PersistentIdentifiers Requirements

See also:

- ObjectUri
- W3 TR on URI/URN/URL (2001): http://www.w3.org/TR/uri-clarification/
- Commons ID: http://jakarta.apache.org/commons/sandbox/id/
  - java.util.UUID
  - Java UUID Generator (JUG): http://jug.safehaus.org/

Following from the recent discussion of persistent identifiers in DSpace, I thought it would be a good idea to define exactly what our requirements are. I will try to avoid using the term "persistent" because it is a loaded term that has limited meaning in the context of long-term preservation – if we assign a global identifier to a resource, it should be assumed that the identifier can and will survive as long as the object does. The resolution aspect of persistent identification is the key thing to be concerned about in this context, but that is far beyond the scope of what DSpace can take responsibility for.

It is important to distinguish between locally assigned & managed identifiers (which may or may not be globally unique) and globally unique identifiers that are external. Such external identifiers are treated as metadata and are not used for identification locally.

Note: When I refer to "objects in DSpace" this will largely be referring to entities that extend DSpaceObject.

Overview: Scoping of Identifier Schemes

The following figure attempts to illustrate the problem of object identification within DSpace. In particular, it highlights the fact that:

- The database key used locally to reference an object might be different from the "local identifier"
- Locally-defined identifiers might be meaningful (name- or path-like) or might be opaque (UUIDs or even hashes)
- Local repositories might instead chose to use external, centrally-managed identifier schemes
- External systems are usually registries (e.g. the Handle System(r)), and resolvable URLs (perhaps based on the local identifier) will still be required

Local & global identification & resolution
All objects captured in DSpace should have at least a locally unique identifier. Resolution to objects is always required, except where it doesn’t make sense (e.g., users, groups, bundles, etc).

**UUIDs**

It has been suggested that DSpace could switch to using **UUIDs** for object identification. This approach would have the following benefits:

- Objects could be moved between archives while retaining their IDs
- Identifier generation would be independent of the storage layer (useful for reconstructing archives from AIPs etc)
- Simple, robust identification mechanism even when universal uniqueness isn’t a requirement

**Local resolution**

The simplest way to achieve local resolution to objects would be to map local identifiers into the DSpace URL space. If this approach is taken, there are a few things to consider:

- We would need to be careful about gracefully handling cases of deleted / withdrawn objects
- As already observed, some objects don’t have a natural web-UI presentation (users, groups, etc)
- If UUIDs are adopted for identification, we would be introducing 128-bit identifiers (which are rendered canonically as 36 text characters, but could be rendered in as few as 22 characters) into object URLs which would make them excessively long
- DSpace currently makes assumptions about the form of identifiers present in the URL (specifically, that they contain exactly one “/”). This assumption will not work with arbitrary identifiers, so the way in which URLs are constructed will have to take a more parameterized form.

**Global identification & resolution**

The assignment of persistent identifiers to objects in DSpace is a matter of some debate. For simplicity, I will assume that all objects have at least one persistent identifier that is assigned and managed locally (similar to the current behaviour with Handles). The prototype implementation I have currently allows for zero to n identifiers per object (and makes no assumptions about how they are generated), but for the purposes of this discussion we need only consider objects with a single, locally generated identifier.

Global (persistent) identifiers are assumed to be similar to Handles, DOIs, PURLs, ARKs, etc. What each of these mechanisms have in common is universal identification coupled with a robust resolution mechanism (hence making them “persistent”). However, there are identifiers that are globally unique and yet have no resolution mechanism (UUIDs, ISBN, ISSN, etc). It is therefore important that we make no assumption about the capabilities of persistent identifiers beyond the actual identification part. That some identifiers also have HTTP proxy resolvers should be exploited where possible, but should not be assumed.

**Use cases**

In order to make solid some of the ideas expressed above, I will outline a few core use cases for internal and external identifiers in DSpace.

**Moving objects between archives**

If we want to transparently migrate objects between DSpace instances, it may be desirable to have them retain their internal identifier, as well as any external identifiers that may refer to them. For this purpose, the only option as I see it is to use UUIDs; while the use of external identifiers for use as the primary internal identification mechanism is possible, it is highly undesirable for several obvious reasons.

However, I’m not convinced that this (global uniqueness) is really a legitimate requirement for our internal identification mechanism. If we really care about tracking objects between archives, then I think it is reasonable to assume that an external identification mechanism is being employed that can perform this function for us. This isn’t to say that I disagree with using UUIDs, I just don’t think that it can be used as an argument in favour of using them.

**Replicating objects between archives**

Similar to the above use case of migrating content, the implications of replicating (copying) objects between archives has implications for the identifier mechanisms used. A corner case that highlights some key issues here is as follows:

PURLs are used as the external identifier mechanism, and so each identifier can only refer to a single resource location (unlike Handles which can have arbitrary name / value pairs for indicating multiple resource locations). An object is replicated in a second DSpace instance, and gets a new external identifier assigned, as per local policy. How do we now make the connection between the two objects? If we used UUIDs internally, then both objects would have this in common, and so it would be possible to relate them. Without this capability, the relationship between the objects would effectively be lost (i.e., we can’t depend on the external identifier mechanism to solve this problem for us).

Something that occurred to me when thinking about this is that the OpenArchives ORE effort may help us here. If we could use ORE to keep track of objects that exist in multiple archives (or even objects that are composed of objects in other archives) then we would have a really powerful infrastructure that is independent of identification mechanisms, both internal and external.