Guide to Resource Linking and Embedding (DRAFT)

Background

Responses to DSpace 7 REST API requests are HAL documents which, in accordance with the specification, contain the following two reserved properties:

- **`_links`**: contains links to other resources
- **`_embedded`**: contains embedded resources

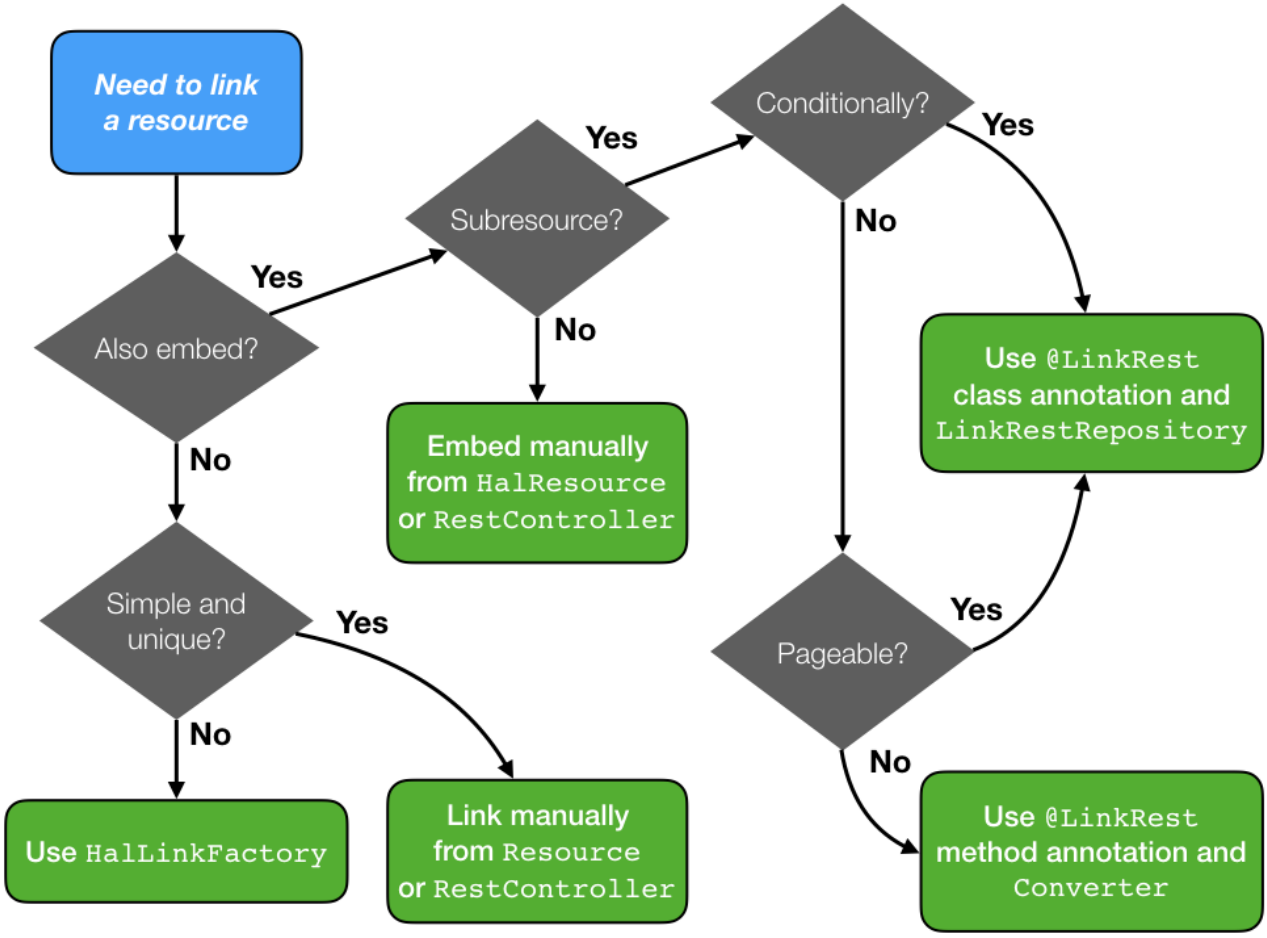
There are multiple ways to add links and embeds within the DSpace 7 server webapp.

The purpose of this guide is to help developers decide which method is appropriate for a given scenario. *(TBD: provide links/examples of each).*

Before following this guide as a developer, you should already have a Rest Model object in mind (usually a subclass of `RestAddressableModel`, with a name ending with *Rest*, by convention). That object should already have a few properties, and you should have an idea of what link you want to create from it. You should also have a `Resource` subclass in mind.

Determining the recommended approach

Follow this chart. See below for details on each question and recommended approach.



Decision Points

Also embed?

Will you ever want to embed a representation of the linked resource within the current one?

Simple and unique?

Are you making a single link to a single resource, and is the link unique to this resource?

Subresource?

Does the URL of what you're linking and embedding begin with the canonical (aka "self") URL of this resource?

Conditionally?

Do you only want to only embed sometimes, based on whether some condition is true?

Pageable?

Are you embedding a list of things that might grow beyond a very small number and thus necessitate paging?

Recommended Approaches

Use `HalLinkFactory`

This provides a pluggable approach to adding links to resources. It is well suited to linking needs that are cross-cutting (apply to multiple types of resources) or non-trivial.

Link manually from `Resource` or `RestController`

When linking to a single resource, especially if that link is unique to the current resource, the creation of a link factory can be overkill. The preferred approach is to add the link from within your Resource (or HalResource) subclass constructor, or directly from your rest controller, if the link is unique to that particular rest endpoint.

Embed manually from `HalResource` or `RestController`

When linking to *and embedding* a resource that exists at a different path in the URL space (not a subpath of the current resource), the preferred approach is to embed it from within the HalResource subclass constructor, or directly from your rest controller, if the embed is unique to that particular rest endpoint.

Use `@LinkRest` class annotation and `LinkRestRepository`

When subresource embedding is optional or pageable, the recommended approach is to create a LinkRestRepository with a method to provide the data.

Then you can refer to the repository and method in the @LinkRest class annotation (inside a @LinksRest annotation)in your Rest Model class. This allow the resource to be exposed as a subresource and permit the optional embedding of it.

This makes it possible to avoid the work of a) querying the database unnecessarily in certain cases, via the entity class used by the Converter, and b) reading in all possible values of a potentially long list into memory.

Use `@LinkRest` method annotation and `Converter`

If the subresource to be embedded is small and required in all cases, the preferred approach is to use a `@LinkRest` method annotation on the Rest Model class. In this case, the `Converter` will be responsible for populating the value(s). This allows you to avoid the overhead of creating a `LinkRestRepository` while still allowing the resource to be exposed as a subresource in the REST API.