hours': hours,

'minutes': minutes,

'frequencyOption': selectedFrequency,

'terminationOption': terminationOption,

'terminationOptionValue': terminationOptionValue,

'modeOfTransfer': modeOfTransfer,

'recurringchkbox': recurringchkbox,

'paymentEnabled': paymentEnabled,

'toRefreshFlag': false,

'fromRefreshFlag': false,

'personalNote': personalNote,

'favouriteTransactionChecked': favouriteTransactionChecked,

'favouriteName': favouriteTransaction,

'isImmediate': isImmediate,

'fromFXpopUp': fromFXpopUp

}

return finalFormFieldValues;

},

// show a demo animation when view is rendered

onRender: function () {

$('html, body').animate({scrollTop: $("#region2").offset().top}, 1000);

},

onSubmit: function () {

// this.ui.nextButton.attr("disabled", "true");

$('html, body').animate({scrollTop: $("#region1").offset().top}, 1000);

},

onSubmitSuccess: function () {

// this.ui.nextButton.remove();

},

onSubmitError: function () {

//this.ui.nextButton.removeAttr("disabled");

},

onShow: function () {

$("input[name\*='Credit-withtext']").keypress(function () {

$("#Credit-withtext-radio").prop("checked", true);

$("#Debit-withtext-radio").prop("checked", false);

$("input[name\*='Debit-withtext']").val('');

});

$("input[name\*='Debit-withtext']").keypress(function () {

$("#Credit-withtext-radio").prop("checked", false);

$("input[name\*='Credit-withtext']").val('');

$("#Debit-withtext-radio").prop("checked", true);

});

require([

'css!platform/lib/prod/pickadate/lib/themes/default.css',

'css!platform/lib/prod/pickadate/lib/themes/default.date.css'

]);

//activate the date picker widget

C3Widgets.WidgetUtils.createWidgets(this);

var self = this;

$('#recurringchkbox-control').change(function () {

if ($(this).prop('checked')) {

$("#personalSperator").addClass("cbolui-hidden");

$('#recurringchkbox-wrapper').addClass('cbolui-form-subsection-visible');

} else {

$("#personalSperator").removeClass("cbolui-hidden");

$('#recurringchkbox-wrapper').removeClass('cbolui-form-subsection-visible');

}

$('#Recurring-datepicker-input').focus(function () {

$('#Recurring-datepicker-input-radio').prop("checked", true);

});

$("input[type='radio'][name='terminationOption']").each(function () {

$(this).attr("checked", false);

});

$('#repeat\_amount, #Recurring\_transferamount,#Recurring-datepicker-input').val('');

$('#repeat\_amount, #Recurring\_transferamount,#Recurring-datepicker-input').removeClass('cbolui-error-background');

self.populateDataForRecurring();

// if (TxnJSONdata.recurringProcessingVO.frequencyOptionslist && TxnJSONdata.recurringProcessingVO.frequencyOptionslist.length == 1) {

// $('#Frequency\_single').show();

// $('#Frequency-button').hide();

// $('#recurr\_transfer').attr('checked', 'checked');

// } else {

// $('#Frequency\_single').hide();

// $('#Frequency-button').show();

// }

if ($('#InterRepeat').find('input:radio').length == 1) {

/\*Single input space\*/

$("#recurringList [type='radio']").prop('checked', true).hide();

$("#recurringList div").css("padding-left", "0px");

if ($('#InterRepeat').children().hasClass('cA-recurringElementContainer')) {

$('#InterRepeat').children().removeClass('cA-recurringElementContainer');

}

$('#InterRepeat').children().addClass('cA-singleRecurring');

}

});

},

populateDataForRecurring: function () {

var self = this;

var checked = $('#recurringchkbox-control').prop('checked');

if (checked) {

if ($('#TmplRecurring\_Transfer .cF-inputChoiceList > \*').length == 1) {

$('#TmplRecurring\_Transfer .cF-inputChoiceList input[type="radio"]').addClass('cbolui-hidden');

$('#TmplRecurring\_Transfer .cF-inputChoiceList input[type="text"].cF-addOnInput').addClass('cF-noRadio').parent().addClass('cM-lastElementVertical');

//Finds the last radio option and applies the cM-lastElementVertical class to it to remove the margin/padding bottom and attaches handler for the radio buttons to be selected when you select the field

} else {

$('#TmplRecurring\_Transfer .cF-inputChoiceList input[type="radio"]:last').parent().addClass('cM-lastElementVertical');

//Radio button is selected if the input box is focused

$('#TmplRecurring\_Transfer .cF-value.cF-inputBoxChoice input:text').on('focus', function () {

if ($(this).parent().find("input:radio").length) {

var radioButton = $(self).parent().find("input:radio");

//If the radio button is already selected, you need to not trigger the change.validation event

if (radioButton.attr("checked") === undefined) radioButton.trigger("change.validation");

radioButton.attr("checked", "checked");

} else {

var radioButton = $(self).parent().parent().find("input:radio");

//If the radio button is already selected, you need to not trigger the change.validation event

if (radioButton.attr("checked") === undefined) radioButton.trigger("change.validation");

radioButton.attr("checked", "checked");

}

});

}

//Hides the elements that do not have any text input and selects the first of the shown elements

var $visiblePymtOptions = $('#amountOptionsWithoutText > div.cF-withText');

if ($visiblePymtOptions.length == 1) {

$visiblePymtOptions.find('input:radio').addClass('cbolui-hidden').end().find('span.cA-otherAmt\_OneTime').removeClass('cF-choiceText');

}

$('#amountOptionsWithoutText span.pymtRadio').css('display', 'none');

$('#amountOptionsWithoutText span.pymtBalInfo').css('display', 'none');

$("#amountOptionsWithoutText span.cA-otherAmt\_OneTime").css('display', 'none');

$("#amountOptionsWithoutText span.cA-otherAmt\_Recurring").css('display', 'inline');

if ($('#amountOptionsWithoutText label:visible').length == 1) {

}

//To remove yellow color on click of checkbox

$('#InterRepeat input[type="text"]').removeClass('jfpw-tooltip-cursor ui-tooltip-shown cF-InvalidInputBackground cF-invalidField');

//4580- Destroying the error tooltip on frquency on toggling the recurring option

$('#Frequency-button.cbolui-error-background').removeClass('cbolui-error-background');

$('#amountOptionsWithoutText div input:radio').each(function () {

$(self).removeClass('cbolui-hidden');

});

$('#amountOptionsWithoutText > div.cF-withText').find('span.cA-otherAmt\_OneTime').addClass('cF-choiceText');

$('.cF-radioButtonOptions input[type="radio"]').removeClass('cbolui-hidden');

$('.cF-radioButtonOptions span').removeClass('cF-singelAmtMargin');

//Shows the elements that have text input

if ($('#amountOptionsWithoutText > div:not(".cF-withText")').length > 0) {

/\*Amount option radio button checked recurring option scenario\*/

//$('input[name="paymentRadio"]').prop('checked', false);

$('.termination\_text\_options1').val('');

}

$('#amountOptionsWithoutText > div:not(".cF-withText")').removeClass('cbolui-hidden');

$('#amountOptionsWithoutText span.pymtRadio').css('display', 'inline');

$('#amountOptionsWithoutText span.pymtBalInfo').css('display', 'inline');

$("#amountOptionsWithoutText span.cA-otherAmt\_OneTime").css('display', 'inline');

$("#amountOptionsWithoutText span.cA-otherAmt\_Recurring").css('display', 'none');

}

var $overallRadio = $('input[name="paymentRadio"]:not(".cbolui-hidden") , input[name="paymentRadio1"]:not(":visible")');

if ($overallRadio.length == 1) {

$overallRadio.addClass('cbolui-hidden');

$overallRadio.attr('checked', 'checked');

}

},

render: function () {

this.$el.html(this.template({data: this.model.attributes.data, validMap: this.validMap}));

},

openFavContainer: function () {

if ($('#favouriteCheckboxExpanded').hasClass('show')) {

$('#favouriteCheckboxExpanded').removeClass('show').addClass('hide');

$('#favouriteName').val("");

} else {

$('#favouriteCheckboxExpanded').removeClass('hide').addClass('show');

}

},

openRecurringContainer: function () {

if ($('#recurringContainer').hasClass('show')) {

$('#recurringContainer').removeClass('show').addClass('hide');

} else {

$('#recurringContainer').removeClass('hide').addClass('show');

}

},

onDestroy: function(){

console.log("Destroy date picker now @tboa additonal input view...")

C3Widgets.WidgetUtils.destroyWidgets(this);

}

});

});

# 14.3 Handlebars and Partial - Intro

How to write Handlebar template and its partials

Since the syntax of Handlebars templates requires more than plain JavaScript, this section will go through some of the basic expressions to help get you started.

More can be read from <http://handlebarsjs.com/>

**Displaying Variables**

Variable values can be output to the template using the form {{name}}. This will print out the value name in the current context, which can be altered by block expressions that loop through collections.

<h1>{{name}}</h1>

If you are dealing with nested variables, you can use dot-separated paths to determine the value.

For example, if the template was dealing with a person object, you could display the details as follows:

<h1>{{person.name}}</h1>

<p>Birthday: {{person.birthday}}</p>

**Comments**

Comments can be inserted into templates using either {{! comment }} or {{!-- comment --}}.

{{! Name of the person }}

<h1>{{person.name}}</h1>

The {{!-- }} style is used when you need to comment out Handlebars expressions.

{{!-- unused expression for now {{name}} --}}

**Block Expressions**

Blocks in Handlebars are represented with a {{#expression}}statement anda {{/expression}} statement to complete the block.

**each**

As we’ve already seen in the library example, the each helper allows a list to be iterated over.

{{#each library}}

<li>

<em>{{name}}</em> by {{author}} published in {{publishDate.year}}

</li>

{{/each}}

Note that you can use {{this}} within the block to reference the current element that is being iterated through.

You can also access objects and properties outside of the current context by using paths. An example of this

would be to use {{../<property name>}} to look at the property that is at the same level as the list you are currently

iterating through.

**Conditionals**

There are a number of conditional statements available for use in your template definition. The following section lists some of these.

**if else**

In the event that there are conditional sections of a template to be rendered, Handlebars includes an if else structure. A parameter is passed through to the {{#if }} expression to be evaluated.

{{#ifbook.published}}

<p>Book now available</p>

{{else}}

<p>Book coming soon : {{book.pubishDate}}</p>

{{/if}}

**unless**

The unless helper works as the inverse of the if statement shown earlier. It takes a parameter and ensures that it evaluates to false before entering the template section.

{{unlessbook.published}}

<p>This book is not yet published</p>

{{/unless}}

**Built-in Helpers**

Handlebars comes with a few built-in helpers to make it easier to use.

**with**

The with helper allows the context to be changed within a template so that a particular object is used. Rather than

using {{object.property}}, this helper allows you to specify that you are dealing with a particular object within the

block in the following form:

{{#with object}}

{{property}}

{{/with}}

To see this in action, let’s consider the following Person object:

{{#with person}}

<h1>{{name}}</h1>

<p>Birthday: {{birthday}}</p>

{{/with}}

**log**

The log helper allows simple logging within the template, which can useful for debugging.

.

{{#with person}}

<h1>{{log name}}</h1>

{{/with}}

**Partials**

Partials are same as template and get included in the main template file via a special syntax

{{> templatekey [optional data to be passed] }}

{{> areyousureModal}}

{{#with data}}

{{c3\_debug}}

<div class="row">

<div class="col-md-1">

</div>

<div class="col-md-12">

<form id="addlInputForm">

{{!-- removed the legend of tooltip since C3 tooltips are still not ready --}}

{{!-- if payment eanbledd, use the input-payment-options else use input-date-amount --}}

{{#if amountOptionLength}}

{{> paymentTemplate this}}

{{else}}

{{> inputDateAmount this}}

{{/if}}

{{> dateAndTime this}}

{{!-- <div class="cbolui-divider-dotted "></div> --}}

{{> transferTime this}}

{{> recurringTemplate this}}

{{>personalNoteAndFav this}}

<p>TBOA Input Disclosure <a href="#learnmore">Learn More.</a></p>

<div class="cbolui-divider-dotted cbolui-divider-nav"></div>

<div class="cbolui-bottom-nav cbolui-clearfix">

<button id="tboaPrimaryNav" type="button" class="cbolui-cta cbolui-primary-cta cbolui-cta-large">

Next

</button>

<!-- <button id="tboaSecondaryNav\_makeChanges" type="button" class="cbolui-cta cbolui-secondary-cta">Make Changes</button> -->

<button id="tboaSecondaryNav\_cancel" type="button" class="cbolui-cta cbolui-secondary-cta">Cancel

</button>

</div>

</form>

</div>

</div>

{{/with}}

# 14.4 Regions vs Layouts

* Regions and layouts are similar, but different: both have “areas” where you can display things, but they serve different purposes.
* Regions are areas in your application that will remain displayed (semi-)permanently as the user navigates.
* For example, you would use a region to display the navigation menu.
* Layouts, on the other hand, are more like “super views”: they behave like views, but *in addition* have areas where you can display sub-views (or sub-layouts!).
* In contrast to regions, layouts are meant to disappear when a user navigates somewhere else and that view is no longer necessary.
* In our case, when we (e.g.) display a contact in a dedicated view, the “list” layout and all the views it contains will be closed, removed, and replaced with the view displaying that particular contact.
* Put another way, the layout contained in the “main” region will be removed (along with the view it contains), but the “main” region itself will still exist: it will simply be displaying a different view (the dedicated contact display view).

# 14.5 Masking

C3 supports masking of text/ digits based on FFIEC rules. The rules can be configured according to application needs.

Note: All masking preferences will occur on blur/ focusout of the textbox/textfield.

Keypressed event is not supported in this version, since all the UI configurations are reflected only on the UI, as new element is created and updated in it, instead of modifying the actual non formatted data.

If key pressed event needs to be captured, then you need to write the same in the view.

C3 does not change the original data to be sent to sever while masking the form data, it provides a way such that the input fields are masked when user does a focus out/ blur from the field. C3 retains the original data and the same is being posted to the server when called for.

It supports mainly three types of input data masking. Multiple types of the below type can be configured within the application. Also the configurations can be based on locale, if some different formatting is required for different locales.

1. Text Masking

Essentially to be used, when required to mask any text field. Like we want to mask the bank name, or hide payee name, etc.

Allowed configuration:

1. Digit Masking

When specific pattern needs to be followed for digits like zip code (xxxxx-xxxx), phone no ( xxx-xxx-xxxx), account no ( 1234-xxxx-xxxx-5678), SSN (xxx-xx-1234) etc then this needs to be used.

Allowed configuration:

If variable is used, when we don’t know the length of the digits entered, eg. (when amount is entered and we want to add a separator after 3 digits, then variable should be used, and in all other cases, like phone no, ssn, zip coede, account no, variable should not be used. Variable attributes takes precedence above the rest.

Multiple digit mask attributes can be used with a unique name, and then can be directly referred from the handlebar helpers.

"DigitMask":{

"amount":{

variable:{

"showFirst":true",

"showLast":false,

"firstLength":4

"digitSeperate":3,

"wrapper": "(|)",

"seperator":","

},

"prefix":"",

"suffix":""

"showlast":false,

"showFirst":true

"wrapAll":true,

"wrapFirst":true,

"wrapLast":true,

"wrapIndex":2,

"wrapper": "(|)",

"seperator":['-','-'],

"digitSeperate":"3|3|4",

"mask":false

}

}

1. Email Masking

Required when we need to mask the email types.( [john.doe@aol.com--](mailto:john.doe@aol.com--)> [j\*\*\*\*\*e@aol.\*\*\*](mailto:j*****e@aol.***))

Allowed configuration

"fromStart":1,

"fromEnd":1,

"keepDomain":true,

"keepSuffix":true

Example of a typical masking configurations in app-config.

maskingConfig : {

commonMask:{

maskChar:"\*"

},

TextMask:{

fromStart:1,

fromEnd: 2

},

DigitMask:{

PhoneNo: {

wrapAll:true,

separator:['-','-',],

wrapper:"(|)",

digitSeparate:"3|3|4",

mask:true,

showIndex:[1]

},

SSN: {

showLast:true,

wrapAll:true,

separator:'-',

wrapper:"(|)",

digitSeparate:"3|2|4"

},

AccountNo: {

prefix:"$",

variable:{

showFirst:true,

showLast:true,

lastLength:4,

digitSeparate:3,

separator:",",

mask:false

}

}

}

How to use them in the helper.

Add the property mask when defining the text helper. If multiple types of a masking config is being used, call the masking configuration using <<mask Type>>-<<defined masking type>>

Eg if AccountNo is a masking type defined inside DigitMask type, then to use it, write mask=”DigitMask-AccountNo”

Typical example

{{#text id = 'payeeAccount' mask = 'DigitMask-AccountNo' label = '{{PayeeAcNo}}' value = '' error = 'InlineBottom' class='form-control' placeholder= '{

"message":{

"contentid":"addPayee\_input",

"appid":"cbol\_addPayee",

"type":"copy",

"attribute":"PLACEHOLDER\_ACCOUNT\_NO"

}}'

rules='{

"required": { "message":{

"contentid":"addPayee\_input",

"appid":"cbol\_addPayee",

"type":"copy",

"attribute":"ERROR\_PAYEEACOUNT\_REQUIRED"

}},

"minLength": {"length":9, "message":{

"contentid":"addPayee\_input",

"appid":"cbol\_addPayee",

"type":"copy",

"attribute":"ERROR\_PAYEEACOUNT\_MINLENGTH"

}}

}'

}}{{/text}}

# 14.6 Form Validation

C3 Supports Group Validation for form fields

**C3.GroupValidation**

{

"valueBased": [

{

"errorPlacementId": "uname",

"commonError": "This is required",

"or": [

{

"id": "uname",

"greater": {

"then": 200

},

"less": {

"then": 1000

},

"message": "this is required for or",

},

{

"id": "pwd",

"less": {

"then": 1000

}

}

],

"and": [

{

"id": "email",

"greater": {

"then": 1000

}

},

{

"id": "zip",

"equal": {

"to": 1000

}

}

],

"notOr": [

{

"id": "phoneno",

"greater": {

"then": 1000

}

},

{

"id": "add",

"equal": {

"to": 1000

}

}

],

"notAnd": [

{

"id": "add1",

"greater": {

"then": 1000

}

},

{

"id": "mob",

"greater": {

"then": 1000

}

}

]

}

]

}

Explanation of group validation memo,

* It accepts an array of ruleBased and valueBased rules.

Value Based:

**or**, if either of the rule passed then it passes.

**and**, all the rules should pass

**notOr**, not or of the rules.

**notAnd**, not and of the rules.

In each of the blocks, dynamic messages can be fit in..

"greater": {

"then": 200,

"message": '{{%then}} {{modelKey}} is required'

},

{{%<<any key in this map}} 🡪 {{%val}}

{{modelKey}} 🡪 any key inside model , a.b.c is also supported.

If common error needs to be specified, the same can be specified for each group, but the messages inside each rule takes precedence.

"**commonError**": "This is required",

A single group rule is an **and** of all the group rules.( **or && and && notOr && notAnd**)

The rule object can be brought from content and dynamically fit into method.

So this can be dynamic based on some configuration, or static rules written directly inside the view.

Method API:

To wire inside the view, call below,

this.executeAllValidationRules(memo, model, failonFirst),

The validation can be called directly even,

C3Helpers.Validation.executeAllValidationRules(memoz,model, false);

The memo element is printed above.

To Extend validation framework and create own validations.

\_.extend(C3Helpers.Validation. CommonFieldValidation, {});

Create validation methods of your own,

minLength : function(map){

}

The map is the object which is passed to the function. The map has key/value params which is being passed to the memo, or rule object.

Restricted keyword

**val -- map[‘val’]** 🡪 will give you $(‘#<<id>>’).val(), so this cannot be a property in the rule.

**\*\*message -- map[‘message’] 🡪contains the error placeholder to work internally, so don’t expect this as param inside the validation custom helper**

Eg.

In memo,

"greater": {

"then": 200

},

So in greater function,

map[‘then’] will give you 200.

Or in helper rule,

rules='{

"required": { "message" : "Something is wrong very wrong!!!"},

"minLength": {"length":8,"message": "{{email}} 8 min length is required"}}'

}

So in minLength, map[‘length’] will give you 8.

Chapter 15 C3 Widgets

This chapter provides an overview of the C3 Widgets

t

his chapter provides an overview of the C3 Widgets and how to use it in the applications.

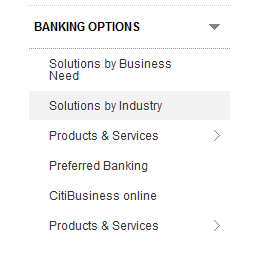
# 15.1 C3 Accordian Widgets

This C3 Accordion Menu widget is designed to display an accordion menu using list elements.

The **accordion** is a vertically stacked list of items, such as labels. Each item can be "expanded" or "stretched" to reveal the content associated with that item.

Samples:

This is a sample C3 Accordion Menu widget

.

**Base Plugin:**

There is no base plugin for C3 Accordion Menu widget. It is completely a CITI customized plugin.

**Javascript Source Files:**

c3.helpers.js

pf.widgets.js

**CSS:**

platform/lib/prod/accordionmenu/c3.widgets.accordion-menu.css

**Implementation:**

* Open the template file of the view in which AccordionMenu needs to be implemented. Insert the AccordionMenu helper in that template file. For example :

{{#accordionmenu id='c3-accordion-menu'}}

<ul >

<li >

<a href="#" >CITI Services</a>

<ul >

<li class="activeAME"><a href='#' >Solutions by Business Need</a></li>

<li ><a href='#' >Solutions by Industry</a></li>

<li >

<a href="#" >Products & Services</a>

<ul>

<li ><a href='#' >Credit Products</a></li>

<li ><a href='#' >Billing</a></li>

<li ><a href='#' >Saving Products</a></li>

</ul>

</li>

<li ><a href='#' >Preferred Banking</a></li>

<li ><a href='#' >CitiBusiness online</a></li>

<li >

<a href="#" >Products & Services</a>

<ul>

<li ><a href='#' >Credit Products</a></li>

<li ><a href='#' >Billing </a></li>

<li ><a href='#' >Saving Products</a></li>

</ul>

</li>

</ul>

{{/accordionmenu}}

* The following two functions need to be called from inside the view js file :

After the DOM is rendered, the following function needs to be called:

C3Widgets.WidgetUtils.createWidgets(this);

The best place to put this function call would be either in the render function of the view after the HTML is rendered or inside the onShow method of the view.

The following function needs to be added to remove method of the view:

C3Widgets.WidgetUtils.destroyWidgets(this);

**Options:**

The following are the list of options that the user can configure for accordion menu widget:

|  |  |
| --- | --- |
| **class="activeAME**" | This class needs to be added to that li element which will be active when the widget is initialized. That particular element will be in expanded state when the widget is created. |

**API Methods:**

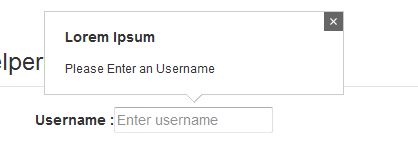
|  |  |  |  |
| --- | --- | --- | --- |
| Widget generic methods | | | |
| Return type | Name | Parameters | Description |
| void | destroy | N/A | Destroy the Accordion Menu widget. |
| void | enable | N/A | Enables the Accordion Menu. |
| void | disable | N/A | Disables the Accordion Menu. |
| Void | expandActiveItem | N/A | Expand the accordion menu to the li element which has activeAME class added to it. |

# 15.2 C3 Tooltip Widget

This C3 Tooltip widget is designed to display a Help Bubble with many configuration options that allow the front end developer to customize the widget and fit the business requirements.

**Samples:**

This is a sample C3 Tooltip widget configured on an input box



**Base Plug-in**

The base plugin for this C3 widget is tooltipster.

It’s a responsive plugin Compatible with Mozilla Firefox, Google Chrome, IE8+ and others. Requires jQuery 1.7+

References can be found at the following link:

<http://iamceege.github.io/tooltipster/>

**Source Files**

c3.helpers.js

pf.widgets.js

**CSS**

platform/lib/prod/tooltipster/css/tooltipster.css

**Implementation:**

* Open the template file of the view in which you need to add the tooltip. Insert the tooltip helper in that template file. For example :

{{#tooltip id='username-tooltip' selector='username'

<<LIST OF OPTIONS>> }}

           <div class="c3-tooltip-title">Lorem Ipsum</div>

              Please Enter Username

{{/tooltip}}

The selector attribute will be the id of the element on which the tooltip has to be initialized.

* The following two functions need to be called from inside the view js file :

After the DOM is rendered, the following function needs to be called:

C3Widgets.WidgetUtils.createWidgets(this);

The best place to put this function call would be either in the render function of the view after the HTML is rendered or inside the onShow method of the view.

The following function needs to be added to remove method of the view:

C3Widgets.WidgetUtils.destroyWidgets(this);

**Options:**

The following are the list of options that the user can configure in the tooltip helper:

|  |  |  |
| --- | --- | --- |
| animation | **fade, grow, swing, slide, fall** | **Determines how the tooltip will animate in and out. Feel free to modify or create custom transitions in the tooltipster.css file. In IE9 and 8, all animations default to a JavaScript generated, fade animation. Default: 'fade'** |
| arrow | **boolean** | **Adds the "speech bubble arrow" to the tooltip. Default: true** |
| arrowColor | **hex code / rgb** | **Select a specific color for the "speech bubble arrow". Default: will inherit the tooltip's background color** |
| autoClose | **boolean** | **If autoClose is set to false, the tooltip will never close unless you call the 'hide' method yourself. Default: true** |
| content | **string, jQuery object** | **If set, this will override the content of the tooltip. Default: null** |

|  |  |  |
| --- | --- | --- |
| contentAsHTML | **boolean** | **If the content of the tooltip is provided as a string, it is displayed as plain text by default. If this content should actually be interpreted as HTML, set this option to true. Default: false** |
| contentCloning | **boolean** | **If you provide a jQuery object to the 'content' option, this sets if it is a clone of this object that should actually be used. Default: true** |
| debug | **boolean** | **C3 Tootlip logs notices into the console when you're doing something you ideally shouldn't be doing. Set to false to disable logging. Default: true** |

|  |  |  |
| --- | --- | --- |
| delay | Integer | Delay how long it takes (in milliseconds) for the tooltip to start animating in. Default: 200 |
| minWidth | integer | Set a minimum width for the tooltip. Default: 0 (auto width) |

|  |  |  |
| --- | --- | --- |
| maxWidth | **Integer** | **Set a maximum width for the tooltip. Default: null (no max width)** |
| functionInit | **function** | **Create a custom function to be fired only once at instantiation. If the function returns a value, this value will become the content of the tooltip. See the** [**advanced section**](http://iamceege.github.io/tooltipster/#advanced) **to learn more. Default: function(origin, content) {}** |
| functionBefore | **function** | **Create a custom function to be fired before the tooltip opens. This function may prevent or hold off the opening. See the** [**advanced section**](http://iamceege.github.io/tooltipster/#advanced) **to learn more. Default: function(origin, continueTooltip) { continueTooltip(); }** |
| functionReady | **function** | **Create a custom function to be fired when the tooltip and its contents have been added to the DOM. Default: function(origin, tooltip) {}** |
| functionAfter | **function** | **Create a custom function to be fired once the tooltip has been closed and removed from the DOM. Default: function(origin) {}** |
| hideOnClick | **boolean** | **If true, the tooltip will close if its origin is clicked. This option only applies when 'trigger' is 'hover' and 'autoClose' is false. Default: false** |
| icon | **string, jQuery object** | **If using the iconDesktop or iconTouch options, this sets the content for your icon. Default: '(?)'** |
| iconCloning | **boolean** | **If you provide a jQuery object to the 'icon' option, this sets if it is a clone of this object that should actually be used. Default: true** |
| iconDesktop | boolean | Generate an icon next to your content that is responsible for activating the tooltip on non-touch devices. **Default: false** |
| iconTheme | CSS class | If using the iconDesktop or iconTouch options, this sets the class on the icon (used to style the icon). **Default: 'c3-tooltip-icon'** |
| iconTouch | boolean | Generate an icon next to your content that is responsible for activating the tooltip on touch devices (tablets, phones, etc). **Default: false** |
| interactive | boolean | Give users the possibility to interact with the tooltip. Unless autoClose is set to false, the tooltip will still close if the user moves away from or clicks out of the tooltip. **Default: false** |
| interactiveTolerance | integer | If the tooltip is interactive and activated by a hover event, set the amount of time (milliseconds) allowed for a user to hover off of the tooltip activator (origin) on to the tooltip itself - keeping the tooltip from closing. **Default: 350** |
| multiple | boolean | Allows you to put multiple tooltips on a single element. Read further instructions down this page. **Default: false** |
| offsetX | integer | Offsets the tooltip (in pixels) farther left/right from the origin. **Default: 0** |
| offsetY | integer | Offsets the tooltip (in pixels) farther up/down from the origin. **Default: 0** |
| onlyOne | boolean | If true, only one tooltip will be allowed to be active at a time. Non-autoclosing tooltips will not be closed though. **Default: false** |
| position | right, left, top, top-right, top-left, bottom, bottom-right, bottom-left | Set the position of the tooltip. **Default: 'top'** |
| positionTracker | boolean | Will reposition the tooltip if the origin moves. As this option may have an impact on performance, we suggest you enable it only if you need to. **Default: false** |
| positionTrackerCallback | function | Called after the tooltip has been repositioned by the position tracker (if enabled). **Default:** A function that will close the tooltip if the trigger is 'hover' and autoClose is false. |
| restoration | 'none', 'previous' or 'current' | Specify if a TITLE attribute should be restored on the HTML element after a call to the 'destroy' method. This attribute may be omitted, or be restored with the value that existed before Tooltip was initialized, or be restored with the stringified value of the current content. Note: in case of multiple tooltips on a single element, only the last destroyed tooltip may trigger a restoration. **Default: 'current'** |
| speed | integer | Set the speed of the animation. **Default: 350** |
| timer | integer | How long the tooltip should be allowed to live before closing. **Default: 0 (disabled)** |
| theme | CSS class | Set the theme used for your tooltip. **Default: 'c3-tooltip-default'** |
| touchDevices | boolean | If set to false, tooltips will not show on pure-touch devices, unless you open them yourself with the 'show' method. Touch gestures on devices which also have a mouse will still open the tooltips though. **Default: true** |
| trigger | hover, click, custom | Set how tooltips should be activated and closed. See the [advanced section](http://iamceege.github.io/tooltipster/#advanced) to learn how to build custom triggers. **Default: 'hover'** |
| updateAnimation | boolean | If a tooltip is open while its content is updated, play a subtle animation when the content changes. **Default: true** |

**API Methods:**

|  |  |  |  |
| --- | --- | --- | --- |
| Widget generic methods | | | |
| Return type | Name | Parameters | Description |
| void | destroy | N/A | Destroy the tooltip widget. |
| void | enable | N/A | Enables the Tooltip. |
| void | disable | N/A | Disables the Tooltip. |
| String | getContent | N/A | Get the content of Tooltip. |
| void | setContent | content - value to be set as content | Set the content of the tooltip. |
| \* | getConfig | optionName – name of the options whose value is required. | Get the value of a particular configuration option. |
| void | setConfig | optionName - name of the options whose value is to be set.  optionValue – Value of the option to be set. | Set the value of a particular configuration option. |
| Void | reposition | N/A | Reposition the tooltip. |
| \* | getTooltipElement | N/A | Returns the DOM containing the tooltip element |
| \* | getTooltipIconElement | N/A | Returns the DOM containing the tooltip icon element |

# 15.3 C3 Progress Bar Widget

This C3 ProgressBar widget is designed to display a progress bar in forms which need user to input data in multiple steps.

Samples:

This is a sample C3 ProgressBar widget configured on an input box



**Base Plug-in**

There is no base plugin for C3 ProgressBar widget. It is completely a CITI customized plugin.

**Source Files**

c3.helpers.js

pf.widgets.js

**CSS**

platform/lib/prod/progressbar/c3.widgets.progressbar.css

**Implementation:**

* Open the template file of the view in which you need to add the tooltip. Insert the tooltip helper in that template file. For example :

{{#progressbar id="c3progressbar" <<LIST OF OPTIONS>> ~}}

<li > ENTER YOUR ACCOUNT INFORMATION </li>

<li > SET UP ONLINE ACCESS </li>

<li class="c3-progressbar-current"> CONFIRMATION </li>

<li > WHATS NEXT </li>

{{~/progressbar}}

The selector attribute will be the id of the element on which the tooltip has to be initialized.

* The following two functions need to be called from inside the view js file :

After the DOM is rendered, the following function needs to be called:

C3Widgets.WidgetUtils.createWidgets(this);

The best place to put this function call would be either in the render function of the view after the HTML is rendered or inside the onShow method of the view.

The following function needs to be added to remove method of the view:

C3Widgets.WidgetUtils.destroyWidgets(this);

**Options:**

The following are the list of options that the user can configure in the tooltip helper:

|  |  |
| --- | --- |
| width | Set the width of the progress bar widget Default: '700px' |
| styleClass | Set the class of the progress bar widget Default: 'c3-progressbar' |
| class="c3-progressbar-current" | The class="c3-progressbar-current" should be added to that li element which is the current active section |
| data-c3progressbar-steps | If there is a case when a particular section has multiple sub-section(steps) inside it, then the data attribute data-c3progressbar-steps should be added to the li element. It should be assigned an integer value. For eg.  <li data-c3progressbar-steps=2> ENTER YOUR ACCOUNT INFORMATION </li>  will cause the above section to have 2 steps. |

**API Methods:**

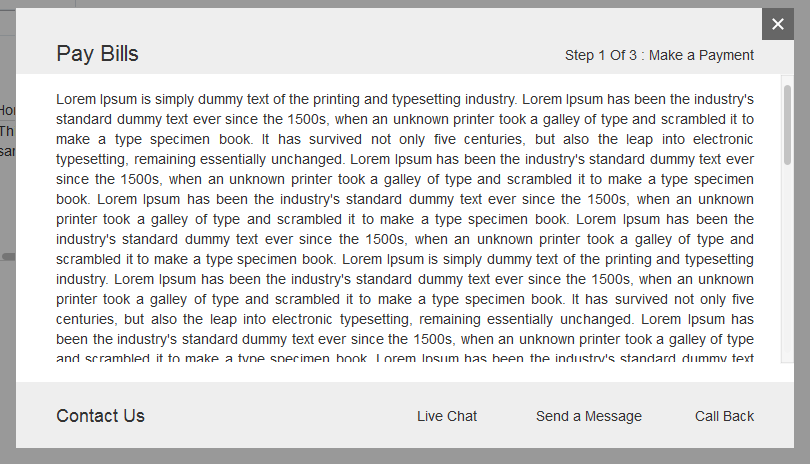
|  |  |  |  |
| --- | --- | --- | --- |
| Widget generic methods | | | |
| Return type | Name | Parameters | Description |
| void | destroy | N/A | Destroy the progress bar widget. |
| void | enable | N/A | Enables the progress bar widget. |
| void | disable | N/A | Disables the progress bar widget. |
| Void | reset | N/A | Reset the progress bar widget. |
| Void | nextSection | N/A | Complete the current section and enable the next section. |
| Void | prevSection | N/A | Reset the current active section and enable the previous section. |
| Void | setSection | section – Number | Set a particular section as active. |
| Void | nextStep | N/A | Enable the next step within the current section. |
| Void | prevStep | N/A | Enable the previous step within the current section. |
| Void | setStep | step– Number | Enable a particular step within the current section. |
| Object | getSectionStep | N/A | Get an object which mentions the current section and the current step of the progressbar. |

# 15.4 C3 OveryLay Widget

This C3 Overlay widget is designed to display a modal dialog with many configuration options that allow the front end developer to customize the widget and fit the business requirements.

Samples:

This is a sample C3 Overlay widget



**Base Plug-in**

It’s a responsive plugin Compatible with Mozilla Firefox, Google Chrome, IE8+ and others. Requires jQuery 1.7+

References can be found at the following link:

http://getbootstrap.com/javascript/#modals

**Source Files**

c3.helpers.js

pf.widgets.js

**CSS**

platform/lib/prod/overlay/css/c3-overlay.css

**Implementation:**

* Open the template file of the view in which you need to add the tooltip. Insert the tooltip helper in that template file. For example :

{{#overlay id="myModal" size="medium" ~}}

<div>

<div class="c3-overlay-header">

<div class="modal-header-content">

<span class="modal-header-label"> Pay Bills </span>

<span class="modal-header-description">Step 1 Of 3 : Make a

Payment</span>

</div>

</div>

<div class="c3-overlay-body">

<div class="c3-overlay-body-content c3-overlay-slider-parent">

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of typeand scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book.

</div>

</div>

<div class="c3-overlay-cta">

<a href="#">Add a payee</a>

</div>

<div class="c3-overlay-footer">

<div>Contact Us</div>

<div>

<span>Live Chat</span>

<span>Send a Message</span>

<span>Call Back</span>

</div>

</div>

</div>

{{/overlay}}

The selector attribute will be the id of the element on which the tooltip has to be initialized.

* The following two functions need to be called from inside the view js file :

After the DOM is rendered, the following function needs to be called:

C3Widgets.WidgetUtils.createWidgets(this);

The best place to put this function call would be either in the render function of the view after the HTML is rendered or inside the onShow method of the view.

The following function needs to be added to remove method of the view:

C3Widgets.WidgetUtils.destroyWidgets(this);

* The overlay has four sections. To implement the content of the overlay, the developer has to pass the content of these individual sections is separate div elements with specific classes
  + Header - c3-overlay-header
  + Body - c3-overlay-body
  + CTA - c3-overlay-cta
  + Footer - c3-overlay-footer

All these div elements have to be passed in the context of the overlay helper as described in the above example.

The developer has the options to skip the CTA or the Footer sections by not having those div elements defined in the context.

* Please note that the ~ is added to the block helpers to remove whitespace from the context data. It is recommended that teams implementing this widget use it.
* A slider will automatically be initialized on the “body” of the overlay if the contents overflow the maximum allowed height.

**Generated HTML:**

The overlay helper will parse the context data and will generate the overlay HTML which compliant with BOOTSTRAP modal classes and HTML structure.

<div aria-hidden="false" aria-labelledby="myModalLabel" role="dialog" tabindex="-1" data-c3widget="true" id="myModal" class="modal modal-overflow in" style="display: block; margin-top: 0px;">

<div class="modal-dialog modal-md">

<div class="modal-content">

<div class="modal-header">

<button aria-label="Close" data-dismiss="modal" class="close" type="button">

<span aria-hidden="true">×</span>

</button>

<div class="modal-header-content">

<span class="modal-header-label"> Pay Bills </span>

<span class="modal-header-description">Step 1 Of 3 : Make a Payment</span>

</div>

</div>

<div id="myModal-c3modalbody" class="modal-body" tabindex="0">

<div class="c3-overlay-body-content c3-overlay-slider-parent">

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of typeand scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book.

</div>

</div>

<div class="modal-body-cta">

<a href="#">Add a payee</a>

</div>

<div class="modal-footer">

<div>Contact Us</div>

<div><span>Live Chat</span>

<span>Send a Message</span>

<span>Call Back</span>

</div>

</div>

</div>

</div>

</div>

**Options:**

The following are the list of options that the user can configure in the overlay helper:

|  |  |  |
| --- | --- | --- |
| size | small,  medium,  large | Sets the size of the overlay.  small - modal-sm - 460 x 240  medium - modal-md - 550 x 420  large - modal-lg - 780 x 550  Default: “medium” |
| backdrop | boolean or the string 'static' | Includes a modal-backdrop element. Alternatively, specify static for a backdrop which doesn't close the modal on click. Default: true |
| keyboard | Boolean | Closes the modal when escape key is pressed  Default: true |
| show | boolean | Shows the modal when initialized.  Default: true |

**API Methods:**

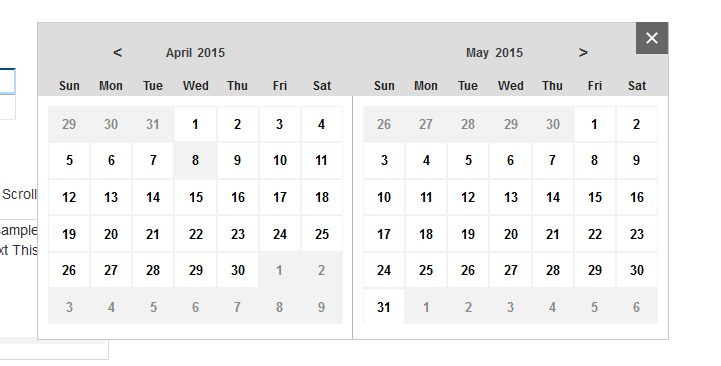
|  |  |  |  |
| --- | --- | --- | --- |
| Widget generic methods | | | |
| Return type | Name | Parameters | Description |
| void | destroy | N/A | Destroy the overlay widget. |
| void | show | N/A | Open the overlay |
| void | hide | N/A | Close the overlay |
| Void | toggle | N/A | Toggles the state of the overlay |

# 15.5 C3 DatePicker Widget

The C3 Datepicker widget is a responsive calendar widget which allows the user to select a particular date from a list of dates.

Samples:

This is a sample C3 Datepicker widget.



**Base Plugin:**

The base plugin for this C3 widget is pickadate.

References can be found at the following link:

<http://amsul.ca/pickadate.js/>

Here are a few features of this plugin:

* Supports [jQuery](http://jquery.com) 1.7 and up.
* Is ARIA-enabled to be WCAG 2.0 compliant.
* Loads a tiny JS and CSS footprint.
* Comes with translations for over 40 languages.
* Has touch & keyboard friendliness.
* Follows [BEM](http://bem.info/) style class naming.
* Utilizes [LESS](http://lesscss.org/) based style sheets.
* Includes a [Grunt](http://gruntjs.com/) based build system.

**Source Files**

c3.helpers.js

pf.widgets.js

**CSS**

lib/prod/pickadate/lib/themes/default.css

lib/prod/pickadate/lib/themes/default.date.css

**Implementation:**

* Open the template file of the view in which you need to add the tooltip. Insert the tooltip helper in that template file. For example :

{{#datepicker id='<<ID OF DATEPICKER>>' ~}}

<<LIST OF OPTIONS>>

{{~/datepicker}}

**<<LIST OF OPTIONS>>** has to be a properly formatted JSON string which will contain all the configuration options needed by the datepicker.

* The following two functions need to be called from inside the view js file :

After the DOM is rendered, the following function needs to be called:

C3Widgets.WidgetUtils.createWidgets(this);

The best place to put this function call would be either in the render function of the view after the HTML is rendered or inside the onShow method of the view.

The following function needs to be added to remove method of the view:

C3Widgets.WidgetUtils.destroyWidgets(this);

**Configurations:**

The overlay helper will parse the context data and will generate the overlay HTML which compliant with BOOTSTRAP modal classes and HTML structure.

The basic setup requires targeting an input element and invoking the picker:

**NOTE:** months in a JavaScript Date object are zero-indexed. Meaning, new Date(2015, 3, 20) is 20 April, 2016.

To stay consistent with this, whenever an integer is used in reference to a month, C3 Datepicker treats it as zero-indexed. Dates as strings are still parsed as expected.

* Options*[§](" \l "options)*

With the basic invocation above, these are the default settings:

*//* [*Strings*](#strings) *and* [*translations*](#translations)

monthsFull: ['January', 'February', 'March', 'April', 'May', 'June', 'July', 'August', 'September', 'October', 'November', 'December'],

monthsShort: ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'],

weekdaysFull: ['Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday'],

weekdaysShort: ['Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat'],

showMonthsShort: undefined,

showWeekdaysFull: undefined,

*//* [*Buttons*](#buttons)

today: 'Today',

clear: 'Clear',

close: 'Close',

*//* [*Accessibility labels*](#accessibility-labels)

labelMonthNext: 'Next month',

labelMonthPrev: 'Previous month',

labelMonthSelect: 'Select a month',

labelYearSelect: 'Select a year',

*//* [*Formats*](#formats)

format: 'd mmmm, yyyy',

formatSubmit: undefined,

hiddenPrefix: undefined,

hiddenSuffix: '\_submit',

hiddenName: undefined,

*//* [*Editable input*](#editable)

editable: undefined,

*//* [*Dropdown selectors*](#selectors)

selectYears: undefined,

selectMonths: undefined,

*//* [*First day of the week*](#first-weekday)

firstDay: undefined,

*//* [*Date limits*](#limits)

min: undefined,

max: undefined,

*//* [*Disable dates*](#disable-dates)

disable: undefined,

*//* [*Root picker container*](#container)

container: undefined,

*//* [*Hidden input container*](#container-hidden)

containerHidden: undefined,

*//* [*Close on a user action*](#close-on-action)

closeOnSelect: true,

closeOnClear: true,

*//* [*Events*](#events)

onStart: undefined,

onRender: undefined,

onOpen: undefined,

onClose: undefined,

onSet: undefined,

onStop: undefined,

// [Classes](#classes)

klass: {

// The element states

input: 'c3datepicker-input',

active: 'c3datepicker-input--active',

// The root picker and states \*

picker: 'c3datepicker',

opened: 'c3datepicker--opened',

focused: 'c3datepicker--focused',

// The picker holder

holder: 'c3datepicker-holder',

// The picker frame, wrapper, and box

frame: 'c3datepicker-frame',

wrap: 'c3datepicker-wrap',

box: 'c3datepicker-box',

// The picker header

header: 'c3datepicker-header',

// Month navigation

navPrev: 'c3datepicker-nav--prev',

navNext: 'c3datepicker-nav--next',

navDisabled: 'c3datepicker-nav--disabled',

// Month & year labels

month: 'c3datepicker-month',

year: 'c3datepicker-year',

// Month & year dropdowns

selectMonth: 'c3datepicker-select--month',

selectYear: 'c3datepicker-select--year',

// Table of dates

table: 'c3datepicker-table',

// Weekday labels

weekdays: 'c3datepicker-weekday',

// Day states

day: 'c3datepicker-day',

disabled: 'c3datepicker-day--disabled',

selected: 'c3datepicker-day--selected',

highlighted: 'c3datepicker-day--highlighted',

now: 'c3datepicker-day--today',

infocus: 'c3datepicker-day--infocus',

outfocus: 'c3datepicker-day--outfocus',

// The picker footer

footer: 'c3datepicker-footer',

// Today, clear, & close buttons

buttonClear: 'c3datepicker-button--clear',

buttonClose: 'c3datepicker-button--close',

buttonToday: 'c3datepicker-button--today'

}

**\*** It is important to not add any styling to the picker’s root element. Instead, target the .c3datepicker-holder element (or any other within) based on the state of the root element.

* Strings*[§](" \l "strings)*

Change the month and weekday labels as you find suitable:

{

weekdaysShort: ['Su', 'Mo', 'Tu', 'We', 'Th', 'Fr', 'Sa'],

showMonthsShort: true

}

* Accessibility labels*[§](" \l "accessibility-labels)*

Change the title attributes to several elements within the picker:

{

labelMonthNext: 'Go to the next month',

labelMonthPrev: 'Go to the previous month',

labelMonthSelect: 'Pick a month from the dropdown',

labelYearSelect: 'Pick a year from the dropdown',

selectMonths: true,

selectYears: true

}

* Translations*[§](" \l "translations)*

The picker can be extended to add support for internationalization. Translations for over [40 languages](http://github.com/amsul/pickadate.js/blob/3.5.5/lib/translations) are available out of the box, which you can include in one of two ways:

*// Extend the default picker options for all instances.*

$.extend($.fn.pickadate.defaults, {

monthsFull: ['Janvier', 'Février', 'Mars', 'Avril', 'Mai', 'Juin', 'Juillet', 'Août', 'Septembre', 'Octobre', 'Novembre', 'Décembre'],

weekdaysShort: ['Dim', 'Lun', 'Mar', 'Mer', 'Jeu', 'Ven', 'Sam'],

today: 'aujourd\'hui',

clear: 'effacer',

formatSubmit: 'yyyy/mm/dd'

})

*// Or, pass the months and weekdays as an array for each invocation.*

{

monthsFull: ['Janvier', 'Février', 'Mars', 'Avril', 'Mai', 'Juin', 'Juillet', 'Août', 'Septembre', 'Octobre', 'Novembre', 'Décembre'],

weekdaysShort: ['Dim', 'Lun', 'Mar', 'Mer', 'Jeu', 'Ven', 'Sam'],

today: 'aujourd\'hui',

clear: 'effacer',

formatSubmit: 'yyyy/mm/dd'

}



When using translations, [specify the formatSubmit and data-value](#formats) to ensure the date parses correctly regardless of locale.

RTL languages*[§](" \l "translations_rtl)*

For languages that flow from right-to-left (RTL), you’ll need to switch the arrows and text direction by linking along the rtl.css file:

<!-- Add the stylings \*after\* the datepicker theme files -->

<link rel="stylesheet" href="lib/themes/rtl.css">

<!-- Add the language \*after\* the datepicker script files -->

<script src="lib/translations/ar.js"></script>

* Formats*[§](" \l "formats)*

Display a human-friendly format and use an alternate one to submit to the server.

This is done by creating a new hidden input element with the same name attribute as the original with an optional prefix/suffix:

{

// Escape any [“rule” characters](#formatting-rules) with an exclamation mark (!).

format: 'You selecte!d: dddd, dd mmm, yyyy',

formatSubmit: 'yyyy/mm/dd',

hiddenPrefix: 'prefix\_\_',

hiddenSuffix: '\_\_suffix'

}



Send the hidden value only*[§](" \l "formats_use_hidden_only)*

A majority of the time, the value that needs to be sent to the server is just the hidden value – and not the visible one. To make this happen, use the hiddenName option.

This essentially nullifies the hiddenPrefix and hiddenSuffix, strips the name attribute from the source input, and then sets it as the name of the hidden input:

{

formatSubmit: 'yyyy/mm/dd',

hiddenName: true

}

Pre-fill values using custom formats or translations*[§](" \l "formats_prefill)*

When using a custom [formatting rule](#formatting-rules) for the format option or when using [translations](#translations), the input element should be given a data-value attribute formatted using the formatSubmit – the element’s value can be left blank. This helps to parse the date from custom formats into various languages:

<input data-value="2015/04/20">

Formatting Rules*[§](" \l "formatting-rules)*

The following rules can be used to format any date:

| Rule | Description | Result |
| --- | --- | --- |
| d | Date of the month | 1 – 31 |
| dd | Date of the month with a leading zero | 01 – 31 |
| ddd | Day of the week in short form | Sun – Sat |
| dddd | Day of the week in full form | Sunday – Saturday |
| m | Month of the year | 1 – 12 |
| mm | Month of the year with a leading zero | 01 – 12 |
| mmm | Month name in short form | Jan – Dec |
| mmmm | Month name in full form | January – December |
| yy | Year in short form **\*** | 00 – 99 |
| yyyy | Year in full form | 2000 – 2999 |

**\*** If you use the yy rule in the format option, you **must** specify the yyyy rule in the [formatSubmit option](#formats) with the appropriate [data-value attribute](#formats_prefill) to ensure the date parses accurately. Never use the yy rule in the formatSubmit option.

* Editable input*[§](" \l "editable)*

By default, typing into the input is disabled by giving it a readOnly attribute. Doing so ensures that virtual keyboards don’t pop open on touch devices. It is also a confirmation that values passed to the server will be of a consistent format.

However, this behavior can be changed using the editable option:

{

editable: true

}

An important thing to note here is that this disables keyboard bindings on the input element, such as arrow keys opening the picker. You will have to add your own bindings as you see fit.

Using HTML5 attributes*[§](" \l "html5-attributes)*

Because each input is [readOnly by default](#editable), HTML5 attributes, such as required and pattern, do not get enforced.

To enable default browser behavior on these attributes, set the [editable property to true](#editable).

* Dropdown Selectors*[§](" \l "selectors)*

Display select menus to pick the month and year. Anything truth-y enables the selectors and anything false-y switches them into text:

{

selectYears: true,

selectMonths: true

}

When selectYears is truthy, the year selector appears *before* the month. [Read here](http://github.com/amsul/pickadate.js/issues/384) for more details on why.

You can also specify the number of years to show in the dropdown using an even integer - half before and half after the year in focus:

{

// `true` defaults to 10.

selectYears: 4

}

* First Weekday*[§](" \l "first-weekday)*

The first day of the week can be set to either Sunday or Monday. Anything truth-y sets it as Monday and anything false-y as Sunday:

{

firstDay: 1

}

* Date Limits*[§](" \l "limits)*

Set the minimum and maximum selectable dates on the picker.

Using JavaScript dates*[§](" \l "limits-use-dates)*

{

min: new Date(2015,3,20),

max: new Date(2015,7,14)

}

Using arrays formatted as [YEAR,MONTH,DATE]*[§](" \l "limits-use-arrays)*

{

min: [2015,3,20],

max: [2015,7,14]

}

Using integers or a boolean*[§](" \l "limits-relative)*

{

// An integer (positive/negative) sets it relative to today.

min: -15,

// `true` sets it to today. `false` removes any limits.

max: true

}

* Disable Dates*[§](" \l "disable-dates)*

Disable a specific or arbitrary set of dates selectable on the picker.

Using JavaScript dates*[§](" \l "disable-dates-use-dates)*

{

disable: [

new Date(2015,3,13),

new Date(2015,3,29)

]

}

Using arrays formatted as [YEAR,MONTH,DATE]*[§](" \l "disable-dates-use-arrays)*

{

disable: [

[2015,3,3],

[2015,3,12],

[2015,3,20]

]

}

Using integers as days of the week (1 to 7)*[§](" \l "disable-dates-use-integers)*

{

disable: [

1, 4, 7

]

}

Using objects as a range of dates*[§](" \l "disable-dates-use-ranges)*

{

disable: [

{ from: [2016,2,14], to: [2016,2,27] }

]

}

The values for from & to can be:

* A JavaScript Date object,
* An array formatted as [YEAR,MONTH,DATE],
* And true to set it as “today”.

The values can also be integers representing dates relative to the other:

* to can only be positive:

{ from: [2016,3,12], to: 10 }

* from can only be negative:

{ from: -10, to: true }

*Disabling all* with a set of exceptions*[§](" \l "disable-dates-all)*

Enable only a specific or arbitrary set of dates by setting true as the first item in the collection:

{

disable: [

true,

1, 4, 7,

[2015,3,3],

[2015,3,12],

[2015,3,20],

new Date(2015,3,13),

new Date(2015,3,29)

]

}

Disabling ranges with exceptions*[§](" \l "disable-dates-inverted)*

Enable dates that fall within a range of disabled dates by adding the inverted parameter to the item within the collection:

{

disable: [

5,

[2015, 10, 21, 'inverted'],

{ from: [2016, 3, 15], to: [2016, 3, 25] },

[2016, 3, 20, 'inverted'],

{ from: [2016, 3, 17], to: [2016, 3, 18], inverted: true }

]

}

* container*[§](" \l "container)*

By default, the picker’s root element is inserted right after the main input element. Specify where to insert the root element by passing any valid CSS selector to this option:

{

container: '#root-picker-outlet'

}

This is especially important when the input falls within a label element because click events bubble up to the label element and re-open the picker.

When using this option, be careful not to set the container to something generic like the document’s body. This will break the document’s keyboard flow, for example when tabbing through a form. Instead, maintain the flow by keeping the container close to the input element.

* container for the hidden input*[§](" \l "container-hidden)*

By default, the picker’s hidden input is inserted right after the main input element. Specify where to insert the hidden element by passing any valid CSS selector to this option:



{

containerHidden: '#hidden-input-outlet'

}

* Close on a user action *[§](" \l "close-on-action)*

When a date is selected or the “clear” button is pressed, the picker closes. To change this behavior, use the following options:

{

closeOnSelect: false,

closeOnClear: false

}

* events*[§](" \l "events)*

All datepicker widget events are available on the C3Widgets.radio channel. To listen to a particular event the following syntax has to be used:

C3Widgets.radio.on(<<EVENT>>,function(eventdata){

<<HANDLER FUNCTION DEFINITION>>

})

The following events are available to be used :

|  |  |
| --- | --- |
| "datepicker:open" | Triggered whenever the datepicker is opened. |
| "datepicker:close" | Triggered whenever the datepicker is closed. |
| "datepicker:render" | Triggered whenever the datepicker is rendered. |
| "datepicker:start" | Triggered whenever the datepicker is initialized. |
| "datepicker:stop" | Triggered whenever the datepicker is destroyed. |
| "datepicker:set" | Triggered whenever the datepicker options is set. |

The datepicker:set event is the only callback that is passed a context argument that provides details as to which properties are being “set”.

Within scope of all six of these events, this refers to [the picker](http://amsul.ca/pickadate.js/date/api.htm).

Chapter 16 Events and Messaging

This chapter gives an overview of Events and Messaging.

t

his chapter gives an overview of Events and Messaging.It covers both the built in events and external messaging system. Having every object reference each other for events creates a tangled mess of object dependencies.

It very tightly couples implementation details of the application’s features to each other by crossing boundaries that should not be crossed. This creates a bad situation where changing one part of an application will cause ripple effects that require changes in other parts of the app.

* Event Aggregator & Mediator
* Façade Pattern
* Command Pattern

Which can be further conceptualized as

* “Something Happened” – Events and Event Aggregation
* “Go and Do this” – Command
* “Do this and let me know” – Request and Response.

One of the core tenets of building a Backbone application of any size is using an event-driven approach. Everything in the Backbone way of writing code and integrating the different object types relies on events. They are used to communicate changes in models and collections, DOM changes and user interaction, and more. But events are not the only communication means that a scalable application needs.

There are times when the code needs to look beyond events - beyond a statement of “something happened” - and start to look at statements like “do this” and “do this, and let me know”. To venture in to these other realms of communication, additional patterns are needed - patterns such as commands and a request/response system.

Below Section will deal about that in detail.

# 16.1 C3 Events

* Events are included in every C3 object.
* Events are from Backbone.Events and as such C3 adopts over it and includes in every object of C3
* This includes the following:

• C3.Model

• C3.Collection

• C3.View

• C3.Router

• C3.History

* When the event is fired, a number of parameters are available in the callback functions, which are listed in the following tables.

# 16.2 C3 Model Events



# 16.3 C3 Router Events



# 16.4 C3 View Events

|  |  |  |
| --- | --- | --- |
| Event | Callback | Fired When… |
| show | onShow | Called on the view instance when the view has been rendered and displayed. |
| before:destroy | onBeforeDestroy | call an onBeforeDestroy event on the view, if one is provided |
| destroy | onDestroy…returns view | * call an onDestroy event on the view, if one is provided * unbind all custom view events * unbind all DOM events * remove this.el from the DOM * unbind all listenTo events |
| attach | onAttach | * Every view in C3 has a special event called "attach," which is triggered anytime that showing the view in a Region causes it to be attached to the document. * The attach event is only fired when the view becomes a child of the document. If the Region you're showing the view in is not a child of the document at the time that you call show then the attach event will not fire until the Region is a child of the document. * This event is unique in that it propagates down the view tree. For instance, when a Layout's attach event is fired, all of its children views will have the attach event fired as well. In addition, deeply nested Layout View structures will all have their attach event fired at the proper time, too. |
| before:attach | onBeforeAttach | * This is just like the attach event described above, but it's triggered right before the view is attached to the document. |
| dom:refresh | onDomRefresh | * Triggered after the view has been rendered, has been shown in the DOM via a Marionette.Region, and has been re-rendered * This event / callback is useful for [DOM-dependent UI plugins](http://lostechies.com/derickbailey/2012/02/20/using-jquery-plugins-and-ui-controls-with-backbone/) |
| before:render | onBeforeRender | * Triggered before an View is rendered. |
| render | onRender | * Triggered after the view has been rendered. |

**General View Events**

A few events are common across all views. When any C3 view is closed, the following sequence of events occurs on the view being closed:

1. The onBeforeDestroy callback is executed, if it exists.

2. The onDestroy callback is executed, if it exists.

3. Unbind all custom view events.

4. Unbind all DOM events.

5. Remove the view’s el from the DOM.

6. Unbind all listenTo events.

Above Listed events are used whenever we want to understand “Something Happened” on the C3 Objects (Model, View, Collection, Router, History)

# 16.5 C3 Model Events – Usage

**When Part:**

* You can use Model events to infer something happened on the model data structure

**How part:**

* Events can be listenTo directly on model (or )
* View can bind to model events

**View.ModelEvents**

Similar to the events hash, views can specify a configuration hash for collections and models. The left side is the event on the model or collection, and the right side is the name of the method on the view.

define([

'jquery',

'underscore',

'c3',

'templates',

'c3helpers',

'msgbus',

'backbone-validation',

'bootstrap'

], function ($, \_, C3, JST, C3Helpers, MsgBus) {

'use strict';

return C3.FlowBaseView.extend({

template: JST['modules/pnt/tboa/templates/tboa-input.hbs'],

selectTemplate: JST['modules/pnt/tboa/templates/select-account-input.hbs'],

areyousureModal: JST['modules/pnt/tboa/templates/partials/are-you-sure-model.hbs'],

// model: new TboaInputModel(),

// UI hash for the entire view. This prevents jQuery selectors from spreading in the view.

// this.bindUIElements() can be called after rendering, so that ui.property returns a jQuery object instead of selector string

// UI elements are not bind in this view because we need to re-bind whenever partial templates are render

ui: {

nextButton: '#tboa\_input\_next',

cancelButton:'#InternalTransfer\_Cancel',

fromAccount: '#fromAccount',

fromAccountSelect: '#fromAccountSelect',

toAccount: '#toAccount',

toAccountSelect: '#toAccountSelect'

},

// Delegates events for the view

events: {

"click @ui.nextButton": "submit",

"change @ui.fromAccountSelect": "\_fromListChanged",

"change @ui.toAccountSelect": "\_toListChanged"

},

modelEvents: {

"change:selectedFromAccount": "\_rePopulateToAccount",

"change:selectedToAccount": "\_rePopulateFromAccount"

},

initialize: function () {

console.log("tboa-input-view:initialize");

var self = this;

C3Helpers.Handlebars.registerPartial("selectAccountInput", this.selectTemplate);

C3Helpers.Handlebars.registerPartial("areyousureModal", this.areyousureModal);

$('#tboamodalyes').off();

$("#tboamodalno").off();

$("#tboamodalyes").on('click', function () {

$('#myModal').modal('hide');

self.model.set({"selectedFromAccount": self.$(self.ui.fromAccountSelect).val()});

self.submit();

});

$("#tboamodalno").on('click', function () {

self.$(self.ui.fromAccountSelect).val(self.model.get("selectedFromAccount"));

});

},

// preprocess data from rendering

// returned data will be used to render against template and show

preprocessData: function () {

return this.\_preprocess();

},

// prepare data for submission

// returned data will be set into model and submit to server

serializeForm: function () {

var srcAcctIndex = '';

var destAcctIndex = '';

var a = $(":input").serializeArray();

$.each(a, function () {

$.each(a, function () {

if ('fromAccountSelect' === this.name) {

srcAcctIndex = this.value;

} else if ('toAccountSelect' === this.name) {

destAcctIndex = this.value;

}

});

});

return {

sourceAccountIndex: srcAcctIndex,

destinationAccountIndex: destAcctIndex,

amount: '',

currencyIndicator: '',

\_flowExecutionKey: this.model.attributes.\_flowExecutionKey,

\_eventId: "interSubmit",

"jfp.layout": "Raw"

}

},

/////////////////////////////////////////////////////////////////////////////////////

// Varies optional event handlers for this view tied directly to UI

/////////////////////////////////////////////////////////////////////////////////////

onSubmit: function () {

this.ui.nextButton.attr("disabled", "true");

this.ui.cancelButton.attr("disabled", "true");

},

onSubmitSuccess: function () {

this.ui.nextButton.remove();

this.ui.cancelButton.remove();

},

onSubmitError: function () {

this.ui.nextButton.removeAttr("disabled");

},

// TODO:

onValidationError: function () {

$nextButton.removeAttr("disabled");

},

\_fromListChanged: function () {

//first check if next button exist or not. If it doesnt exist then it means bottom screen is rendered

if ($('#tboa\_input\_next').length) {

if ($("#fromAccountSelect").val()==="") {

this.ui.nextButton.attr("enabled", "false");

$("#tboa\_input\_next").prop("disabled", true);

} else if ($("#toAccountSelect").val()!=="") {

this.ui.nextButton.attr("enabled", "true");

$("#tboa\_input\_next").prop("disabled", false);

$("#cbolui-iconDomID-pageErrorTBOA-iconText").html("");

}

else {

$("#tboa\_input\_next").prop("disabled", true);

}

this.model.set({"selectedFromAccount": this.$(this.ui.fromAccountSelect).val()});

}

else {

//show the confirmation Model.

// $("#mybutton").trigger('click');

$('#myModal').modal('show');

}

},

\_toListChanged: function () {

if ($('#tboa\_input\_next').length) {

if ($("#toAccountSelect").val()==="") {

this.ui.nextButton.attr("enabled", "false");

$("#tboa\_input\_next").prop("disabled", true);

} else if ($("#fromAccountSelect").val()!=="") {

this.ui.nextButton.attr("enabled", "true");

$("#tboa\_input\_next").prop("disabled", false);

$("#cbolui-iconDomID-pageErrorTBOA-iconText").html("");

}

else {

$("#tboa\_input\_next").prop("disabled", true);

}

this.model.set({"selectedToAccount": this.$(this.ui.toAccountSelect).val()});

}

else {

//show the confirmation Model.

// $("#mybutton").trigger('click');

$('#myModal').modal('show');

}

},

\_rePopulateFromAccount: function () {

console.log("\_rePopulateFromAccount:");

var beforeRefreshFrom = $(this.ui.fromAccountSelect).val();

console.log("Before Selection:" + beforeRefreshFrom);

var fromAccounts = this.model.attributes.destSrcMap[this.model.attributes.selectedToAccount];

console.log("Selected TO Account ::" + this.model.attributes.selectedToAccount);

$(this.ui.fromAccount).html(this.selectTemplate(this.\_formatAccountList(fromAccounts, this.model.attributes.defaultSelectionText, 'fromAccountSelect', 'FROM:')));

$(this.ui.fromAccountSelect).val(beforeRefreshFrom);

},

\_rePopulateToAccount: function () {

console.log("\_rePopulateToAccount:");

var beforeRefreshTo = $(this.ui.toAccountSelect).val();

console.log("Before Selection:" + beforeRefreshTo);

var toAccounts = this.model.attributes.srcDestMap[this.model.attributes.selectedFromAccount];

$(this.ui.toAccount).html(this.selectTemplate(this.\_formatAccountList(toAccounts, this.model.attributes.defaultSelectionText, 'toAccountSelect', 'TO:')));

$(this.ui.toAccountSelect).val(beforeRefreshTo);

},

\_checkToRender: function () {

var errorMsg = this.model.attributes.errorMsg;

if (errorMsg != null) {

this.render();

}

},

\_preprocess: function () {

console.log("tboa-view-input.js:preprocess");

var data = this.model.attributes;

var output = {

fromElement: {

displaylabel: '',

id: '',

element: []

},

toElement: {

displaylabel: '',

id: '',

element: []

}

};

output.fromElement = this.\_formatAccountList(data.sourceAccountVBList, data.defaultSelectionText, 'fromAccountSelect', 'FROM:');

output.toElement = this.\_formatAccountList(data.destinationAccountVBList, data.defaultSelectionText, 'toAccountSelect', 'TO:');

\_.extend(output, data);

// console.log("preprocess returns:"+JSON.stringify(output))

var errorObj = this.model.get('error');

if (errorObj.status) {

return \_.extend(output, {error: errorObj});

}

return output;

},

\_formatAccountList: function (accountList, defaultSelectionText, id, displaylabel) {

console.log("\_formatAccountList:");

var output = {

displaylabel: '',

id: '',

element: []

};

output.displaylabel = displaylabel;

output.id = id;

output.element = \_.map(accountList, function (account) {

console.log("tboa-input-view:map");

var element = {};

element.itemDescription = account.accountIdentifierText.split('{')[0];

if (account.accountResolverVO[0]) {

if (account.accountResolverVO[0].negativeHandler1 && (account.accountResolverVO[0].negativeHandler1 == 'DISPLAY\_ZERO' || account.accountResolverVO[0].negativeHandler1 == 'DISPLAY\_WITH\_MINUS')) {

element.itemDescription += " -";

}

if (account.accountResolverVO[0].resolverBalanceText1 && account.accountResolverVO[0].resolverBalanceText1 === 'Field Label') {

element.itemDescription += " " + account.accountResolverVO[0].resolverBalance1;

}

if (account.accountResolverVO[0].resolverBalanceText2 && account.accountResolverVO[0].resolverBalanceText2 === 'Field Label') {

element.itemDescription += account.accountResolverVO[0].resolverBalanceText2;

}

}

element.instanceID = account.instanceID;

return element;

});

var defaultElement = {};

defaultElement.itemDescription = defaultSelectionText;

defaultElement.instanceID = ""; //hope no account is named default... might need to clean this

output.element.splice(0, 0, defaultElement);

// console.log("tboa-input-view:output:"+JSON.stringify(output));

return output;

}

});

});

**Events can be listenTo Directly on Model**

* Out of the two ways Backbone gives us to bind callbacks to events, C3 prefer to use of listenTo rather than on:
* Both methods make use of the same information, but the difference is where we’re recording it.
* Note that on is called on the event bus itself. We’re saying “event bus, whenever this event passes by, be sure to invoke this callback on this object.”
* This works but is at risk of leaking memory. For instance, say the event bus has bound a callback to a child view, and then the parent view closes that child and creates a new view in its place. The old child view will still be retained in memory because the event bus will still have a reference to it (and will still be invoking callbacks on it in response to events). To avoid this problem, be sure to remove all previously-bound callbacks from any object that should be destroyed using off (a hard thing to do).
* Alternatively just use listenTo, and the above problem will be managed for you. Note that listenTo is invoked on the object that would have otherwise been at risk of a memory leak. By keeping track of callback bindings on this object instead of just the event bus, C3 via Backbone can clean up the objects bindings itself.
* You can pretty much listenTo all the Events listed in the above section for model
* For Eg: Model change event fired when there is change in the model

var ImageModel = C3.Model.extend(

{

initialize: function()

{

this.listenTo(this,'change',this.someChange); //when some property in this object has changed, run this.someChange() function

},

defaults : { title : 'untitled', description : 'no description available', owner : 'anonymous', date : 'no date supplied' },

someChange: function(model,options){ alert(‘something has changed’); } });

var photo = new ImageModel({title:’awesome image’}); //create new instance

photo.set(‘title’,’really awesome indeed’); //change the title attribute, this will trigger ‘change’ event

photo.set({title:’well i agree’},{agree: ‘yep’}); //alternative way to change model attribute with optional option passed, this will also trigger change event. </p>

# 16.6 C3 View Events – Usage

**When Part:**

* You can use View events when we want to tap something happened at view and DOM elements of View.

**How Part:**

* You can use Event hash for any C3 View.
* all jQuery events are valid events that can be used inside the “events” hash in a view
* Here is the complete list of all jQuery events you can listen to in a Backbone.js view:  
  <http://api.jquery.com/category/events/>

define([

'jquery',

'underscore',

'c3',

'templates',

'c3helpers',

'msgbus',

'backbone-validation',

'bootstrap'

], function ($, \_, C3, JST, C3Helpers, MsgBus) {

'use strict';

return C3.FlowBaseView.extend({

template: JST['modules/pnt/tboa/templates/tboa-input.hbs'],

selectTemplate: JST['modules/pnt/tboa/templates/select-account-input.hbs'],

areyousureModal: JST['modules/pnt/tboa/templates/partials/are-you-sure-model.hbs'],

// model: new TboaInputModel(),

// UI hash for the entire view. This prevents jQuery selectors from spreading in the view.

// this.bindUIElements() can be called after rendering, so that ui.property returns a jQuery object instead of selector string

// UI elements are not bind in this view because we need to re-bind whenever partial templates are render

ui: {

nextButton: '#tboa\_input\_next',

cancelButton:'#InternalTransfer\_Cancel',

fromAccount: '#fromAccount',

fromAccountSelect: '#fromAccountSelect',

toAccount: '#toAccount',

toAccountSelect: '#toAccountSelect'

},

// Delegates events for the view

events: {

"click @ui.nextButton": "submit",

"change @ui.fromAccountSelect": "\_fromListChanged",

"change @ui.toAccountSelect": "\_toListChanged"

},

modelEvents: {

"change:selectedFromAccount": "\_rePopulateToAccount",

"change:selectedToAccount": "\_rePopulateFromAccount"

},

initialize: function () {

console.log("tboa-input-view:initialize");

var self = this;

C3Helpers.Handlebars.registerPartial("selectAccountInput", this.selectTemplate);

C3Helpers.Handlebars.registerPartial("areyousureModal", this.areyousureModal);

$('#tboamodalyes').off();

$("#tboamodalno").off();

$("#tboamodalyes").on('click', function () {

$('#myModal').modal('hide');

self.model.set({"selectedFromAccount": self.$(self.ui.fromAccountSelect).val()});

self.submit();

});

$("#tboamodalno").on('click', function () {

self.$(self.ui.fromAccountSelect).val(self.model.get("selectedFromAccount"));

});

},

// preprocess data from rendering

// returned data will be used to render against template and show

preprocessData: function () {

return this.\_preprocess();

},

// prepare data for submission

// returned data will be set into model and submit to server

serializeForm: function () {

var srcAcctIndex = '';

var destAcctIndex = '';

var a = $(":input").serializeArray();

$.each(a, function () {

$.each(a, function () {

if ('fromAccountSelect' === this.name) {

srcAcctIndex = this.value;

} else if ('toAccountSelect' === this.name) {

destAcctIndex = this.value;

}

});

});

return {

sourceAccountIndex: srcAcctIndex,

destinationAccountIndex: destAcctIndex,

amount: '',

currencyIndicator: '',

\_flowExecutionKey: this.model.attributes.\_flowExecutionKey,

\_eventId: "interSubmit",

"jfp.layout": "Raw"

}

},

/////////////////////////////////////////////////////////////////////////////////////

// Varies optional event handlers for this view tied directly to UI

/////////////////////////////////////////////////////////////////////////////////////

onSubmit: function () {

this.ui.nextButton.attr("disabled", "true");

this.ui.cancelButton.attr("disabled", "true");

},

onSubmitSuccess: function () {

this.ui.nextButton.remove();

this.ui.cancelButton.remove();

},

onSubmitError: function () {

this.ui.nextButton.removeAttr("disabled");

},

// TODO:

onValidationError: function () {

$nextButton.removeAttr("disabled");

},

\_fromListChanged: function () {

//first check if next button exist or not. If it doesnt exist then it means bottom screen is rendered

if ($('#tboa\_input\_next').length) {

if ($("#fromAccountSelect").val()==="") {

this.ui.nextButton.attr("enabled", "false");

$("#tboa\_input\_next").prop("disabled", true);

} else if ($("#toAccountSelect").val()!=="") {

this.ui.nextButton.attr("enabled", "true");

$("#tboa\_input\_next").prop("disabled", false);

$("#cbolui-iconDomID-pageErrorTBOA-iconText").html("");

}

else {

$("#tboa\_input\_next").prop("disabled", true);

}

this.model.set({"selectedFromAccount": this.$(this.ui.fromAccountSelect).val()});

}

else {

//show the confirmation Model.

// $("#mybutton").trigger('click');

$('#myModal').modal('show');

}

},

\_toListChanged: function () {

if ($('#tboa\_input\_next').length) {

if ($("#toAccountSelect").val()==="") {

this.ui.nextButton.attr("enabled", "false");

$("#tboa\_input\_next").prop("disabled", true);

} else if ($("#fromAccountSelect").val()!=="") {

this.ui.nextButton.attr("enabled", "true");

$("#tboa\_input\_next").prop("disabled", false);

$("#cbolui-iconDomID-pageErrorTBOA-iconText").html("");

}

else {

$("#tboa\_input\_next").prop("disabled", true);

}

this.model.set({"selectedToAccount": this.$(this.ui.toAccountSelect).val()});

}

else {

//show the confirmation Model.

// $("#mybutton").trigger('click');

$('#myModal').modal('show');

}

},

\_rePopulateFromAccount: function () {

console.log("\_rePopulateFromAccount:");

var beforeRefreshFrom = $(this.ui.fromAccountSelect).val();

console.log("Before Selection:" + beforeRefreshFrom);

var fromAccounts = this.model.attributes.destSrcMap[this.model.attributes.selectedToAccount];

console.log("Selected TO Account ::" + this.model.attributes.selectedToAccount);

$(this.ui.fromAccount).html(this.selectTemplate(this.\_formatAccountList(fromAccounts, this.model.attributes.defaultSelectionText, 'fromAccountSelect', 'FROM:')));

$(this.ui.fromAccountSelect).val(beforeRefreshFrom);

},

\_rePopulateToAccount: function () {

console.log("\_rePopulateToAccount:");

var beforeRefreshTo = $(this.ui.toAccountSelect).val();

console.log("Before Selection:" + beforeRefreshTo);

var toAccounts = this.model.attributes.srcDestMap[this.model.attributes.selectedFromAccount];

$(this.ui.toAccount).html(this.selectTemplate(this.\_formatAccountList(toAccounts, this.model.attributes.defaultSelectionText, 'toAccountSelect', 'TO:')));

$(this.ui.toAccountSelect).val(beforeRefreshTo);

},

\_checkToRender: function () {

var errorMsg = this.model.attributes.errorMsg;

if (errorMsg != null) {

this.render();

}

},

\_preprocess: function () {

console.log("tboa-view-input.js:preprocess");

var data = this.model.attributes;

var output = {

fromElement: {

displaylabel: '',

id: '',

element: []

},

toElement: {

displaylabel: '',

id: '',

element: []

}

};

output.fromElement = this.\_formatAccountList(data.sourceAccountVBList, data.defaultSelectionText, 'fromAccountSelect', 'FROM:');

output.toElement = this.\_formatAccountList(data.destinationAccountVBList, data.defaultSelectionText, 'toAccountSelect', 'TO:');

\_.extend(output, data);

// console.log("preprocess returns:"+JSON.stringify(output))

var errorObj = this.model.get('error');

if (errorObj.status) {

return \_.extend(output, {error: errorObj});

}

return output;

},

\_formatAccountList: function (accountList, defaultSelectionText, id, displaylabel) {

console.log("\_formatAccountList:");

var output = {

displaylabel: '',

id: '',

element: []

};

output.displaylabel = displaylabel;

output.id = id;

output.element = \_.map(accountList, function (account) {

console.log("tboa-input-view:map");

var element = {};

element.itemDescription = account.accountIdentifierText.split('{')[0];

if (account.accountResolverVO[0]) {

if (account.accountResolverVO[0].negativeHandler1 && (account.accountResolverVO[0].negativeHandler1 == 'DISPLAY\_ZERO' || account.accountResolverVO[0].negativeHandler1 == 'DISPLAY\_WITH\_MINUS')) {

element.itemDescription += " -";

}

if (account.accountResolverVO[0].resolverBalanceText1 && account.accountResolverVO[0].resolverBalanceText1 === 'Field Label') {

element.itemDescription += " " + account.accountResolverVO[0].resolverBalance1;

}

if (account.accountResolverVO[0].resolverBalanceText2 && account.accountResolverVO[0].resolverBalanceText2 === 'Field Label') {

element.itemDescription += account.accountResolverVO[0].resolverBalanceText2;

}

}

element.instanceID = account.instanceID;

return element;

});

var defaultElement = {};

defaultElement.itemDescription = defaultSelectionText;

defaultElement.instanceID = ""; //hope no account is named default... might need to clean this

output.element.splice(0, 0, defaultElement);

// console.log("tboa-input-view:output:"+JSON.stringify(output));

return output;

}

});

});

* C3.View extends Marionette.View. It is recommended that you use  
  the listenTo method to bind model, collection, or other events from C3, Backbone  
  and Marionette objects.
* The context (this) will automatically be set to the view. You can  
  optionally set the context by using \_.bind.
* You can make use of all of the view events listed above
* A common use case for the onShow method is to use it to add children views.

var LayoutView = C3.Layout.extend({

regions: {

Header: 'header',

Section: 'section'

},

onShow: function() {

this.Header.show(new Header());

this.Section.show(new Section());

}

});

* By providing an onDestroy method in your view definition, you can  
  run custom code for your view that is fired after your view has been  
  destroyed and cleaned up.
* The onDestroy method will be passed any arguments that destroy was invoked with. This lets you handle any additional clean up code without having to override the destroy method.
* Every view in C3 has a special event called "attach," which is triggered anytime that showing the view in a Region causes it to be attached to the document.
* Like other Marionette events, it also executes a callback method, onAttach, if you've specified one.
* The "attach" event is great for jQuery plugins or other logic that must be executed after the view is attached to the document.
* The attach event is only fired when the view becomes a child of the document. If the Region you're showing the view in is not a child of the document at the time that you call show then the attach event will not fire until the Region is a child of the document.
* View.Triggers
* Views can define a set of triggers as a hash, which will convert a DOM event into a view.triggerMethod call.
* The left side of the hash is a standard C3.View DOM event configuration, while the right side of the hash is the view event that you want to trigger from the view.

define([

'jquery',

'underscore',

'c3',

'templates',

'handlebars',

'msgbus'

], function ($, \_, C3, JST, HandleBars, MsgBus) {

'use strict';

return C3.FlowBaseView.extend({

template: JST['modules/pnt/tboa/templates/tboa-recap.hbs'],

recapHeaderTemplate: JST['modules/pnt/tboa/templates/partials/tboa-recap-header.hbs'],

accountdetailsTemplate: JST['modules/pnt/tboa/templates/partials/tboa-recap-account-details.hbs'],

transactiondetailsTemplate: JST['modules/pnt/tboa/templates/partials/tboa-recap-transaction-details.hbs'],

fxdetailsTemplate: JST['modules/pnt/tboa/templates/partials/tboa-fxfee-details.hbs'],

ui: {

confirmButton: '#InternalTransfer\_Confirm',

changeButton : '#loctfrRecap\_MakeChanges'

},

events: {

"click @ui.confirmButton": "submit"

},

triggers: {

'click @ui.changeButton': 'changeButton:clicked',

'click @ui.terms': 'changeTerms:clicked'

},

initialize: function () {

console.log("tboa-recap-view:initialize");

C3Helpers.Handlebars.registerPartial("recapHeader", this.recapHeaderTemplate);

C3Helpers.Handlebars.registerPartial("recapAccountDetails",this.accountdetailsTemplate);

C3Helpers.Handlebars.registerPartial("recapTxDetails",this.transactiondetailsTemplate);

C3Helpers.Handlebars.registerPartial("recapFxFeeDetails",this.fxdetailsTemplate);

},

preprocessData: function () {

//console.log("TBOA Recap:::" + JSON.stringify(this.model.attributes.data));

return {currentModel: this.model.attributes.data, previousModels: this.previousModelReference};

},

serializeForm: function() {

// TODO: This need to be replace by actual data

return {

\_flowExecutionKey: this.model.attributes.data.\_flowExecutionKey,

\_eventId: "finalSubmit",

"jfp.layout": "Raw"

}

},

onSubmit: function() {

this.ui.confirmButton.attr("disabled", "true");

$('html, body').animate({ scrollTop: $("#region1").offset().top }, 1000);

},

onSubmitError: function() {

this.ui.confirmButton.removeAttr("disabled");

},

onDomRefresh : function() {

$('#TermsAndCondition').on('click',function(){

if($("#TermsAndCondition").is(':checked')){

$('#InternalTransfer\_Confirm\_Disabled').addClass('cbolui-hidden');

$('#InternalTransfer\_Confirm').removeClass('cbolui-hidden');

}else{

$('#InternalTransfer\_Confirm\_Disabled').removeClass('cbolui-hidden');

$('#InternalTransfer\_Confirm').addClass('cbolui-hidden');

}

});

},

// Animation start

// trigger 'animateIn' to let the region know that you're done

animateIn: function() {

this.$el.addClass('slide in');

this.trigger('animateIn');

},

// Same as above, except this time we trigger 'animateOut'

animateOut: function() {

this.$el.removeClass().addClass('fade out');

this.trigger('animateOut');

}

});

});

* A trigger event handler will receive a single argument that includes the following:

view

model

collection

* These properties match the view, model, and collection properties of the view that triggered the event.
* Having access to these allows more flexibility in handling events from  
  multiple views. For example, a tab control or expand/collapse widget such  
  as a panel bar could trigger the same event from many different views  
  and be handled with a single function.

var MyView = C3.View.extend({

// ...

triggers: {

"click .do-something": "some:event"

}

});

var view = new MyView();

view.on("some:event", function(args){

args.view; // => the view instance that triggered the event

args.model; // => the view.model, if one was set on the view

args.collection; // => the view.collection, if one was set on the view

});

# 16.8 Event Aggregator

Events are an integral part of Backbone because they provide a simple yet powerful way of connecting application pieces.

One object - a listener - can be notified of something happening from another object - a publisher.

The listening object can then responds appropriately and get things done.

The vast majority of events in a Backbone application are set up so that the object listening to an event has a direct reference to the object triggering the event. A Model -> View event relationship is a common example.

But events must be viewed beyond two objects that have a direct reference or relationship.

**It is not always possible for the listener to have a reference to the publisher.**

**We use Event Aggregator when we don’t have this direct reference of Listener and Publisher.**

**Event Aggregators:**

An Event Aggregator is an intermediate object that allows other objects to publish events, and objects interested those events to subscribe to them without needing a reference to the publishing object.

C3 provides Event Aggregator via its Wrapper C3.Radio over the Backbone.Radio.

Events provide Radio's publish-subscribe functionality. Multiple objects can subscribe to an event using on or listenTo. And any object can publish an event using trigger. For those of you unfamiliar with Backbone's events, they have two features that you might not expect. First, unlike native browser events or jQuery events, Backbone events are synchronous; when you trigger an event, all callbacks that are registered on that event run immediately. Second, it's easy to pass data with Backbone events. Unlike native browser events which tend to simply consist of an event and a target, any data you pass when you trigger an event gets passed directly to the callback function, making it a true messaging system.

**Functions Provided - on, off, trigger, once, listenTo, listenToOnce, stopListening**

**Sample Usage:**

***Decouple your SubApps with Radio Events***

Loose coupling is one of the most commonly cited design principles in Object Oriented Programming. In Backbone, loose coupling can be applied to mean that your Views should know as little about each other as possible. We can implement this by allowing Subapps to know about and reference their own child Views, but do not allow them to directly reference between each other views. If Views need to communicate up or across a tree of Views, let’s use Radio present in the subapp.

Radio Events are great for alerting other parts of the application about a change of state. For instance, in an application where we have a main content area and a sidebar that provides context for the main area (for instance, in CBOL that shows Navigation in the sidebar next to an main Area), we could use Radio Events to keep our sidebar in sync with the main content.

That might look something like this.

event2ButtonFunction : function()

{

console.log("event2ButtonFunction");

var myChannel = this.getChannel();

myChannel.trigger("SubApp2:View1:Event2");

},

myChannel.on("SubApp2:View2:Event1", function()

{

var element = $("#RadioSubApp2View1EventInfo");

element.html("<h4>SubApp2:View2:Event1</h4>");

//this.$el.find("#RadioSubApp1View2EventInfo").html("<h4>SubApp1:View1:Event1</h4>");

}

);

***When not to use Event Aggregator***

Event aggregators are an important concept and are a necessary part of building a scalable C3 application.

However, it doesn’t make sense to replace all event bindings with an event aggregator.

The primary use case for them is when the object that needs to handle an event does not have a natural relationship or reference to the object that triggers the event.

For example, a view that is binding to a model’s events does not need to use an event aggregator because the view already has a reference to the model.

A menu view, though, may want to use an event aggregator to say a menu item was clicked. This would allow other parts of the application to handle the menu click without needing a reference to the menu.

# 16.9 C3 Radio

Problem:

How to make use of Event Aggregator and other object C3 Radio provides viz., Command, Request and Response Object.

Solution:

A Channel from C3 Radio is required to make use of the above object(s).

Discussion:

* Three types of channels exist
  + Application channel
  + Module Channel
  + Subapp Channel
* These channels provide the proper necessary encapsulation and shield for messages from being unnecessarily propagated to uninterested objects.
* When a message need to be propagated across the application and can be listened by any of the objects without restriction, we can make use of Application Channel.
* Module Channel enables subapp to subapp communication. It will not allow passing the message to subapp present in other modules.
* Subapp Channel enables communication between leaf objects present in a subapp viz.., Model, View. It will not allow cross subapp leaf objects communication.
* A leaf objects can request communication with leaf objects present in other subapp by registering its interest in module channel.
* All the above channels are baked inside the controllers.
* Controllers provides the channel to leaf objects appropriately, eg: ModuleController provides module channel to its leaf objects viz., SubApp, Module specific Model and Module specific View.
* Subapp Controller provides subapp channel to its leaf objects viz., subapp specific Model and subapp specific view.

# 16.10 C3 Command Object

Commands are something to be used for “go do this” semantic instead to rely on “Something Happened”

***Centralize Logging Logic with Commands***

While Events are a way of notifying other components about a change in state, commands are useful when you want to cause a change in state without having to know the implementation.

For instance, we can make use of Commands to have a centralized event-logging module that can take application events that we want logged and send them to a backend API.

We could use events for this, but there's value in using commands. We're explicitly saying that we want our application to log a specific piece of data, whereas a similar event would have the connotation that an event had occurred, and anything that wanted to respond to it could. We can be more confident of what's happening here.

command2ButtonFunction : function()

{

console.log("command2ButtonFunction");

var myChannel = this.getChannel();

myChannel.command("SubApp2:View1:Command2");

},

------------

# 16.11 C3 Request/Response Object

Sometimes we need to do something more than just say “go do this.” Sometimes

we need to say, “go do this, and let me know.”

C3 provides you Request/Response object for these kinds of scenarios.

The same core reasons for building a Command system also apply to a Request/Response system.

The additional requirement of “and let me know”, however, adds a new dynamic: return values.

When it comes to the implementation, the only real difference between a Command system and a Request/Response system is that the Request/Response system has a return statement for the handler execution. But this one single return statement is part of a larger semantic understanding.

When we say request ("price:check"), there is value in understanding that a request has a response.

We expect to receive a response because the semantics of “request” include a response. By comparison, execute ("price:check") doesn’t have the same semantics. It just says “go do a price check” but doesn’t say anything about a response.

Eg:

Requesting for a component to be shared between views.

It may be a good practice to populate the header views, Navigation and footers views as direct HTML in the Application.js. We have to do this for SEO(Progressive Enhancement) – Please refer the section “Where and How to handle SEO Rendering” to know more details about this.

There may be sometime a need to request for this Navigation to be updated inside one of the subapp component(s).

request2ButtonFunction : function()

{

console.log("event1ButtonFunction");

var myChannel = this.getChannel();

var data = myChannel.request("SubApp2:View1:Request2");

alert(data);

},

***How not to use request/response object***

Care should be taken not to abuse request/response object as a Global hashtable to put and retrieve data.

When there is an ultimate need to share data because it doesn’t have direct references, always ensure only your top level object (App/Module/Subapp) is the one that determine from which component to get the data and to which component(s) share the data.

Top level object should act as a controller to determine the data sharing.

# 16.12 Decision Tree

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| S.No | Event Type | Event Source | Event Destination | Source and Destination have direct Reference? | Where to Handle | Comments |
| 1 | DOM Based Event(Local Events) | View | Same as Source View | Yes(Source and destination is same view) | View(Source and Destination one and same) | Use View Events |
| 2 | Change in Model | Model | Model | Yes(Source and destination is same Model) | Model ( Source and destination one and same) | Use listenTo on Model |
| 3 | Change in Model data | Model | View | Yes | View | Use View.ModelEvents |
| 4 | Dom Events(Cross Subapp Events) | One subapp View | Different Subapp View | No | Source View captures the events and forward the event via Radio Event Aggregator present in Subapp Controller | Use Radio Event Aggregator as mentioned above. Note: it should be ideally subapp controller takes care of passing the event between unrelated views |
| 5 | Custom Events | View | View | Yes | Source View | Use Triggers and triggerMethod |
| 7 | Any Custom Events | Any C3 object | Any C3 Object | No | Any Source C3 Object going to use the Radio event aggregator to pass on the events to destination C3 Object | Use Radio Event Aggregator with its parent (App,Subpp) acting as point of control |
| 8 | Events to Persist/retrieve the data | Model | Any C3 Object | No | Any C3 Object requesting for data will use Radio.Request and any c3 object replying for data will use the Radio.Reply | Even though, it may be found to be a good practices to retrieve model via request and response object. Most of the model is specific to the view so it may be advised to use just the direct references instead of request and reply paradigm |
|  |  |  |  |  |  |  |

Chapter 17 Models and Collections

This chapter provides and overview of the requirements and approach for C3 Models and Collections

t

his chapter describes the requirements for C3 Models and collections.

# 17.1 C3.Model Initialize

When creating new model objects, you may want some behavior to be executed on construction, such asserting up different handlers or setting some initial state in your model

To achieve this in C3, just define an initialize function when creating your model object.

To see this initialize function in action, let’s create a simple Book object where a line is written to the console on object creation.

Book = C3.Model.extend({

initialize: function(){

console.log('a new book');

}

});

…..

This initialize function gets called as soon as we create a new instance of the Model object.

In this case, on creation of a new Book object, we’ll see the line “a new book” on the console.

# 17.2 C3 Model Defaults

A common requirement for models is to have some default attributes available. You may want to do this so that optional parameters, not passed through on object creation, have some definition.

In C3, this is done using the defaults object literal.

Book = C3.Model.extend({

initialize: function(){

console.log('a new book');

},

defaults: \_.extend({},C3.Model.prototype.defaults,

        name: 'Book Title',

author: 'No One'

    )

}

});

Now when the object is created, these default values are provided for each instance.

Please ensure to use .extend to for the default attributes. This will ensure you didn’t overwrite any attributes available in the parent class.

# 17.3 C3 Model Attribute Values

Problem:

How to access the attributes Value present in the model

Solution:

The attributes in any model object can be easily retrieved by using the .get function and passing through the name of the attribute that you want to access.

console.log(myBook.get('name'));

During object creation, it is possible to pass through the values of the attributes in the model so that each instance can be unique.

Var thisBook = new Book({name : ‘Test’,

author: ‘testA’});

Now when you retrieve the values of either attribute, the value passed through on creation will be used in place of the default.

You can also use the .attributes property to get a JSON object that represents all of the model data.

console.log(thisBook.attributes); // a JSON representation of all attributes

# 17.4 C3 Model Changing Attribute Values

**Problem:**

How to change the model attributes

**Solution:**

Changing attribute values outside of the constructor is done in a similar way, using the structure of a function call in the format .set('<variable name>', <value>).

You can add new attributes to your object in the same manner as shown previously.

thisBook.set('year', 2015);//creates a new attribute called year

# 17.5 C3 Model Deleting Attribute Values

Problem:

How to delete attributes from the Model

Solution:

You can delete an attribute from a Model using unset method

thisBook.unset('year');

C3 provides a neater way to check for the presence of an attribute in a model, using the .has function, which returns a Boolean.

You can also delete all attributes from your model using the clear function.

newBook.clear();//remove all attributes

Both the unset and clear functions take an additional options object, in which you can add a Boolean indicating that no change event should be triggered from the model when the operation is complete.

# 17.6 C3 Cloning Models

Problem:

It’s common that you might want to make a complete copy of your C3 model, keeping all the same attributes.

Solution:

Rather than needing to worry about the details of how to create a deep copy, you can simply use the .clone() method to create a cloned model instance.

Var clonedBook = thisBook.clone();

# 17.7 C3 Model – Handling CRUD

Problem:

You want to do CRUD operation with your data from the AppServer.

Solution:

Use C3.Model to Create, Read, Update and Delete data.

Discussion:

C3 Supports both REST and NON-REST APIS for the CRUD operations.

The default **sync** handler maps CRUD to REST like so:

* **create → POST**/collection
* **read → GET**/collection[/id]
* **update → PUT**/collection/id
* **delete → DELETE**/collection/id

For NON REST APIS

* C3 Provides Support to emulateHTTP and emulateJSON
* C3 also provides partial update to model via Patch attribute set as true.
* In nutshell, following are the property being set as true by defaults
  + emulateHTTP - Setting this option will fake PUT and DELETE requests with a HTTP POST, setting the X-HTTP-Method-Override header with the true method.
  + emulateJSON- setting this will cause the JSON to be serialized under a model parameter, and the request to be made with aapplication/x-www-form-urlencoded MIME type, as if from an HTML form.
  + PATCH
* Empty Model without any data will trigger GET call to the URL provided in the model class
* When the model is set with data, it will POST data to the URL provided in the model class
* ID Attributes is set as unique by platform to ensure when model is updated with Data, it doesn’t send entire attributes but instead only the changed attributes

# 17.8 C3 Model – Validation

C3 provides a validation mechanism for model data, meaning that you can have all the logic that determines whether the state of the model is correct or not within the model, rather than in some external JavaScript or form-processing code.

If you provide a validation method, it will be run every time the .save function is invoked and during every set/unset operation when {validate:true} is provided as an optional parameter.

define([

'underscore',

'c3'

], function (\_, C3) {

'use strict';

return C3.Model.extend({

url: '/c3/sb/accounts/payment/transfer/tboa1',

//url: 'NGAMA/GBGCB/pnt/tboa/SourceAndDestinationAccounts',

defaults: {

id: 'tboa-input-model',

selectedFromAccount: '',

selectedToAccount: ''

},

validation: {

selectedFromAccount: {

required: true

},

selectedToAccount: {

required: true

}

},

sync: function(method, model, options) {

if(method==='update' || method==='patch') {

model.url='/c3/sb/accounts/payment/transfer/tboa2';

//model.url = 'NGAMA/GBGCB/pnt/tboa/AmountAndDate?sourceAccountIndex='+model.get('sourceAccountIndex') +'&destinationAccountIndex='+model.get('destinationAccountIndex');

}

return C3.Model.prototype.sync.apply(this, arguments);

},

initialize: function () {

}

});

});

If you break any of these rules when manipulating the model, the operations will fail to change the attribute

values.

When a validation error has been detected, an event is fired. By adding an “invalid” event handler, you can provide feedback on the validation error. As with all event handlers, this should be added to your initialize function.

define([

'underscore',

'c3'

], function (\_, C3) {

'use strict';

return C3.Model.extend({

url: '/c3/sb/accounts/payment/transfer/tboa1',

//url: 'NGAMA/GBGCB/pnt/tboa/SourceAndDestinationAccounts',

defaults: {

id: 'tboa-input-model',

selectedFromAccount: '',

selectedToAccount: ''

},

validation: {

selectedFromAccount: {

required: true

},

selectedToAccount: {

required: true

}

},

sync: function(method, model, options) {

if(method==='update' || method==='patch') {

model.url='/c3/sb/accounts/payment/transfer/tboa2';

//model.url = 'NGAMA/GBGCB/pnt/tboa/AmountAndDate?sourceAccountIndex='+model.get('sourceAccountIndex') +'&destinationAccountIndex='+model.get('destinationAccountIndex');

}

return C3.Model.prototype.sync.apply(this, arguments);

},

initialize: function () {

this.on("invalid", function(model, error){

console.log("\*\*Validation Error : " + error + "\*\*");

});

}

});

});

Without the validation flag, the validation function will not be executed on set. However, you can check whether

the model is in a valid state at any time with the isValid() function.

define([

'underscore',

'c3'

], function (\_, C3) {

'use strict';

return C3.Model.extend({

url: '/c3/sb/accounts/payment/transfer/tboa1',

//url: 'NGAMA/GBGCB/pnt/tboa/SourceAndDestinationAccounts',

defaults: {

id: 'tboa-input-model',

selectedFromAccount: '',

selectedToAccount: ''

},

validation: {

selectedFromAccount: {

required: true

},

selectedToAccount: {

required: true

}

},

sync: function(method, model, options) {

if(method==='update' || method==='patch') {

model.url='/c3/sb/accounts/payment/transfer/tboa2';

//model.url = 'NGAMA/GBGCB/pnt/tboa/AmountAndDate?sourceAccountIndex='+model.get('sourceAccountIndex') +'&destinationAccountIndex='+model.get('destinationAccountIndex');

}

return C3.Model.prototype.sync.apply(this, arguments);

},

initialize: function () {

this.on("invalid", function(model, error){

console.log("\*\*Validation Error : " + error + "\*\*");

});

console.log('Is model valid: ' + this.isValid());

}

});

});

# 17.9 C3 Model – Saving

* The save function invokes the operation to save the model to the server, invoking the C3.sync function.
* Using C3.FlowBaseview will invoke the Save function automatically once the validation of view is returned without any errors inside the view form submit event.
* The save function can be called with no parameters or can take the set of attributes you want to persist to the server, along with an options hash that contains handlers for both success and error cases.
* Success and error handlers are important when making calls to remote API endpoints, and you cannot be certain that a call to save a model will always be successful.
* Once the call is complete and has returned, the appropriate callback will be invoked, either success or error.
* Don’t forget that calls are made asynchronously, so any lines of code after the save method won’t wait for the save to be completed first.

model.save(this.attributes,

{

success: function(model, response, options){

console.log('Model saved');

},

error: function(model, xhr, options){

console.log('Failed to save model');

}

});

# 17.10 C3 Model – Reset Data

* If you want to reset the state of your model object to the same as it is on the server side, you can invoke the fetch() function.
* Again, this function accepts an options hash that includes success and error callbacks.
* If the execution of the fetch function detects that there is a difference in the models between the server and client sides, a change event will be triggered.
* This can be useful when you want to ensure that the application is in sync with the back-end service or when you need to populate your model objects on application start-up.

model.fetch({

success: function(model, response, options){

console.log('Fetch success');

},

error: function(model, response, options){

console.log('Fetch error');

}

});

# 17.11 C3 Model – Delete

* If you want to delete your model object, you can invoke the destroy() function.
* If the model is new and doesn’t yet exist on the server, the destroy operation will fail.
* Adding a wait:true option will ensure that the model is successfully removed from the server before it is removed from any C3.Collection that contains it on the client side.

model.destroy({

success: function(model, response, options){

console.log('Destroy success');

},

error: function(model, response, options){

console.log('Destroy error');

},

wait: true

});

# 17.12 C3 Model – Parsing Server Response

Problem:

When invoking the save() or fetch() function, you may want to parse the model to enrich your data model by adding additional attributes or removing unnecessary attributes.

Solution:

* You can do this by adding a parse() function to your model definition.
* Note that as well as adding new attributes, you could make other changes to make the data returned from the API work for your front-end application, such as changing the currency of an attribute.

Account = C3.Model.extend({

parse: function(response, xhr) {

response.accountType = checking;

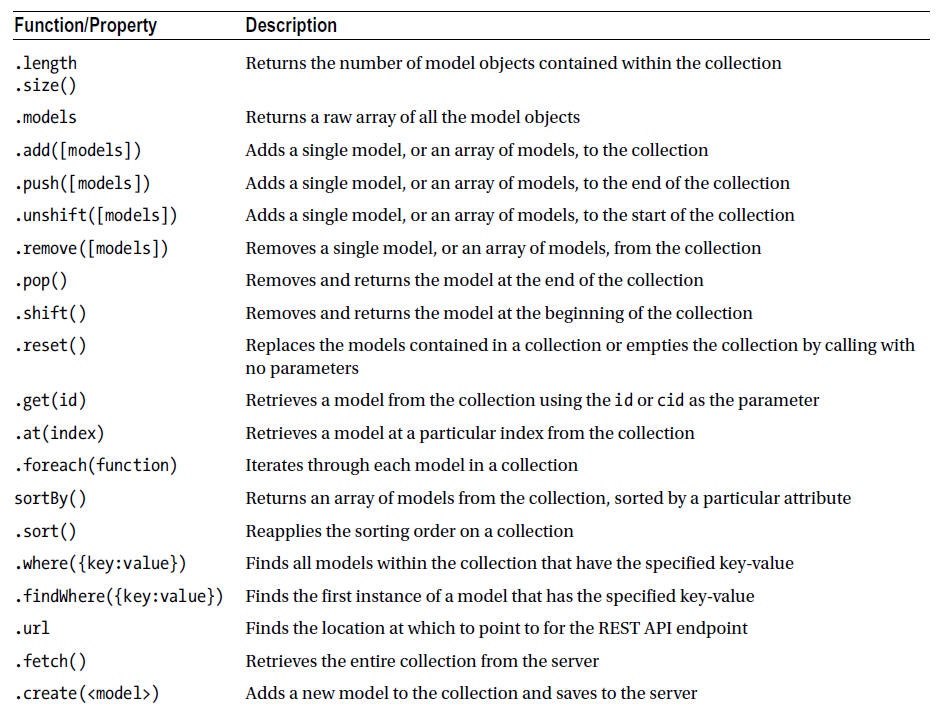
return response;

}

});

# 17.13 C3 Collection

* When creating an instance of the collection, you can pass through an array of model objects to populate it with some initial content.
* var checkingAccount = new Account({name: ‘Checking', number: ‘1234'});
* var savingAccount = new Account({name: ‘Saving, number: ‘5678});
* var accounts = new Accounts([checkingAccount, savingAccount]);
* The .length property allows you to get the number of models currently contained within the collection. Note that the .size() function returns the same number.
* Adding a new model to your collection can be easily done using the .add method.
* If a model is already present in the collection when passed through to .add, it will be ignored. However, if {merge: true} is included in the call, the attributes will be merged into the duplicate model.
* You can also use the .push() function to add a model to the end of the collection, providing either an array or a single model, as in the .add function.
* To do the opposite of .push()and add the model to the beginning of the collection, use the .unshift function.
* a .remove function is available for removing a single model or array of models.
* A remove event is fired when models are removed. An options object used in the listener can access the index of the element that has been removed.
* The .pop() function removes and returns the last model in the collection.
* To remove the first model in the collection, use the .shift() function rather than .pop(). This will also return the model you are removing.
* The reset function exists to provide the ability to replace the set of models in a collection in a single call.
* You can empty the collection in one go by calling the reset method with no parameters. accounts.reset();
* Using reset fires a single reset event rather than a sequence of remove and add events. This is useful for performance reasons because your application needs to react just once to a reset operation.
* The set function is described by the official Backbone documentation as a way to perform smart updates on a collection. By passing through an array of models, set abides by the following rules:
* if a model doesn’t yet exist in the collection, it will be added. The rule will be ignored if {add: false} is provided.
* If a model is already in the collection, the attributes will be merged. The rule will be ignored if {merge: false} is provided.
* If there is a model in the collection that isn’t in the array, it will be removed. The rule will be ignore if {remove: false} is provided.



Collection Event



# 17.14 Handling Data Latency

Problem:

How to handle Data latency before displaying a view?

Solution:

* What we need is some way to wait until the data has been returned before instantiating our view.
* One way of achieving this is using a callback function, but that doesn’t scale well: what if we need to wait for multiple data sources to be returned before displaying them? Recursively providing callbacks to callbacks starts getting very painful, very fast.
* Instead, we’ll use jQuery Deferreds, which will allow us to use a much cleaner mechanism to wait for the required data before instantiating a view.
* Let’s use a deferred object to return a promise from our “entity” handler

getContactEntity: function(contactId){

var contact = new Entities.Contact({id: contactId});

var defer = $.Deferred();

setTimeout(function(){

contact.fetch({

success: function(data){

defer.resolve(data);

}

});

}, 2000);

return defer.promise();

}

* A deferred object is essentially “something that will happen later”
* it is used to synchronize code by having it react as the promise is updated (typically with succeess/failure).
* By returning the deferred object’s promise, we’re basically saying “I promise I’ll do something, and I’ll update you as things progress”. This contract allows code elsewhere to simply monitor the promise and react appropriately to any changes (e.g. fresh data coming in).
* You use a Deferred object instance where you need to be able to update the internal request’s status (e.g. indicate when it’s done), and you provide a promise to dependent code.
* That way, dependent code can monitor the deferred’s progress, but cannot modify it.
* For all practical purposes, you can think of a promise as a read-only version of a Deferred object instance.
* How do we send that fresh data to the code monitoring the promise? By our call to resolve when the fetch call succeeds, we resolve the deferred object, and we forward the received data.

# 18.1 C3 tool to generate css using less

* If the project is using less, C3 has an automated tool to generate less files on compile time with just minimal configurations.
* The tool can be configured to generate multiple css files based on less folders or some specific files.
* Muliple levels of css can be generated using differnet set of configurations for dev, prod, sit for easy devolupment.
* Less plugins can be directly injected in the configuration to beautify/uglify the css results.

Meaning of the configuration.

basePath : refers to the path of the project

build path: where you want the css to be generated.

lessOptions: various types of less options, ( dev/prod/sit) can be configured with different less options.

combinedCss: put the key as the input and the value can holds less files/folders. Examples shown below.

Sample configuration

{

"basePath": "../Less-Boilerplate",

"buildPath": "C:/Citi Stuff/projectTest/Less-Boilerplate/build/",

"lessOptions": {

"dev": {

"compress": true,

"sourceMap":true

},

"prod": {

"compress": false,

"plugins": [

"new (require('less-plugin-autoprefix'))({browsers:['last 2 versions']})",

"new (require('less-plugin-clean-css'))(cleanCssOptions)"

]

}

},

"combinedCss": [

{

"bestbuy.css": {

"files": [

"lessProject/bestbuy/BestBuy.less"

]

}

},

{"sears.css": {

"folder": "lessProject/sears/"

}

}

]

}