COmanage Style Guide

This style guide is a living document for COmanage 4.0.

See also: COmanage Coding Style and Registry PE Style Guide (for Registry version 5)

Background

As larger numbers of developers participate in the COmanage project, the project must provide a set of guidelines to help developers:

- create consistent interface elements within core code and plugins;
- understand the reasoning behind UI/UX choices; and
- theme the application to meet the needs of a specific deployment.

Additionally, as COmanage gets included in a suite of Internet 2 tools that may benefit from coherency in theme or element design, the project can likewise provide guidance for how this may be accomplished while adopting common approaches where possible.

This Style Guide includes layout guidance, element screenshots, and occasional code samples with explanations for how to implement them. When appropriate, the guide explains the reasoning behind specific elements.

Contents

Constraints and Assumptions

The COmanage UI/UX is itself guided by a number of important considerations that impact the choices of user interface elements, the tooling used to build them, the theme, layout, and fonts, and even client/server interactions which impact the user experience.

1. **Synchronous request / response cycle**

   Provisioning (using Registry data to create or remove access to applications and services) is a fundamental feature of COmanage Registry. For now, the project requires that all actions that result in provisioning be synchronous so that users receive feedback on them in real time.

2. **Keep tooling to a minimum**

   COmanage's adopters have significantly different development resources. In an effort to lower the boundaries to plugin development or theme customization, the project keeps tooling requirements to a minimum. Driven by that aim, COmanage's user interface code is intentionally simple, structured within the CakePHP framework, and requires only a text editor to modify. Developers must understand the basics of PHP and, specifically, CakePHP. Beyond this, we aim to adopt few and widely understood libraries, and we weigh the selection of these both by their ability to serve the project's needs as well as by the library's stability and size of its user base. Development that requires separate tooling or a separate

build step (e.g. Node/NPM, CSS pre-processing, etc) is avoided.

3. **COmanage is a project in motion**

   Like all software, COmanage must grow to meet the feature needs of its adopters while keeping up with dependencies that change over time. COmanage Registry 4.0 uses the Bootstrap framework (currently at version 4). An interface library such as Bootstrap provides useful features such as mobile breakpoints, common interface elements, and common design patterns. Bootstrap was selected because of its wide adoption, simplicity, and existing dependency on jQuery - which is already heavily leveraged by COmanage. However, COmanage also attempts to strike a balance between reliance on such libraries and the use of COmanage-specific markup laid out using CSS. Structures for forms, tables, and common layouts are documented in this Style Guide. (Note that COmanage Registry versions 2 and 3 were built on the now end-of-life MDL library; guidance is provided (*link coming)* for how to convert Plugins that were built in earlier versions.)

   As the number of features in COmanage Registry continues to grow, the navigation and layout paradigms must address and simplify added complexity. In other words, the Style Guide and interface elements presented within, though fixed for a time, will always remain a work in progress.

4. **Accessibility**

   COmanage's user interface is built to conform to WCAG 2.1 AA. Colors, UI features, and underlying markup are chosen and constrained with accessibility at the top of mind.

Custom Themes

COmanage Registry may be styled without touching the underlying code by creating a custom Theme. Registry Themes allow for custom header and footer markup as well as custom CSS which is loaded last among styles. This allows a Theme to override any aspect of the Registry styles directly within the COmanage application.

While the use of direct CSS within custom themes is powerful, it has the disadvantage of being brittle: if the underlying markup changes significantly between versions, the CSS of a theme will need to adapt. As a future enhancement, the project is considering the creation of a simple theme panel for colors and a logo that will protect less development-minded deployers from such changes.

Fonts

The font chosen for the COmange project is **Noto Sans.** The Noto fonts were designed specifically to cover all scripts found in the Unicode standard and was adopted by COmanage to simplify and enhance internationalization. Noto Sans comes in only two weights: Regular (400) and Bold (700).

Regular 400

Noto Sans is the main font used by COmanage

Regular 400 italic

Noto Sans is the main font used by COmanage

Bold 700

Noto Sans is the main font used by COmanage

Bold 700 italic

Noto Sans is the main font used by COmanage

Iconography

COmanage makes use of the **Material Icons** font library for most icons. Material Icons are licensed under the Apache license version 2.0.

Font Awesome (free) is also present and used in a number of places, and jQuery UI Icons have been carried forward as part of legacy styling from versions prior to 1.0. The use of jQuery UI Icons will be deprecated in coming versions.

Colors

Interior background colors in COmanage should be few and tend toward grey-scale in an effort to make Custom Theming simpler. That is, deployers should be able to place a logo and color scheme in a custom theme without the need to significantly modify interior colors of the application. Guidance for changing the primary colors in a custom theme is below, and development will move forward with an eye on making this ever simpler. Link and text colors are set to comply with WCAG AA 2.0 contrast guidelines.

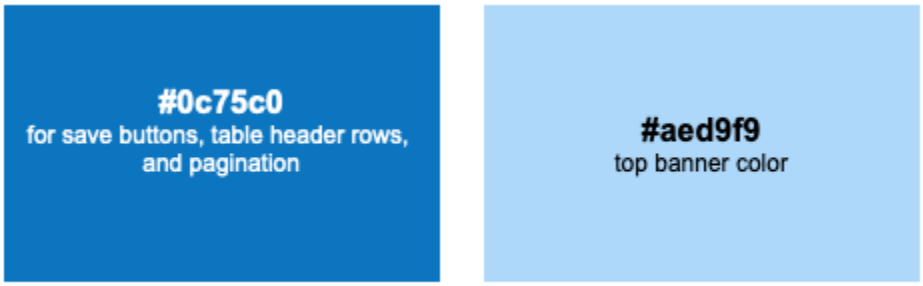
Text colors

- The color for general hyperlinks is #06c. This color passes accessibility contrast checking on white and gray backgrounds down to #eee.
- The standard body font color is #222.
- Color for headings (h1, h2, h3, etc) is #555.
- Please note: text colors should never be lighter than #666 on backgrounds from white down to #eee. On a #ddd background, text colors should not be lighter than #555. The ".text-muted" class (version 4.0 and greater) may be used to lighten text on backgrounds down to #ddd.

Form Field Colors

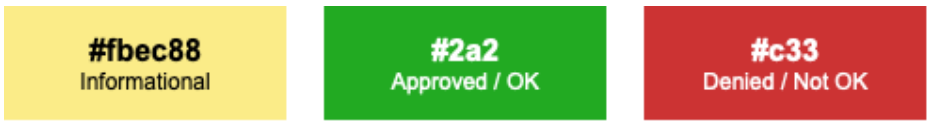
- Input focus color: #ffd (light yellow)
- Input error color: #fee (light pink)
- Input error outlines, and requirement markers (*): #f00 (red)

Primary colors



Background and border colors

Highlight colors



Changing primary colors in a custom theme

While this document can not cover the scope of all color changes you may wish to implement in a custom theme, the following CSS rules can be added to a custom theme and modified to change the most obvious default colors:

CSS for Custom Theme Colors - COmanage version 4.0+

```
/* hyperlinks */
a, #main-menu > li li a,
#main-menu li li a.selected, .widget-actions .material-icons {
  color: #06c;
}

/* light blue banner background color */
#banner, #peopleAlphabet a:hover {
  background-color: #aed9f9;
}

/* primary dark blue background color, mostly for table headings and pagination */
th, .ui-widget-header, .co-grid .co-grid-header, #pagination {
  background-color: #0c75c0;
  color: white;
}
```

Loading Animation

Full-page loading animation

The loading animation from version 3.3.0 and up is built from CSS and is intentionally simple so that it uses very little code and can fire during authentication without the need to reference any external resources. To fire the loading animation from a button or link, add “spin” to the css classes for that link and the ever present #co-loading div that contains the loading animation will be shown.

To launch or hide the loading animation programmatically, call displaySpinner(); or stopSpinner(); with javascript.

Mini loading animation

A smaller loading animation may be rendered with the following markup. Hiding it is up to the page that makes use of it.

```
<span class="co-loading-mini"><span></span><span></span><span></span></span>
```

An example of the mini loading animation can be seen when a dashboard widget is loaded.

Buttons

COmanage buttons found within page content can be classified into four general categories:

1. Buttons used for Saving forms
2. Buttons used to Approve or Deny a petition
3. Buttons used to Edit, View, Delete (etc) a row in a listing
4. Buttons used to access area specific actions (generally links found at the top-right of a section)

1. Save button

Cake PHP

COmanage Save Button

```
<?php if($e): ?>
  <li class="fields-submit">
    <div class="field-name"></div>
    <div class="field-info">
      <?php print $this->Form->submit($submit_label); ?>
    </div>
  </li>
<?php endif; ?>
```

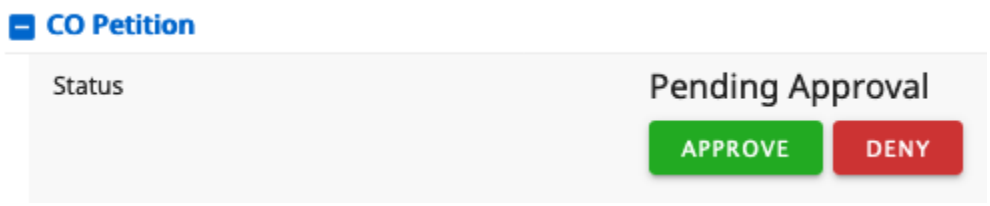
Output

COmanage Save Button Output

```
<div class="submit">
  <input type="submit" value="Save" class="spin submit-button btn btn-primary">
</div>
```

Note: all submit buttons have the "spin" class applied dynamically using jQuery. At the moment, the other classes are added at the same time. This latter approach dates back to the early days of COmanage and will be deprecated.

2. Pending and Approval buttons



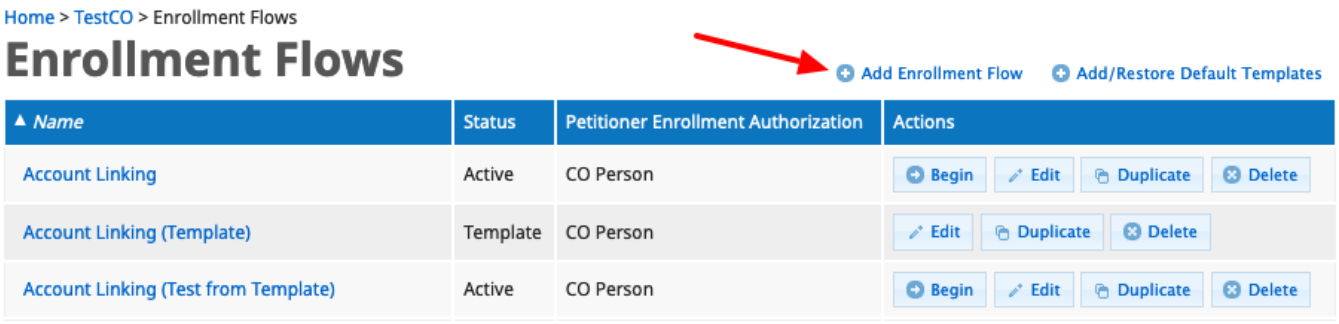
Output

COmanage Pending and Approval Buttons

```
<div class="submit">
  <input class="checkbutton approve-button spin submit-button btn btn-primary ui-button ui-corner-all ui-
widget" name="action" type="submit" value="Approve" role="button">
</div>
<div class="submit">
  <input class="cancelbutton deny-button spin submit-button btn btn-primary ui-button ui-corner-all ui-widget"
name="action" type="submit" value="Deny" role="button">
</div>
```

Note: upon the the deprecation of jQuery UI, the classes on these buttons will be much simplified.

3. Top action buttons



Top action buttons currently use jQuery UI icons and text within a hyperlink. The button icon is determined by class and is established by a block of jQuery that loads at the top of each page in Elements/javascript.ctp. ***Note: this approach dates back to the early days of COmanage and will be deprecated.***

Cake PHP

Top links are specified at the top of pages in an array and are rendered in the pageTitleAndButtons.ctp element.

COmanage Top Links

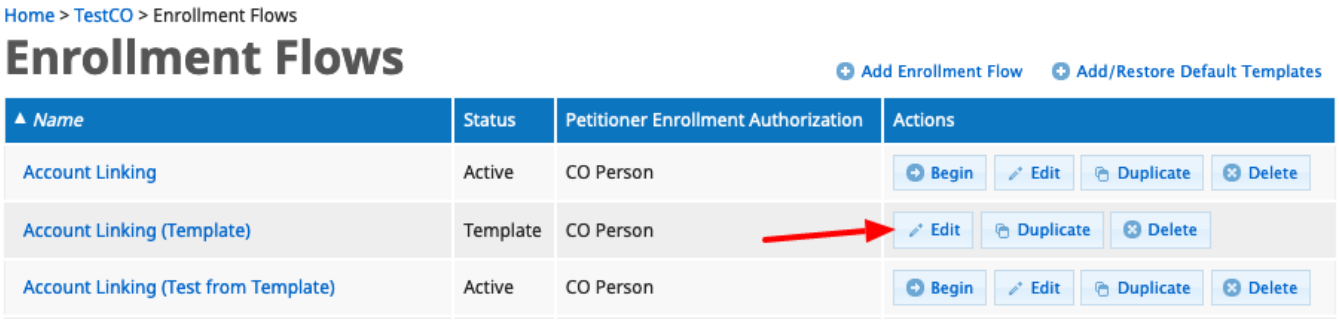
```
// Add page title
$params = array();
$params['title'] = $title_for_layout;

// Add top links
$params['topLinks'] = array();

if($permissions['add']) {
  $params['topLinks'][] = $this->Html->link(
    _txt('op.add-a', array(_txt('ct.co_enrollment_flows.1'))),
    array(
      'controller' => 'co_enrollment_flows',
      'action' => 'add',
      'co' => $cur_co['Co']['id']
    ),
    array('class' => 'addbutton')
  );
  $params['topLinks'][] = $this->Html->link(
    _txt('op.restore.ef'),
    array(
      'controller' => 'co_enrollment_flows',
      'action' => 'addDefaults',
      'co' => $cur_co['Co']['id']
    ),
    array('class' => 'addbutton')
  );
}

print $this->element("pageTitleAndButtons", $params);
```

Output

COmanage Top Links Output

```
<ul id="topLinks">
  <li>
    <a href="/registry/co_enrollment_flows/add/co:2" class="addbutton ui-button ui-corner-all ui-widget" role="
button">
      <span class="ui-button-icon ui-icon ui-icon-circle-plus"></span>
      <span class="ui-button-icon-space"> </span>
      Add Enrollment Flow
    </a>
  </li>
  <li>
    <a href="/registry/co_enrollment_flows/addDefaults/co:2" class="addbutton ui-button ui-corner-all ui-
widget" role="button">
      <span class="ui-button-icon ui-icon ui-icon-circle-plus"></span>
      <span class="ui-button-icon-space"> </span>
      Add/Restore Default Templates</a>
  </li>
</ul>
```

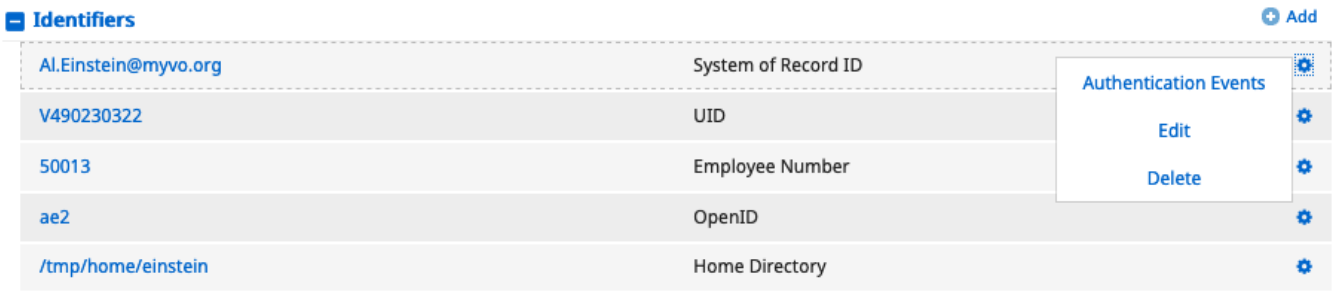
4. Row action buttons



Like the top action buttons, row action buttons currently use jQuery UI icons and text within a hyperlink. Examples of the CakePHP for these buttons can be found in most app/Views/*viewname*/index.ctp files.

Futures

In the future, the layout of the top and row action buttons will change. To simplify information density, we will condense row actions (when appropriate) into a single drop-down of actions for the row, and jQuery UI Icons will be deprecated. For example:



For further examples, see https://miro.com/app/board/o9J_ktm3nUY=/. See also CO-2054 - Getting issue details... STATUS

Forms

COmanage's form markup allows a form page to gracefully alter its layout when moving from desktop to mobile views. It consists of a list of field names and form elements (text input, checkboxes, text areas, drop-downs) with optional descriptive text. Form elements should be kept simple and generally produced by CakePHP. Forms are not built in tables.

Registry uses the following structure to layout forms. CSS classes are used to achieve specific layout effects.

The base structure looks like this:

COmanage Form Markup Basic

```
<ul id="..." class="fields form-list">

  <!-- a field -->
  <li>
    <div class="field-name">
      <div class="field-title">...</div>
    </div>
    <div class="field-info">
      ... <!-- put the form input here -->
    </div>
  </li>

  ...
  ... <!-- include as many fields as are needed -->
  ...

  <!-- Submit button -->
  <? if($e): ?><!-- hide entire row if not editable -->
    <li class="fields-submit">
      <div class="field-name">
        <span class="required"><?php print _txt('fd.req'); ?></span>
      </div>
      <div class="field-info">
        <?php print $this->Form->submit($submit_label); ?>
      </div>
    </li>
  <?php endif; ?>
</ul>
```

Use the following CSS classes in the form-list structure:

- **fields form-list** (on base ul) : top-level classes used to specify a form. The class "fields" is used within explorer structures and should be kept on all form-lists for consistency and future proofing.
    - **form-list-admin** (optional on base ul) : used primarily for configuration pages where the balance between the columns needs to be more even

- **fields-header** (on li) : used at the top of a set of fields for informational text pertaining to the set; typically stands alone without further internal structure.
- **field-stack** (on li) : used to make the field name and field info (form elements) stack vertically; this is particularly intended for textarea fields
- **fields-submit** (on li) : used on final list element for the submit button. Its primary use is to turn off background color.

structures within the list element:

- **field-name** (div) : signifies the field name left column
    - **field-title** (div) : wrapper for the field title (typically renders in bold)
    - **vtop** (optional on any div) : sets vertical-align top (field names are normally aligned middle or bottom).
- **field-info** (div) : signifies the field input right column
    - **checkbox** (optional on field-info div) : used for special treatment of checkboxes
        - **subfield** (div) : used within checkbox div for a small subfield, typically hidden or shown by the checkbox
- **field-children** (ul) : a sibling to field-name and field-info containing a large set of fields (or data) rendered inside the parent form-list. The internal list elements use the same classes (field-name, field-info, etc) as the form-list structure.

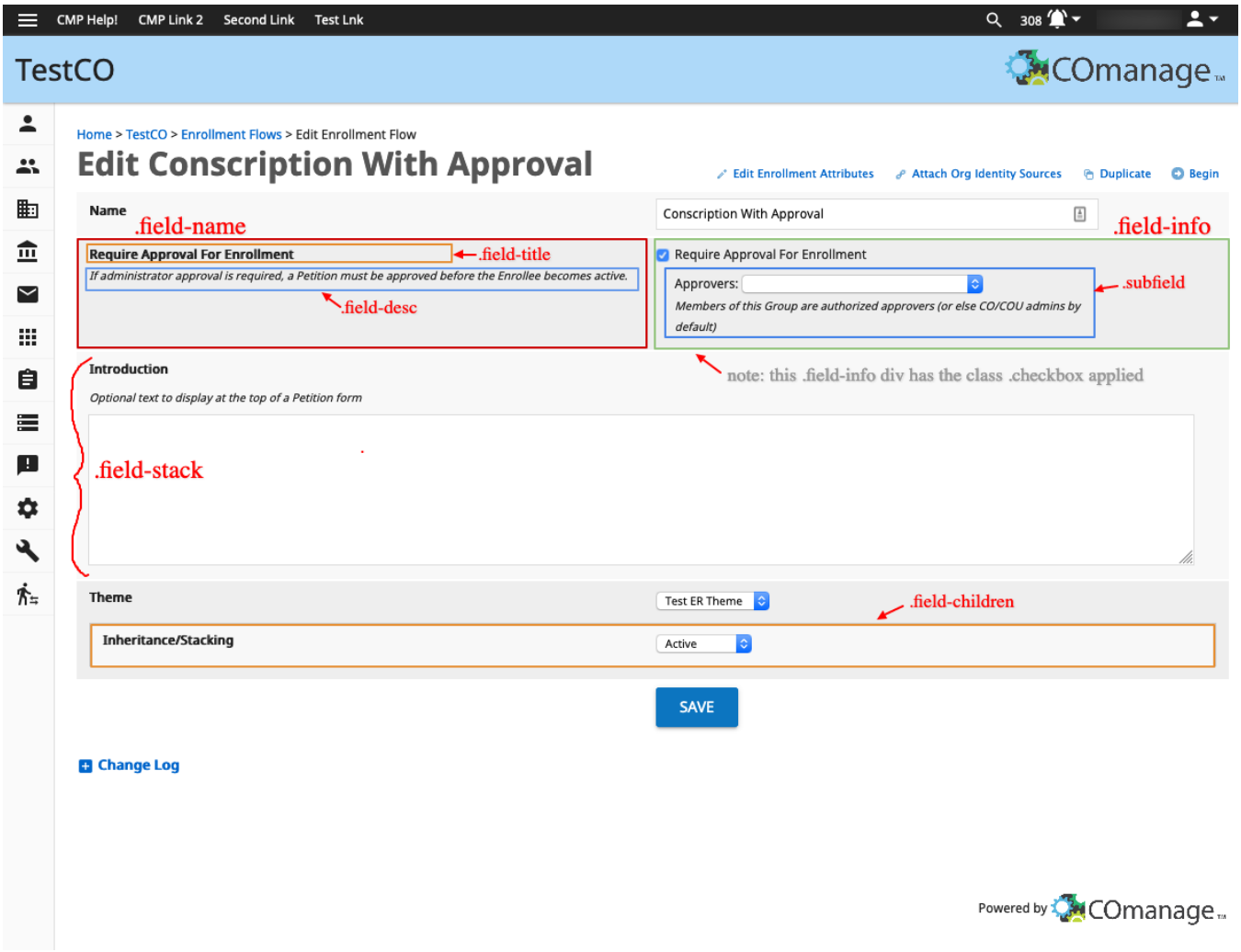
    - **field-desc** (div) : used within field-name or field-info divs to provide descriptive text

A more complete / complex structure might look like this:

COmanage Forms Markup with Options

```
<ul id="..." class="fields form-list form-list-admin">
  <!-- "form-list-admin" provides more equal balance between the field-name and field-info columns;
       it is used primarily for configuration forms where the field-desc block is used heavily in
       the field-name block. -->
 <li class="fields-header">
   ... <!-- simple, descriptive text used as a heading for a collection of fields -->
 </li>
 <li>
    <div class="field-name vtop">
      <!-- vtop aligns the field-name to the top of its container.
           This is useful when the field-info block is large. -->
      <div class="field-title">...</div>
      <div class="field-desc">...</div><!-- optional descriptive text - can be placed
         here or within the "field-info" block, under the form input -->
    </div>
    <div class="field-info checkbox"><!-- the "checkbox" class applied here helps checkboxes align with labels
-->
      ...
      <div class="subfield">...</div>
          <!-- used for small field collections (e.g. drop-downs) that are exposed or hidden based on a
checkbox interaction -->
    </div>
  <li>
  <li class="field-stack">
        <!-- field-stack causes the field-name and field-info blocks to stack vertically;
         this is particularly useful when the field input is a textarea
         (the textarea will expand to fill the horizontal space) -->
    <div class="field-name">
      <div class="field-title">...</div>
      <div class="field-desc">...</div>
    </div>
    <div class="field-info">
      ...
    </div>
  <li>
  <li>
    <div class="field-name">
      <div class="field-title">...</div>
    </div>
    <div class="field-info">
      ... <!-- form input -->
      <div class="field-desc">...</div><!-- the optional descriptive text can also go here -->
    </div>
    <ul class="field-children">
      ... <!-- used for a large subsection contained under the current field; typically hidden or shown based
on
               a form interaction. The list elements use the same structure as form-list. -->
    </ul>
  </li>

  <? if($e): ?>
    <li class="fields-submit">
      <div class="field-name">
        <span class="required"><?php print _txt('fd.req'); ?></span>
      </div>
      <div class="field-info">
        <?php print $this->Form->submit($submit_label); ?>
      </div>
    </li>
  <?php endif; ?>
</ul>
```

This screenshot illustrates how some of these classes render (excerpt of fields from "Edit Enrollment Flow"):



See the code for more examples. This structure can be commingled with other structures. Note also that as COmanage takes advantage of Bootstrap, more Bootstrap classes may be attached to form elements.

Date Pickers

jQuery date pickers are used for selecting dates in COmanage date fields. Due to the way jQuery inserts a datepicker widget into the DOM, we must use some positional styling to ensure that the datepicker appears next to the appropriate field when that field is focused. To allow for this, you must wrap any date picker in a container with the class "modelbox-data".

The datepicker has four different formats defined within COmanage by default, and these are specified by assigning a specific datepicker class. They differ primarily by what date format is used with each:

1. .datepicker (dateFormat: "yy-mm-dd")
2. .datepicker-f (dateFormat: "yy-mm-dd 00:00:00")
3. .datepicker-m (dateFormat: "mm-dd")
4. .datepicker-u (dateFormat: "yy-mm-dd 23:59:59")

View app/View/Elements/javascript.cpt for the full definitions.

Here is an example of a datepicker field rendered in a plugin. Note that the outer <li> has the class "modelbox-data" and we pass in the class "datepicker-u":

Using a Datepicker in COmanage

```
<li class="modelbox-data">
  <div class="field-name">
    <?php print _txt('fd.valid_through'); ?>
  </div>
  <div class="field-info">
    <?php
      if($e) {
        $args = array(
          'class' => 'datepicker-u'
        );

        if(isset($co_announcements[0]['CoAnnouncement']['valid_through'])
          && $co_announcements[0]['CoAnnouncement']['valid_through'] > 0) {
          if(!empty($vv_tz)) {
            // We need to adjust the UTC value to the user's local time
            $args['value'] = $this->Time->format($co_announcements[0]['CoAnnouncement']['valid_through'], "%F %
T", false, $vv_tz);
          } else {
            $args['value'] = $co_announcements[0]['CoAnnouncement']['valid_through'];
          }
        }

        print $this->Form->text('valid_through', $args);
      } else {
        if(isset($co_announcements[0]['CoAnnouncement']['valid_through'])
          && $co_announcements[0]['CoAnnouncement']['valid_through'] > 0) {
          print $this->Time->format($co_announcements[0]['CoAnnouncement']['valid_through'], "%c $vv_tz",
false, $vv_tz);
        }
      }
    ?>
  </div>
</li>
```

Tables

Tabular data is common in COmanage and tables, when used, should be wrapped in a div with class “table-container” to allow for overflow and/or display changes on small devices.

COmanage Table Markup

```
<div class="table-container">
  <table id="meaningful_id">
    ...
  </table>
</div>
```

Legacy note: you may notice some row classes produced in COmanage legacy code: “line1” and “line2” (i.e. <tr class=”line1”>). These were used for zebra striping long ago and can be safely ignored - and ultimately will be removed. All current striping is managed in CSS.

Tabs

Tabs may be used for navigation of COmanage subsections when warranted. As of COmanage 4.0 this is achieved using Bootstrap classes: https://getbootstrap.com/docs/4.0/components/navs/#tabs.

For example, the group subnavigation tabs look like this (limited to just the first two tabs for simplicity):

COmanage Tabs

```
<nav id="cm-group-subnav-tabs" class="cm-subnav-tabs">
  <ul class="nav nav-tabs">
    <li class="nav-item">
      <span class="nav-link active">Members</span>
    </li>
    <li class="nav-item">
      <a href="/registry/co_groups/nest/23" class="nav-link">Nested Groups</a>
    </li>
  </ul>
</nav>
```

and produce this output:

Dialog Boxes

It is sometimes convenient to have dialog-box boilerplate in a view that can be populated on demand. If this approach is taken, apply the ".co-dialog" class to the dialog box to help hide the box on first page load - the class will apply 'display: none' to the box.

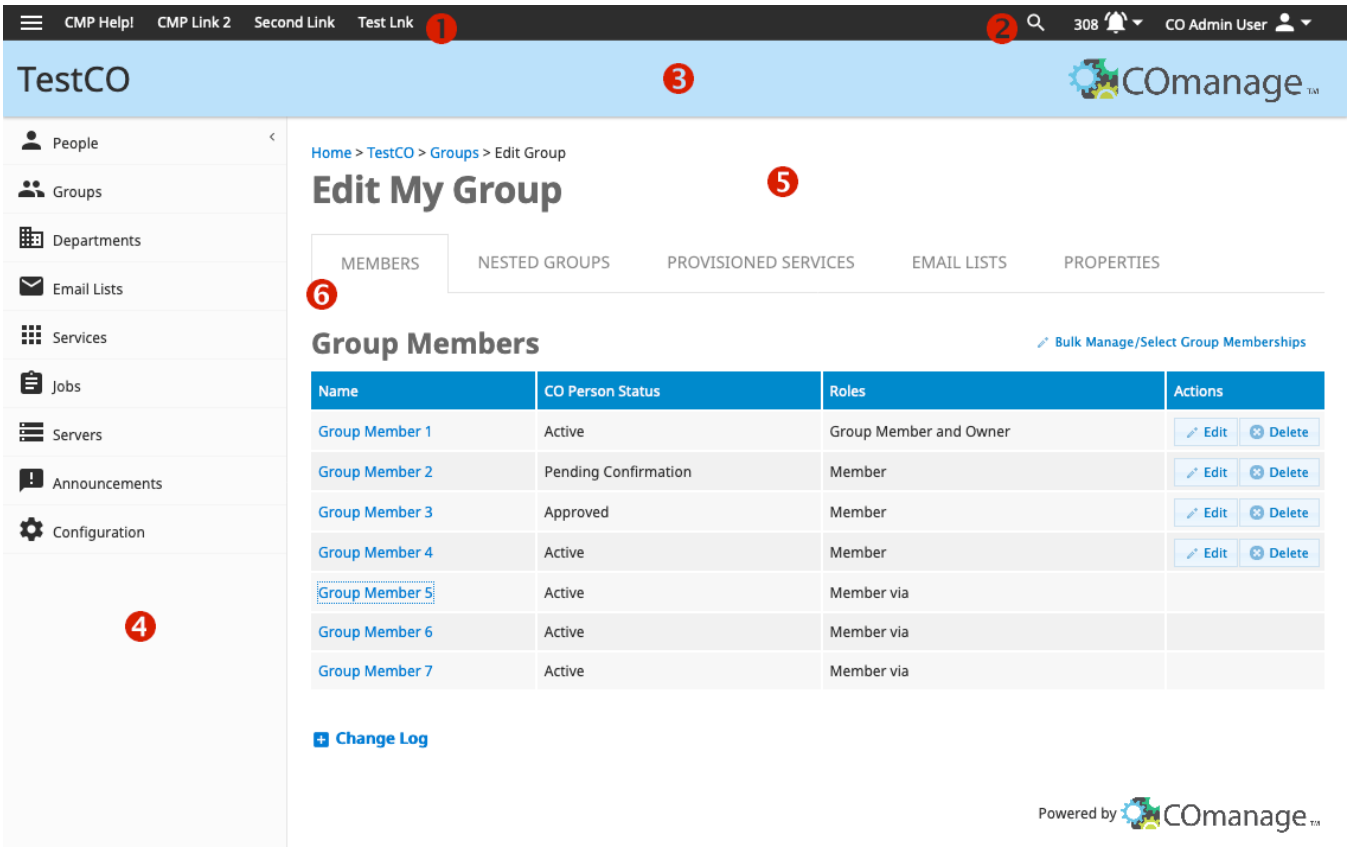
Layout and navigation

Primary Layout

The COmanage interface is divided into roughly three sections,

1. the header containing the user menu, navigation controls, and the banner with CO Title and logo,
2. the primary navigation drawer, and
3. the main content area.

The file that lays out these fundamental sections is *app/View/Layouts/default.ctp*.



Major components include:

1. Navigation controls and optional links: the hamburger menu which controls the sliding left menu drawer, and platform-wide and co-wide navigation links (listed consecutively)
2. Search and user navigation: global search, user notifications, and the user menu providing access to the currently authenticated user's profile, groups, and enrollments. Navigational items in this section are intended to persist across all views. The user menu can be found in the file *app /View/Elements/menuUser.ctp.*
3. Banner: containing the CO Title and the COmanage logo
4. Menu drawer: exposing operational tasks for the current CO (as well as platform-wide tasks for the CMP administrator). The main menu can be found in the file *app/View/Elements/menuMain.ctp.*
5. Main content: containing page title, wayfinding (breadcrumbs), and the page content
6. Page sub-navigation: when appropriate, tabs may be used to further divide up a page or sections activities

Person Canvas

The person canvas is used to get a single-page overview of a Person within COmanage. We are working on simplifications to the person canvas. See https://miro.com/app/board/o9J_ktm3nUY=/ and **CO-2054** - Getting issue details... STATUS

Dashboard

The dashboard is a stacked series of widgets that may be presented to an authenticated user as the front page of a CO.