PersistentIdentifiers

See also:

- PersistentIdentifiers Requirements
- ObjectUri

Proposal to "fix" DSpace persistent identifier handling

*I've "top-posted" this so that it gets read; the rest of the information below is still relevant, but it's pretty well established. I should note that code from patch #1690912 could be used in this work. --*JR

Update (25/06/07): _I've made considerable progress on the code, and both internal and external identifiers are supported in a very general fashion. I'm confident that with very little effort, it should be possible to plug in arbitrary identifiers for objects. I've stripped the code examples from the page, as they were out of date and misleading. The code is available as part of the DAO prototype branch of the dspace-sandbox repository on Google Code: http://code.google.com/p/dspace-sandbox/source_ --JR

There has been some discussion recently about how to abstract the persistent identifier mechanism used by DSpace to something more "pluggable". Core requirements seem to be:

- persistent identifiers should always exist in some form for the same objects as they do currently
- they should be configurable & stackable, with a default (probably defaulting to Handles)
  - *What does "stackable" mean here - implemented in e.g. a sequence plugin that allows alternate persistent-ID types to coexist? LarryStone 22:24, 16 May 2007 (EDT)*
    - *I'm currently using a sequence plugin. --JR*
- several people are interested in assigning persistent identifiers to arbitrary

  ```
  DSpaceObject
  ```

  s, possibly even object metadata (Claudia Jürgen noted that objects may contain metadata that is stored in normative reference bases, and as such could be eligible for persistent identification).
- identifiers are *not* necessarily resolvable outside of DSpace, although it is better if they are. LarryStone 22:24, 16 May 2007 (EDT)
  - Yes - I think this is an exceptionally important point if we're serious about supporting long-term preservation. I take this proposal as being for providing pluggable identifier schemes and improving internal handling of identifiers rather than improving id management in a preservation context. If we're talking long-term preservation, I would expect that all the identity management (allocation/resolution) would need to be very separate from the repository software (or at least separated as far as is possible). Else how are custody changes going to work at all - e.g. if I move a journal from a DSpace at institution X to a DSpace at institution Y, previous citations (assuming use of the handle proxy and not local URL for citation) to journal at X will no longer resolve because the resolution mechanism is currently tied to each individual repository. The problem is exacerbated if moving from unlike repositories/CMSs with their own id mechanisms (e.g. DSpace to Fedora, OJS to DSpace). To solve this really I think requires an external service like the CNRI handle service but more comprehensive handling resolution lower than prefix level. But then comes the fun issues of who "owns" the service, funding, resourcing, etc. Some work is being undertaken to look at this (see http://www.arrow.edu.au/PILIN) but it's a really big issue (SY)
- *every* DSpaceObject of certain classes (e.g. Collection, Community, Item) *must* be bound to one canonical persistent identifier. This is not currently a requirement. LarryStone 22:24, 16 May 2007 (EDT)
- DSpaceObjects *may* be bound to more than one identifier (currently implemented in DC metadata, but needs to be recognized as a PersistentIdentifer) LarryStone 22:24, 16 May 2007 (EDT)
  - *This is a desired outcome of the implementation. --JR*
- ... (someone fill in the gaps please --JR)

Here is a *very* rough draft of how I think we could handle persistent identifiers.

What makes a persistent identifier?

I've had a go at defining exactly what is required of a persistent identifier, and how this might map to the API outlined below:

```
NS VALUE PROTOCOL BASE
hdl 1234/56 http hdl.handle.net
doi 1234/56 http dx.doi.org
purl 1234/56 http purl.oclc.org
ln-s 123456 http ln-s.net
```

The ln-s.net example is a bit artificial (I don't think anyone would use a public url shortening system as a basis for persistent identifiers), but it demonstrates that we won't always be able to break a

```
VALUE
```

up into

```
prefix/suffix
```

.

Ideally, *value* should be treated as an opaque string. Just URL-encode it when putting it into a URL. Unfortunately, many HTTP implementations are buggy when interpreting escaped "/" (slash) characters, and end-users will be sloppy about transcribing them, so any Web interface has to take slashes in the Persistent ID's into consideration. Maybe we should encode this somehow with a predicate method, `PersistentIdentifier.hasSlashes()` so servlet code would know what to expect. LarryStone 22:24, 16 May 2007 (EDT)

Note that Handles (and DOIs) are resolved through the Handle System protocol which is distinct from HTTP. The `hdl.handle.net` prefix is just an HTTP proxy server. This is actually an advantage of Handles; anything that depends on HTTP is not very "persistent". Handles and DOIs will outlive the Domain Name system and Internet protocols. LarryStone 22:24, 16 May 2007 (EDT)

- *Agreed. The fact that persistent identifiers built on the Handle System only support HTTP for convenience was a motivating factor in including "protocol" as an attribute of the persistent identifier types. --JR*

`org.dspace.content.uri`

">

```
org.dspace.content.uri
```

New packge containing all the classes and interfaces for managing identifiers.

`org.dspace.content.uri.ExternalIdentifier`

">

```
org.dspace.content.uri.ExternalIdentifier
```

I'm not sure about

```
getLocalURL()
```

. I don't really think it should exist, since it isn't a persistent identifier (by definition), and as such has no place here.

Update (25/06/07) *I've worked around this problem by making a clean separation between internal and external identifiers. Only internal identifiers may know the "local" URL of the object. External identifiers only know what the "global" URL is --*JR

The

```
getMetadata()
```

methods are there to support systems similar to the Handle System that allow arbitrary name / value pairs to be store with the persistent identifier record. If such a mechanism isn't supported with a given implementation, we can always just return

```
null
```

.

How should persistent identifiers be configurable?

For now, I have put a simple configuration option in

```
dspace.cfg
```

:

```
plugin.sequence.org.dspace.content.uri.ExternalIdentifier = \
org.dspace.content.uri.Handle
```

This is where the list of supported external identifiers will be kept. Issues arise when objects have more than one option for what to use as an external identifier. A few simple scenarios are:

- More than one identifier is supported. How do we define which gets used? First off the stack; all supported identifiers; per-collection configuration, etc...
- An object arrives in the system with an identifier that is supported locally. Do we add our own identifier?

How should the URLs be modified?

I propose a simple switch from (eg):

```
http://pomona.hpl.hp.com/dspace/handle/1234/56
```

to

```
http://pomona.hpl.hp.com/dspace/resource/hdl/1234/56
```

It has been suggested that maybe OpenURL-style URLs may be the way forward in this respect.

Escaping issues

All versions of Tomcat before version 6 (I think) have issues with URLs containing escaped slashes ("

```
%2F
```

"). This will cause problems for a lot of people if we want to stick to the spec and escape slashes in our identifiers (though legal for use in the URL, they have semantics attached to them that make their unescaped use in identifiers technically incorrect). The fix was backported from version 6.0.x to 5.5.x in revision 507117 of the Tomcat SVN repo (details here). Looking at the tags in the Tomcat SVN repo, it looks like 5.5.22 was the first release after this fix was applied. I'm currently running 5.5.23 and it is still broken, so I think the default behaviour is still to throw the error, but you can set some configuration to allow the escaping.

Here is the relevant code from

```
tomcat/connectors/trunk/util/java/org/apache/tomcat/util/buf/UDecoder.java
```

:

```
Boolean.valueOf(System.getProperty("org.apache.tomcat.util.buf.UDecoder.ALLOW_ENCODED_SLASH", "false")).booleanValue();
```

See also:

- https://www.covalent.net/download/patch2.0/README-ers-3.1.0-patch-tomcat-20070315.txt

Bitstream

issue">The

```
Bitstream
```

issue

Currently, DSpace only assigns persistent identifiers to

```
Communities
```

,

```
Collection
```

s, and

```
Item
```

s. Among other things, this means that we currently have the rather bogus method of accessing the files associated with an

```
Item
```

:

```
http://pomona.hpl.hp.com/dspace/bitstream/1234/56/x/fileName.foo
```

where

```
x
```

is the (internal) sequence id of the file itself. What would be far better would be for

```
Bitstream
```

s to have their own URIs so they were resolvable in the same way as anything else. After a discussion with Richard Jones, Claudia Jürgen, and Stuart Lewis, it became apparent that this would require all sorts of shenanigans to implement (including versioning for

```
Bitstreams
```

, as well as a withdraw / delete feature similar to

```
Item
```

s). This isn't necessarily a Bad Thing (in fact, it's definitely a Good Thing), but it means that the implementation would need to be thoroughly considered, and implementation on top of the current information model would be tricky.

Handle System

Many issues here:
Some people don't want to use Handles.
Handles are rather deeply ingrained in the system:

- Plenty of references to 'Handles' in names of variables, fields, methods, database tables etc. (should be just 'Identifier' or somesuch)
- Ideally, IDs should just be treated as opaque strings of characters by DSpace, making no assumptions. However, because of the way Handles are stored and used, various parts of code 'parse' and manipulate them in a way that would mangle or break with other ID schemes:
    - They're stored in the 'handle' table as 'raw' Handles, e.g. `1721.1/1234`
    - In various places they are used in the URN-syntax form, e.g. `hdl:1721.1/1234`
    - Since browsers cannot resolve either of the above, Handles are usually displayed embedded in a URL for an HTTP/Handle 'proxy', e.g. http://hdl.handle.net/1721.1/1234.
    - Tools which accept Handles as part of command line parameters, etc., in order to be resilient to any of the above forms of Handle as input, look inside (parse) the Handle to see which form it is and extract the 'raw' form. This code would probably break with any other ID scheme.
    - Handles contain slashes '/'. Whenever they're embedded in URLs, the code may need to take this into account, and hence can't just treat the ID as an opaque string of characters. E.g. when you're searching or browsing inside a collection, the URL is: https://dspace.mit.edu/handle/1721.1/1783/browse-title As you can probably tell, the servlet code needs to know about the slash in the Handle to be able to parse this.
    - Another example of this is OAI-PMH set specs, where a 'safe' version of the Handle is used. (slashes etc. turned into _).

Should bitstreams get persistent IDs? At the moment they just get semi-persistent URLs, which include the item Handle and a sequence number; e.g.: https://dspace.mit.edu/bitstream/1721.1/4852/6/toc.pdf. This is probably something that should be configurable.
Should be possible to play with Handle system "properly".

DSpace's integration with the Handle server is rather clunky. DSpace 'pretends' to be a storage backend implementation for the Handle server, and DSpace is actually managing / administering the Handles itself. This basically means a Handle server running this way can really only resolve Handles for things in that DSpace, and do nothing else. This approach was taken for reasons of expediency and efficiency, and to make setting the whole thing up simpler.
This would involve building a Handle client for DSpace that requests new Handles from a Handle service, and updates the Handles using that service with the appropriate URL (and maybe other metadata).

Why it is like it is now

To understand the ID decisions in DSpace right now, one really needs to understand the object model and reasoning behind it.
The basic reason behind using Handles is that they identify an abstract intellectual object and will continue to do so over time, despite the state of the underlying 'physical' object changing – underlying formats, custodian, location, perhaps being stored in multiple locations, access/delivery mechanism and so forth.
Given this, we made the decision to give the following things Handles:

Communities – the original intent behind communities was to group together content and services pertaining or of interest to a particular group of people; e.g. a department in a university, or perhaps a subject area. Although over time, such things are perhaps more likely to really change than individual intellectual works, it was felt that they were still worth preserving.

Collections – groups of related Items. Some may change a lot (in terms of the intellectual works they contain), many won't.

Items – these are intellectual objects or 'archival atoms'. In my opinion they correspond to 'expressions' in http://www.ifla.org/VII/s13/frbr/frbr.pdf the FRBR model. The underlying manifestation of these might change over time (e.g. MS Word to PDF to PDF/A) and there might be many manifestations of a particular item available at one point in time. The intention is that if in 50 years time I come across a Handle, I will want to get to a manifestation appropriate for consumption using contemporary technology rather than the particular version of Microsoft Word or media player etc. that was used to create the object in the first place.

Below that in the DSpace object model, we have Bundles, which were originally intended to correspond to manifestations (e.g. HTML w/GIF image manifestation, PDF manifestation etc), though in practice this isn't how they've been used (mainly because not many people have multiple manifestations in a single Item, since the software isn't particularly mature in dealing with that yet). In any case, the bundle structure of an item is likely to change over time, and individual bundles are only like to be of use for a limited period of time. Hence the idea that a Handle for a Bundle/manifestation isn't appropriate.

Then we have the bitstream, which is a constituent part of a bundle/manifestation. While there is a possibility that one might want to refer to a specific bitstream, i.e. a set of 1's and 0's that will never ever change, the administrative overhead and costs in assigning **every** bitstream a Handle was deemed to much. If a single bitstream is likely to be thus cited, it can be 'promoted' so that an item contains nothing but that bitstream and then the Handle for that can be used.

A complication which has led to the situation where use of Handles is rather embedded in the DSpace code, is the desire that these persistent identifiers be resolvable. i.e. that the identifiers can just be shoved into a browser's address bar, and bang, something useful for the end user happens. (This is as opposed to having to go to some search tool, say a DSpace instance or Google, and enter the ID there to find the identified object). Handles in their native, URN-syntax form (e.g. hdl:1721.1/1234) are not understood by browsers. So, URL proxy of the Handle is displayed, e.g. http://hdl.handle.net/1721.1/1234. In various parts of the system, either form is used, or may be received as part of some request, so some parsing must be done. This really needs to be untangled somehow if the persistent ID part of DSpace is to really be 'pluggable'.
The above only really considers the use case of external references, e.g. in journal articles. It does not speaks to what I consider separate issues:

1. how DSpace identifies and manages the relationships between things internally
2. how external systems interoperate with DSpace.

Although in some cases it may be possible and desirable for the persistent identifiers used for a long-term preservation purpose to be the same as those used for 1/ and/or 2/, there is certainly no requirement for this, and should not be assumed.
At the moment for 1/ DSpace uses database primary keys to manage the relationships between these objects in the system. These are rather volatile – any rebuilding of the database, importing/exporting, or software changes could well mean these database keys change and so they are totally inappropriate to use outside of the system itself. However they are of course very efficient for querying in the usual SQL SELECT/JOIN sort of way.

Given this, to facilitate 2/, DSpace currently assigns 'semi-persistent' IDs to bitstreams, of the form: http://host.name/bitstream/(item_handle)/(sequence_number)/filename.ext
I say semi-persistent, since although this is more resilient than database primary keys, it is still tied to the host name, and thus if the item changes custody, is moved to another server etc., the above URL would no longer work.

Unknown macro: {latex}

\begin

Unknown macro: {eqnarray}

{
Insert formula here
}
\end