Extranet User Manager

Version 4.1:

Developer Guide

May 14, 2019

Extranet User Manager
7145 West Credit Avenue
Suite 200, Building 3
Mississauga, ON L5N 6J7

TABLE OF CONTENTS

INTRODUCTION **1**

Extranet User Manager Overview 1

About this Guide 2

DOCUMENTATION **3**

PREPARING FOR DEVELOPMENT **4**

HOW TO CUSTOMIZE **4**

MasterPage (for Landing) 4

Images 5

CSS Style Sheets 5

Registration Fields 7

Removing existing fields *7*

Adding new custom fields *7*

Enabling the Available for Registration Groups *9*

Branding 11

Email Configuration *11*

Identity Server and LandingAdmin *11*

Navbar and Button Colors *11*

PASSWORD FIELDS **13**

Password Complexity 13

USING BOOTSTRAP GRID SYSTEM **13**

Rules: *14*

CUSTOMIZING THE IDENTITY SERVER **20**

Overview 20

Modify the Disclaimer page text 21

Mobile 21

DisplayError 22

Additional Pages with Identity Server Branding 22

INTEGRATING EUM INTO CUSTOM APPLICATIONS **23**

Integrating Registration into your own App 24

Integrating My Profile into own App 24

Integrating Forgot Password and Change Password into own App 25

Integrating Login and Logout Features into own App 25

CUSTOM API DEVELOPMENT **26**

Setup OIDC client for your API 26

Optional – Setup claims for OIDC client *27*

C# samples 27

Call IdentityServer to get a token to use when calling EUM *27*

Attaching token to client used to call EUM API *27*

Introduction

Extranet User Manager Overview

Extranet User Manager v4 allows our customers to easily customize the UI for user facing pages and LandingAdmin as well as utilize our API to automate and/or enhance the user experience to meet business requirements.

The LandingAdmin application was developed with html and utilizes JavaScript/JQuery to get and post data using our API.

To customize the pages in landing admin, you will need to update a html file and possibly the corresponding JavaScript file. EUM utilizes a custom object defined in EUM.js that simplifies the majority of actions required when retrieving or posting data using the EUM API. Adding custom data fields can be done by simply adding the html element (e.g. <input>) to the page. Applying attributes to the field will trigger the javascript to read/write the field value from the API.

The Identity Server v4 application can also be restyled by editing the razor cshtml files and the CSS. All of the user facing pages such as Login, Register, Forgot Password are found in the Identity Server application.

About this Guide

This guide provides instructions on customizing Extranet User Manager V4. It is intended to provide instructions for typical customizations as well as providing additional details on enabling the out-of-the-box features.

Documentation

This document is part of the following set of documents available at https://www.extranetusermanager.com/Support.

Document	Description
Extranet User Manager Pre-Requisite Guide	Document including the pre-requisites to install the current release of EUM
Extranet User Manager Deployment and Configuration Guide	Installation and configuration guide for the current release of EUM
Extranet User Manager User Guide	User guide for the end user and admin sections of EUM
Extranet User Manager Branding Guidelines	Branding Guidelines to customize Extranet User Manager
Extranet User Manager Developer Guide	Guide on how to customize Extranet User Manager

Preparing for Development

We recommend that you use your favorite web editor. We use Visual Studio 2017 and Visual Studio Code primarily for integration with Team Foundation Server. Since majority of the customizations do not require compilation, you can use your preferred editor for .cshtml, .js, .css, and .html files

How to Customize

MasterPage (for Landing)

As of version 4.1.7059.3, EUM has rolled all the End user pages, into the IdentityServer application. This instruction may be not be required for your particular EUM version.

EUM.master controls the shared header and footer on any of the Landing pages (including registration). The welcomeMenu is a server control that is rendered by the code file for the Master Page. Generally, you should just be changing html on the EUM.master and should not require any changes to the .cs file.

Many people customize the site, with the master page and CSS only – This type of customization can be done directly on the server and doesn't require visual studio.

- Update the web icons (ico) – these are the small icons in the web browser tabs.

```
<link rel="apple-touch-icon" href="./eum/img/favicon.ico" />
<link rel="shortcut icon" href="./eum/img/favicon.ico" />
```

- Change the main image in the header.

```
<a id="logo" href="/"><img src="/Landing/eum/img/EUM_logo.jpg"></img></a>
```

- Navigation can be added – out of the box, there is a Home, My Profile, Change Password. If you want more languages, or more menu items, update the navigation section (<nav> </nav>) of the master page.

- Footer text is in the masterpage at the bottom – there is text and hyperlinks that can be updated. (<footer></footer>)

The basic HTML5 page layout used in the MasterPage:

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>
        <asp:ContentPlaceHolder ID="title" runat="server">
        </asp:ContentPlaceHolder>
    <!-- </title> -->
    <asp:ContentPlaceHolder ID="head" runat="server">
    </asp:ContentPlaceHolder>
</head>
<body>
    <form id="EUMForm" runat="server">
        <div id="eum">
            <header>
                <nav class="navbar navbar-eum">
                    <div class="container-fluid">
                        <!-- Brand and toggle get grouped for better mobile display -->
                        <div class="navbar-header">
                        </div>

                        <!-- Collect the nav links, forms, and other content for toggling -->
                        <p class="navbar-text navbar-right">
                            <span id="welcomeMenu" runat="server"><a runat="server" id="anchorWelcome"
                        </p>
                        <div class="collapse navbar-collapse" id="bs-example-navbar-collapse-1">
                            <ul class="nav navbar-nav">
                            </ul>
                        </div>
                    </div>
                </nav>
                <div class="container">
                    <a href="/" id="logo">
                        <img src="/Landing/eum/img/EUM_logo.jpg" />
                    </a>
                </div>
            </header>

            <div id="page">
                <asp:ContentPlaceHolder ID="body" runat="server">
                </asp:ContentPlaceHolder>
            </div>
            <!-- /#page -->

           <footer>
                <div class="container-fluid">
                </div>
            </footer>
        </div>
        <!-- /#eum -->
    </form>
</body>
</html>
```

Images

Any additional image files (jpg, gif, png) that you are using should be added to the **eum/img** folder.

CSS Style Sheets

The **eum/css** folder contains two css files **– eum.css** and **custom.css**

You should not modify eum.css – you should override any styles in c**ustom.css**

In the future, when you want to upgrade to a newer version of EUM, you should use the distributed **eum.css** and your own modified custom.css. This way you get any updates, as well as maintaining your changes from the previous version.

The **custom.css** has some commented out classes.

These can be enabled during development to visualize where the rows and form-groups are laid out.

- **rows** are outlined in grey dots
- **form-groups** are blue dashed and show a label/field/validator combination

- **form-horizontal** red dashed collections of form-groups
- any bootstrap **col**umns are a light purple, that gets darker as items are nested

Please read more about rows and columns in **Using Bootstrap Grid System** section.

Registration Fields

The Registration form, and any custom fields you want to add are modified in IdentityServer/views/account/Register.cshtml, IdentityServer/views/account/MyProfile.cshtml and in LandingAdmin/eum/html/UserDetails.html.

Since we've moved away from compile code to html/JavaScript pages, you will need to ensure fields that only fields you want editable are marked accordingly, likewise any field you do not want to edit/appear on each of the forms are marked as read only and/or hidden.

As Registration is anonymous, you can probably test the registration on the development machine, but you will need to deploy your changes to the fully configured EUM installation, to test the admin, and the logged in user versions.

Removing existing fields

To remove a field, locate it in the html, delete or comment out the entire element that contains the control (e.g. <input>).

IdentityServer/wwwroot/eum/js/**register.js** contains code to hide some address fields depending on the country.
If you are removing address fields, or adding validators for province/state you may need to modify register.js for registration. There are corresponding .js files for My Profile (IdentityServer/wwwroot/eum/js/myprofile.js) and User Details (LandingAdmin/eum/js/user-details.js) pages.

Adding new custom fields

Custom fields are stored in the UserExtra Table in the EUM database. The database should first be updated with the new fields you want to store.

PublicProfile and eNewsletter are the example fields shipped in the application.
You can remove these two fields while you are adding the others.
Leave the User_FK and the Entered, EnteredBy, LastModified, and LastModifiedBy fields alone.

In the html, any custom fields come in JSON form from the API in the UserExtra property.

The JSON is parsed and populate the customized values in the html form. The html control **id** and **name** must match the name of the field in the database. For User Extra fields, you must include a **data-extra** attribute with the value "**user**" in the control.

i.e. `<input id="eNewsletter" type="checkbox" name="eNewsletter" data-extra="user" tabindex="21"`

Any new fields, will need a label, a control and possibly one or more validators. These are added as standard html controls to the html page, and configured accordingly.

To make a field required, you will update the <label> control with the class "**required**" and also add the attribute **required="required"** to the <input> control.

EUM uses JQuery Validation plugin for validation. Depending on the control you are validating, you may require to update the corresponding .js file with additional validation rules.

The only rule we set as a default on the validator is to ignore hidden items.

Adding new field

Example of adding an Occupation field in the Register.html

- Add a with class **form-group** within the column you want to render the field in. This column ( ) is found within and

```
<div class="form-group">
    <label class="col-sm-4 control-label">Occupation</label>
    <div class="col-sm-8">
        <input class="form-control" type="text" id="Occupation" name="Occupation" placeholder="eg. Doctor"  />
    </div>
</div>
```

- It'll display like this on the page

Make the Occupation field required

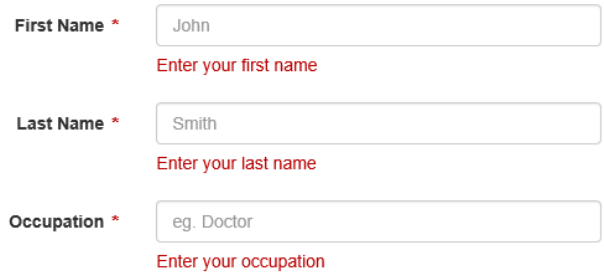
- Add the **required** class to the label
- Add an attribute **required="required"** to the input filed

```
<div class="form-group">
    <label class="col-sm-4 control-label required">Occupation</label>
    <div class="col-sm-8">
        <input class="form-control" type="text" id="Occupation" name="Occupation" placeholder="eg. Doctor" required="required" />
    </div>
</div>
```

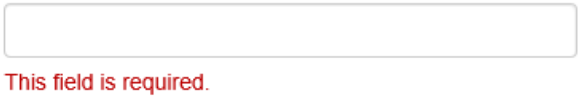
- If you would like to customize the validation error text, you can add a title to the input field

```
<input class="form-control" type="text" id="Occupation" title="Enter your occupation" name="Occupation" placeholder="eg. Doctor"  />
```

- It'll display like this on the page



Without the title attribute on the input field, the required validation error text will display as a generic error



Enabling the Available for Registration Groups

EUM has the ability to place users within Groups upon registration. This requires that these groups are marked for “Available for Registration”. Group management is done through the LandingAdmin/group-details.aspx page. Optionally, these groups can have one or more “Approver Email” address assigned for Approvers to validate the registration and activate the user’s account.

- For a dropdown list ensure the following snippet of code exists on the register.html

<select class="form-control" id="RegistrationRoles" name="RegistrationRoles" title="Group Access Request " tabindex="13" data-lookup="registrationroles" data-type="HTML" data-include-blank="true"></select>

The key here is that this snippet tells the EUM.js to databind the values returned by the API for the registration roles and include a blank entry.

- Ensure that the register.js also has the following highlighted snippet of JavaScript code within the submitHandler function.

  submitHandler: function (form) {

```
        if (typeof (grecaptcha) !== 'undefined' && grecaptcha.getResponse() === "") {
            EUM.message('error', 'Please verify you are not a robot.');
        } else {
            EUM.clearMessage();
            var postData = {};
            postData = $('#EUMPage *:visible, #EUMPage *[type=hidden]').not('[data-extra]').serializeFormJSON();
            postData.UserExtra = JSON.stringify($('#EUMPage [data-extra*="user"]').serializeFormJSON());
            var userMFAInfo = JSON.stringify($('#EUMPage [data-extra*="mfainfo"]').serializeFormJSON());
            if (userMFAInfo != null && userMFAInfo != "{}") {
                postData.UserMFAInfoJSON = userMFAInfo
    }
    var registerRoles = $('#RegistrationRoles').val();
    if (registerRoles != null && registerRoles != "") {
      var myRoles = [];
      myRoles.push(registerRoles);
      postData.RegisterRoles = registerRoles;
    }
```

Branding

Email Configuration

You can add your own images to the automated emails your **EUM** will send to your users.

1. On **EUM LandingAdmin**, sign in as an admin.
2. Go to **Email Settings**
3. Set the **Image Path** to the folder on your **EUM** that contains your branding images.
4. Go to **Email Configuration** and select the email you would like to add your branding to.
5. Click the part of the body of the email you would like to add your image and click the **Image** icon. A popup will appear.
6. On the popup, enter the name of the image you want to add, and select it's dimensions. Click **OK** when done.

   NOTE: Make sure the image exists in the **Image Path** above. ([path set in email settings]\[name entered on email configuration]). There is a good chance that if it is not, the email will be placed in the user's Junk Mail folder by their email client.

   By default, there are images at the top of the body of each email. If you click on the image (or box outlining the image), and click the **image** icon, you will be able to follow the step 6, and get your branding on the email.

Identity Server and LandingAdmin

You can add your branding image to the header of **Identity Server** by going into the **Identity Server** folder **> Views > Shared > _Layout.cshtml.**

For **LandingAdmin** go to the root folder and edit the **EUM.master** file. In this file you can do the same changes outlined above in **MasterPage** section of **How to Customize**.

Navbar and Button Colors

You can change the colour scheme of the navigation bar, primary buttons, and dropdown menus of the navbar in the **custom.css** file in **IdentityServer/wwwroot/eum/css**. There are rules that you can change with the colors for your company:

```
/* EUM Branding Colors */
.navbar-eum,
.navbar-eum .dropdown-menu > li > a:focus,
.navbar-eum .dropdown-menu > li > a:hover,
.app,
.btn-primary {
    background-color: #83275D;
}
```

Sets the background color of the navigation bar, dropdowns of the navbar, and primary buttons. Sets the background color when an item in the dropdown menu is moused over or clicked.

```
.btn-primary:focus,
.btn-primary:hover,
.app:hover {
    color: #83275D;
    border-color: #83275D;
}
```

Sets the color and border color when a primary button is moused over or clicked.

```
.btn-primary {
    border-color: #83275D;
}
```

Sets the border color of primary buttons.

```
.navbar-eum,
.navbar-eum .dropdown-menu > li > a:focus,
.navbar-eum .dropdown-menu > li > a:hover,
.navbar-eum a {
    color: #fff;
}
```

Sets color of items on the navbar and dropdown menus of the navbar. Sets the color for dropdown items when the mouse goes over or when it is clicked.

Password Fields

By default EUM comes with the Password and Confirm Password boxes commented out on the Registration form. The “SetPassword” div can be uncommented if your users should preset their password as part of the registration process.

Password fields can only be used if the user is going to be activated without any workflow approvals. If workflow approvals are required then you should leave these fields commented out on the form.

Password Complexity

You can review the client side rules by navigating to IdentityServer/wwwroot/eum/js/setpassword.js.

You should ensure that your client side validation adheres to your Provider password complexity requirements as configured in the IdentityServer.exe.config element “membership”.

You can update these requirements or bring them to the registration form by updating the variables / rules and applying them to the register.js

The checkPasswords function is where the logic is implemented. This function is called for each keyup event on the password field and the confirm password field.

There are 4 rules by default.

Rule 1 is to ensure that the first character is a letter or number

Rule 2 is to ensure that the password contains at least 1 upper-case letter, lowercase letter and a number

Rule 3 is to ensure that the password is between 8 and 16 characters in length

Rule 4 is to ensure that you typed the same password correctly.

If you would like to remove any of the rule(s), you can remove the rule from the html (li element) and remove the condition from the js. Note: the js refers to the rule by child numbering, so if you remove for example rule 1, ensure that the nth-child referenced for rules “2, 3, and 4” have been re-numbered to match (i.e. nth-child(2) becomes nth-child(1), nth-child(3) becomes nth-child(2) etc.)

Using Bootstrap Grid System

All the pages are using the bootstrap grid system: rows and cols so it helps scaling all mobile, tablet and desktop devices.

All the forms (Labels, inputs, structure) use the bootstrap classes "form-horizontal", "form-group", "form-control".

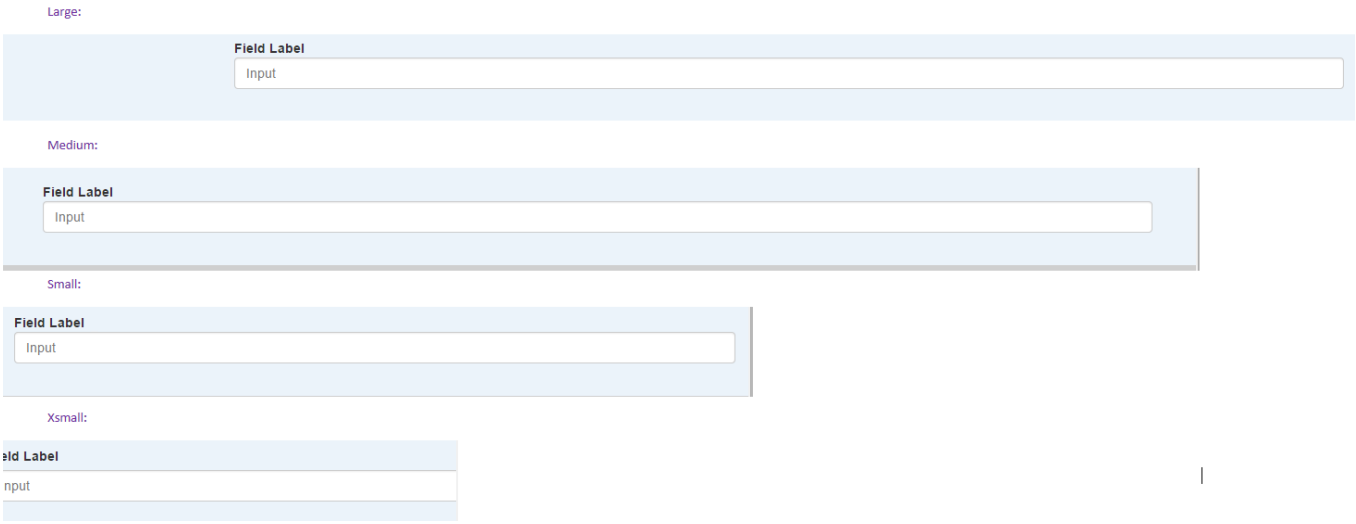
Rules:

- There are total of 12 columns in a row
- If you add a style to xsmall for eg: **.col-xs-4** - this will apply same styling to all the greater sizes (.col-sm- .col-md- .col-lg-).
- If you add a style to **.col-sm-** , then it will apply same style to **.col-md- .col-lg-** but not apply on **.col-xs-**

"Grid classes apply to devices with screen widths greater than or equal to the breakpoint sizes, and override grid classes targeted at smaller devices. Therefore, applying any .col-md- class to an element will not only affect its styling on medium devices but also on large devices if a .col-lg- class is not present." *From http://getbootstrap.com/css/#grid*

1) **Simple Row and forms**

```
<div class="row">
    <div class="form-horizontal">
        <div class="form-group">
            <label class="control-label">Field Label</label>
            <div class="">
                <input id="txt1" maxlength="50" class="form-control" placeholder="Input">
            </div>
        </div>
    </div>
</div>
```

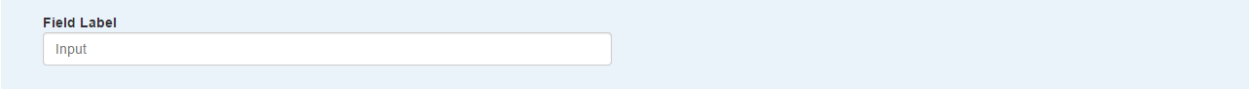


In xsmall Field label is cut-off because .row in bootstrap has negative margins for left and right

```
.row {
  margin-right: -15px;
  margin-left: -15px;
}
```

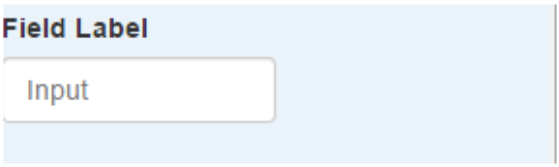
2) **Specifying columns**

```
<div class="row">
  <div class="col-md-6">
    <div class="form-horizontal">
        <div class="form-group">
            <label class="control-label">Field Label</label>
            <div class="">
                <input id="txt1" maxlength="50" class="form-control" placeholder="Input">
            </div>
        </div>
    </div>
  </div>
</div>
```



If the has .col-xs-6 then it will reflect the same styling to all the greater sizes

```
<div class="row">
  <div class="col-xs-6">
    <div class="form-horizontal">
        <div class="form-group">
```

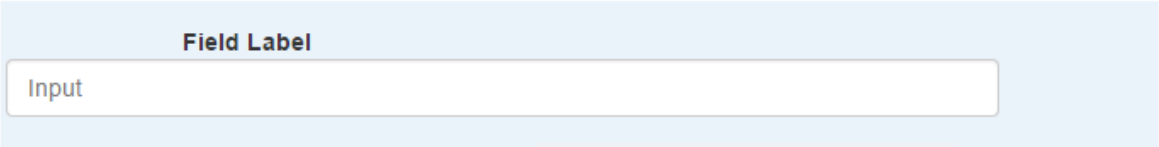


3) **Wrapping elements within a <form-group>**

Wrapping Label and Input field in same line:

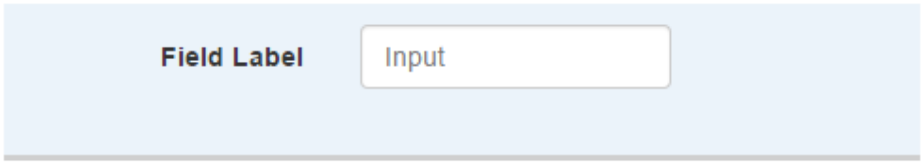
If the <label> is specified to be **.col-sm-4** then it is still not wrapping label and input field in the same line

```
<div class="form-group">
    <label class="col-sm-4 control-label">Field Label</label>
    <div class="">
        <input id="txt1" maxlength="50" class="form-control" placeholder="Input">
    </div>
</div>
```



One needs to specify the that contains the <input> with a specific **.col-sm-** (since label is taking 4 columns, then input should take 6 or less than 6 columns to wrap then both in the same line. Remember there are total of only 12 columns in a row)

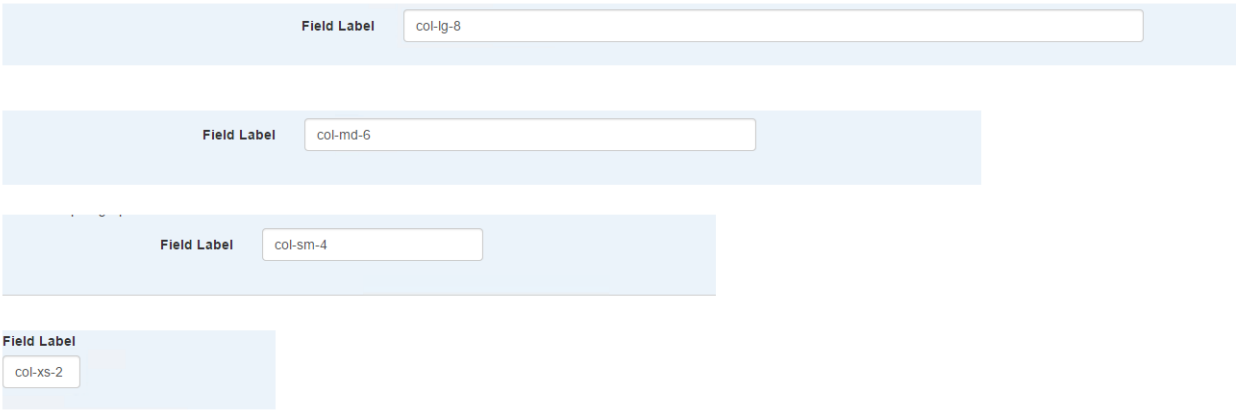
```
<div class="form-group">
    <label class="col-sm-4 control-label">Field Label</label>
    <div class="col-sm-4">
        <input id="txt1" maxlength="50" class="form-control" placeholder="Input">
    </div>
</div>
```



4) **Different styling**

If one wants different styling on different screen sizes then they can also specify all the styles in one class

```
<div class="form-group">
    <label class="col-sm-4 control-label">Field Label</label>
    <div class="col-lg-8 col-md-6 col-sm-4 col-xs-2">
        <input id="txt1" maxlength="50" class="form-control" placeholder="Input">
    </div>
</div>
```



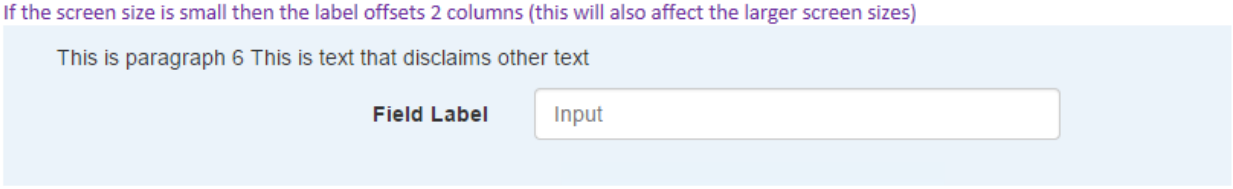
5) Using Offsets

Indent/ Offset

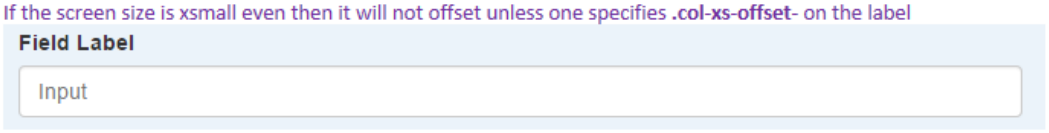
If one wants to indent their label then they can add **.col-sm-offset-** along with other styles (can replace "sm" with any other sizes)

```
<div class="form-group">
    <label class="col-sm-4 col-sm-offset-2 control-label">Field Label</label>
    <div class="col-sm-4">
        <input id="txt1" maxlength="50" class="form-control" placeholder="Input">
    </div>
</div>
```

If the screen size is small then the label offsets 2 columns (this will also affect the larger screen sizes)



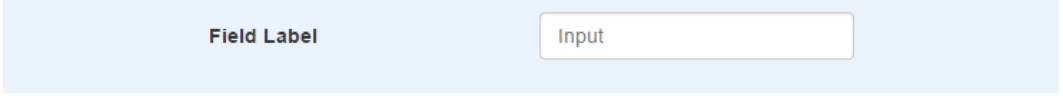
If the screen size is xsmall even then it will not offset unless one specifies **.col-xs-offset-** on the label



If one wants to indent their that has an <input> they can use the same offset class on it

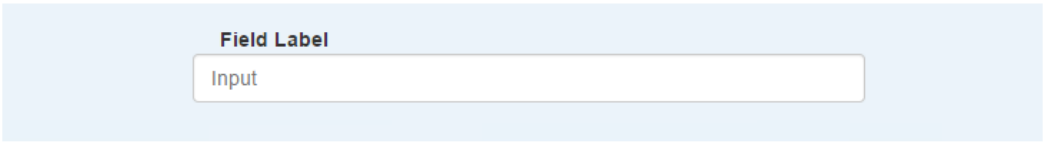
```
<div class="form-group">
    <label class="col-sm-4 control-label">Field Label</label>
    <div class="col-sm-4 col-sm-offset-2">
        <input id="txt1" maxlength="50" class="form-control" placeholder="Input">
    </div>
</div>
```

If the screen size is small then the input field offsets 2 columns (this will also affect the larger screen sizes)

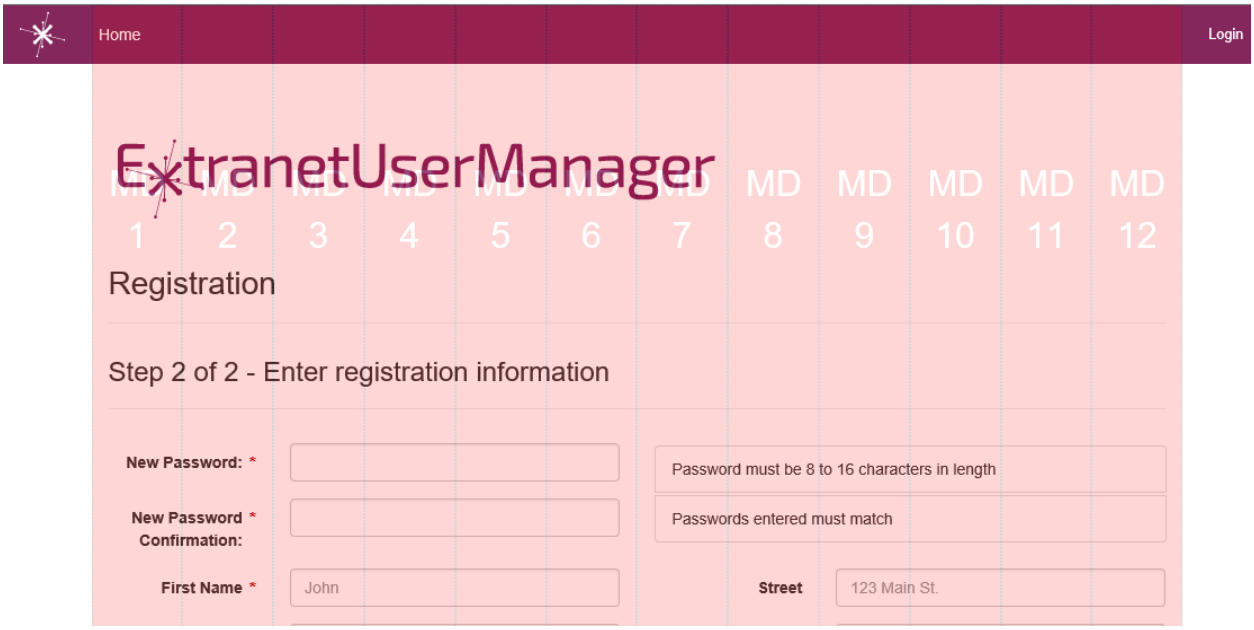


Note: If label is .col-sm-**4**, input div is .col-sm-**8** .col-sm-offset-**2** then total columns is 4+8+2 = 14 (this is more than 12 columns) - then the input field will wrap on the next line

```
<div class="form-group">
    <label class="col-sm-4 control-label">Field Label</label>
    <div class="col-sm-8 col-sm-offset-2">
        <input id="txt1" maxlength="50" class="form-control" placeholder="Input">
    </div>
</div>
```



6) **LandingAdmin and Register contain Eum.utilities.js which includes a bootstrap_overlay function.**
In the browser console type bootstrap_overlay(); and a red responsive grid will be displayed over the page, indicating where the 12 bootstrap columns lay and which adaptive display mode is currently in use **(XS, SM, MD, LG)**



Customizing the Identity Server

Overview

The Identity Server portion, is completely separate from the LandingAdmin application.

- Identity Server is a .Net Core 2.1 application, which compiles down to a single IdentityServer.exe. There are no single page assemblies so you cannot easily add new pages as you can in the Landing Admin web site.
- Identity Server does not have a Master page. Instead the base layout is in **Views\Shared_Layout.cshtml**.
- Each view has one or more .cshtml files, which are written in Razor syntax, and interpreted at runtime. These files can be edited to change the layout of the GUI using any text editor. This is analogous to editing the HTML in the .aspx pages, without recompiling the dll in the back.
- You can find some guidance on the razor syntax from Microsoft http://www.asp.net/web-pages/tutorials/basics/2-introduction-to-asp-net-web-programming-using-the-razor-syntax
- Identity Server caches items in memory (including cshtml files). After making any changes, you should Recycle the application pool from the Identity Server administration pages, or use IIS to reset the App pool or the entire webserver.

Modify the Disclaimer page text

1. The disclaimer page (if configured) will be shown to the user logging in after they have successfully completed the earlier login steps. To enable the disclaimer ensure that IdentityServer.exe.config has the AppSetting “ShowLoginDisclaimer” set to “true”. You can also control how frequently the user will need to re-accept the disclaimer by setting the “LoginDisclaimerExpiryDays” to the desired number of days.

2. Modifying the text is done on IdentityServer/views/shared/disclaimer.cshtml

Mobile

The site uses bootstrap to ensure that the UI is responsive. This includes the end user pages in IdentityServer and the administrator pages in LandingAdmin.

DisplayError

The EIT_ErrorUtil library has been incorporated into Identity Server so that any exceptions will be logged in c:\Logs as they are for Landing and LandingAdmin.

The **IdentityServer\Views\Shared\Error.cshtml** is the template that is displayed when an error is trapped and logged.

The default is similar to Login. It can be restyled by adding css or changing the template as you see fit.

Additional Pages with Identity Server Branding

The default IdentityServer **_Layout.cshtml** is used as the template for all pages. You can either override the shared layout with your own, or modify the default layout with a different logo. Or even just change logo.jpg.

Integrating EUM into Custom Applications

There are some steps that integrating any part of **EUM** into your **application** will need to go through.

1. Add your **application's URL** to the list of accepted origins for you deployment of **EUM**. Your **application** will need to access your **EUM's database** and **API**, which will give back CORS errors if this is not done. This includes going to the **Web.config** file for the **EUM API**, and adding your **application's URL** in the **originHeader** key, under **<appSettings>**. If you're using **EUM** on **Microsoft Azure**, go to you **EUM App Service**, go to the **CORS** tab under **API**, and add your **application** as an allowed origin.

2. Go onto your **EUM LandingAdmin** site as an administrator. Go to **Manage Clients**. You will need to create or update clients for the **application**. You should have a client for the site, and one for the JavaScript. The following needs to be in the **website client**:

    a. **Client Redirect Uris: [site URL]/signin-oidc**

    b. **Client CORS Origins: [site URL]**

    c. **Post Logout Redirect Uris: [site URL],** and **[site URL]/signout-callback-oidc**

3. The following needs to be in the **JavaScript client**:

    a. **Client Redirect Uris: [site URL]/silent**

    b. Client CORS Origins: [site URL]

    c. Post Logout Redirect Uris: [site URL]

4. Create an **appsettings.json** file. Create an object: **Application Settings.** The fields you need are:

    a. **Authority:** "[URL of your **EUM**]/idsrv",

    b. **ClientId:** "[ID of **website's client**]",

    c. **ClientIdJs:** "[ID of **JS client**]",

    d. **RequireHttpsMetadata=** "[true/false]",

    e. **SilentRedirectUri:** "[should be **application's URL**]/[**path** to **silent** page]",

    f. **PostLogoutRedirectUri:** "[should be **application's URL**]"

    g. **Scopes:** "[Found under **Scopes** in **website's client**]"

    h. **ScopesJs:** "[Found under **Scopes** in **JS client**]"

    i. EumApiUrl: "[URL of your **EUM**]/_api/v4"

5. You can change the links that are received by the user through the automated emails (i.e., the Forgot Password email) to the new pages on your **application** (more on those pages below) by going to your **EUM LandingAdmin** site and signing in as an admin. Go to **Email Configuration** and select the email you want to change. You can add your link directly into the body of the email, edit the source of the email to add in the link, or highlight the desired text, click the **hyperlink** button on the top toolbar of the body and add your link through there.

6. You can also change the links on your **EUM Identity Server Login** page. In the **Identity Server** folder, go to **Views > Account > Login.cshtml**. You can change the **href** values for the buttons, like **Register,** to link to your **application's** pages. If the pathnames on the page match the pathnames on your **application**, you can just go to the **IdentityServer.exe.config** (**app.config**) and change the **SiteURL** key's value under **appSettings** to your **application's** URL.

7. Add the **main.js, eum.js,** and **eum.utilities.js** files up on your website (preferred path: **\eum\js**).

Integrating Registration into your own App

1. You will need to have a registration page that takes the place of the **Register.cshtml** page on your **IdentityServer** site. On this page, or a layout that the register page is using, will need to reference, **eum.js, eum.utilities.js,** and **register.js**. See **Registration Fields** in the **How to Customize** section of this guide to see how you can customize this page.

2. Have **eum.js, eum.utilities.js,** and **register.js** on your site, so that your registration page can reference those files (should be in the same folder as the other JavaScript files taken from **IdentityServer**).

Integrating My Profile into own App

1. You will need a page on your **application** that mimics the **MyProfile.cshtml** page from **IdentityServer**. You may choose to write your own javascript but we recommend that you reuse our javascript files if possible. This page will need to reference **eum.js, eum.utilities.js,** and **myProfile.js**. The fields can be customized the same as in **Registration Fields** in the **How to Customize** section of this guide.

2. Make sure **eum.js, eum.utilities.js,** and **myProfile.js** are on your **application**, so that your **profile** page can reference those files (should be in the same folder as the other JavaScript files taken from **EUM Landing**).

Integrating Forgot Password and Change Password into own App

1. You will need pages on your **application** that mimics the **ForgotPassword.cshtml and SetPassword.cshtml** pages from **IdentityServer**.

2. You may choose to write your own javascript but we recommend that you reuse our javascript files if possible. Make sure **eum.js,** and **eum.utilities.js** are on all of these pages on your **application**. Your **Forgot Password** page needs to reference **forgotPassword.js,** your **Set Password** page needs to reference the **setPassword.js**, and your **Change Password** page needs to reference **changePassword.js** (should be in the same folder as the other JavaScript files taken from **IdentityServer**).

Integrating Login and Logout Features into own App

1. The pages that need to act like the respective pages on **EUM Landing** are:

    a. **Authenticate**

    b. **SignOut**

    c. **Silent**

    All of these need to reference **eum.js** and **eum.utilities.js.**

2. If you want email validation on your **application**, make sure you have **emailValidation.js** up on your **application** with the other **EUM Landing** JavaScript files.

3. You will need a sign in button that has **href="/authenticate?returnurl=[url]"** and a sign out button with **href="/signout"** to allow users to login and logout directly through your **application.** These pages need to reference **oidc-client.js** for logout to work properly. You need a function that an **Oidc.UserManager** object with an object containing the app settings from **appsettings.json** and **response_type: 'id_token token',** passed in to create the **Oidc.UserManager** object (i.e. mgr = new Oidc.UserManager([object with app settings]). Then, upon signout, call the function **[name of Oidc.UserManager object].signoutRedirect()** (i.e. mgr.signoutRedirect();).

    NOTE: the sign in and sign out will redirect you to your **EUM's identity server**. The **returnurl** on **authenticate** dictates where you will get redirected to after a successful sign in. You will need to pass in the id_token_hint if you want to be automatically redirected back to your **application**.

Custom API Development

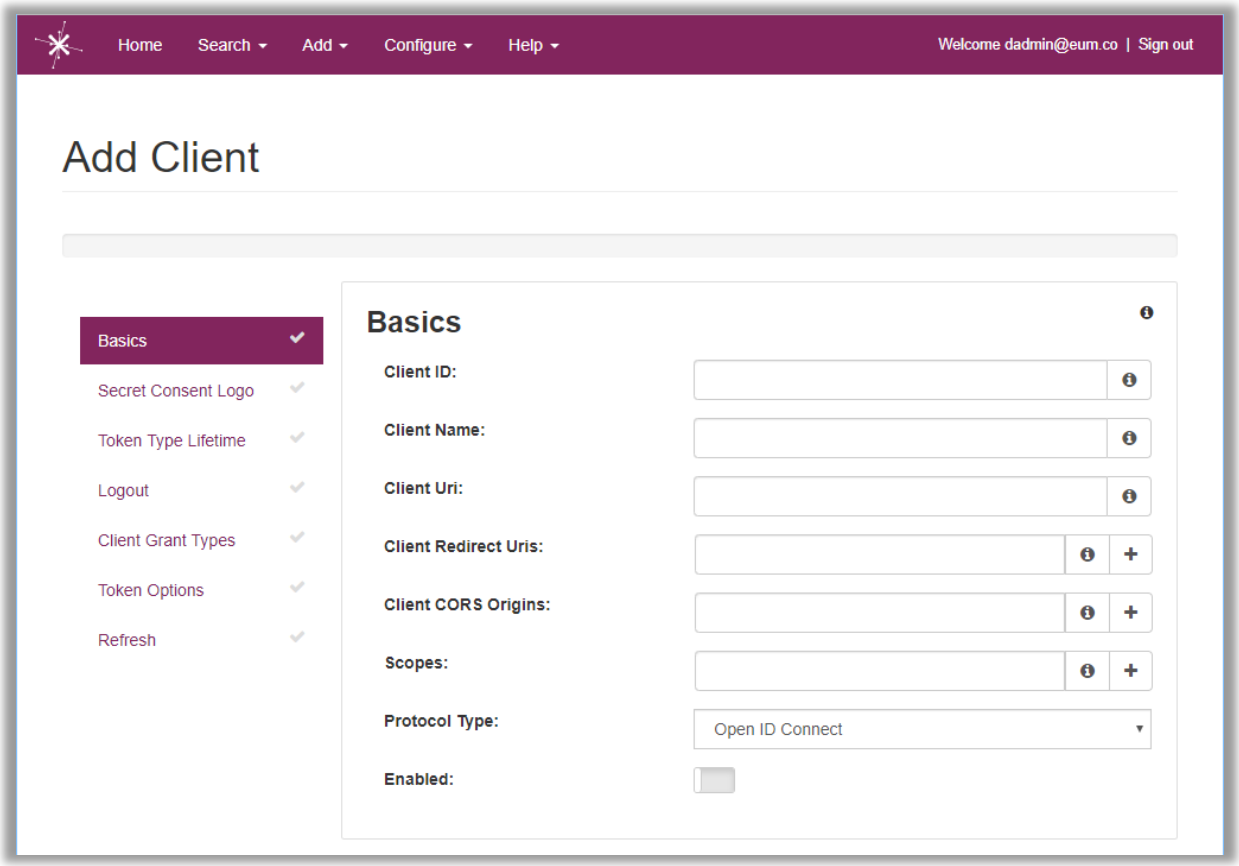
If there are custom business rules that must be enforced you will need to develop a custom API that accepts the requests, validates the business rules and then passes the validated data through to the EUM API. The EUM API requires OIDC tokens that have been issued by IdentityServer. We recommend using IdentityServer as the identity provider for your custom API.

In C# this is done during application startup. E.g.

```
app.UseIdentityServerBearerTokenAuthentication
(new IdentityServerBearerTokenAuthenticationOptions
{
        Authority = "https://<your site>/idsrv",
        RequiredScopes = new[] { "mycustom_api" }
});
```

Setup OIDC client for your API

The LandingAdmin application can be used to setup the most common required parameters for an OIDC client. This is done by selecting Configure > Manage Clients from the menu or Federation Clients tile on the home page. Click Add Client:



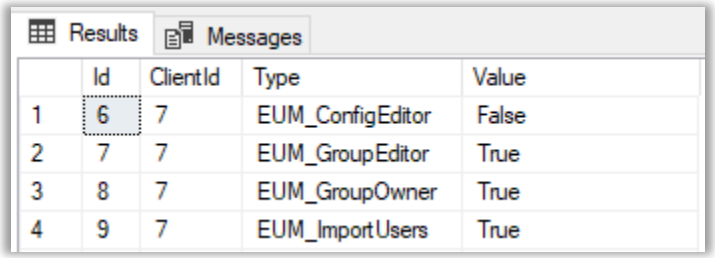
See our Deployment and Configuration guide if you are unfamiliar with setting up OIDC clients.

In the Scopes field you should include the normal OIDC scopes “openid” and “profile”. Since your client will be calling the EUM API include the scope “extranet_api_v4” as well as your own custom scope used by your API.

Optional – Setup claims for OIDC client

Depending on the EUM operations that your custom API will be performing it may be necessary to configure the OIDC client to have elevated permissions for the EUM API. In order to do this you will need to access the EUM database directly and insert records into the ClientClaims table. First look in the Clients table to find Client.Id value for the client you created through LandingAdmin. You will need to supply that value in the ClientId field of the rows you create.

E.g.



Results | Messages

	Id	ClientId	Type	Value
1	6	7	EUM_ConfigEditor	False
2	7	7	EUM_GroupEditor	True
3	8	7	EUM_GroupOwner	True
4	9	7	EUM_ImportUsers	True

In this example Client 7 will have EUM_GroupEditor rights on any operations it performs. This level allows adding/editing of any users or groups managed by EUM. This should be sufficient for any User or Group operations your custom API will need to do.

C# samples

Call IdentityServer to get a token to use when calling EUM

```
        TokenClient tokenClient = new TokenClient(
                              "<your site>/idsrv/connect/token",
                              "<your clientid>",
                              "<your client secret>");

        var cancelToken = new System.Threading.CancellationToken();
        var tokenResponse = tokenClient.RequestClientCredentialsAsync("extranet_api_v4",
null, cancelToken).Result;
        token = tokenResponse.AccessToken;
```

Attaching token to the client used to call EUM API

```
        HttpClient client = new HttpClient();
        client.BaseAddress = new Uri("<your site>/_API/v4/");
        client.DefaultRequestHeaders.Accept.Clear();
```

```
        client.DefaultRequestHeaders.Accept.Add(new
MediaTypeWithQualityHeaderValue("application/json"));
        if (client.DefaultRequestHeaders.Authorization == null)
        {
                client.DefaultRequestHeaders.Authorization = new
AuthenticationHeaderValue("Bearer", token);
        }
```