ion: Developer's Guide v4.1

Updated August, 2013

So you want to create plug-and-play designs for the ion platform? Excellent! Here's what you need to know to get started…

Terminology

First, learn the lingo:

You create one or more **frameworks** that offer one or more **themes** and also contain one or more **master pages**.

What are all these different pieces?

Think of the **framework** as a collection of master pages that are all structurally compatible with each other. In other words, a marketing manager should be able to construct a creative that contains any of the master page layouts, and that sequence should look like it flows. In general, this usually means that all of the master pages in a framework have the same "outer geometry" structure.

A **master page** is formatted like an actual HTML file. You can lay it out using your favorite design tool such as Dreamweaver. Any of the elements of the page that you want to let the marketing manager control—such as headlines, content-specific images, body copy, call-to-action links, etc.—you simply add `ID="fieldname"` (note: each ID in your master page must have a unique value.) and `runat="server"` parameters on the tags.

A **theme** is a collection of images and style sheets that are shared by master pages. The theme is responsible for providing the look-and-feel that is common across all of the master pages in the framework. This often includes colors, fonts, graphics, and branding images (i.e. logo). What is cool is that you can create as many themes as you want for a framework—although you need at least one—and the marketing manager can pick which theme they want for a particular creative.

Each theme contains theme specific images and all of the same .css filenames, but you can have these images and classes differ as subtly or as radically as you want. Maybe there's a "blue" theme and a "green" theme in your framework; maybe there's a "conservative" theme and a "wild, Californian surf party" theme. The important principle is that all themes work with all master pages in your framework—so a marketing manager can switch the look-and-feel of a creative from "blue" to "green" without changing anything else.

Companion Framework

Throughout this document you will find helpful references to example code in the "SampleFW_v4-1" companion framework that came with this guide. These references primarily focus on the code in SampleMaster.ascx (file located in

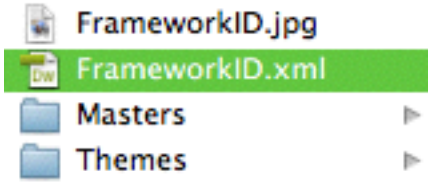
/SampleFW_v4-1/Masters/SampleMaster/SampleMaster.ascx) and SampleMaster_Responsive.ascx (file located in /Sample-FW_v4-1/Masters/SampleMaster_Responsive/SampleMaster_Responsive.ascx), but useful tips and notes can be found in the .css and .xml files as well.

Directory Structure

We recommend that you build your pages locally in a directory structure that mirrors how the ion frameworks are organized. Note that FrameworkID, MasterPageID, and ThemeID must each be unique, case insensitive strings of no more than 28 characters with no spaces: only letters, numbers, and underscore (_) and dash (-) characters.

For instance, in the following, you might substitute your FrameworkID as “Floating_Wizard”, your ThemeID as “Blue_Chrome”, and your MasterPageID as “Big_Image_With_Headline”.

Two files that are required in the root of FrameworkID directory are `FrameworkID.xml` and `FrameworkID.jpg` (note: the name of the these files are the same as the FrameworkID.)



`FrameworkID.xml` specifies the title, description and version of a framework. The format of this XML file should be as follows (also see “/SampleFW_v4-1/SampleFW_v4-1.xml” in the companion framework for an example):

```
<?xml version="1.0" encoding="utf-8"?>

<framework_properties>

<developer>ion</developer>

	<label>Sample framework label</label>

	<description>Sample framework description.</description>

	<version>1.0</version>

	<mode>d</mode>

</framework_properties>
```

Note: The label, description, & version are the only values that should be edited.

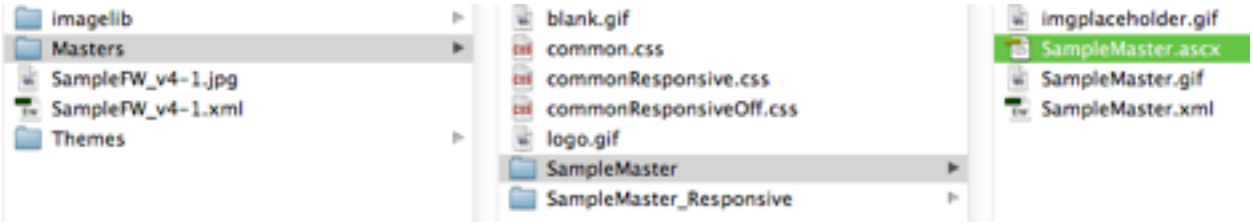
`FrameworkID.jpg` is a thumbnail representation (must be 150x100px) of this framework and appears in the ion console when the marketing manager selects which framework they would like to use (see “/SampleFW_v4-1/SampleFW_v4-1.jpg” in the companion framework for an example).

Generally, your framework files will be placed in two types of directories (`/FrameworkID/Masters/` & `/FrameworkID/Themes/`:

`/FrameworkID/Masters/MasterPageID/`

Each master page in your framework gets its own directory. Your page is just like a plain HTML file—think of it as a static mock-up of how you want the page to appear. The name of the page file is `MasterPageID.ascx`. Note that the name of

the file is the same as the subdirectory name in which it is placed, but with the extension of ".ascx". (Even though this file ends in ".ascx", it is for all practical purposes a ".html" file—with just a few exceptions that we will cover in a bit.)



All of your other resources for this page—primarily GIF, JPEG and PNG images—should go in this directory with the exception of images or CSS files that are associated with a theme and common.css that is associated with all master pages in your framework (more on this in a bit). So there are two types of images you might include on a master page:

- images that can be dynamically chosen as content by the marketing manager (see example #1 & #1b);
- images that will be substituted according to theme, in theme directories (see example #8);

Sample "placeholder" (see example #1) images for the images that marketing managers can choose should be placed in this directory if you choose not to let ion auto-generate them.

NOTE: Only dynamic placeholder images should reside within a master directory (i.e. `/FrameworkID/Masters/MasterPageID/)`.

Non-theme CSS class/IDs that appear in all your master pages can be declared in the CSS file common.css. This file is located in the root /Masters/ directory like so:

```
/FrameworkID/Masters/common.css
```

Responsive master pages should also reference the CSS file commonResponsive.css. This file overrides styles that are specified within the main common.css file and should be referenced after the common.css file in your html. For example, you may want to override fixed-width
columns with percentage-based columns (see example #1 within the SampleMaster_Responsive.ascx file).

This file is located in the root /Masters/ directory like so:

```
/FrameworkID/Masters/commonResponsive.css
```

Note that there are other (required and non-required) files stored in this directory to help with the administration of master pages by marketing managers, such as a thumbnail representation (not required) of this master page (.gif or .jpg, 150x100px) and a descriptive help text file (required). See " /SampleFW_v4-1/Masters/SampleMaster/SampleMaster.xml " and "/SampleFW_v4-1/Masters/SampleMaster/SampleMaster.gif" in the companion framework for examples. If you choose to not include the master thumbnail .gif/.jpg, ion will automatically generate one for you based on the page layout and 'label' value from the master's .xml file.

`MasterID.xml` specifies the title and description of a master page (and optional category and responsive parameters). The format of this XML file should be as follows (see " /SampleFW_v4-1/Masters/SampleMaster_Responsive/SampleMaster_Responsive.xml " in the companion framework for additional comments/notes.):

```
<?xml version="1.0" encoding="utf-8"?>

<master_properties>

       <label>Theme name</label>

       <description>description of the theme</description>

       <category>Sample master category</category>

       <responsive>true</responsive>

</master_properties>
```

NOTE: Adding `<responsive>true</responsive>` to both the responsive master .xml file AND the theme .xml file will communicate to the ion platform that this layout is responsive. If configured in both of these .xml files a button will be added, within the creative studio editor, to toggle responsive on/off. Allowing responsive layouts to be used for both responsive and non-responsive cases. If this parameter is not included within the master and theme .xml files, then the creative studio toggle will not appear in the editor.

`/FrameworkID/Themes/ThemeID/`

Each theme in your framework has its own directory too. Typically, the only files in here will be image files, CSS files and an XML file. The idea is that when a marketing manager selects a particular theme for a creative, all we need to do is swap the subdirectory name we're pointing to for theme elements.

Two files that are required in every theme directory are `theme.css`, and `ThemeID.xml` (note: the name of the theme xml file is the same as the subdirectory name which it is placed). If any of your master pages contain a form, you can also include `themeform.css` to your theme directory. This CSS file governs the look and feel of your forms. See more on setting up your forms on page 6. And if any of your master pages are responsive, you can include `themeResponsive.css` to your theme directory as well.

`theme.css` specifies the look and feel of the theme. This often includes font, colors, element placement, etc...

Every master page in your framework should be supported by the theme, although obviously some master pages will use some theme elements and others will use different ones.

`themeResponsive.css` specifies responsive elements of the theme. Such as media queries, percent-based overrides, max-widths, etc... This file is referenced after the theme.css file within the master page layout (see example #2 within the SampleMaster_Responsive.ascx file).

`ThemeID.xml` specifies the title and description of a theme (and optional responsive parameter) and appears in the ion console. The format of this XML file should be as follows (see " /SampleFW_v4-1/Themes/SampleThemeOne/SampleThemeOne.xml " in the companion framework for additional comments/notes.):

```
<?xml version="1.0" encoding="utf-8"?>

<theme_properties>

       <label>Theme name</label>
```



```
    <description>description of the theme</description>

    <responsive>true</responsive>

</theme_properties>
```

Remember, a framework must contain at least one theme.

Creating a Master Page

Please note that this section will be most applicable to users whose console was provisioned before March 1st, 2014. If your console was provisioned after this date, you may build custom layouts using the creative studio and the base_layout.

Now the fun part, authoring a new master page. It's quite simple: create an HTML page in your favorite program such as Dreamweaver and you're 98% done. So let's start there.

Say you have a static mock-up of a page, `myFirstMaster.html`, here's what you do to make it ready to be a master page:

1. Make sure that every element you want to be dynamically configurable by the marketing manager using your master page is captured by one of these tags:

`<span>, <div>`	for text elements
`<h1>, <h2>`, etc.	for semantic text elements (see example #2)
`<blockquote>`	for semantic text elements
`<li>`	for bullet lists inside `<ul>` or `<ol>` tags (each individual bullet is editable) (see example #3)
`<img>`	for images (don't surround with `<a>` tags, though) (see example #1)
`<a>`	for explicit text links (see example #4)
`<div>`	for widgets/iframes if ID begins with "widget", e.g. "widgetsidebar" (see example #5)
`<meta>`	for meta data such as keywords, description (see example #6)
`<form>`	for a form (one per master page) with ID="FormSocket" (see example #13)

   As of right now, these are the only tags that are dynamically supported by ion. You can have other tags in your master page, but they will be passed on "as is" to the end-user of every creative that uses this master page. Note: we do not recommend using `<p>` tags in your template since the platform automatically inserts `<p></p>` tags as paragraph breaks within multi-paragraph dynamic content.

   With both `<img>` and `<div>` tags used for widgets & images, you can add parameters for `minwidth`, `minheight`, `maxwidth`, and `maxheight` that will constrain the dimensions of content that can be placed in those slots. For example (also see example #5):

   ```
   <div id="widgetmain" maxwidth="600" maxheight="400" runat="server">[widget]</div>
   ```

NOTE: Regardless of element-type specified within the master page layout, the user has the ability to place any kind of element (image, text, links, widgets, etc...) within these dynamic areas using creative studio in the ion platform.

2. Every tagged element that will either be part of a theme or dynamically configurable by the marketing manager needs these two parameters:

   ```
   ID="uniqueID"
   runat="server"
   ```

For example, you might have:

```
<span class="heading" id="Headline" runat="server">[headline]</span>
```

Whatever values you associate with these tags in your master page, for instance the text "[headline]" in the above example, will appear as the default placeholders when a marketing manager first starts editing a new page based on this master page template. Note: If the marketing manager does not place text into this dynamic text field, it will not appear on the live page. And some dynamic elements appear as empty containers within the editor, indicating the element is editable.

3. Two optional parameters can be applied to every tagged element:

   ```
   advancedUI="yes"
   friendlyName="friendly name"
   ```

   For example, you might have (also see example #7):

   ```
   <h1 id="Sidebar_Headline_1" friendlyName="Sidebar Headline 1" advancedIU="yes"
   runat="server">[headline]</h1>
   ```

   When one or more tagged element in a master page contains the `advancedIU="yes"` parameter, the platform enables its advanced mode toggle. By default, any element containing this parameter is hidden from the page editor—you won't see it when editing the page in the platform. Switching to advanced mode, in turn, shows all the elements containing the `advancedIU="yes"` parameter. If you do not want your tagged element to be an advanced UI element, simply leave out the `advancedUI` parameter.

   Think of the `friendlyName="friendly name"` parameter as a user friendly version of the `ID="id_name"` tag. When included on your master page(s), the platform uses this tag to provide an easier to read version of the element's name. For example, if a headline tag has an ID of `id="Sidebar_Headline_1"`, you might use `friendlyName="Sidebar Headline 1"` as your friendly name tag.

4. Optional parameters that can be applied to dynamic images, widgets and forms to auto-generate placeholder graphics (also see example #1b):

   ```
   data-placeholder-width="200"
   data-placeholder-height="75"
   ```

5. Optional parameter to include multiple ion parameters to a dynamic element (also see example #7b):

   ```
   data-liveball="minwidth:10; maxwidth:294; minheight:10; maxheight:294; friend-
   lyname:Form Image Right; advancedui:yes; data-placeholder-width:114; data-
   placeholder-height:114"
   ```

   Note: The `data-liveball=""` parameter can be used to replace any of the existing ion specific parameters, **except** for:

   ```
   ID="uniqueID"
   runat="server"
   ```

6. Optional parameters that can be applied to non-dynamic elements:
   `data-liveball="styleonly"`
   `data-liveball="wrapper"`

   The data-liveball="`styleonly`" value allows for a non-dynamic element to be styled without requiring editable content (great for background elements and the <body> tag). This parameter value will allow for custom inline CSS (using css code or the editor UI) on elements in both the page and MVT editor within the ion console. (For an example of `styleonly`, see example #11.) If a element is tagged as style-only, a drop-down menu will appear within the page editor bar in the platform.

   Note: All dynamic elements automatically allow for custom CSS styles within the page and MVT editors in the ion console. Include the `data-liveball="styleonly"` parameter only to non-dynamic elements.

   The data-liveball="`wrapper`" value allows for an element to only be rendered if dynamic content is contained within the "wrapper" element (great for a content pod with rounded corners that contains dynamic elements. This pod will not appear if the user does not input dynamic content within it). (For an example of `styleonly`, see example #11.)

   Note: Both of these `data-liveball` parameter values require a `ID="uniqueID"` and `runat="server"` parameters.

7. Themes can be applied to `<img>` or `<link>` tags. To indicate that an element is controlled by a theme, rather than dynamically set by the marketing manager, simply include a

   relative path to the `src` or `href` parameter respectively. For example (also see example #8):

   ```
   <img src="../../Themes/Blue_Chrome/topbar.gif" alt="" id="TopBar" runat="server">
   ```

   The "../../Themes/" preface on the relative URL serves as the signal to the ion platform to swap out the sample theme you referred to in your master page template with the dynamic one chosen by the marketing manager.

   Note that themes can have a dramatic impact on the look-and-feel of the page by putting a CSS file inside the theme collection, so that a different .css is linked to depending on the theme selected by the marketing manager. We strongly recommend having your framework CSS be maintained at the theme level to support this.

8. Non-dynamic or theme-based images can be included within your master page. To do this, place the image file (.jpg, .gif, or .png) within the root of the main /Masters/ directory.

   ```
   /FrameworkID/Masters/
   ```

   In order for these type of images to appear, you must specify the full relative path to this file within your ion console. For example (also see example #8b):

NOTE: ion's full (`/Templates/ion/FrameworkID/Masters/`) relative path is needed when referencing any files (.js, .xml, .swf, etc...) with a `src` attribute within the root of the main /Masters/ directory.

9. Now you have to make a few minor changes that help ion dynamically take over the rendering of this master page.

   At the very top of the source of your page, add this line (also see example #9):

   ```
   <%@ Control Language="C#" Inherits="LiveBallPage" EnableViewState="false" %>
   ```

   This line must be inserted before your `<!DOCTYPE>` tag.

   Next, add a `runat="server"` parameter to your `<html>, <head>` and `<body>` tags. If you want to let marketing managers dynamically control the `<title>` value—which we highly recommend you do—you need to also add an `ID` parameter to `<head>` as well. For example (also see examples #9 & 10):

   ```
   <head id="Page_Title" runat="server">
   ```

   Within your `<head></head>` block, add the `<link>` to `common.css,` and `theme.css` files in the sample theme directory that you are using for your other resources and the common master css file (also see example #12):

   ```
   <link href="../common.css" rel="stylesheet" type="text/css" />
   <link href="../../Themes/gray/theme.css" rel="stylesheet" type="text/css" />
   ```

   If the master page is responsive, then include the following within your `<head></head>` as well.

   ```
   <link href="../commonResponsive.css" rel="stylesheet" type="text/css" />
   ```
   (Note: Include after the reference to common.css)

   ```
   <link href="../../Themes/gray/themeResponsive.css" rel="stylesheet" type="text/
   css" />
   ```
   (Note: Include after the reference to theme.css)

   If a form is included on your master page, then include the following within your `<head></head>` as well.

   ```
   <link href="../../Themes/gray/themeform.css" rel="stylesheet" type="text/css" /
   >
   ```

   Finally, rename your file extension from `.html` to `.ascx`.

That's it, you're done—you've successfully built a ion master page. You should still be able to view the template as a standalone HTML page.

A Special Note About Forms

The one thing you don't get to control in a master page is the inclusion of forms. That's handled by a different facility to give marketing managers dynamic control of what type of data they're collecting and what they'd like to do with it.

All you get to decide is whether or not a given master page that you create will support having a form plugged into it.

If the answer is "no"—which makes sense for splash-style or information-presentation master page formats—then you don't have to do anything special: just make sure you don't have any `<form>` elements in your page.

If the answer is "yes", you can include a placeholder image between the form's opening and closing tags in your page or include the optional data-placeholder parameters to the starting `<form>` tag (Example #13), and then the entire form will be substituted dynamically by a new form that is selected by the marketing manager when they build their creative. To support this dynamic form substitution, you just need to add two parameters to your `<form>` tag (note: only one form can be included in a master). As you've probably guessed:

```
ID="FormSocket"
runat="server"
```

Note: The ID for all dynamic forms must be "FormSocket".

You can also control the look and feel of your form by including a `themeform.css` file to your theme directory (see example #12). Again, each `themeform.css` file must have the same classes for each theme in your framework.

Within your `<head></head>` block, add a `<link>` to `themeform.css` file in the sample theme directory that you are using for your other resources.:

```
<link href="../../Themes/gray/themeform.css" rel="stylesheet" type="text/css" />
```

The following is a list of all possible classes that can be controlled in the `themeform.css` file (also see /SampleFW_v4-1/Themes/SampleThemeOne/themeForm.css).:

Class	Description
`.pf_form_table`	CSS class for form table
`.pf_field_row`	CSS class for field rows
`.pf_field_row_bad`	CSS class for bad field rows
`.pf_label_cell`	CSS class for label cells
`.pf_label_cell_bad`	CSS class for bad label cells
`.pf_label_text`	CSS class for label text
`.pf_label_text_bad`	CSS class for bad label text

Class	Description
.pf_field_cell	CSS class for field cells
.pf_field_cell_bad	CSS class for bad field cells
.pf_hint_cell	CSS class for hint cells
.pf_hint_text	CSS class for hint text
.pf_hint_text_bad	CSS class for bad hint text
.pf_hr_cell	CSS class for <hr> separator cell
.pf_hr	CSS class for <hr>
.pf_prefield_cell	CSS class for pre-field cells
.pf_prefield_text	CSS class for pre-field text
.pf_postfield_cell	CSS class for post-field cells
.pf_postfield_text	CSS class for post-field text
.pf_text	CSS class for regular <input> text
.pf_text_narrow	CSS class for narrow <input> text
.pf_text_wide	CSS class for wide <input> text
.pf_dropdown	CSS class for <select> drop-down
.pf_dropdown_narrow	CSS class for <select> drop-down narrow
.pf_dropdown_wide	CSS class for <select> drop-down wide
.pf_multiline	CSS class for <textarea> multiline text
.pf_multiline_narrow	CSS class for <textarea> multiline text narrow
.pf_multiline_wide	CSS class for <textarea> multiline text wide
.pf_listbox	CSS class for <select> listbox
.pf_listbox_narrow	CSS class for <select> listbox narrow

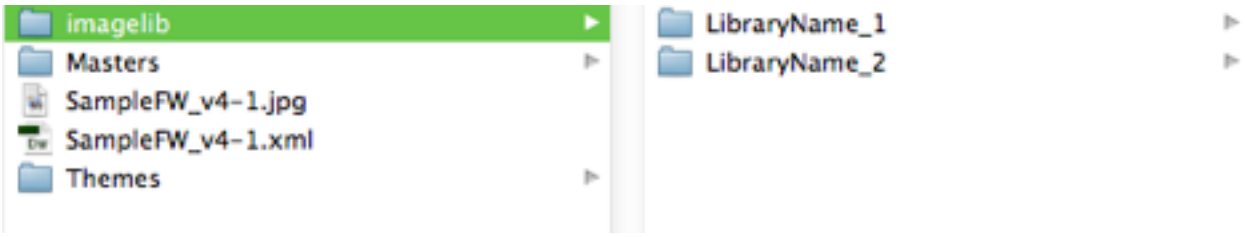
Class	Description
.pf_listbox_wide	CSS class for <select> listbox wide
.pf_checktable	CSS class for <table> of checkbox <input> options
.pf_radiotable	CSS class for <table> of radio <input> options
.pf_submit_cell	CSS class for submit button cell
.pf_submit_button	CSS class for submit button
.pf_submit_image	CSS class for submit image
.pf_upleft_corner	* CSS class for top left form hook
.pf_upright_corner	* CSS class for top right form hook
.pf_loleft_corner	* CSS class for bottom left form hook
.pf_loright_corner	* CSS class for bottom right form hook

* These classes can be used to add design style around your forms. For instance, rounded corners.

Image Libraries

You can optionally include framework specific image libraries within your framework files. These images would then be available to the marketing manager while publishing pages dynamically through the ion editor.

Within the root of your of the FrameworkID directory, include a directory for "imagelib" (/FrameworkID/imagelib/). Within this directory, you will need to include at least one subdirectories.



Notes: All of your images must be within a subdirectory in the main "imagelib" folder. Only .jpg, .gif and .png files are accepted. These image libraries are framework specific... they will only be available for use within dynamic image spots using the master page layouts of the framework.

Uploading into the ion platform

Save all your framework files in a .zip. It is very important that the root level .xml/.jpg files (i.e. SampleFW_v4-1.xml and SampleFW_v4-1.jpg) and /Theme/ and /Master/ directories are all present in the root of this .zip, named appropriately and maintain the same directory structure as described in this document on page 2. The zip file name is the FrameworkID (i.e. SampleFW_v4-1.zip).

Note: Some zip/compress softwares add an additional directory in the root of the .zip, which will prevent you from uploading your framework into the ion platform. To avoid this, zip the contents of your framework and rename the .zip accordingly (remember the .zip must have the same name as the root level .jpg and .xml).

To upload (and download) frameworks in the ion platform, select 'Frameworks' from the main navigation drop down menu 'Libraries'. Click “+upload”, find your framework .zip file and “save”.

You’re now ready to begin building creatives!

In-console framework management

Once a framework has been uploaded into the ion platform, the ability to manage/edit the framework files is available through your console.

Navigate to the framework library and click the icon to the left of the framework you would like to manage.

Editing the framework: Within each framework, you can edit the .css, .ascx, and .js files directly within your ion console. Navigate to the file you would like to edit, click on it, make your edits and save!

There is a “Save & Preview” option for .ascx files that allow for previewing (will render the first theme in the framework) of **saved** changes before leaving the file editor.

Notes: Any saved changes while editing the framework code will affect ALL creative using this file. Be careful!

Managing files: The ability to add/replace, preview and delete files within a framework.

To add/replace a file you will see a green pop-up saying ‘+ file’ when hovering your mouse over the framework folders. Click it and you will get a pop-up window to browse and locate the files. You can also drag and drop multiple files (Firefox newer version and Chrome) to the 'add feature'.

To delete a file, click on the trash icon next to the item you would like to delete.

Notes: If an image is in use you won't be able to delete it. Same applies for other files that are important to the framework, you won't be able to delete them. Such as .xml, .css, .ascx, etc…You also can't add or replace an .xml file within a framework.

Copy/Delete Masters and Themes: The ability to copy/create and delete masters and themes.

To copy a master or theme you will see a green pop-up saying ‘copy’ when hovering your mouse over the master and theme folders. Click it and you will get a pop-up window to name it.

To delete a master or theme, click on the trash icon next to the item you would like to delete.

Notes: If a master or theme is in use you won't be able to delete it.

Preview Masters: The ability to preview master page layouts (in read-only format).

To preview a master you will see a blue pop-up saying ‘preview’ when hovering your mouse over the master folders. Click it and you will get a new window displaying the master page layout.

The previewed master renders the first theme in your framework. To change the theme, you can do so manually within the url query string. For instance, change (in bold below):

/MasterPage.aspx?snapkey=snap&interact=y&dID=ion&fID=Dev_testing&mpID=form_right_xl&tID=**Brand_1**

to

/MasterPage.aspx?snapkey=snap&interact=y&dID=ion&fID=Dev_testing&mpID=form_right_xl&tID=**Brand_2**

The ability to toggle the ‘show advanced’ functionality is available within the ‘master preview’ bar at the top of the page. This will display the elements, within the master’s .ascx file, that are tagged with the `advancedUI="yes"` parameter (see page 5 above).

Import page from external URL

In the framework library, you now have the ability to import a master page template by entering the URL of any external page! This powerful feature will be further developed, but enables you to easily import an existing page into the ion platform.

Navigate to the framework library and click the icon to the left of the framework you would like to import the page to. Click on the red 'import master' button, name it and enter the URL of the page you want to import, confirm the assets that will be included in the upload and the page will be momentarily be available within your console.

Note: The import can take a while depending on how many files there are.

About imported master templates:

- After importing your master template, the content on your page will **not** be dynamic. Images, text and forms must be manually converted into dynamic elements before they can be used as such in the ion platform.
- Pages with more than one HTML form cannot be imported (including form tags that may be commented out).
- Imported master templates will not be automatically associated to any existing themes or the common.css file within the framework.
- Some code may be modified. For instance, form 'onsubmit' and body 'onload'. But, will be rewritten.

Note: It's recommended that the page you import is added to a dedicated framework for imported pages. Although can be imported to any existing framework.

What gets imported?

- Your page's content, including images, stylesheets and scripts that are referenced from within your page's domain, are imported. For example, if you import a page from "mydomain.com", resources from "www.mydomain.com" and "sub.mydomain.com" will be imported. Resources from "anotherotherdomain.com" will not be imported.
- the ion platform will import images referenced from your page's HTML and stylesheets, but images and other resources referenced from Javascript will not be imported.

Note: All of the external page assets (.css, .js, image, etc...) will reside within the new master directory created on import.

Additional Resources

Visit our support section to view FAQs and developer resources.

View the framework development training video.