All About Plug-ins for ArchivesSpace

Laney McGlohon

Lora Woodford

(special guest Christine Di Bella)

Agenda

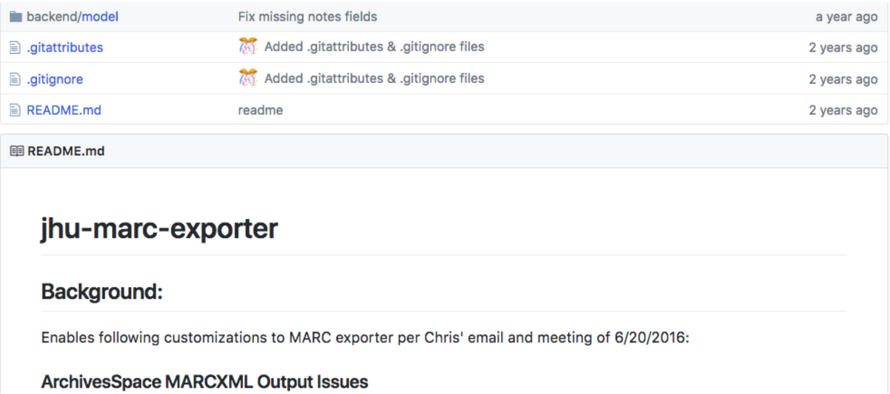
- What is an ArchivesSpace plug-in?
  - Overview
  - When - and when not - to use one
  - Examples and finding plug-ins
- What if I can’t use plug-ins?
- Project management and plug-ins
- Steps to create and use plug-ins
- Getting help with developing plug-ins and finding a wider audience

What is an ArchivesSpace plug-in?

ArchivesSpace plug-ins provide a mechanism to customize ArchivesSpace by overriding or extending functions without changing the core codebase. As they are self-contained, they also permit the ready sharing of packages of customization between ArchivesSpace instances.

When should you use a plug-in?

- To add **institution-specific** extended functionality, style, branding, and customizations
- Examples:
  - Customizing exporters
  - Setting required fields
  - Auto-generating DOIDs, accession numbers, etc.
  - Modifying labels
  - Customizing staff or public display



backend/model	Fix missing notes fields	a year ago
.gitattributes	Added .gitattributes & .gitignore files	2 years ago
.gitignore	Added .gitattributes & .gitignore files	2 years ago
README.md	readme	2 years ago

README.md

jhu-marc-exporter

Background:

Enables following customizations to MARC exporter per Chris' email and meeting of 6/20/2016:

ArchivesSpace MARCXML Output Issues

When shouldn’t you use a plug-in (in a perfect world)

- Obvious bugs
  - Bugs should be reported to the program and fixes incorporated into the core code
  - You may still wish to expedite the fix by implementing a plug-in in the interim, and may even wish to submit a pull request back to the core code!
- One-off data needs
- When you can’t commit to a maintenance plan

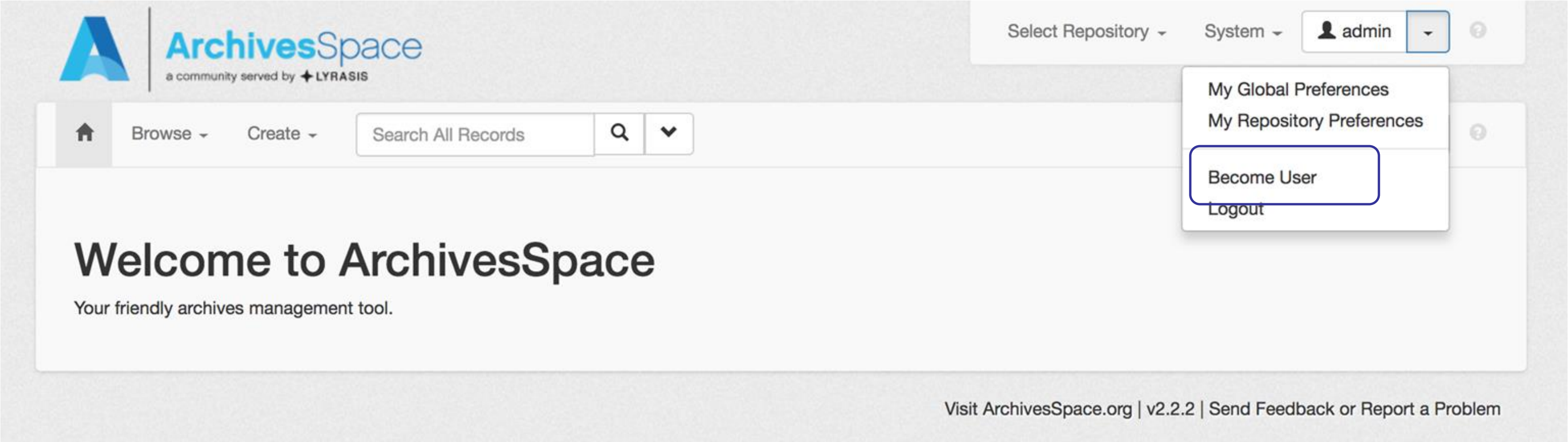
Maintenance considerations

- New releases may impact the efficacy (or even the need for) your plug-in
- Pay attention to release notes and always confirm your plug-ins are still working as intended before upgrading
- Example:
  - JHU MARC exporter initially included bug fix for AR-1189
  - JHU PUI plug-in (not surprisingly) broken post-2.1

Some simple plug-in examples

- Customizing field and option labels
- Moving branding image from right to left
- Customizing MARC exporter

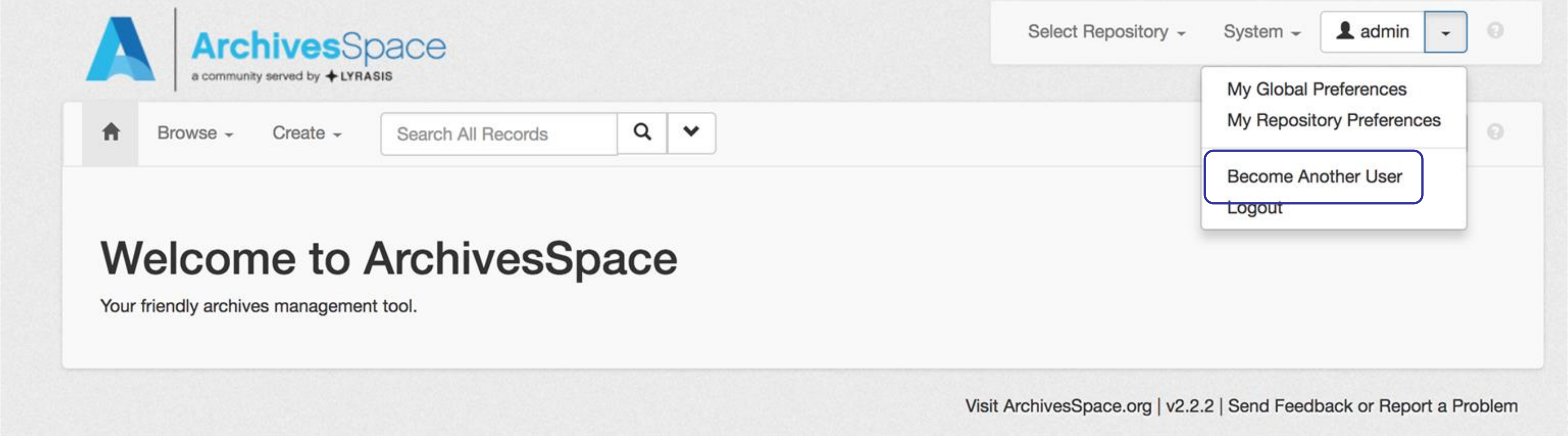
Customizing field and option labels



Welcome to ArchivesSpace

Your friendly archives management tool.

Visit ArchivesSpace.org | v2.2.2 | Send Feedback or Report a Problem



Welcome to ArchivesSpace

Your friendly archives management tool.

Visit ArchivesSpace.org | v2.2.2 | Send Feedback or Report a Problem

Customizing field and option labels

- To override some part of a locale file for the staff interface, you can just add the following structure to the local plug-in:

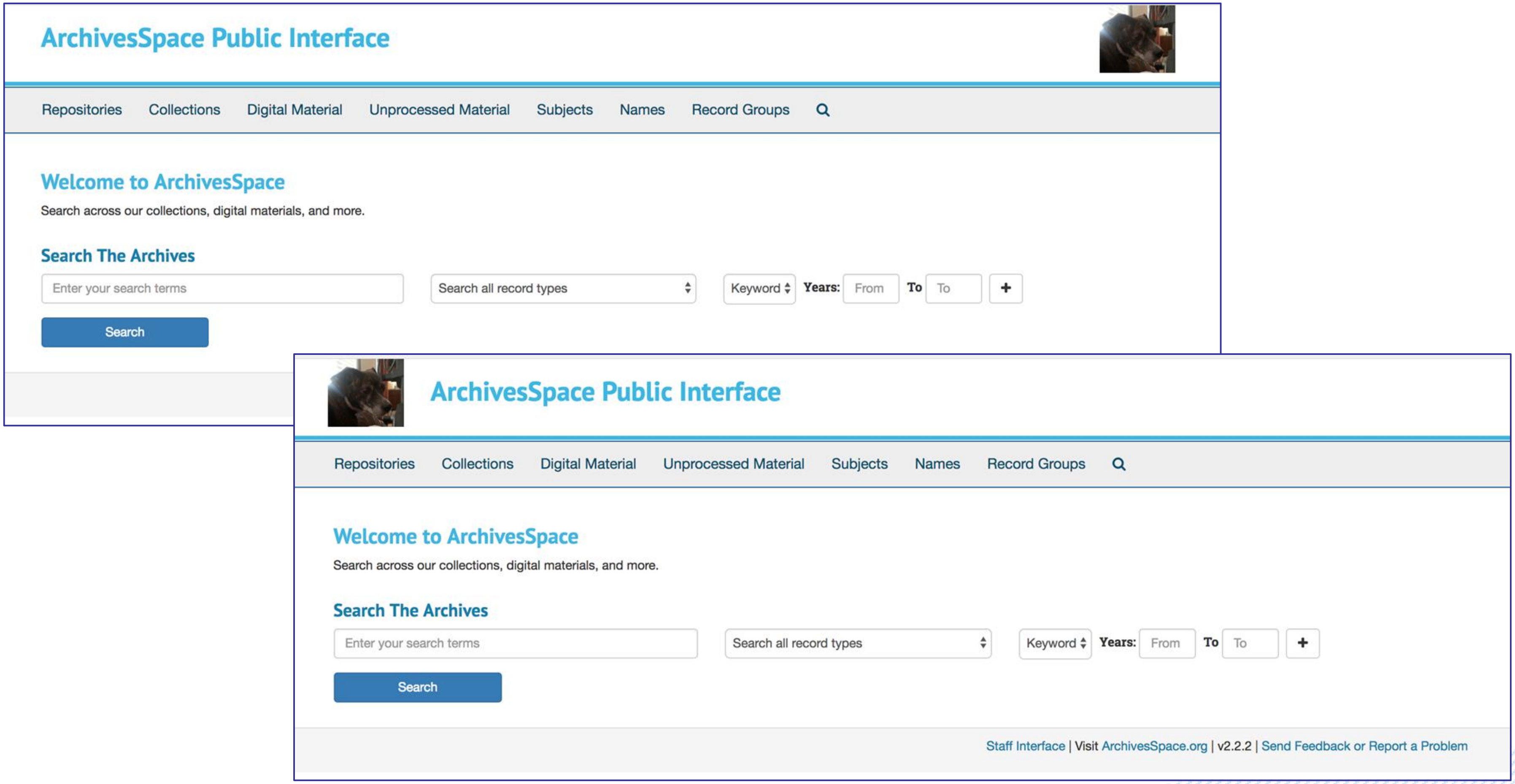
```
plugins/local/frontend/locales/en.yml
```

- For example, to change the text in the become-user pull-down entry, put this in the en.yml file mentioned above:

```
en:
  navbar:
    become-user: Become Another User
```

- Restart ArchivesSpace

Moving branding image from right to left



ArchivesSpace Public Interface

Welcome to ArchivesSpace

Search across our collections, digital materials, and more.

Search The Archives

ArchivesSpace Public Interface

Repositories Collections Digital Material Unprocessed Material Subjects Names Record Groups

Welcome to ArchivesSpace

Search across our collections, digital materials, and more.

Search The Archives

Enter your search terms | Search all record types | Keyword | Years: From | To To | + | Search

Staff Interface | Visit ArchivesSpace.org | v2.2.2 | Send Feedback or Report a Problem

ArchivesSpace
a community served by LYRASIS

Moving branding image from right to left

The placement of the branding image is handled by the **`public/app/views/shared/_header.html.erb`** file, so in order to change the position from right to left, the file needs to be overridden in a plug-in.

The contents of the core code **`public/app/views/shared/_header.html.erb`** is:

```
    <section  id="header">
     <div class="row">
       <div class="col-sm-9 h1">
         <% unless request.original_fullpath == '/' %>
           <a title="<%=t('brand.title_link_text') %>"
              href="<%= AppConfig[:public_proxy_url] %>">
         <% end %>
A        <%= t('brand.title') %>
         <% unless request.original_fullpath == '/' %>
           </a>
         <% end %>
       </div>
       <div class="col-sm-3 hidden-xs">
         <img class="logo" src="<%= asset_url(AppConfig[:pui_branding_img]) %>" alt="" />
B      </div>
     </div>
    </section>
```

Moving branding image from right to left

In the **`plugins/local/public/views`** directory, add a new directory called shared and create a file called “_header.html.erb” there.

Add this snippet to the **`plugins/local/public/views/shared/_header.html.erb`** file:

```
     <section  id="header">
       <div class="row">
         <div class="col-sm-2 hidden-xs">
B          <img class="logo" src="<%= asset_url(AppConfig[:pui_branding_img]) %>" alt="" />
         </div>
         <div class="col-sm-9 h1">
           <% unless request.original_fullpath == '/' %>
             <a title="<%=t('brand.title_link_text') %>"
                href="<%= AppConfig[:public_proxy_url] %>">
           <% end %>
A             <%= t('brand.title') %>
           <% unless request.original_fullpath == '/' %>
             </a>
           <% end %>
         </div>
       </div>
     </section>
```

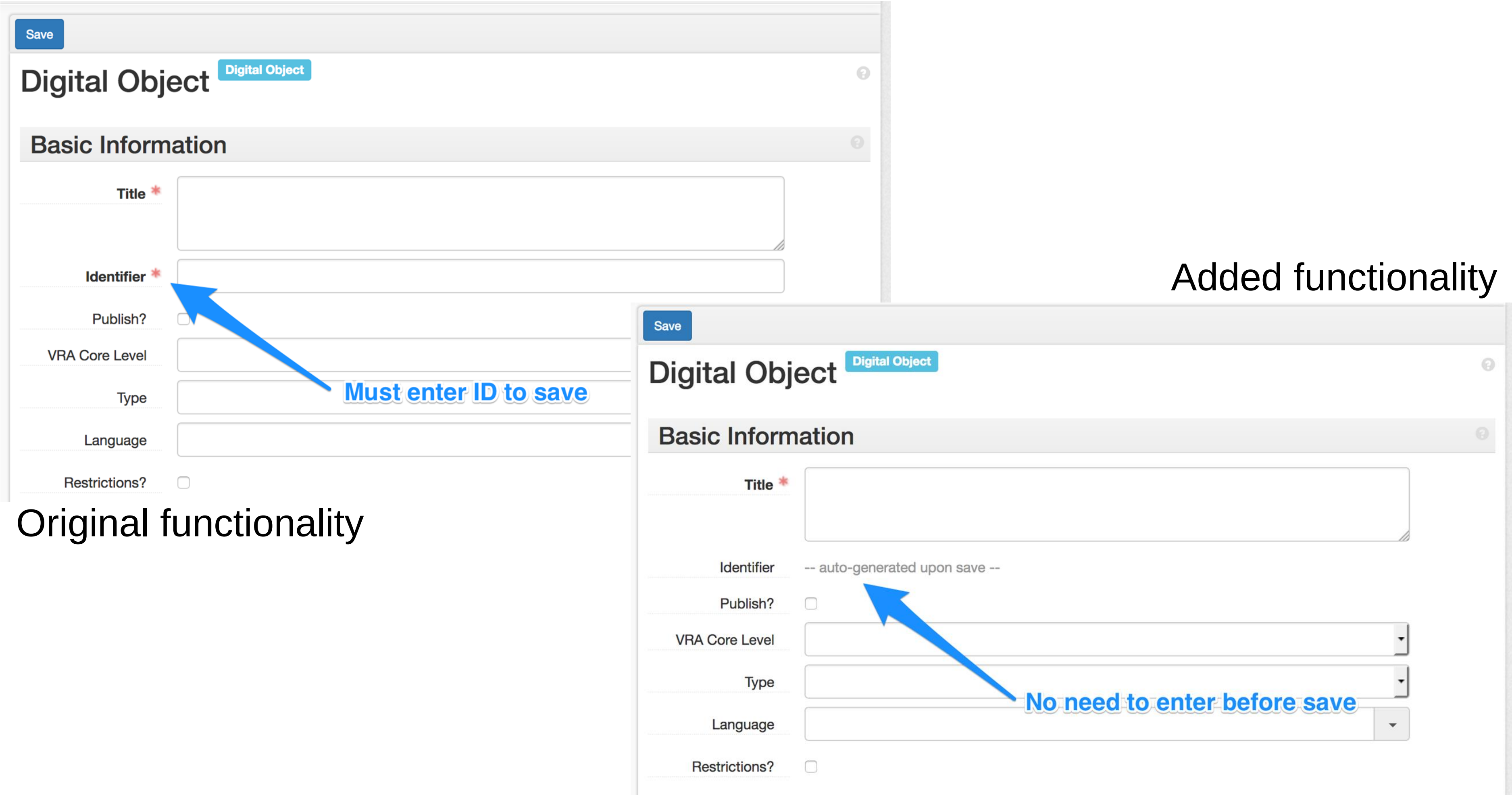
Moving branding image from right to left

So what did this change do?

It swapped the order of the two <div> tags in the erb file. By putting the <div> that handles the branding image before the <div> that handles the text that displays, the columns where these <div> tags are rendered are rearranged.

Don’t forget to restart ArchivesSpace to be able to see the change!

Autogenerating Digital Object IDs



Original functionality

Save

Digital Object

Basic Information

Title *

Identifier *

Publish? □

VRA Core Level

Type

Language

Restrictions? □

Must enter ID to save

Added functionality

Save

Digital Object

Basic Information

Title *

Identifier -- auto-generated upon save --

Publish? □

VRA Core Level

Type

Language

Restrictions? □

No need to enter before save

Autogenerating Digital Object IDs

- ArchivesSpace's Digital Object records require an identifier, defined as:
  - *A unique identifier for the digital object as a whole. May be an ARK, HANDLE, a URI, or **any string that uniquely identifies the digital object**. The field needs to be completed for a valid METS record to be exported.*
- What if you have nothing obvious to use to populate that field? Can we autogenerate a random hash to fill this field?
  - Yes!
- ArchivesSpace already uses the **`SecureRandom.hex`** Ruby method to do exactly this elsewhere in the application
- Create a plug-in that modifies:
  - **`backend/model/digital_object.rb`**
    - Call **`SecureRandom.hex`** on initial save.
  - **`frontend/views/digital_objects/_form_container.html.erb`**
    - Display text informing user that field will be autogenerated on save
  - **`schemas/digital_object.rb`**
    - Set hex pattern and do not require field for initial save

Autogenerating Digital Object IDs

- Go to https://github.com/lorawoodford/autogenerate-doid
- Navigate to **`archivesspace/plugins`** and either git clone or download and unzip **`autogenerate-doid`**
- Add ‘custom-marc-exporter’ to **`config/config.rb`** under **`AppConfig[:plugins]`**
- Restart ArchivesSpace

Finding plug-ins

- If you have a need and it’s not met in the application, chances are someone else has had it too - and may have created a plug-in for it
  - Look at https://github.com/archivesspace/archivesspace/tree/master/plugins and https://github.com/archivesspace-labs
  - Search github for “archivesspace”
    - Lots of cool development going on by many community members

What if I can’t use plug-ins?

If you don’t host your own ArchivesSpace or if your IT support is skeptical or overwhelmed, what then?

- Talk to your IT folks/host
  - Build trust/confidence with small steps
  - Try to work with them to use a plug-in that does something small or is only needed one time
- Submit individual feature requests to build what you want into the application
  - https://archivesspace.atlassian.net/wiki/spaces/ADC/pages/19202060/How+to+Request+a+New+Feature

What if I can’t use plug-ins?

- Build community support for what you want in the application
- Build community partnerships to convert a plug-in that does what you want to a standard distribution plug-in (the way the LCNAF plug-in works) or core code

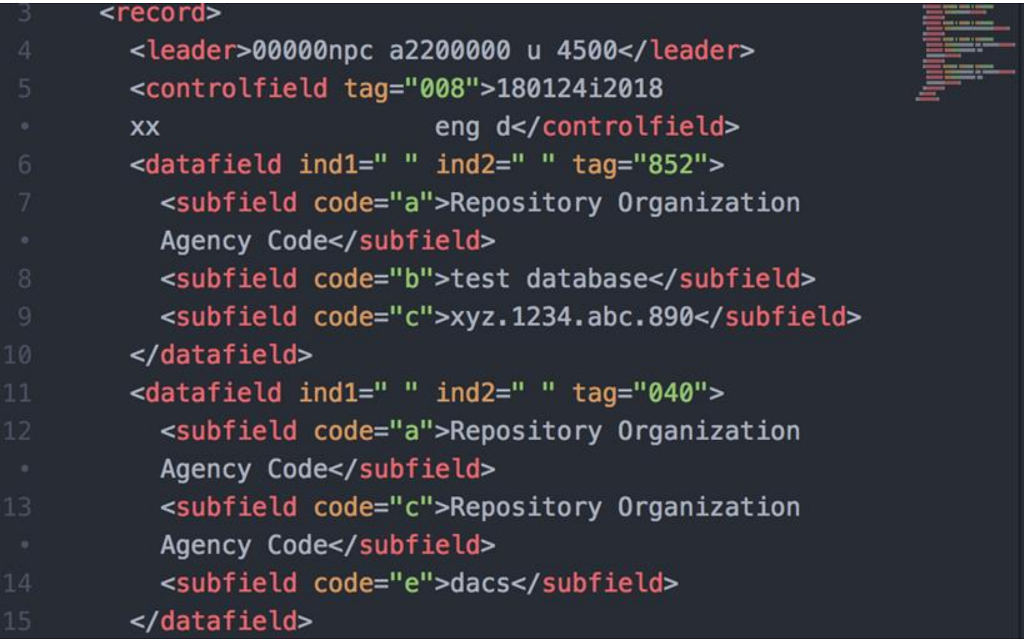
ALSO

- The project management principles we outline next apply to any kind of project, and will come in handy no matter your specific situation with respect to plug-ins

Project management and plug-ins

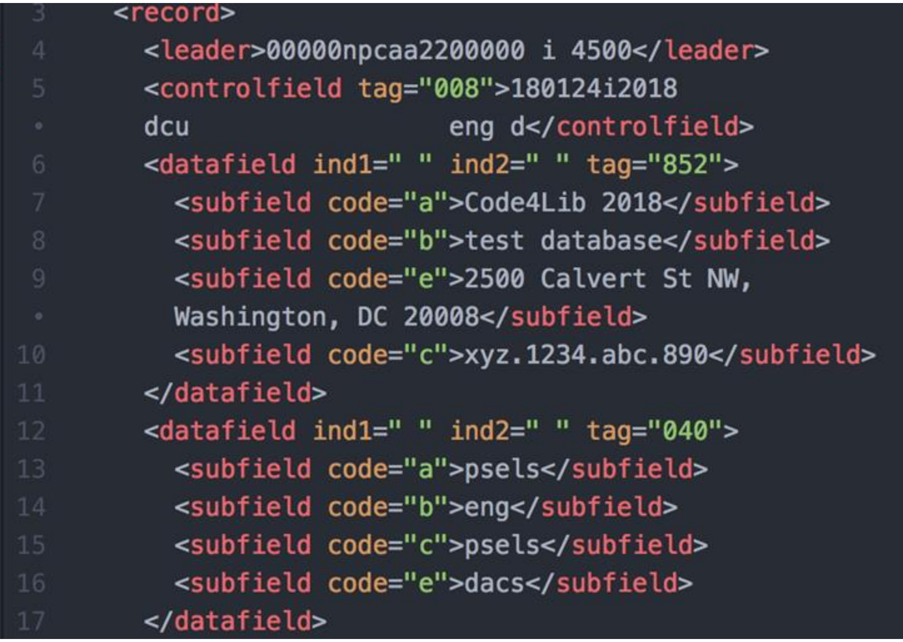
- Project management: the key to any successful project
  - Identify the need
  - Put together a written plan of work
    - What happens in the application now?
    - What do you want to happen/what should it look like?
    - What are the ramifications?
      - Think about unintended consequences
    - Who’s going to do the work and who’s going to test it?
  - Work with stakeholders to build and test the plug-in
    - Involve the wider ArchivesSpace community when useful

Example: Customizing the MARC exporter



```
<record>
  <leader>00000npc a2200000 u 4500</leader>
  <controlfield tag="008">180124i2018
  xx                  eng d</controlfield>
  <datafield ind1=" " ind2=" " tag="852">
    <subfield code="a">Repository Organization
    Agency Code</subfield>
    <subfield code="b">test database</subfield>
    <subfield code="c">xyz.1234.abc.890</subfield>
  </datafield>
  <datafield ind1=" " ind2=" " tag="040">
    <subfield code="a">Repository Organization
    Agency Code</subfield>
    <subfield code="c">Repository Organization
    Agency Code</subfield>
    <subfield code="e">dacs</subfield>
  </datafield>
```

Default exporter



```
<record>
  <leader>00000npcaa2200000 i 4500</leader>
  <controlfield tag="008">180124i2018
  dcu                 eng d</controlfield>
  <datafield ind1=" " ind2=" " tag="852">
    <subfield code="a">Code4Lib 2018</subfield>
    <subfield code="b">test database</subfield>
    <subfield code="e">2500 Calvert St NW,
    Washington, DC 20008</subfield>
    <subfield code="c">xyz.1234.abc.890</subfield>
  </datafield>
  <datafield ind1=" " ind2=" " tag="040">
    <subfield code="a">psels</subfield>
    <subfield code="b">eng</subfield>
    <subfield code="c">psels</subfield>
    <subfield code="e">dacs</subfield>
  </datafield>
```

Overridden exporter

Customizing the MARC exporter

- Exporters packaged with the application are necessarily non-specific in nature
- Exporters can be customized by *overriding* the defaults set in **`backend/app/exporters/models`**
- Create a plugin with custom settings with the following two files:
  - **`backend/model/custom-marc21-overrides.rb`**
  - **`backend/model/utils-marc-overrides.rb`**
- Copy the existing relevant exporter model from the core code and use it as your guide as you make overrides
  - Delete out any block you're not changing
  - Leave in and alter any blocks you are changing
  - Pro-tip: Comments are your friends.
- You can see what has been changed in the MARC exporter we're using here in the plug-in's README.md

Customizing the MARC exporter

- Go to https://github.com/lorawoodford/custom-marc-exporter
- Navigate to **`archivesspace/plugins`** and either git clone or download and unzip **`custom-marc-exporter`**
- Add ‘custom-marc-exporter’ to **`config/config.rb`** under **`AppConfig[:plugins]`**
- Restart ArchivesSpace

Steps to create a plug-in

- Identify what change is required. Should the implementation be extended or overridden?
- Determine where implementation is in the code
- To override behavior, rather than extend it, match the path to the file that contains the behavior to be overridden.

NOTE: The name layout_head.html.erb is special:
anything you put in a file under

```
[plugin_name]/frontend/views/layout_head.html.erb
```

or

```
[plugin_name]/public/views/layout_head.html.erb
```

will be inserted at the top of every page delivered by ArchivesSpace.

Plug-in directory structure

The directory structure within a plug-in is similar to the structure of the core application. The following shows the supported plug-in structure. Files contained in these directories can be used to override or extend the behavior of the core application.

```
local ............... Enabled by default
backend ............. Database and API
  controllers ......... backend endpoints
  model ............... database mapping models
  converters .......... classes for importing data
  job_runners ......... classes for defining background jobs
  plugin_init.rb ...... if present, loaded when the backend first starts
frontend ............ Staff Interface
  assets .............. static assets (such as images, javascript) in the staff interface
  controllers ......... controllers for the staff interface
  locales ............. locale translations for the staff interface
  views ............... templates for the staff interface
  plugin_init.rb ...... if present, loaded when the staff interface first starts
public .............. Public User Interface
  assets .............. static assets (such as images, javascript) in the public interface
  controllers ......... controllers for the public interface
  locales ............. locale translations for the public interface
  views ............... templates for the public interface
  plugin_init.rb ...... if present, loaded when the public interface first starts
migrations .......... Database migrations
schemas ............. JSONModel schema definitions
search_definitions.rb Advanced search fields
```

How to enable the plug-in

- Plug-ins are enabled by placing them in the ArchivesSpace installation plugins directory and referencing them in the ArchivesSpace configuration, **`common/config/config.rb`**.
  - For example: **`AppConfig[:plugins] = ['local', 'my_plugin']`**
- Note that the order that the plug-ins are listed in the :plugins configuration option determines the order in which they are loaded by the application. Be mindful of how plug-ins “play” together.
- Make sure that you uncomment the line with **`AppConfig[:plugins] = ['local', 'my_plugin’]`**

Getting help

- The ArchivesSpace community wants to help make your plug-in dreams come true
  - Core Committers
  - Users Group listserv and Google Group
- Slack channels, especially Archivists Working with Archival Data (shoes-untied.slack.com)
- Wider GitHub community

Getting a wider audience

- Think about whether your plug-in could be helpful beyond your own institution (most can!)
- If it can
  - Promote it via the listservs
  - Consider a write-up for a monthly update or blog post
  - Bring it up in an open call
  - Work with the community to make the case for it to be part of the standard distribution of ArchivesSpace or core code

Developer (and plug-in afficianado) resources

- https://github.com/archivesspace/archivesspace/blob/master/plugins/PLUGINS_README.md
- https://archivesspace.atlassian.net/wiki/spaces/ADC/pages/17137734/Plugins+and+Scripts
- http://campuspress.yale.edu/yalearchivesspace/category/what-archivesspace-does/
- http://libraryblogs.is.ed.ac.uk/librarylabs/tag/archivesspace/
- http://archival-integration.blogspot.com/2015/07/archivesspace-donor-details-plugin.html
- https://guides.nyu.edu/archivesspace/development
- https://blogs.library.duke.edu/bitstreams/2016/09/21/archivesspace-api-fun/
- https://blogs.harvard.edu/archivaldescription/2017/01/26/spreadsheet_to_ead_to_as/
- https://rubyexample.com/user/djpillen
- https://library.osu.edu/blogs/it/category/archivesspace/
- https://www.youtube.com/watch?v=hWP430Q5EWM
- Turning a plug-in into core code: https://archivesspace.atlassian.net/wiki/spaces/ADC/pages/349995159/Turning+an+ArchivesSpace+Plugin+into+Core+Code

Thank you!

Any questions?

Contact Information:

Laney McGlohon – laney.mcglohon@lyrasis.org
Lora Woodford – lora.woodford@lyrasis.org
Christine Di Bella – christine.dibella@lyrasis.org