# USING TAGLIBS AND PREFERENCES IN TEMPLATES

# STYLE ALL THE CONTENT

- Content Templates can be extremely powerful.
- Our current templates take content from a Structure and style it in a consistent way.
- We've used Lexicon CSS to create beautiful content displays like a banner and thumbnails.
- As we style more content throughout the site, we may need more advanced tools:
  - Taglibs
  - Repeatable fields from Structures
  - Passing settings between templates

# S.P.A.C.E. NEEDS A PLACE

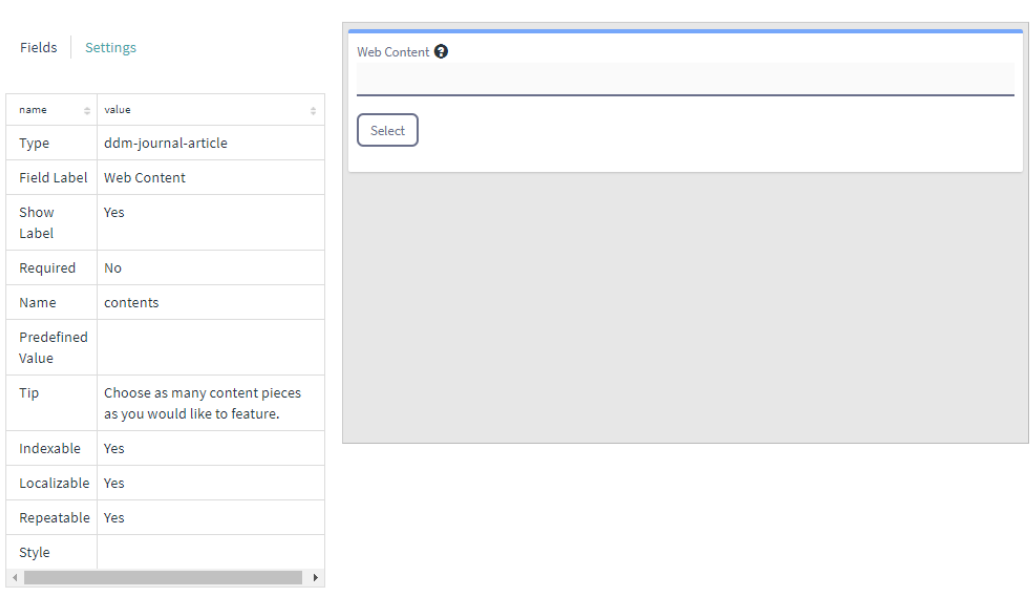
- The Content Team in S.P.A.C.E. needs a dedicated section to show upcoming events.
- Between incoming student orientations, tours, summer sessions, field trips, and career fairs, anyone could get lost in all that information.
- They'd like to organize the events so that:
  - It's easy to see the most recent events.
  - Important events are featured and stand out.
  - Events are created once, but restyled in different places.

# CREATING AN EVENTS PAGE

- They've decided to create a new *Events* page on the site.
- Events will be created through Web Content and displayed in a number of different styles.
- Our Content Team has already created a Structure for events, but needs to style it.
- Some events will be featured, or highlighted, on the *Events* page.
- Instead of creating multiple Structures for events (whether they're featured or not), the Content Team has come up with something a little different.

# THE COMPOSITE SOLUTION

- The Content Team has created an additional Structure, *Content List*.
- The *Content List* Structure contains:
  - A *ddm-journal-article* field called `contents`
- This field is different from what we saw before because it:
  - Allows the writer to choose a Web Content Article for the field
  - Can be repeated many times



# DEVELOPING A NEW TYPE OF TEMPLATE

- Using the structured content from the Content Team, we'll need to:
  - Develop a Template for the *Event* Structure
  - Develop a Template for *Content List* that embeds other Web Content Articles
  - Develop Templates to display the *Content List* in different ways

# EXERCISE: IMPORTING EVENTS

- The S.P.A.C.E. Content Team has already provided the Structures for us to build from.

1. **Open** the Menu.
2. **Go to** *Content→Web Content* in the Site Administration panel.
3. **Open** the *Options* menu.
4. **Choose** *Export/Import*.
5. **Click** on the *Import* tab.
6. **Choose** the `02-space-events.lar` from *exercises/front-end-developer-exercises/ 06-web-content-templates*.
7. **Click** *Continue→Import*.
8. **Close** the pop-up once it's successful.

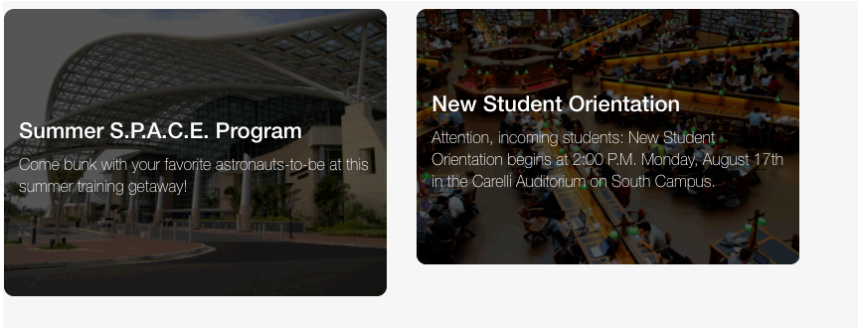
# STYLING EVENTS

- Our first task is to provide basic styling for the *Events*.
- We'll build a nice display with Lexicon CSS components, like *Card*, with the Structure fields.
- *Events* contain:
  - **Cover Image:** A beautiful image to display when listing or displaying the event
  - **Headline:** The name of the event
  - **Date:** When the event is happening
  - **Lead Text:** Brief information to display about the event
- We can style these into a nice card, displaying all the event details.

# EXERCISE: BASIC EVENT STYLING

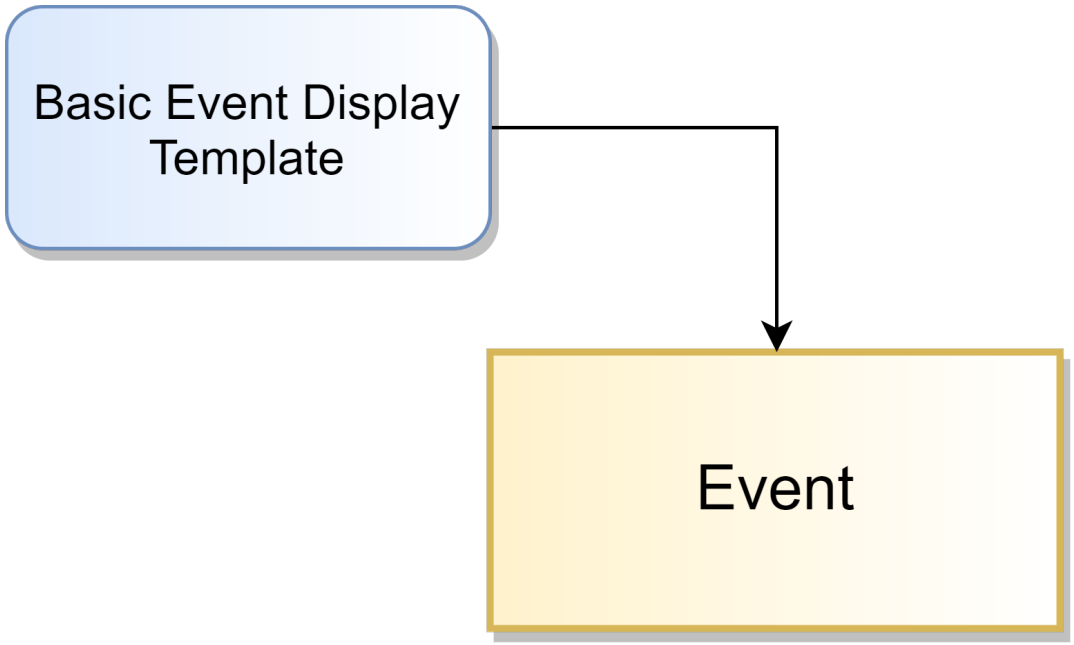
1. **Open** the *Options* menu in *Content→Web Content*.
2. **Choose** *Templates*.
3. **Click** on the *Event Display* template and scroll down.
4. **Click** *Choose File* under the editor.
5. **Choose** the *event-basic-display.ftl* found in *exercises/front-end-developer-exercises/06-web-content-templates/template-src*.
6. **Click** *Save*.

✓ Now we have our styled events!



# THE EVENT DISPLAY TEMPLATE

- Events will now be styled using the basic styling outlined in the *Event Display* template.



# FEATURING EVENTS

- Now that we have our basic events styled, any time we want to show an event, it will look clean and consistent.
- The Content Team would also like to feature some events.
- A *Content List* Structure allows for selecting Web Content Articles to feature.
- We'll need to write a Template that can:
  - Embed content from another Web Content Article
  - Style multiple fields from a list
- To embed Web Content Articles, we'll need to use some other Liferay features.

# EMBEDDING WEB CONTENT

- If you tried to use `getData()` on each Web Content Article, you'd only get:
  - The *Java class* that article uses
  - The *primary key* of the article
- If you're a Java developer, or have access to the `serviceLocator`, this might be useful.
- You could call the service for the Web Content Article (`JournalArticleServce`) and ask for its content.
- But what if we can display articles *without* needing to call more back-end services?

# USING TAGLIBS IN TEMPLATES

- *Taglibs* provide reusable components that simplify complex displays in Liferay.
- Normally, Java developers can use these in JSPs to make building application views easy.
- Templates have access to Taglibs using a simple syntax in FreeMarker:

```
<@taglib_name["tag-name"] attribute=value />
```

- There are a lot of taglibs available for displaying data and building complex UIs:
  - `liferay_ui`: some general components for displaying data
  - `liferay_frontend`: contains some Lexicon CSS components
  - `aui`: components for AlloyUI
- More information can be found in the appendix.

# DISPLAYING CONTENT WITH A TAG

- Taglibs are so useful that applications like the *Asset Publisher* use them for displaying Web Content and other assets.
- Can we use the same tags as the *Asset Publisher* for displaying Web Content?
- The taglib `liferay_ui` contains the tag `asset-display` that takes three attributes:
  - `className`: the *Java class* of the asset
  - `classPK`: the *primary key* of the asset
  - `template`: how to display the asset
- We already have most of this information from our Structure field, `contents`.

# DISPLAYING OUR ASSET LIST

- Using this tag, we can easily display the Web Content Article in FreeMarker:

```
<@liferay_ui["asset-display"]
    className=article.className
    classPK=getterUtil.getLong(article.classPK, 0)
    template="full_content"
/>
```

- The template called `full_content` just needs to render our Web Content Article as if it were being displayed on its own.
  - This way, the Content Template for the article gets used.
- We can use the tag `<@liferay_ui["asset-display"] \>` to display each article in our list.

# USING REPEATABLE FIELDS

- Our list of Web Content Articles is implemented as a *repeatable field* in the Structure.
- When content creators are building an article, they can click a button to add as many copies of the field as they want.
- We could have 1, 2, or 10 articles, depending on how many fields there are.
- To use a list of fields, we need to be able to:
  - Get the list
  - Iterate over the list
- This is actually pretty easy to do.

# GETTING THE LIST OF FIELDS

- When we want data from a field, we usually call its `getData()` method.
- Repeatable fields have *siblings*.
- Each *sibling* is a copy of the field.
- Each field would have a `getData()` method.
- In our case, each *sibling* would have the `className` and `classPK` for the article.
- Getting our list of content is pretty easy:

```
$contents.getSiblings()
```

- So how do we iterate over the list?

# LIST BY LIST

- We see list iteration in a number of our templates, using the directive `<#list>`.
- Listing items means we need to know the name of the *list* and each individual *item*:

```
<#list items as item>
$item.getData()
</#list>
```

- *items* is the list of variables, and *item* represents each item in the list.
- We can apply this to our articles:

```
<#list contents.getSiblings() as cur_contents>
$cur_contents.getData()
</#list>
```

- We now have enough information to create a template that can *iterate over a list* of Web Content Articles and *display each article*.

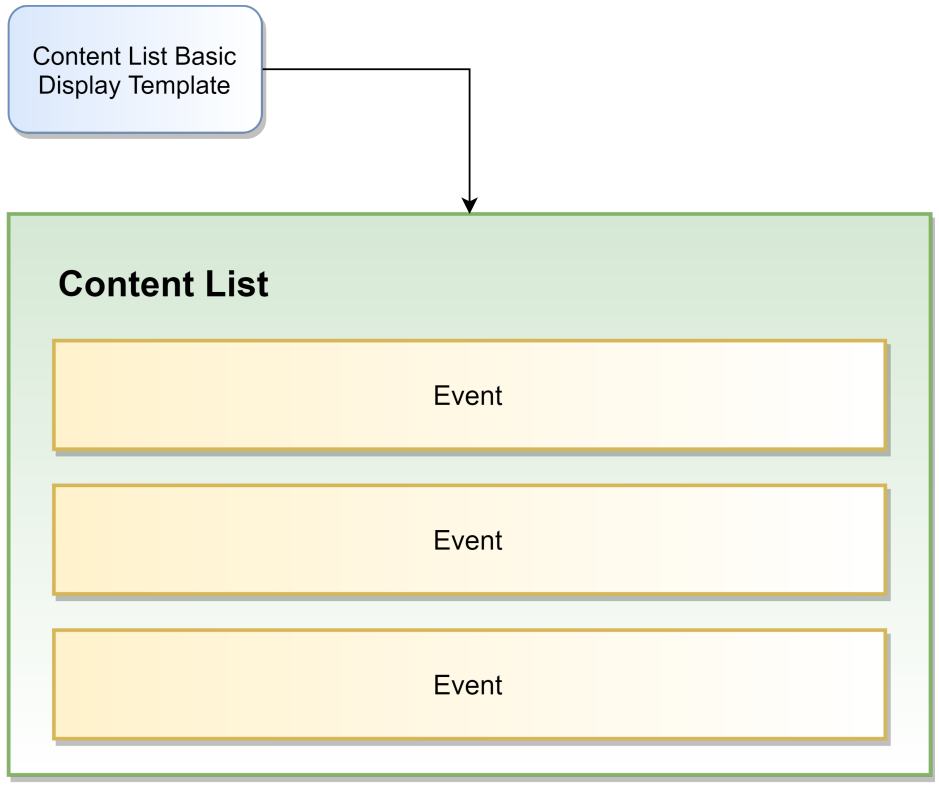
# EXERCISE: DISPLAY A CONTENT LIST

1. **Click** on the *Content List Display* template and scroll down to the editor.
2. **Click** *Choose File* under the editor.
3. **Choose** the *content-list-basic-display.ftl* found in *exercises/front-end-developer-exercises/06-web-content-templates/template-src*.
4. **Click** *Save*.

✓ Now we can show repeatable content.

# CONTENT LIST DISPLAY TEMPLATE

- With the *Content List* strucure, we can feature multiple Web Content Articles.
- We've set the basic styling in the *Content List Display* template.

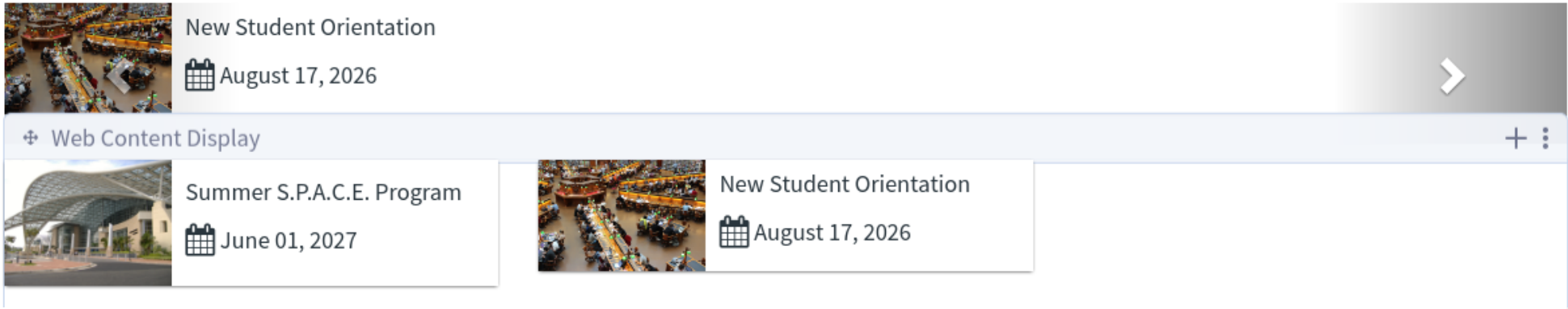


# BUILDING DIFFERENT DISPLAYS

- It's really useful to be able to reuse the same Web Content Articles and just display them differently.
- We'd like to be able to provide multiple displays for featured events:
  - Basic display
  - Thumbnail display
  - Carousel display
- We've just completed the basic display, and could create new templates to build out a thumbnail and carousel display.

# A CAROUSEL AND A THUMBNAIL

- What would it look like if we built carousel and thumbnail templates and displayed the Web Content Articles inside them?
- Not pretty.
- We need to change how the article displays in each component.



# SETTING VARIABLES ACROSS TEMPLATES

- The *Events Display* template is responsible for how the article is displayed.
- But it doesn't know anything about how the carousel or thumbnail display or what changes they need.
- We need a way to *tell* the events template *how* to change its display for each template.

# USING PREFERENCES TO SHARE SETTINGS

- All of our Web Content Articles are being displayed in the *Web Content Display* application.
- Each application has a way to store *preferences*, or settings, as variables.
- Setting *preferences* allows those settings to be used by any other template in the application.
- These preferences are available through an object in FreeMarker called `$freeMarkerPortletPreferences`.
- We can set and get different values using this object:

```
<#assign VOID = freeMarkerPortletPreferences.setValue("mySetting", "stuff") />
...
<#assign mySetting = freeMarkerPortletPreferences.getValue("mySetting") />
```

- **Note:** we use an empty variable called `$VOID` to run the method silently. We wouldn't want return values displayed on the page.

# A CAROUSEL, A LIST, AND THUMBNAILS...

- We can now design each of the *Web Content Templates* to set a setting we'll call `view`:
  - `thumbnailListView`: We'll set `view` to this when we're displaying thumbnails.
  - `carouselListView`: We'll set `view` to this when we're displaying a carousel.
- When we're displaying the content in full, we won't set the `view` setting.
- Now, let's build each template to tell the *Event Display* template which style we need.

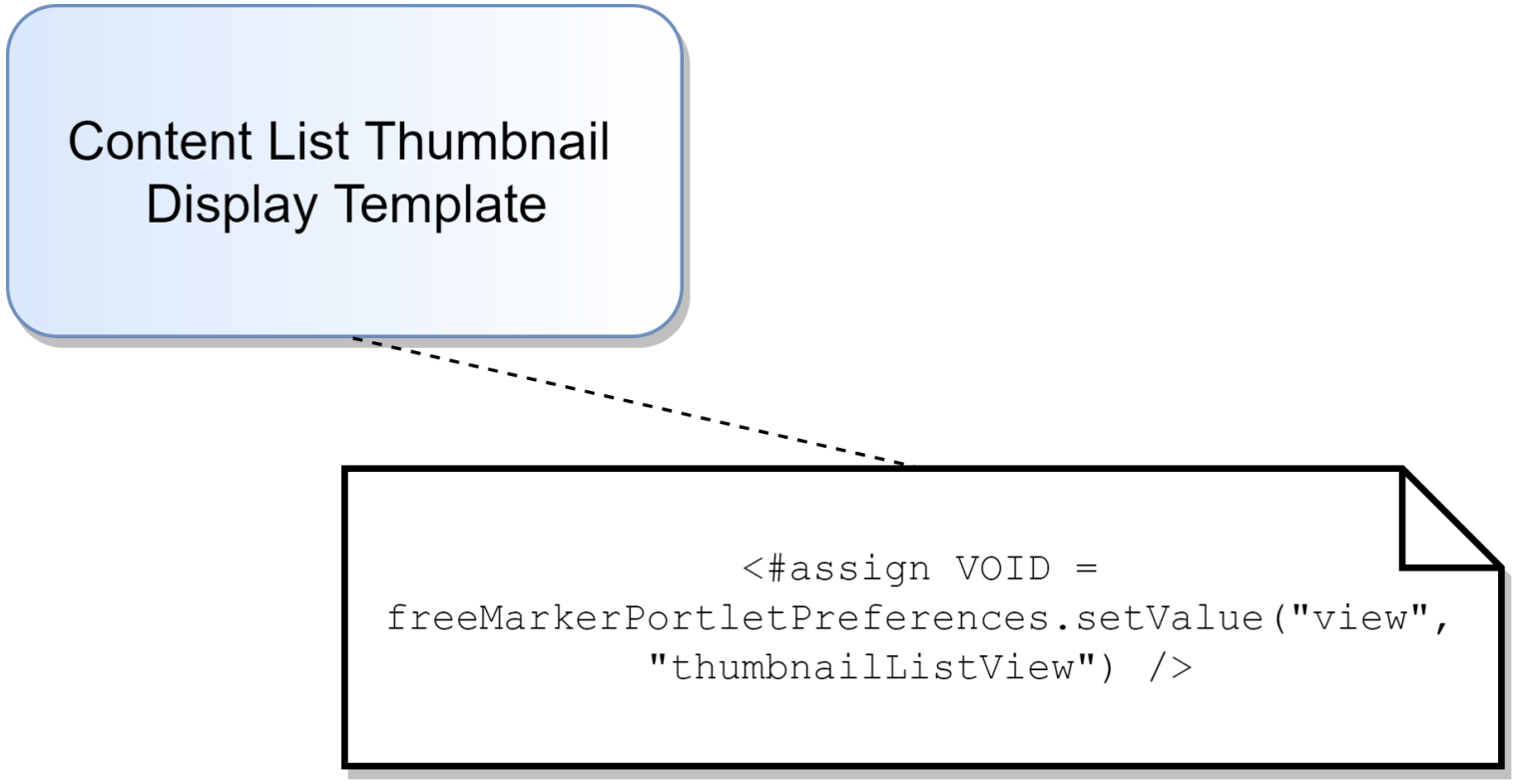
# EXERCISE: BUILDING THUMBNAILS

1. **Click** the *Content List Thumbnail Display* template and scroll down to the editor.
2. **Click** *Choose File* under the editor.
3. **Choose** the *content-list-thumbnail-display.ftl* found in *exercises/front-end-developer-exercises/06-web-content-templates/template-src*.
4. **Click** *Save*.

✓ Now we have a new display showing thumbnails of featured events!

# THUMBNAIL DISPLAY TEMPLATE

- We've set the `thumbnailListView` setting in the *Thumbnail Display* template.
- This setting will be referenced in logic we'll add to the *Event Display* template.



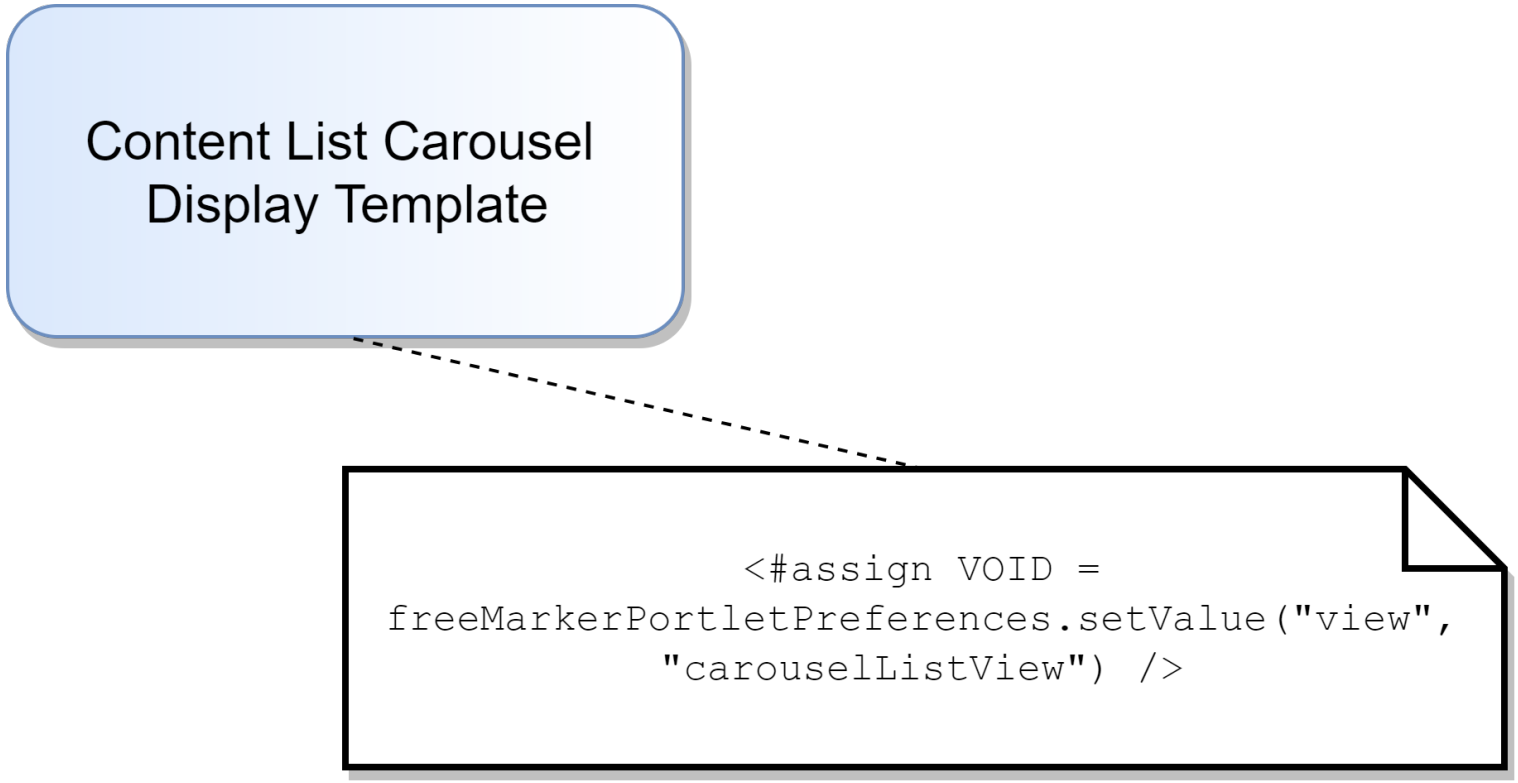
# EXERCISE: BUILDING A CAROUSEL

1. **Click** on the *Content List Carousel Display* template and scroll down to the editor.
2. **Click** *Choose File* under the editor.
3. **Choose** the *content-list-carousel-display.ftl* found in *exercises/front-end-developer-exercises/06-web-content-templates/template-src*.
4. **Click** *Save*.

✓ Now we have a new display showing a carousel of featured events!

# CAROUSEL DISPLAY TEMPLATE

- We've set the `carouselListView` setting in the *Carousel Display* template.
- This will be referenced in logic we'll add to the *Event Display* template.



# MAKING OUR EVENT DISPLAY SMART

- We're going to smarten up our *Event Display* template so that it responds to the value of the `view` setting.
- Using `<#if>` directives, we can build our event display using the right Lexicon CSS components for the containing template.
- If there is no `view` setting, then we'll just display the event the way we originally did.

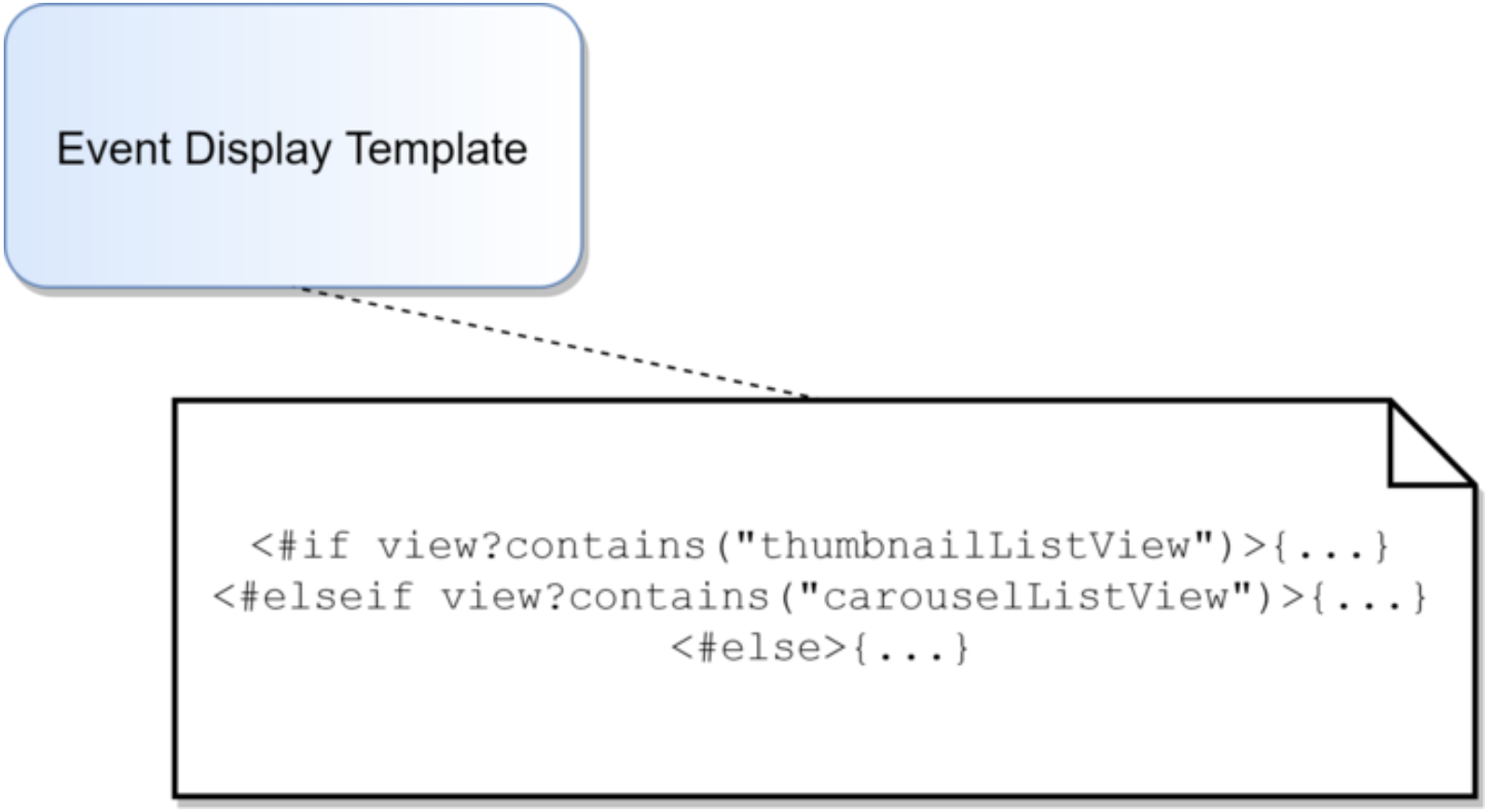
# EXERCISE: SHARING IS CARING

1. **Click** on the *Event Display* template and scroll down to the editor.
2. **Click** *Choose File* under the editor.
3. **Choose** the *event-advanced-display.ftl* found in *exercises/front-end-developer-exercises/06-web-content-templates/template-src*.
4. **Click** *Save*.

✓ Now the events will display differently in each list type!

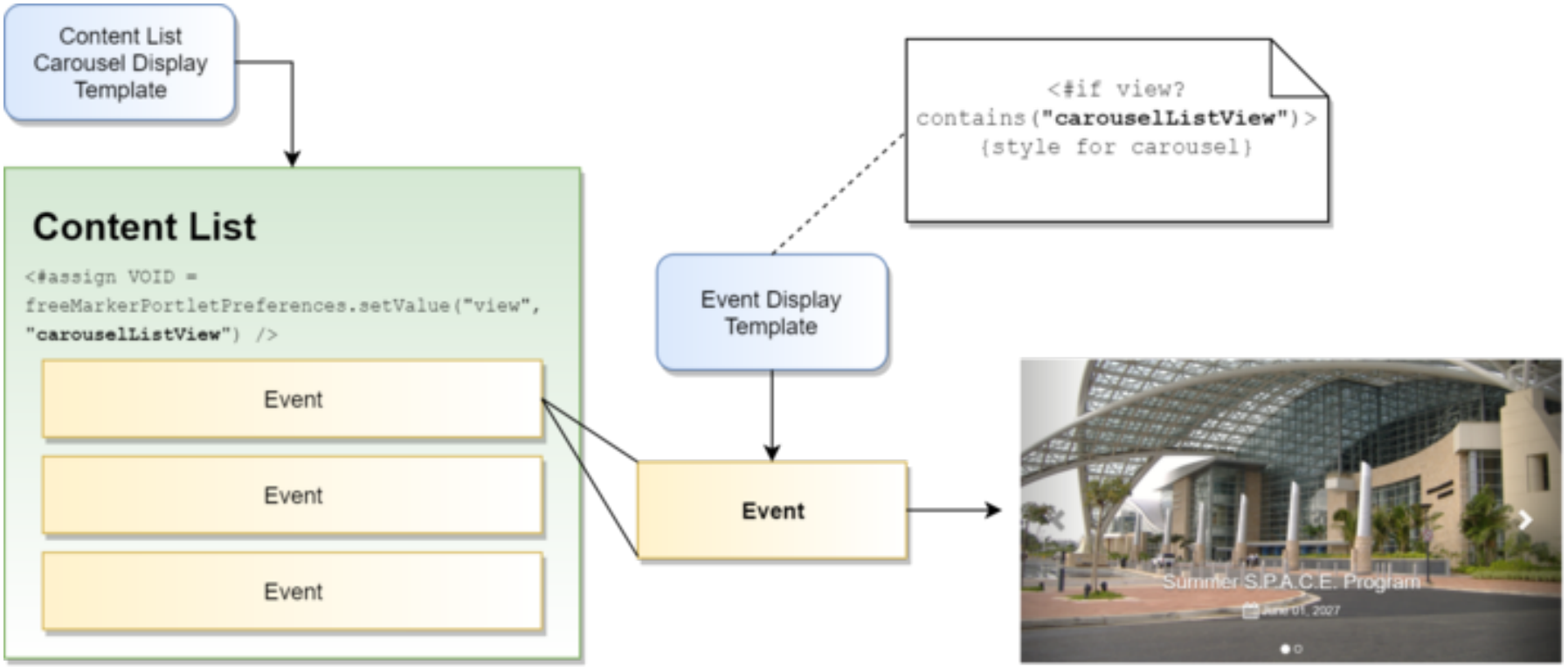
# ADVANCED EVENT DISPLAY TEMPLATE

- In our updated *Event Display* template, we've added logic to allow the template to respond to the `view` preference set in the Content List templates.



# TEMPLATES IN ACTION

- With the logic we've added, the *Event Display* template will respond to whichever template the Content List structure is using.



# DISPLAYING LOTS OF CONTENT

- Now that our displays are smart, respond to different environments, and use the same type of Web Content (*Event*), let's import some events.
- Our Content Team has provided us with events that are featured on the new *Events* page.
- Let's update our site to display these events using our new templates.

# EXERCISE: IMPORTING EVENTS

1. **Go to** *Menu→Site Administration→Publishing*.
2. **Click** on *Import*.
3. **Click** *Add* at the bottom right.
4. **Choose** the `SPACE_Events_Page_Complete.lar` from *exercises/front-end-developer-exercises/ 06-web-content-templates*.
5. **Click** *Continue→Import*.

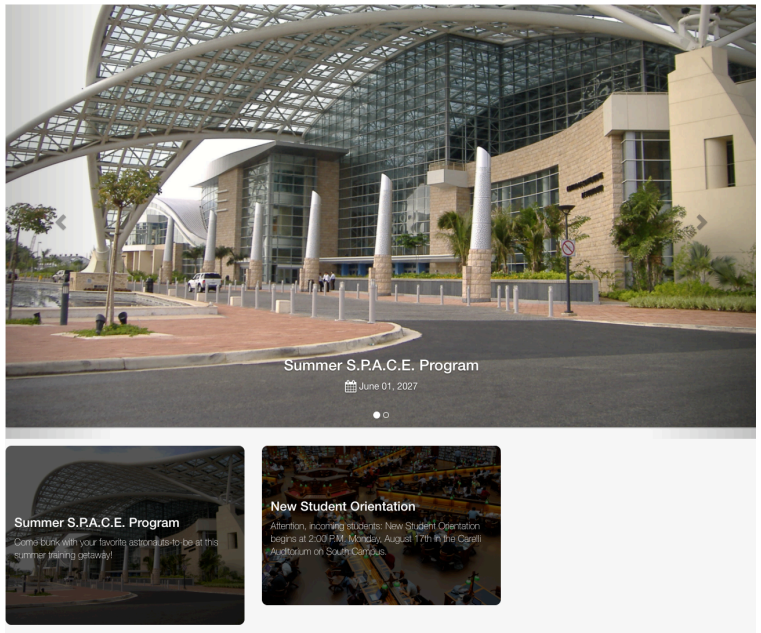
✓ Now we have content on display with our templates!

# EXERCISE: BRAVE NEW TEMPLATES

1. **Open** the *Menu*.
2. **Click** *Go to Site* in Site Administration.
3. **Click** *Events* in the navigation.

✓ Now we can see our new events displays!

- **Note**: Templates may look different depending on the browser.



# EXERCISE: APPLYING THE THEME

- When we imported our events, it reverted the theme selection.
- Let's go ahead and re-apply the theme.

1. **Go to** *Menu→Site Administration→Navigation*.
2. **Open** the *Options* menu next to *Public Pages*.
3. **Click** *Configure*.
4. **Click** *Change Current Theme*.
5. **Select** the *Space Program Theme*
6. **Click** *Save*.

✓ Now our theme is applied!

Notes: