Embargo 1.6

Embargo Functionality

Another area of interest identified in the DSpace+1.6 survey was embargo. Several efforts have already been made against recent versions of the codebase, so the expectation that one or a combination of them could be made ready for 1.6 is not unreasonable. Please use this page to help us understand the range of desired functionality, or if you have done work that has not been advertised to the community, bring it to our attention. Record your comments below, or if you are more comfortable using email, send them to rrodgers@mit.edu Richard Rodgers, and they will be added to the page. Since we hope to have 1.6 in the not too distant future, the sooner you respond the more likely your input will shape the released feature set by creating a superior papers.

A few examples in the requirements space:

- Scope: all Item bitstreams, selected bitstreams, metadata, etc??
- Policy Source: by collection, per individual Item(bitstream) only? Should submitter supply, or administrative action?
- Policy Terms: fixed (configurable) set of intervals, a date, moving wall, indefinite embargo, etc?

Responses

Harvard Univ. Library

Requirements

We have an additional complication that only some Items are eligible for embargo. Our Items have a *license* keyword value chosen from a small controlled vocabulary, and only some license types can be embargoed (more about this later..). However, I'm content to just implement that restriction in the submission UI and hope that Administrator users are careful making changes.

Other requirements:

- The presence and expiration date of the embargo must be easily visible on an Item, e.g. on its Item view page.
- Every embargo includes an end date, at which time the Item is released. There is no other transition.
- When an Item is under embargo:
    1. The Item metadata page and all metadata fields are still visible to the world (and through OAI-PMH).
    2. *All* of its Bitstreams, including full text, are *not* readable to the world. Administrators can bypass the protection.
    3. Any *new* Bitstreams added to the Item (and generated by MediaFilters) are likewise protected.
    4. There is an Option to let anyone with the COLLECTION_ADMIN role on the Item's owning collection to also read all of its Bitstreams. (not that important)
    5. The Item should still have its full text indexed.
- When the embargo expires (or is revoked), all Bitstreams are changed so they are world-readable.

So the UI operations we forsee needing are:

1. Choose and specify date of embargo at submission time.
2. Set embargo terms on an Item created by unattended submission, e.g. SWORD, LNI.
3. Inspect the embargo status of an Item (OK if this is visible to the world)
4. Administrator can change embargo status: either add embargo, or release it early.
5. Search and/or browse embargoed Items only.
6. When embargo expires, email is automatically sent to a user designated as owner.

Implementation

I'm in favor of implementing embargoes using the existing data model, i.e. Item metadata fields and access control policies. Most of the requirements are easily met this way. For example:

1. Use a dedicated (configurable) metadata field to record the embargo expiration date, in regular "DCDate" format. Recommend sites use a field in a separate schema, e.g.

    ```
    site.embargo.until
    ```

    , so it doesn't leak out in OAI DC crosswalks and such.
    - The presence of that field means the Item is under embargo.
2. Enforce embargo by setting ResourcePolicy access controls on Bitstreams and Bundles (Bundles control policy of new Bitstreams)
    - Employ an Event consumer to ensure that changes to the Item's metadata get tracked in resource policies.
3. Add a periodic task (cron job on Unix) to check for expired Embargoes and fix Policies.

Using a metadata field to record the embargo has many benefits:

- Display comes for free.
- Browsing embargoed items, even by date, is simply a matter of configuration.
- Altering embargo through the Edit Item admin UI is already done, if a bit risky (no protection from entering bad date format).
- Unattended ingest is also done, since a crosswalk can set the embargo.

There are some drawbacks of this design:

- Access control is inflexible - content Bitstreams are either all on, or all off.
- All metadata and indexing is still available. This is one of our goals, though.

The logic to set Bitstream access controls might need to be customized at each site, so it could be a good idea to make that a plugin so it's easy to change. It would have to check the ResorucePolicies on all Bitstreams and change them if necessary to syncrhonize with the embargo status.

Dryad implementation

The Dryad project was recently mentioned on the lists. It includes (among other things) a fine prototype implementation of a similar embargo design. There is even a detailed writeup by @mire. They also take the approach of storing the date in a metadata field, and a cron daemon does all the policy updates. It's well worth a good look.

LarryStone 02:00, 29 May 2009 (EDT)

Johns Hopkins

Requirements

Scope

- The entire Item should be embargoed. If the eventual implementation allows bitstream level embargoes, that is fine with us but all we require is Item level embargoes.

In our implementation, an embargoed item returns empty

```
Bundle[]
```

s, and does not allow any bitstreams to be created on the Item (by throwing

```
AuthorizeException
```

in response to the bitstream creation methods).

Policy Source

- Configured at the Collection level
  - Whether or not embargoes are enabled for the Collection
  - Whether or not embargoed items can be discovered
  - Users who have permission to bypass the embargo should always be able to discover the item, regardless of the discovery setting
  - Configured set of embargo intervals

It should be noted that in our implementation, the policy (represented by DspaceEmbargoConfig) was used at the UI layer to drive workflow. At the API layer, the developer can do whatever they want (e.g. set an embargo on an item that resides in a collection that has embargoes disabled). It is up to the developer to consult the policy and do the right thing.

We define *discovery* to be visibility of Item metadata in browse, search, and syndication (rss/atom and OAI-PMH). In our original design (see DspaceEmbargoConfig), we had individual discovery flags for these. The reality is that we never had a need for that level of granularity, so we added a convenience method isDiscoverable() which represents the abstract concept.

Policy Terms

- Configurable set of intervals. Indefinite should be an allowed interval. We have no need to support any other type of embargo (but it would be nice if the implementation allowed more types to be plugged in).
- The submitter should be able to select the embargo term at submission.
- The submitter should not be able to update the terms of the embargo without approval. In our implementation, we do not allow the submitter to update the embargo term. They can only request the term at submission (and then it is approved by an administrator before it is accepted into the archive).
- The administrator should have privileges to add/remove/update the embargo term.
- Certain users should be able to bypass the embargo. It would be nice if the users could be configured. But at the least the Collection admin, DSpace admin, and the submitter should be able to bypass the embargo.

Other Requirements

- There must be an audit trail, recording who embargoed what item, when the embargo was put in place, and when the embargo is set to expire. When embargoes are lifted (or updated) (e.g. by administrative action) an audit event should be recorded who lifted the embargo, on what item, and when the embargo was lifted. The audit trail should be viewable by administrative staff.
- The Item view should carry the embargo expiration date, and it should be displayed in a simple item view. In other words, it needs to be clear to the user that they are viewing an embargoed item, and when the embargo will will expire (if known).

In our implementation, we record audit information in dc.provenance metadata fields. Anyone can view these audit trails.

Implementation

Our implementation is covered here, including links to the source code and Maven website.

We took a lot of liberties with the DSpace codebase. The biggest change was to modify HandleManager to consult the embargo service and return instances of EmbargoedItem if the Item was embargoed. There were a lot of other bits touched in the browse, search, oai, and XMLUI BitstreamReader.

I agree with Larry and other implementations, including @mire's, that using

```
ResourcePolicy
```

s to represent embargoes are a good idea.

We also created an eventing system which is used to manage metadata when an embargo is enacted or lifted. Event producers and listeners are wired up in

```
dspace.cfg
```

.

NESCent/Dryad

These are the requirements that were used by @mire to implement the embargo functionality in Dryad:

- Embargo is at the level of DSpace items. While an item is under embargo, all of its bitstreams are restricted.
- Members of a privileged group (e.g., Hidden_Bitstream_Viewers) can view/download bitstreams that are embargoed.
- Non-members of the privileged group are not able to access embargoed bitstreams in the repository (though they still see the item-level metadata).
- Metadata is always visible for embargoed items. This applies to search results, item display, and OAI export. (Note: Dryad requires metadata visibility, but it is fine for this to be configurable, to meet the needs of other DSpace users.)
- For items under embargo, a single metadata field stores the end date of the embargo. The submission system should take responsibility for correctly setting this field (in Dryad, the submission system may automatically calculate this field from other date fields). Once the embargo date has expired, the item should be fully visible to all users.

Université de Montréal Libraries

Requirements (context)

We came up with an embargo option while implementing our ETD pilot project.
(Starting October 1st 2009, ETD's will be mandatory on campus. Meaning, students will need to archive their ETD in our institutional repository to get their diploma.)

Students submit their ETDs in DSpace and select the time length of the embargo they want directly in the submission form. Students must have written permission from department before selecting this submission option.

Personnel in the department can double check if student selected an embargo and make sure they were granted permission to do so.
(This is standard protocol for ETD submission since we use the 3 steps validation process in DSpace. and submissions are verified by the department administrative staff at the 2nd step of validation.)

Implementation

We added a *dc.date.available* field in the

```
input-forms.xml
```

to store the embargo data. We use a dropdown menu to present 5 options to users. The dropdown menu is on "Publish immediately" by default with the

```
<stored-value>
```

being "NO_RESTRICTION". Students can then choose 4 different lengths of embargo: 6 months, 1 year, 2 years & 5 years. This is coded in the <stored-value> with each embargo period encoded with *MONTHS_WITHHELD#months*. ("MONTHS_WITHHELD:6" = 6 months and "MONTHS_WITHHELD:60" = 5 years.)

This means that once an ETD item is published in DSpace, it contains 2 *dc.date.available* fields: 1 with the entry date in the repository and 1 with the embargo length.

We then modified 4 different classes:

```
AuthorizedManager.java
```

/

```
Item.java
```

/

```
ItemTag.java
```

/

```
BitstreamServlet.java
```

- ```
  AuthorizedManager.java
  ```

  now contains a method called

  ```
  checkEmbargo
  ```

  that checks for a *dc.date.available* containing *MONTHS_WITHHELD:#months* and then adds the amount of months specified to the date in the *dc.date.issued* field which is the date the University grants the diploma. (*dc.date.issued* + *#months* of embargo) There's also a second method called

  ```
  dateEmbargoEnd
  ```

  that creates the date when the embargo ends so we can display it to the users.

- ```
  Item.java
  ```

  uses a method called

  ```
  isThisitemInEmargo
  ```

  to ask the method

  ```
  checkEmbargo
  ```

  in

  ```
  AuthorizeManager.java
  ```

  if there's an embargo on the item.

- ```
  ItemTag.java
  ```

  asks the method
  ```

```
isThisitemInEmargo
```

in

```
Item.java
```

if there's an embargo on the item. If yes, it then asks for the end date to display instead of the view/Open link.

You can check an example right here: http://hdl.handle.net/1866/5085

- We also modified the

```
BitstreamServlet.java
```

, who calls the methods in the

```
AuthorizedManager.java
```

file. This blocks the possibility of someone trying to access the PDF file by directly accessing the bitstream in the URL. (i.e. Using a functioning URL from another document and replacing the handle number with the item under embargo.)
You can check an example right here:
https://papyrus.bib.umontreal.ca/dspace/bitstream/1866/5085/6/Lefevre_Xavier_2011_Memoire.pdf

I must specify that credit for this work should be granted to my 2 colleagues and that I'm only explaining this after they implemented it last fall. I wanted to share this information to help in the creation of the best embargo option for DSpace.

Design Proposal Strawman

Before any implementation work is undertaken, it is useful to have a fairly detailed description of the functionality and design of a proposed solution. Further input is welcome, and can now also address the narrower topic of how well this design addresses the expressed needs.

Assumptions and Objectives

The goal not to be a union of all desired functionality; rather, it is both directly to address the core requirements common to all implementations, and also to make extending the solution for custom purpose easy and conformant to the DSpace architecture and APIs.

Findings

- There seems to be agreement that utilizing DSpace

```
ResourcePolicy
```

s is a natural way to manage access restrictions, since there are standard interfaces to them, and they are consulted when access is required at the API, not application, level.
- There is no requirement to provide bitstream-level embargo - all bitstreams are restricted under an embargo. Conversely, item metadata is not restricted.
- Embargo terms should be visible to all.
- The imposition of terms may be performed by the submitter, or by adminstrative configuration.
- The preceding 3 findings argue prima facie for the use of

```
Item
```

metadata field or fields to encode the terms, since their display and input is already built into DSpace.

Design

The lifecycle of an embargo may be roughly divided into the following phases:

Determine Embargo & Assign Terms

For an embargo to occur, the item must be determined to be eligible and a specific policy (terms) attached to it. This may occur in a variety of contexts, e.g. the submission UI, in metadata passed through a SWORD deposit manifest or ItemImport batch, etc. Also, the terms themselves may vary: from a selection in a controlled vocabulary ("3 months", "6 months", etc), to a specific date. Given this multiplicity, the design places no restrictions on how embargo eligibility is determined, nor on how policy is expressed ('encoded') in a metadata field, except that a single configurable field be used for this purpose
(the name of the field can be placed in 'dspace.cfg', e.g).

Impose Embargo

This process occurs only when an eligible Item is

```
installed
```

into the repository (technically, an

```
inArchive
```

flag is set).
The process consists of two steps:
1. Interpretation (decoding) of policy terms. This means taking the 'encoded' representation of the terms and - together with any other applicable factors like which collection the Item is being installed into - assigning a specific date upon which the embargo will be lifted (the 'lift date').
2. Assigning a set of Resource Policies to the Items bitstreams consistent with the policy. Typically, only administrative access it allowed, but exception may be may made for the submitter (e.g.).

Since the design does not dictate the term encoding rules, it must allow a user-configurable means to decode them. Thus we declare an interface - call it

```
SetEmbargo
```

, whose configured implementation is invoked whenever an Item is installed.

```
SetEmbargo
```

implementations are responsible for performing both the interpretation (decode) of terms and assignment of Bitstream resource policies. A reference implementation will be provided that accommodates the simple interpretation cases (exact date, controlled list), but implementers are free to extend, alter or rewrite the functionality entirely.

Revise Terms

After an embargo is in effect, it's terms may be adjusted. This is considered to be an administrative action, and is thus available only to actors with administrative access to the collection and Item (the DSpace administrator, and collection administrator). It is assumed that since the terms have been 'decoded' into a standard formatted lift date, the change will be to another lift date (not encoded terms), and obey the restriction that the revised date be no earlier that the earliest of: (a)time of revision (b)current lift date.

Lift Embargo

Another interface -

```
LiftEmbargo
```

will manage the lifting of the embargo. This will typically involve instating a set of ResourcePolicies that non-embargoed Items obtain when installed, but again implementers will be free to customize it's logic. As with

```
SetEmabargo
```

, a reference implementation will be provided, together with a script suitable for use in cron-scheduled invocation.

Prototype Implementation

Here is the first version of a prototype implementation, as patches against the 1.5.2 source:

Downloads

1. Embargo-1.6-new.zip - new files (including Harvard-specific plugins as examples)
2. Embargo-1_6-diff.txt - diffs to existing source
3. Embargo-extra-diff.txt - another diff to existing source, to correct bug uncovered by fixes to DCDate logic.

In addition to these files, you will need the following patches from JIRA:

1. DS-254
2. DS-255

Procedure

These commands are intended for a Unix-like environment such as MacOS X, Linux, Solaris.

Given an existing checkout of the DSpace 1.5.2 source,

1. cd to the root of the checkout (e.g. the directory above
   ```
   dspace-api
   ```
)
2. unzip -o embargo-1.6-new.zip
   - This will overwrite DCDate.java, that is expected.
3. patch -p0 -l < embargo-1.6-diff.txt
4. Apply the diffs in DS-254 and DS-255

Note that the files in the

```
dash-api
```

subdirectory are provided for *reference only*,
as an example of a plugin implementation. They are not included in the build.

Build and deploy DSpace as usual.

Configuration

Add these lines to your

```
dspace.cfg
```

configuration file:

```
## ------- Embargo configuration

# DC metadata field to hold the user-supplied embargo terms
embargo.field.terms = SCHEMA.ELEMENT.QUALIFIER

# DC metadata field to hold computed "lift date" of embargo
embargo.field.lift = SCHEMA.ELEMENT.QUALIFIER

# string in terms field to indicate indefinite embargo
embargo.terms.open = forever

# implementation of embargo setter plugin
plugin.single.org.dspace.embargo.EmbargoSetter = CLASSNAME

# implementation of embargo lifter plugin
## plugin.single.org.dspace.embargo.EmbargoLifter = org.dspace.embargo.DefaultEm
plugin.single.org.dspace.embargo.EmbargoLifter = CLASSNAME
```

In practice, you must substitute a metadata field name for each

```
SCHEMA.ELEMENT.QUALIFIER
```

, and a class properly implementing each plugin for

```
CLASSNAME
```

.

The *embargo terms* field may be the same as the *lift date* field – in such a case, when the embargo is set, the value of the terms field gets deleted and replaced by the lift date.

Periodic Task

You must run the *embargo lifter* task periodically to check for Items with expired embargoes and lift them. It is a command-line application invoked with the command:

```
dspace/bin/dsrun org.dspace.embargo.EmbargoManager
```

Run it with the

```
--help
```

option to get a list of options.

We recommend running this once per day.