DRS2 APIs Harvard Library Lab Project – 11/1/13 Final Report

Spencer McEwen, Randy Stern, Chip Goines

Overview

The Digital Repository Service 2 (DRS2) APIs project was designed to explore and prototype 2 types of Application Programming Interfaces to the new DRS2 system:

1. Public DRS metadata APIs
    a. The ability to search for objects and files in the DRS
    b. The ability to access the metadata for objects and files in the DRS
2. Public content delivery APIs
    a. An enhanced API for the DRS Image Delivery Service to support the International Image Interoperability Framework image API.

Substantial progress was made on the content delivery API (~80% of the IIIF Level 1 API is functional). Significant progress (~40% of the work) was also completed on the DRS metadata API, however we were not able to complete a test deployment, expose it for use by stakeholders, and obtain and incorporate feedback. The blocking factor on the DRS metadata API was that the key software developer leading the project was assigned by library management to another very high priority project during this period. Hence, all of the funds allocated by library lab to this project were not used.

The metadata API project will be an important part of the DRS2 in the future, and we would like the opportunity to re-open this project during the Library Lab showcase year to complete the work.

Work to date

This project was planned as follows:

Phase	Planned Start date	Planned End date	Status
1. Planning and scoping	11/1/12	3/1/13	DONE
2. Technical design, including authentication/authorization for access to restricted DRS resources	5/1/13	8/1/13	Not complete

3. Implementation –Implementation of search, metadata read, and content read APIs, including authentication /authorization for access to restricted DRS resources. Development work will utilize existing LTS development server. *NOTE: This estimate assumes Spencer does the API development work and we use the contractor to backfill on his DRS2 work.*	8/1/13	12/1/13	Not complete
4. Implementation of content delivery API that conforms to the International Image Interoperability Framework for the DRS Image Delivery Service.	6/1/13	9/1/13	Partially complete

Phase 1 – Planning and Scoping

- A strawman "Public API" for publicly accessible content and metadata in DRS2 was defined. See appendix 1.
- A wiki site was constructed (https://wiki.harvard.edu/confluence/display/libdrsapi/DRS+API+Project) to document the vision for the project, collect user stories and requirements for the APIs, and outline the evolving API specification
- 3 stakeholder meetings were held to gather input – one with Library Technology Services staff, and 2 with a range of potential consumers of DRS APIS – Library Imaging Services, Berkman Center, Harvard Law Innovation Lab, HUIT Academic Technology Services, Schlesinger Library, Medical Informatics, and DataVerse Network developers. Feedback influenced the initial proposed design:
    - The initial proposal was for a public-only API. There were substantial Harvard staff requirements to be able to search and access entire collections, including "restricted to Harvard" data. So, we plan to add optional HTTP authentication to the API, as well as an API access key. See appendix 2.
    - There were requests to be able to access master content files, such as TIFF or JP2 images, so that these could be delivered to end users for certain collections or so that special delivery systems could be created. This will be enabled for authenticated and authorized API users.
    - There were requests for the ability to bulk download DRS metadata and content. This could come in the form of an OAI data provider. Since DRS2 content and metadata are very large, and a program that iterates over a list of content could effectively produce a bulk download, we decided to defer this requirement.
    - There were requests for a IIIF API for image content delivery. We have included this in the proposal as a separately fundable phase.

Phase 2 and 3 – Technical design and implementation of the metadata API

Due to limited availability of Spencer McEwen who was assigned by library management to other high priority library work, some work has been done on Phase 2, but the majority of work on phase 3 was not completed before the ending of library lab funding in November.

The basic framework for a software layer to expose public DRS2 APIs has been designed, and software implementing it has been prototyped. The initial focus was on APIs for content owners. A software prototype was designed and implemented. The API prototype includes authentication and authorization, which was requested by stakeholders during our planning meetings. The software source code has been checked into the Library Technology Services source code control system. (The prototype is not currently operational, but with a small amount of work it could be restored, and an owner of content who wished to exercise the API could do so.) Access to DRS metadata is API limited to content within owner codes that an authenticated user would normally be able to access in DRS2. Authorization is based on roles stored in our existing policy authentication database. A basic set of object, file, and search API calls was implemented to retrieve and search metadata and download content files. XML or JSON is returned depending on the Accept HTTP header value. Using this API a content owner can get access to the full metadata and file contents of all their content in DRS2.

Work was also begun on a public version of the API that could be used to access public data in DRS2 (access flag = P) by non content owners. In the current prototype, object and file metadata can be retrieved without passing through the same authentication/authorization channels as content owners. Content access flags and other rights restrictions (embargo dates, secure storage, max image size, stream vs download) are also checked. While getting this working several questions were raised. More analysis is needed to determine which metadata fields should be exposed in the public API.

If we are able to continue the project during the showcase year, remaining work includes:

- implement rate limit throttling for authenticated and non authenticated requests
- decide if current metadata format for authenticated requests is sufficient -implement addition API methods for object and file metadata access
- implement addition API methods for public/non authenticated requests, implement a public search API.

Now that the basic software framework of the application has been designed and implemented, adding additional API methods should be relatively straightforward. With a week's work, the app could be made available for testing by external DRS2 content owners. The next immediate steps would be to finalize the authenticated API call and data response formats.

Implemented APIs

Both private and public APIs were prototyped by designing, implementing, and deploying the software to a test server at Library Technology Services. All APIs support content negotiation, and return both XML or JSON when requested and appropriate. Some complex return values are only supported as XML, for example MODS..

Private URLs restricted by authentication:

- /object/{id}
  - Returns object metadata for the supplied ID
- /object/{id}/mods
  - Returns descriptive metadata for the supplied ID as mods XML
- /file/{id}
  - Returns file metadata for the supplied ID
- /file/{id}/mix
  - Returns technical metadata for the supplied image ID, in MIX format
- /file/retrieve/{id}
  - Returns the file itself for the supplied ID

Public URLs that check premis rights and access flags:

- /public/object/{id}
- /public/object/{id}/mods
- /public/file/{id}
- /public/file/retrieve/{id}

These all can send back XML or JSON when asked for and appropriate. For example asking for MODS as Json isn't supported.

Phase 4 - Implementation of content delivery API that conforms to the International Image Interoperability Framework for the DRS Image Delivery Service.

Significant progress was made on Phase 4. A prototype IIIF API for JPG 2000 images delivered though the existing DRS Image Delivery Service (IDS) was developed and deployed to production. The API has been tested with the Stanford University IIIF validation server, and the validation server reports that the code is now IIIF Level One compliant. At the completion of the library lab period, we have begun testing whether the Level One compliance we have achieved will be adequate for integrating into the Mirador multi-Image manuscript viewer being developed at Stanford University as part of the Digital Medieval Manuscript Interoperability project. http://lib.stanford.edu/iiif

At present the IIIF API has been both prototyped and deployed to the production Image Delivery Service.

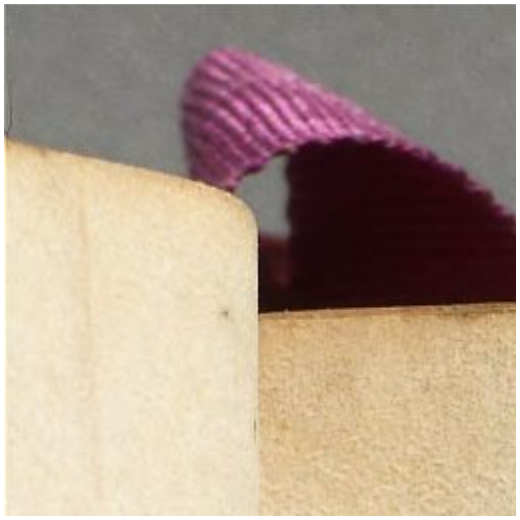
Example production URLS for the image API:

http://ids.lib.harvard.edu/ids/iiif/5981214/full/full/full/native.jpg

returns a full image using the IIIF API

http://ids.lib.harvard.edu/ids/iiif/5981214/0,0,256,256/full/full/native.jpg

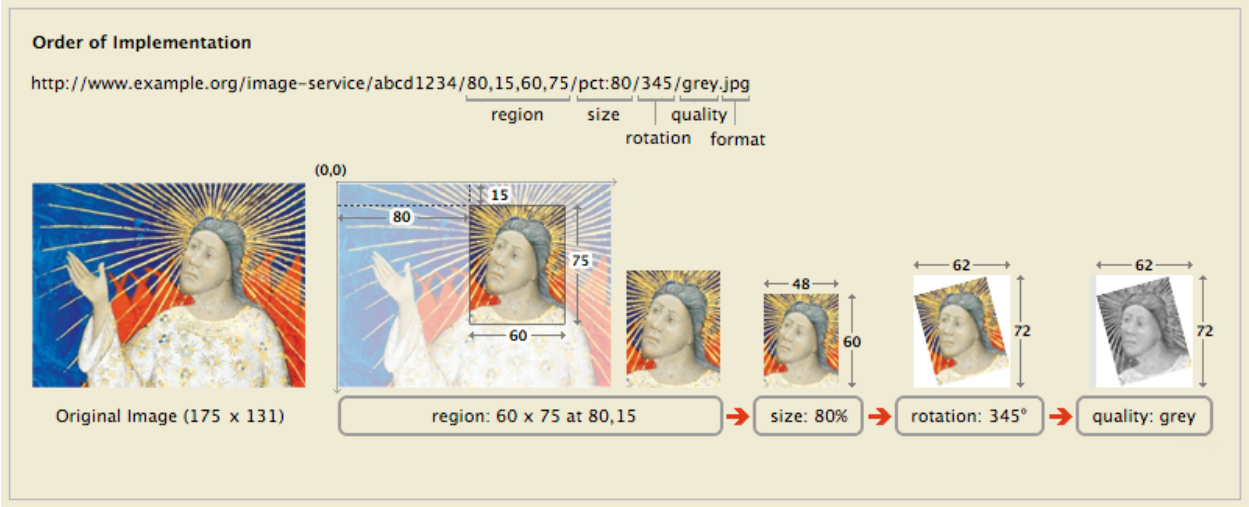
will return a 256x256 image tile from the upper left corner of the image tile using the IIIF API.



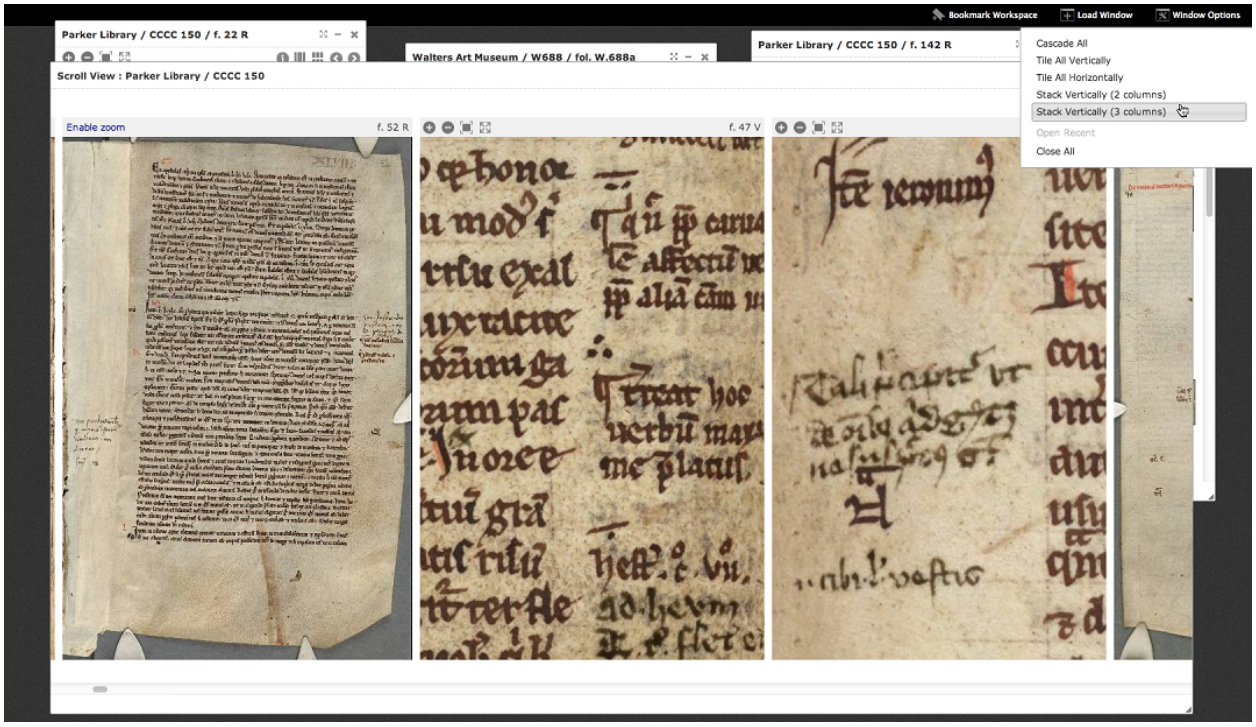
http://ids.lib.harvard.edu/ids/iiif/5981214/full/0,0,150,150/full/native.jpg

will create a thumbnail.

The API slows selection of a region of the image that has been scaled and rotated as necessary (see the example image from the IIIF documentation below.). There are still a few bugs involved in our implementation in the translation from IIIF parameters to the underlying Luratech image server, but we believe these can be fixed.



A goal is for the IIIF implementation to allow any external application that wishes to re-use a Harvard public image to embed that image using the IIIF API. In particular, we are testing with the Stanford Mirador image viewer, an open source project that promises to be the a platform for interoperable access to images from multiple institutions in the same viewer simultaneously. See a Mirador screen shot below (not yet with Harvard supplied images.)



Appendix 1 – Strawman API

This original strawman from early in the project provides some detail about possible options for fleshing out the existing DRS2 API prototype.

The API is RESTful. It will support content negotiation: API calls may return alternate MediaTypes as requested, such as "*application/xml*" and "*application/json*".

All APIs listed here are subject to change during the design phase/

/drs/apiV10/searchMetadata/<query>

Item types	objects, files, batches, events
Input	expose solr queries directly (with some field renaming)
results schema	solr document (with some field renaming and redaction)
results formats	XML, JSON, CSV

/drs/apiV10/searchContent/<query>

Item types	full text OCR data for page turned objects
Input	keyword strings, much more (see FTS API Spec)
results schema	Full Text search service XML format, including links to specific pages in PDS documents that match the query text (see FTS API Spec)
results formats	XML, JSON, CSV

/drs/apiV10/getMetadata/<URI>|<drsID>|<delivery URN>

Item types	objects, files
Input	URI, DRSID, or delivery URN
results schema	DRS specific, (with some field renaming and redaction)
results formats	XML, JSON

Issues

We May add helper APIs as well, such as:

1. getMetadata/<object or file URI> (essentially the entire METS object for objects)
2. getAdminMetadata/<object or file URI>
3. getDescriptiveMetadata/<object or file URI>
4. getRightsMetadata/<object or file URI>
5. getEventsMetadata/<object or file URI>
6. getTechnicalMetadata/<file URI>
7. getStructure<object URI>
8. getFileList<object URI>

getFileContent/<URN>|<file URI>

Item types	files (Access Flag=P only)
Input	URN, file URI
results schema	file format dependent – IIIF for images
results formats	file format dependent, deliverable formats only (e.g. no JP2, JPEG only at max image size)

Appendix 2 – API Security

Authentication

The DRS2 API will make a distinction between anonymous public users and authenticated users. Authenticated users have full access to their content, including the metadata and original quality images. Anonymous public users would be restricted to certain metadata fields and content with a P access flag. Max image sizes and audio stream/download rules would also be enforced.

Several options exist for authentication

1. Basic HTTP authentication. LTS will create and store api_key = app_key; api_secret_key = app_key_pass for each registered application. These can be passed in via HTTP headers or as query string parameters
2. Query authentication. Sign query string parameters - LTS creates app keys and passwords. Query string must be ordered, include a timestamps, encrypted with the password, then sent to the API.
3. Client certificate authentication. We would need to create and distribute certificates. Jboss validates the client, the API code reads the api_key (or equivalent) from the certificate.
4. Use an API key only?

HTTPS will be required regardless. LTS will need to store the api key and password secrets. Example) https://www.stormpath.com/docs/api/authentication

Ideally the DRS2 Web Admin would allow a user to regenerate their API secret key.

Since all requests will be over HTTPS then option 1, basic HTTP Authentication should provide sufficient security and be easy for any client to use.

Authorization

Applications must be registered and associated with the specific DRS2 owner codes that they should be able to access. The existing LTS Policy service may be able to store the application ID, owner codes, and DRS2 API role.

API Rate Limiting

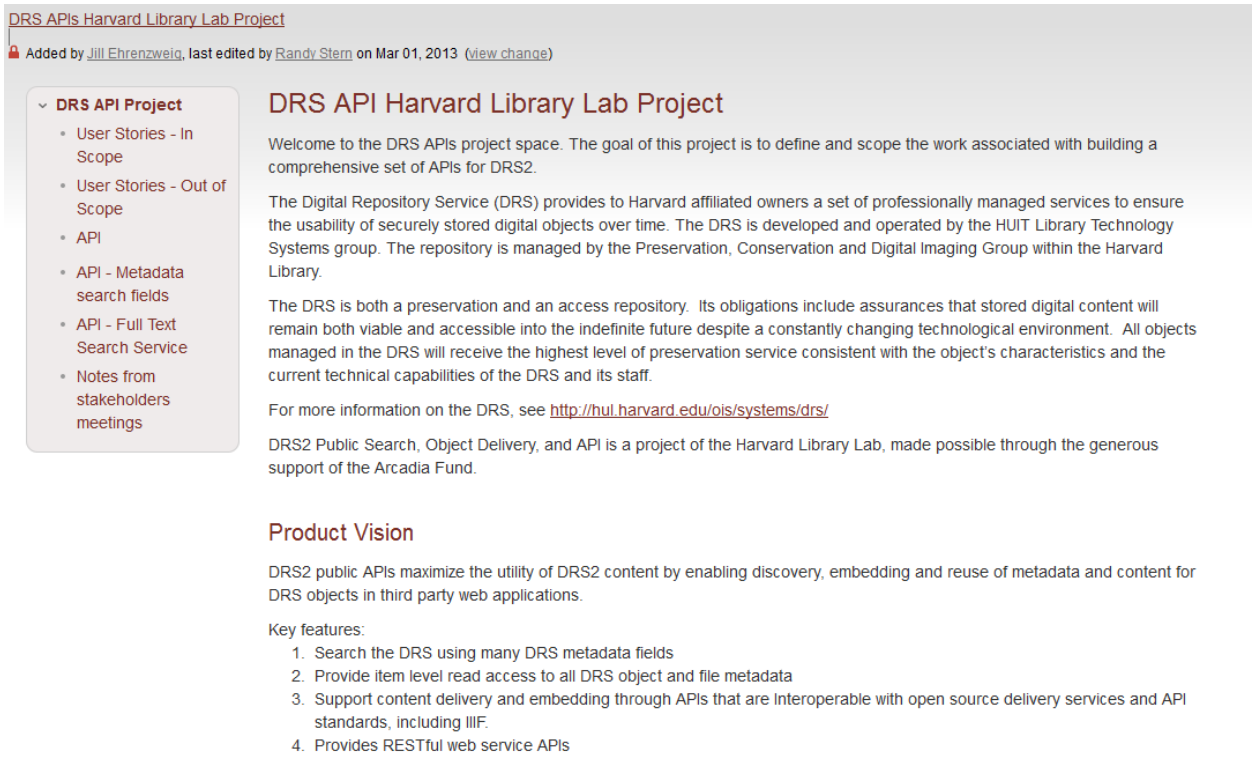
Several options exist for rate limiting API access:

1. Allow authorized users to have x requests per minute or hour. Unauthorized (public) users would be tracked by IP address and have a lower number of requests.
2. Use a leaky bucket algorithm: http://stackoverflow.com/questions/1375501/how-do-i-throttle-my-sites-api-users Authorized users would be allowed a certain rate, unauthorized (public) users would be tracked by IP and be allowed a lower rate.

To accommodate future load balancing these values need to be either tracked in a database or a distributed memory cache (EHCache, Redis, Infinispan).

Appendix 3 – Wiki Page

The wiki is at https://wiki.harvard.edu/confluence/display/libdrsapi/DRS+API+Project



DRS APIs Harvard Library Lab Project

Added by Jill Ehrenzweig, last edited by Randy Stern on Mar 01, 2013 (view change)

- **DRS API Project**
  - User Stories - In Scope
  - User Stories - Out of Scope
  - API
  - API - Metadata search fields
  - API - Full Text Search Service
  - Notes from stakeholders meetings

DRS API Harvard Library Lab Project

Welcome to the DRS APIs project space. The goal of this project is to define and scope the work associated with building a comprehensive set of APIs for DRS2.

The Digital Repository Service (DRS) provides to Harvard affiliated owners a set of professionally managed services to ensure the usability of securely stored digital objects over time. The DRS is developed and operated by the HUIT Library Technology Systems group. The repository is managed by the Preservation, Conservation and Digital Imaging Group within the Harvard Library.

The DRS is both a preservation and an access repository. Its obligations include assurances that stored digital content will remain both viable and accessible into the indefinite future despite a constantly changing technological environment. All objects managed in the DRS will receive the highest level of preservation service consistent with the object's characteristics and the current technical capabilities of the DRS and its staff.

For more information on the DRS, see http://hul.harvard.edu/ois/systems/drs/

DRS2 Public Search, Object Delivery, and API is a project of the Harvard Library Lab, made possible through the generous support of the Arcadia Fund.

Product Vision

DRS2 public APIs maximize the utility of DRS2 content by enabling discovery, embedding and reuse of metadata and content for DRS objects in third party web applications.

Key features:

1. Search the DRS using many DRS metadata fields
2. Provide item level read access to all DRS object and file metadata
3. Support content delivery and embedding through APIs that are interoperable with open source delivery services and API standards, including IIIF.
4. Provides RESTful web service APIs