DSpace 2.0 Pluggable Storage

Pluggable Storage

Considerable thought, discussion, design, prototyping, etc has focused on what a DSpace+2.0 asset store should look like. In particular, whether and how such an asset store should model archival information packages (AIPs) in a more substantial manner than in the current architecture. This model is conceived as a DSpace asset store API/interface. Of course one big advantage of such an interface is the ease of writing implementations for different storage systems (file-based, grid, etc) and plugging them into DSpace. However, it should be noted that this benefit redounds to just having an interface, not having an AIP-aware one in particular.

Modeling an AIP asset store is very important (and hard), and it has proven difficult to achieve consensus - and this has led to holding the benefits of pluggable storage for DSpace hostage to agreement on an AIP model. The work this page describes attempts to circumvent this problem by refactoring the existing asset storage system to drive a very thin API wedge between the storage manager and the actual storage back-end. It conspicuously does **not** attempt to model an AIP, only the low-level storage primitives. If successful, it should make it far easier than it is today in DSpace to attach different storage solutions. It should also feed the AIP asset store design process by providing insights into the powers and limits of different storage systems.

I call this API a **BitStore** to emphasize that it is not an AIP model, and have provided a refactored **BitstreamStorageManager** that utilizes this interface, rather than the direct calls it had into the file-store or SRB store. In addition, I have provided 3 implementations of the interface:

- DSBitStore - this is simply the current DSpace file system store.
- SRBBitStore - the existing SRB store.
- S3BitStore - an asset store using Amazon's Simple Storage Service. NB: this is a commerical (not free) service

Another advantage of this approach is modularity: we will no longer have to include all the code and required libraries for (e.g.) Storage Resource Broker unless we actually want to use it. These storage modules also present some new use-cases for the Add-on mechanism work, since they are optional modules, but not separate applications.

Detailed notes on using each store will follow. Note that this is prototype code, **not** production quality. Feedback welcome, including other possible store implementations (e.g. a RBDMS store). (See the Discussion page for some thoughts on an RDBMS implementation.)

Installation and Configuration

- Download the interface and refactored storage manager, and cleanup:
  - BitStore.java
  - BitstreamStorageManager.java
  - Cleanup.java
  - Add all files to the DSpace source tree at
- ```
  org/dspace/storage/bitstore
  ```

  . The storage manager and cleanup will replace the existing source files.

- Create a new source directory

  ```
  org/dspace/storage/bitstore/impl
  ```

  for BitStore implementations.

- Download the implementation or implementations you want to use, and place into above directory:
  - DSBitStore.java
  - SRBBitStore.java
  - S3BitStore.java

- Install bitstore dependencies:
  - DSBitStore has none
  - SRBBitStore requires jargon.jar and a broker instance. The former is part of the standard DSpace distribution, so no further requirements
  - S3BitStore requires the *jets3t* java library (and its dependencies) available at

    ```
    http://jets3t.s3.amazonaws.com/index.html
    ```

- Place all required jars in

  ```
  dspace/lib
  ```

- Configure bitstores for use in DSpace:
  - each 'assetstore' number in

    ```
    dspace.cfg
    ```
    ```

can be assigned to an instance of a bitstore as follows:

```
assetstore.<n>=<prefix>:<config>
bitstore.<prefix>.class=<fqcn>
```

- where inputs are:
  - 'n' is the store number, from 0 to any number, sequentially.
  - 'prefix' identifies a bitstore - e.g. 'ds' (dspace native), 'srb', 's3', etc. You can define any prefix you like.
  - 'fqcn' is a fully qualified class name of the bitstore implementation. For example,

    ```
    org.dspace.storage.bitstore.impl.DSBitStore
    ```

  - 'config' is the configuration string passed to the bitsore at initialization.
    - For the DSBitStore, it is just the asset store directory, i.e. what was in

      ```
      assetstore.dir=
      ```

      in current

      ```
      dspace.cfg
      ```

    - For S3 and SRB bitstores, it is the path name of the configuration file, e.g.:

      ```
      $\{dspace.dir\}/config/s3.properties
      ```

- Configure bitstores internally:
  - DSBitStore requires no configuration
  - SRBBitStore should have a config file with all the values currently in

    ```
    dspace.cfg
    ```

  - S3BitStore should have a config file with the properties:
    - access.key=
    - secret.key=
  - both values are obtained when you establish an account with S3

</html>