Islandora Object Typification

We've had a hard time pinning down exactly what it means to be "in Islandora". It's not a pressing concern in the sense that this renders the software unusable, but it certainly doesn't add any coherency or "product-ness" to the software. [We still think it's useful to maintain a distinction between content in Drupal that is "not islandora" and the content in drupal that *is*. However, *it has been stated repeatedly that all islandora configuration is optional piecemeal and was designed so that it can be applied to any Drupal content. -RL]* If we can sort this out once and for all, we can solve a perception problem while also paving the way for further improvements (like a wizard to Islandora-ize stuff for you). So far we have considered many options*[, all of which have been argued against at some point… ;) -RL]*

- The presence of a particular field
    - field_model and field_member_of for nodes
    - field_use and field_media_of for media
    - field_external_uri for taxonomy terms (Although not 'core' Islandora, Controlled Access Terms, which uses field_authority_link for taxonomy terms, has become Islandora de facto.)
- Whether or not it's sent to Fedora
- Whether or not it has an RDF representation
- Whether or not we do "Islandora-y things" to a content type, media type, or vocabulary (derivatives are the big one here)
    - Given a content model, what datastreams/media should the object have
    - What derivatives should get generated - for example, for books pdf and OCR (these cannot always be generated at the media level)
    - How the object get viewed (viewers) - Display Hints!
    - How the metadata form should look like? (not sure if this should be per content model, or should be driven by other factors such as collection policy etc).
    - Ability to create a bag or an AIP of an intellectual/conceptual object
    - In 7.x
        - What metadata gets displayed in the details tab (via metadata display, which is tied to content models)
    - Reporting
        - How many intellectual/conceptual objects (i.e books) are in a given collection?
        - How many intellectual/conceptual objects have we successfully created AIPs and synced to external storage?
    - Audits
        - We need to be able to say who and when an Islandora Object was modified

We've also theorized on work we could do to solve this problem, such as

- Giving each object a boolean (I am a repository item)
- Giving each content type, media type, and vocabulary a boolean (I am a repository content type)

Notes & Questions - Nat

- Node -> Media: Currently there is a proposal to switch the membership lookup from media -> node to node -> media. Some suggested node -> media is more Drupaly and the earlier performance considerations does not apply any longer. Switching to node -> media may simplify the viewer setup.

  However, it is not clear how this will impact the representation in Fedora 5.x and specially Fedora 6.x. This may assist with oxford common file layout (OCFL). This maybe an opportunity to think more about how Islandora can model to support OCFL.

- Multiple Files in One Media: It seems that is easier to have 1 node -> 1 media -> N files. This may require extending default media types and/or creating custom media types. If so, we should decide this as the preferred approach.

  Paged Content is the difficult case here. Even currently, paged content does not support page level descriptive metadata in the same way it is supported in 7.x.

  However, we do know that we need much more metadata at the file level such as:
  - Checksum
  - File Format
  - Mime Type
  - Last Edited On
  - Last Edited By
  - Audit Trail etc

- Model/Collection Driven UX/Form: In Islandora 8, we don't have model information until the user selects the model. Even if they selected an image, it does not restrict the user from adding a video. Asking the user to specify a model will enable us to set up the metadata profile as well as possibly guide the user with respect to media they need to upload.

- Select Metadata Profile/Forms based on Criteria - We need a way to show custom metadata profiles based on certain criteria, possibly including model. In 7.x forms are associated with content models, because they also drive other UX events such as what required files to upload.

- Ability to Index and Surface Any Type of Structured Data (XML, Json)/Decoupling/Indexing Files: 7.x allowed for any xml datastream to be indexed easily. This allowed for any data to be ingested and index. Many of the community contributed solution packs were developed due to this flexibility. A user can easily develop a custom content model to build viewers as well. In 8.x this process is much more complexed due to its content model/architecture.

- Intellectual/Conceptual Object vs Digital Object: For many reporting purposes and even functional operations (i.e AIP generation), we are concerned about the intellectual object and not the digital objects composing the intellectual object.

Notes and philosophical ramblings - Rosie

On one hand there is the Drupal worldview of nodes (pages on a website / blog posts / widgets you want to sell), media (reusable files to snazzy up those blog posts), and taxonomy terms (website categories/tags for finding ‘kitchen’ widgets or ‘overly personal’ blog posts).

On the other hand there is the worldview of ‘digital assets’ (files that need some degree of meticulous management/preservation done in an auditable/transparent way that is future-friendly), “descriptions of those assets” (i.e. metadata about those files or the objects / entities depicted/instantiated/represented by those assets”, and ‘controlled vocabularies’ (“Things, ideas, or people that those objects are related to”)

Goals of a Drupal person: SEO-friendly website. No dead links or out-of-date pages. Content of a ‘factually up-to-date’ or ‘explicitly authored’ variety (serving, say, a library website with an ‘hours’ page where it’s the (de facto) institutionally authoritative source or a blog with a ‘posts by me’ page). Media may function as content delivery (thinking of embedding youtube, or serving important PDFs) but media are not generally considered to be “the website content” and access _to_ them is more important than information _about_ them. Also, it’s very common for a website to be pretty bad at managing its uploaded files. Having them carefully organized in meaningful folders is not necessary for a functional website (most i’ve seen just sites/default/files and chill). This, and php upload forms, make it easy for lots of people who you sorta-trust to add (multi)media content to a website (just click upload it via the wysiwyg by clicking the [...lol…] Browse Repository button). And website people think of “putting content up on the website” much more than “taking content down from the website” (which, often, is just removing-the-links-to-it or making-it-not-displayed on that page) - it’s not a full lifecycle management philosophy. Thus ‘automatic garbage collection’ of files getting cleaned up when no longer ‘in use’ by a web page makes sense.

Goals of a Repository person: I will never lose the content of my valuable digital assets. They will never be accidentally deleted by the system. My sysadmin will never accidentally detach or overwrite the filesystem where they are stored because “the repository” is seen fundamentally

as “a pairing between a drupal webserver and one or more file storage backends” - every filesystem added to the repository is explicitly and consciously considered as a management project and its maintenance is integral to my repository system (also, I have budgeted for this and/or have a policy saying that budget will be made for this). The filesystem will be backed up in a way that makes sense for files that rarely-if-ever change their content. Few people have access to the repository file system. Bitrot will be caught because I implement periodic file system checks, checksum checking, etc. I can implement my institution’s preservation policy, which involves managing the mimetypes / software needed for my files and doing file conversions as needed. I can ask questions about my files, such as “tell me about all my repository PDFs”? I want to know that i can trust the accuracy of the answer to reflect what’s actually in my repository.

Example: The institution changed its previous policy of “convert WordPerfect files to MS Word” to “Convert WordPerfect files to PDF”. After configuring islandora to do the new derivative behaviour, I need to go back and potentially convert the previously added files. So, things that my repository should be able to tell me: I need to find all original files that are word perfect files. For those, I need to know which ones do not already have a PDF derivative. Also, useful information would be when that PDF derivative was made, and with what software. That is, I want to be able to query technical metadata about files that is aware of the files’ relationships to each other. Potentially useful is the relationship to “the object” they both are connected to, but for preservation, to each other.

I will be able to find my content because it is described by metadata. It is important to me that i will also always have access to this metadata, and there will be a “link” between the metadata and the binary (especially if they’re stored in different places). The metadata must be editable but is edited by few, trusted people. At the same time, a workflow where less-trusted users add metadata that will be “vetted” is also desired.

I want to be able to see when something that was “supposed to happen” to one of my objects, didn’t, because there are automated processes in the background that modify (add) content, which are part of my preservation system, and inevitably sometimes don’t do what they’re supposed to. I can also audit what successfully happened to my (file) objects. I want it to be obvious when “nothing is configured to happen” so that i don’t confuse it with an error case.

[This is not served, in my clearly biased drupal user’s opinion, by using the same media types for both things-that-should-undergo-preservation and things-that-should-not-undergo-preservation. If I, Rosie, came across an empty field named something that i expected to be populated-by-a-derivative-process, i would read that situation as:

- Maybe there’s something in the karaf logs? Ok, but first:
- What’s the media type? (not always evident).
- Is this media type configured to have derivatives sent out? If not, was this a media type that was intended to be used for repo content at all? (If it was supposed to be configured

for derivatives, do that and figure out why it wasn't at the same time as people were expecting that to happen. What training or documentation or abstraction or constraints or communication lines need to be improved?)
- If the intended derivative was configured for this media type, check the configuration of where the derivative was supposed to end up. Did it go somewhere else?
- If the derivative process was configured to have happened for things of this media type into this field, (and it configured at time of ingest?) then is the particular object in hand compliant with the shibboleths (e.g. media use is populated correctly, media_of has a value, the element referred to in media_of is of the correct type and has the correct field values populated)
    - If not, what form was used? How do i need to change it so that the user making a repository object, who doesn't know what derivatives means, doesn't accidentally forget one of these required-for-derivatives shibboleths, while also being able to use this content type for non-repo content?
- What filesystem is the actual file in for this exact item? Where is the file, and was it successfully uploaded? Was the file accessible to the derivative microservice? If the file could not be successfully added to the repo because the user erroneously added it to a non-repo storage layer while expecting to make a storage object, … did they not have permission? Did the upload not work? what forms/training/documentation/abstraction layer/constraints/communication lines need to be improved?
- If the user intended it to ***not*** be a repository object but someone misread that it was supposed to be? Did they after-the-fact attach it to a repo object? what forms/training/documentation/abstraction layer/constraints/communication lines need to be improved?

If I were not Rosie, or didn't have time, I would read this situation as:

- The magic thing didn't happen. File a ticket with the devs.

I'm not saying this hypothetically. This is literally what happened on the RDM project, by each of the three non-dev project members, who had either:
- *written the user documentation for Islandora*
- *been a pro-level drupal wizard*
- *been named Rosie.*

This software, as it is right now, is super not-useable by someone who is not an islandora committer because there are too many conditional behaviours, and those conditions are opaque. Making them "part of" the types as they are visible to a content creator (by which I mean the bundles, not the value of a field called 'type') would go a long way to making the system legible.

Other "goals" of a repository:

- I can enter, view, and search metadata in various semantically meaningful fields.
- This can be exported as semantic data in standardized formats that are meaningful for repository objects. Semantic markup that is presented “about a repository object” without extraneous fields that “you just have to know to ignore” (e.g. created date and authors that are contradictory because the markup makes it semantically indistinguishable, to someone with no external data, what’s “about the node” vs “about the object”, and “about the file” and “about the metadata that describes the file”
- I can organize my content into collections
- I can make “exhibits” of my collections, which involves extra highlighting or annotating or featuring of repository items while not requiring me to duplicate those repository items, and that the exhibit is (can be) structurally and interactively distinct s.t. I can remove an exhibit without removing the repository content that it references.
- I can create reports of what’s in my repository
- I can create private areas within the repository, where the files and metadata are not publicly available, and cannot be viewed through any direct links that may have slipped out (i.e. even if i have an un-guessable hashed link, access to that information can be controlled.)

8.1.1 is great if you want to set up a site that does one OR the other. But if you want to have both, in a way that makes it clear which ones are which, we are doing a disservice by trying to combine them into “fewer types”.

What’s (in) a content type/solution pack? What shouldn’t be?

- A default metadata form that you don’t have to use but can provide an out of the box example.
- Out of the box, a way to create a thing “of this type” with very few “gotchas” or having to go back and do other things first.
- Configurations on what file type(s) are allowed, and other validation criteria
- Configurations on what derivatives should be done, and triggers that will cause them to happen automatically. - note, this may care deeply about the filetype (e.g. pdf to pdfa) and/or the function (e.g. this newspaper tiff needs OCR; this astronomy image does not)
- A way to distinguish an “object of X type” e.g. the hasModel: islandora:sp_pdf (mostly a file-processing-behaviour-based content type) or hasModel: ir:citationCModel (mostly a function/context processing content type) each are categories. And every object had to fit into a category. And (almost all) expected behaviours could be expected to be fairly consistent for this category, rather than based on the-values-of-various-options-that-are-available-within-this-category.
- Based on the category, try to display a viewer to display multimedia content. (this works if you have the right kinds of derivatives or files available- not always guaranteed.)
- Based on “content model” and relationships between this object and others, display a way to navigate to related objects. Note that in our 7x practice, the relationships were not

semantic enough to entail much about how the objects were related. For example, isMemberOf could be for Compound objects, collections, or issues of a publication. These not only had different displays, they had different management interfaces with different affordances as they needed. Collections “are things you add new content to. They constrain the types of content that can be added. They let you share/move things between collections. And they let you set permissions on existing and/or new children.” Compound objects are “do you want to give this object a parent?”

- Display extra stuff on the page based on (metadata-centric) content type. E.g. IR content (citation and theses) have a citation formatter, but there’s no reason other content models (e.g. book) can’t display a citation formatter.
- Shortcut workflows (that sometimes obscure the real nature of things) e.g. in 7x to add a newspaper issue, you MUST upload a PDF of the whole issue, which is then automatically split (back?) into images, those images run through OCR if you want, and then saved as individual page objects. If you have a stack of Tiffs, you have to make a PDF just so that the system can unpack the PDF - not useful.

What do we need:

- Derivatives to happen, configurable by any/combo of:
    - File extension
    - A more generic “what kind of thing is this file” such as Media Bundle
    - For this individual file, do you want to… (e.g. 7x newspaper issue: “Do not extract text? Extract text? Run OCR?” with an explanation of when/why you’d want to use them)
    - ...I guess, the context in which this file is being created. What is its parent’s “type”? What collection is it in? These are more opaque, and I don't think should be the default for most simple cases.
- Files need upload forms. But it’d be nice to (be able to) “tie” the node form and the media form together in some way… noting that while most objects have media, not all do, and ‘metadata first then add files later’ is a reasonable workflow to have and should not be impossible. This was done in 7x by making the file upload form field optional. This led to, in some cases, objects that got started-then-forgot-about-then-coulnt’-find-again so if there’s an “expectation of files” that you want to be able to make (i.e. for reporting/auditing), an “in progress” state where they’re not required would be good for reporting and management (getting back to the drafts).
- Viewers! A way to display one or more files in viewers. For any viewer to work, it will need a certain file type present in a predictable location. For the same viewer (e.g. a pdf viewer) the file may not be always in the same “place” in terms of media use - a PDF might be an original file in one case and an access derivative in another, or might be different in relation to the node that you want it displayed on (e.g. maybe you have multiple “versions” that aren't revisions, e.g. archival version and redacted version).
- Things that are representations of the same “kind of object” might have different file types available. For example, a “Book” type might have children-and-pages, AND/OR have a full-book PDF. Which is displayed? Is there a link to the other?. A “3D model”

type might have an open-source-file-type that can be displayed in an open source viewer, and/or a proprietary file type that can only be downloaded, but a resized thumbnail image that can be displayed instead.
- Meta: https://xkcd.com/2348/
- I guess i'm saying that judicious use of coding (interfaces, plugins, tools-for-managing-configuration) would be better than a brittle system that tries too hard to re-use components of Drupal that were designed for a different purpose.

Things I am Typing - Dan A

The way I see it, compared to I7, I8 is more like a suite of tools, and then it ships with a content type and config specifically designed to make use of all those tools. Other content types can currently *kind of* opt in to the framework by using a combination of those tools. But then sometimes things don't break or work as expected because your content type isn't doing something something unspecified behaviour that would make the suite of tools work correctly.

The point is, like, is it even Islandora's responsibility to define what this is? Rather, it feels like the responsibility to kinda just provide the tools perhaps?

Example of this, like:

To use the derivative framework, you need a content type that you can attach media to. The attached media need to specify a derivative term and a destination term. Also there's a sort of undefined expectation that there is a 1:1 relationship between said terms and media associated with a specific piece of content.

As long as you explain those restrictions, and someone implements them, you can use the derivative generation tool in the suite of tools.

Same kind of stuff applies to things like RDF, some of the context pieces, submodules, etc.

I think what I'd like to see is for some of the actual tools baked into I8 to be more explicitly broken out so that there's a clear distinction of what each piece is doing; e.g., the indexer piece is broken out, the derivative generation piece is broken out, context reactions, all that stuff. I know this adds some complexity to the structure of Islandora as a Drupal module, but I think maybe part of the disconnect currently is that there's a suite of features built into the base Islandora Drupal module that aren't necessarily related to each other, and it's like … do I have to implement all of those components for something to be Islandora? Certain ones? It's quite a bit different than how other suite-of-tools Drupal modules are structured - I'm thinking specifically of https://www.drupal.org/project/bibcite as an example of this.

If they're clearly delineated and broken out, it's left to the implementer to pick and choose what they want, and define for themselves, 'hey, X and Y represent my repository, i.e., X and Y are the things I'm making derivatives for, exposing RDF with, exposing in my OAI endpoint, sending to my backend, etc., and for me, the use of those features represent my repository.' On top of that, if someone needs an actual field that can be used to determine if something is 'in your repository', they can totally implement that without severing functionality with the suite of tools.

Also in terms of a wizard this I think makes a lot of sense too. I have X content type, I want Y and Z and Q components from the suite of tools to apply to it, run through the wizard, check off those tools, bam, now X content type has been 'Islandora-ized' with the selected tools.

Opinions that probably don't help that much - Bryan B.
Getting into the philosophical angle of "What is an Islandora object" and "what even is Islandora" that Rosie brought up at the 8/26 tech call, my opinion is that Islandora 8 *could* be used as monolithic system that demands that you do everything the Islandora Way, and in fact a lot of users might find that helpful, but that is not the way the the system has been built; the current architecture is super modular to allow a "do what you want" ethos, and that's a good thing that doesn't necessarily preclude giving users an "Islandora Way" set of default configs that give you everything all at once. But considering so much work has gone into making everything loosely coupled, I think we can run with that and we don't necessarily have to make the distinction of things being an "Islandora Object" or not. With so many people using Islandora 8 in such different ways, defining an "Islandora Object" seems like a fool's errand because whatever method you decide, some people won't want to do that but will still want to use Islandora 8 and think of their stuff as "Islandora Objects". It also makes all the components tightly coupled to a arbitrary shibboleth that may or may not even be relevant to their use case.

With this in mind, I think of Islandora 8 as more of an ecosystem for doing digital asset management in Drupal, and all the modules being tools in a toolbox for Drupal users to assist with DAM work. From that perspective, I don't think we need to define things so much as an "Islandora Object" than as an "Asset Content Type" for nodes that are describing files, or an "Authority Content Type" for describing metadata entities without files like authors or organizations (I7 entity cmodels). We could do that from an Islandora admin control panel to eliminate the need for potential users to alter their existing data model to have specific fields or booleans just to get it to work with Islandora tools, and the distinction is just an Islandora config that signals to the rest of the components how the object should be treated. Some folks who only want to use one or two Islandora modules in their Drupal site might be confused or even irritated that their custom content type gets branded as an "Islandora Object", but they might not be offended by "Asset Content Type" because that's generic but still semantically valid for them.

I'll admit that this take doesn't answer the question this doc is asking, and is in fact asking a completely different question, but I still think its a question worth pondering. We have two demographics here: the first is the Islandora 7 community which sees Islandora as a monolithic framework (which it was more of in 7), and the second is other Drupal users who might have

uses for the tools coming out of the Islandora community. My humble opinion is that we should be building I8 so that it serves both by architecting it in a way that supports the first group (Drupal users looking for handy contrib modules), but can be deployed and preconfigured in a way that supports the second group (7x users who want a unified view of what Islandora is). In this context, “what is an Islandora object” becomes less about any specific field on a content type and more about what that content type was created to do (store metadata about a file), and we should let that inform our decision. With that in mind, if this is a distinction that is only relevant for Islandora tools, then I think the data should be stored as an Islandora module settings configuration. That's what I would expect from other D8 modules, and if I found they required me to add fields and change my data model in order to use them, I would immediately be skeptical unless they had a very good reason to do so.

Notes from the Sept 8 2020 zoom conversation - RL

- The concept of a “Digital Object”. Have a repository s.t. we can say “yes, *this* (thing) is in the repository”, while also maintaining the concept that “this other thing is just web content in drupal, it’s not in the repository”. (“modeling clarity”)(“able to understand a thing that’s in our repository and represent it as “a thing” for the purposes of discovery and digital preservation, whether the thing simple or complex like a book.”) (“Be able to export content *as a thing*, and get things into and out of the repo easily”)(“what a digital asset is” - e.g. the node-media coupling that is currently tenuous and easily broken down.)
- Make all features modular and can be enabled or not, including within the same repo. (“Framework for working with digital objects”)(“suite of tools”)(“buffet-style, not locked-in. Features don’t require a specific data model.”)
- My ‘collection manager’ / ‘archivist’ should be able to create a metadata profile fairly simply, and have different ones for (potentially many) different collections. (“describing different collections differently with customized forms”)
- Allow for different types of repository content that behave in different ways.
- We have a lower entry threshold / smooth enough learning curve that a non-dev can “set up an islandora” (and do some basic configuration)
- Have community-built definitions of shared concepts
- Have community-built ‘default behaviours’
- Have ‘defaults’ that ‘make your stuff do a defined set of expected behaviours”, and can be used out of the box, if really desired (“sensible, configurable default views, viewers, pages”)(“solution packs”). [note: there’s a difference between “this is a community-led interpretation of a best practice for X problem that works coherently” and “Turn on

literally everything that you might ever want available in relation to X problem as a demonstration that it works."]

- Make those 'expected behaviours' modular, configurable (not reliant on hard-coded machine names as much as possible), manageable (building a mental model of how to configure it should not take a PhD in Islandora), and transparent (expected behaviours, as they are currently defined in the live config, are communicated to the user). ("I need to know what is needed to do to make it work")("followable documentation")("upgrade with confidence, knowing that we're not hard-coding on specific things i might not have")
- Given a "set of clearly defined expected behaviours" (e.g. an islandora default), it should be straightforward [read: not a trial-and-error filled dive into 80 config screens] to configure a "like this, but changed slightly" set of clearly defined expected behaviours, and use it.
- If new modules/features/affordances come out in an Islandora default, I want to [optionally] be able to turn them on in my altered version too. ("features agree to work on objects with a certain… what does an object need for features to be able to work on it… guarantee that my objects will continue to have these features available to them, and that new features we develop won't be incompatible with my (custom?) object.
- The way we do things should "make sense" from the perspective of a Drupal-familiar person who's new to Islandora. ("drupal user feels confident")
- Cohesive / understandable access controls on media and their descriptions

What are the "Goals" of Islandora as a project?

- Lower the overall barrier to entry to running a digital preservation platform
    - Ease installation and maintenance updates
    - Give repository managers an interface that makes sense, as opposed to disjoint Drupal modules/pages
    - Give repository managers a suite useful tools to help them preserve data "buffet style", without requiring total buy-in to the Islandora ecosystem
    - Provide good documentation
    - Provide reasonable defaults and expected behaviours
    - Provide guidance for modeling content for preservation

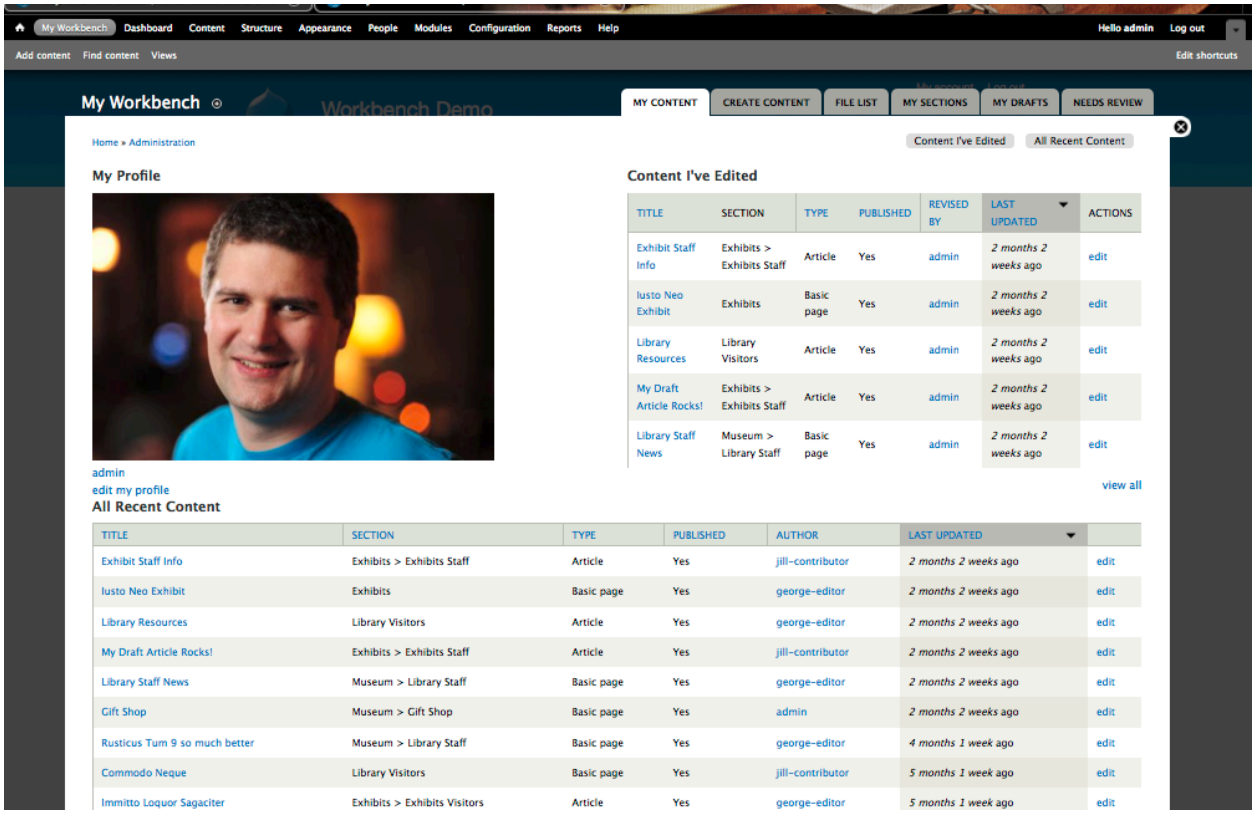
- Modular/extendable/packagable - Basic and custom content models, viewers and other functions can be developed/used/maintained as independent modules (i.e drupal modules) without changing default modules/configs/contexts etc.
- OCFL compliance!

What exactly would this suite of tools be?

- A “repository content type” generator
- [front-end framework]: pluggable/configurable derivative generation (i see this as a framework like a multi-head screwdriver)
    - Create derivative as a File and a new Media, and associate the Media with a Node
    - Create derivative as a file and set to a file field on an entity
    - Create derivative file, instead set to a text field on the entity
- [front-end tools]: integration with ffmpeg to create derivatives of video(/audio?) files (like philips head)
- [front-end tools]: integration with imagemagick to create image derivatives (flathead?)
- [front-end tools]: integration with tesseract to do OCR derivatives
- Message Broker Integration - less of a tool in itself than a workbench, that we can use to configure tools?
- Fcrepo Integration
- Triplestore Integration
- Image Tools
- Audio/Video Tools
- PDF Tools
- Viewers
    - Pdf.js
    - Video.js
    - Openseadragon
    - Mirador
- Tools to calculate checksums, and to do periodic checksum checking
- Tools to export as (various xml or rdf ways)
- Batch ingest/update/export of metadata/media/digital objects.
- Customizable metadata profiles per collection or based on a set of criteria.
- Set of “out of the box” default content types/workflows/configurations/documentation that gives non-dev folks an inroad to understanding the system (solution packs?)
- Multilingual UI and metadata/index/search support

What exactly would this interface look like?

I’m not sayin’ we use workbench, but some kind of landing page when you login with relevant stuff would be a good start if we’re trying to unify this interface.



What exactly would a solution pack look like?

So they used to do everything, and we've broken them apart into tiny little pieces

- Content type / fields
- Logic around the content type and fields
- A microservice for generating derivatives
- An action to pump microservice messages to the queue
- A context for triggering said action to the queue
- A viewer for the content
- Views, specifically EVAs
- Display mode
- Form mode
- … I'm sure there's more

How much of that should be lumped into a solution pack?