Content Types

Amidst a number of successful PoCs, experiments and even productized work (MCTP), this concept aims at formalizing early support for content types in Magnolia, and help feature development move on from there.

Requirements

There's general consensus about the following:

> *[Magnolia needs] a formal definition for a type of content [...] including the fields that type may contain, and its relationships to other types of content.*
> *Based on this definition, other things can be created with little or no work such as a content app, an editing dialog, and REST endpoints.*

—via Content Types

Initial action plan / Spike 0

Based on a registry on one hand, and starting off with a field-based model on the other hand, we could start implementing a minimal user story with few concrete actions.

Status

MGNLCT-9 - Getting issue details... STATUS

MGNLCT-10 - Getting issue details... STATUS

MGNLCT-11 - Getting issue details... STATUS

MGNLUI-4246 - Getting issue details... STATUS

MGNLUI-4238 - Getting issue details... STATUS

MGNLREST-93 - Getting issue details... STATUS

PRs were created, yet they're not deemed to be in a "productized" state atm. Still a bit too reminiscent of the previous PoC efforts.

1. Configuration

Registry

At this stage, **the format does not matter as much** as what we're going to do with it.
It's also unclear why our YAML format should be considered a misfit—we've been pushing it all around the place—or for what use cases it falls short, concretely.

Either way, **Magnolia needs awareness of content types**. Covering that with a **`ContentTypeRegistry`** is a natural starting point, implying a `ContentTypeDefinition` as well.

With Magnolia 5.4+ config & resource-loading APIs, this is trivial to set up. We get significant value out of the box:

- light-module conventions
- visibility inside the definitions app
- problem reporting
- familiar YAML format

Should we hit the wall with the registry or format, and get an alternative on par with benefits above, nothing is set in stone. It just allows us to move on with downstream technical challenges.

Status

CT01. Pick `ContentTypeRegistry` + `ContentTypeDefinition` into a new module

MGNLCT-8 - Getting issue details... STATUS

- Module will sit above core & config, but below UI
- Move registry <-> config-source bindings to the new module class
- tentatively YAML-only to start with

Definition outlook

—what does a content-type consist of?

This is also a never-ending topic. We should **start small, and add more aspects/features as we go**.
As per the high-level vision at the top of this page, **the most prominent part of a content-type is its model**, its fields.

In fact, there is potential for *many* existing Magnolia features to shift under the umbrella of content types, eventually.

Content type all-the-things (allthethings)

Just for reference, here's a non-exhaustive list of aspects a content type *could* specify:

- **data source** (envisioning an evolution of content-connectors, without the Vaadin bits)
  - kind
  - coordinates
  - workspace / table / collection name
  - node types
- **model / fields**
  - relationships, 1-to-1, 1-to-many, etc.
  - evolution, content version-handling
- **locales**
  - see MGNLUI-3616 - Getting issue details... STATUS
- folder-support
- tagging
- versioning
- publication
- personalization
- renditions / URI bindings / previews
- cache policies
- naming / id strategy
- ownership model / security
- consistency model
  - all consistent against latest schema (hence content migration)
  - vs. mixed-model version w/ compatibility
- ...

2. Fields

Modeling

As foreseen, content-types are shifting upstream of apps, that is upstream of any UI/UX consideration.

On the one hand, modeling purists advocate for a strict, dry representation of the model, with almost exclusively primitive property types.
On the other hand, Magnolia has always provided users with pretty loose modeling, driven by field-definitions. The author experience comes first, the data model is deduced from it.

There may be some middle-ground there. **With few key improvements, we can keep our familiar field-definitions, while decoupling them from UI/UX considerations**.

Again, nothing set in stone. This in turn helps progressing towards generation of content apps and endpoints.

Status

We start simple, with a plain map of named fields. We make FieldDefinitions less UI/UX-bound / more UI-agnostic.

CT12. Infer field-types vs. types both ways

MGNLCT-10 - Getting issue details... STATUS

- setting "fieldType": FieldDefinition impls have hard-coded types (already)
- setting "type": infer a simple field type upon form creation (dialog, app)

CT13. Detach Validators from Vaadin

MGNLUI-4246 - Getting issue details... STATUS

- boilerplate: must implement:
  - Vaadin Validator
  - Magnolia ValidatorFactory
  - Magnolia ValidatorDefinition
- switch towards standard Bean Validation (well supported by Vaadin too)
- provide config short-hands for standard bean validators (max-length, range, etc.)
  - validator-registry?

Under the radar (unranked)

- Support semi-structured content types (leveraging the content-editor)
- Defining & resolving references to other content-types (leaving linking out first)
- Groups of fields and/or *1:n* relationships

UI bits

In addition to making fields more UI-agnostic, we seek to maintain presentation flexibility on app & form level.

Under the radar

CT11. Once CT gives us a flat list of fields, layout can be configured on app level

MGNLUI-4245 - Getting issue details... STATUS

- ditch `TabDefinitions` in favor of an optional layout config

3. Data source

CT21. Reuse automatic workspace creation

MGNLCT-9 - Getting issue details... STATUS

MGNLCT-11 - Getting issue details... STATUS

CT22. Connect an existing app or sub-app descriptor to a ContentTypeDefinition

MGNLUI-4247 - Getting issue details... STATUS

- experimented in Sang's PoC, via plain *contentType* reference id

Status

To be discussed / defined.

DEV-617 - Getting issue details... STATUS

4. Content apps

App generation

We seem to have consensus **against re-introducing definition builders maintained by hand** (like those we reverted before 5.0)

- A. Either verbosely build/nest configured definitions w/ plain constructors (no new def code)
- B. And/or we give a shot at leveraging Sasha's code-generation efforts for this

Status

CT23. Generate apps on-the-fly, upon app startup

MGNLUI-4238 - Getting issue details... STATUS

- AppDescriptorRegistry listens/delegates fluently to the ContentTypeRegistry
- broadly speaking, CTR is a source for apps
- resolve a placeholder DefinitionMetadata
- use a dynamic DefinitionProvider building the app on-the-fly from the type

Options under the radar

- Support a new, less-boilerplate app descriptor type, conveying higher-level semantics, leveraging content-types
- Definition-less app on the fly; dynamically built based on CT definition features.
- Either way, both would cause major changes to app sub-components (e.g. workbench, actions, all presenters heavily relying on defs atm; views hardly usable standalone)

App defaults

- yet to formalize config patterns, actions, columns, labelling...
- crud, publishing, import-export
- different stock apps have different flavors of actionbar section availability
- even the groovy generator is not fully "canonical", vs. M5 UX guidelines

Status

To be formalized.

Under the radar

- Composing mixin configuration defaults per content-types capabilities

5. Consuming content

REST endpoint defaults

Status

We have a productized, JCR-based, basic delivery endpoint. Currently this is just one "global" endpoint, taking the workspace name as path parameter. Therefore at the moment, there's no such thing as one endpoint per content type (might come again later), so nothing extra to register.

MGNLREST-93 - Getting issue details... STATUS

Options under the radar

- Side-by-side endpoint implementations for CT-based vs. JCR-based.
- Single endpoint registrations (more content-semantic APIs than with the generic one)

REST improvements will be covered in more details in a separate concept.