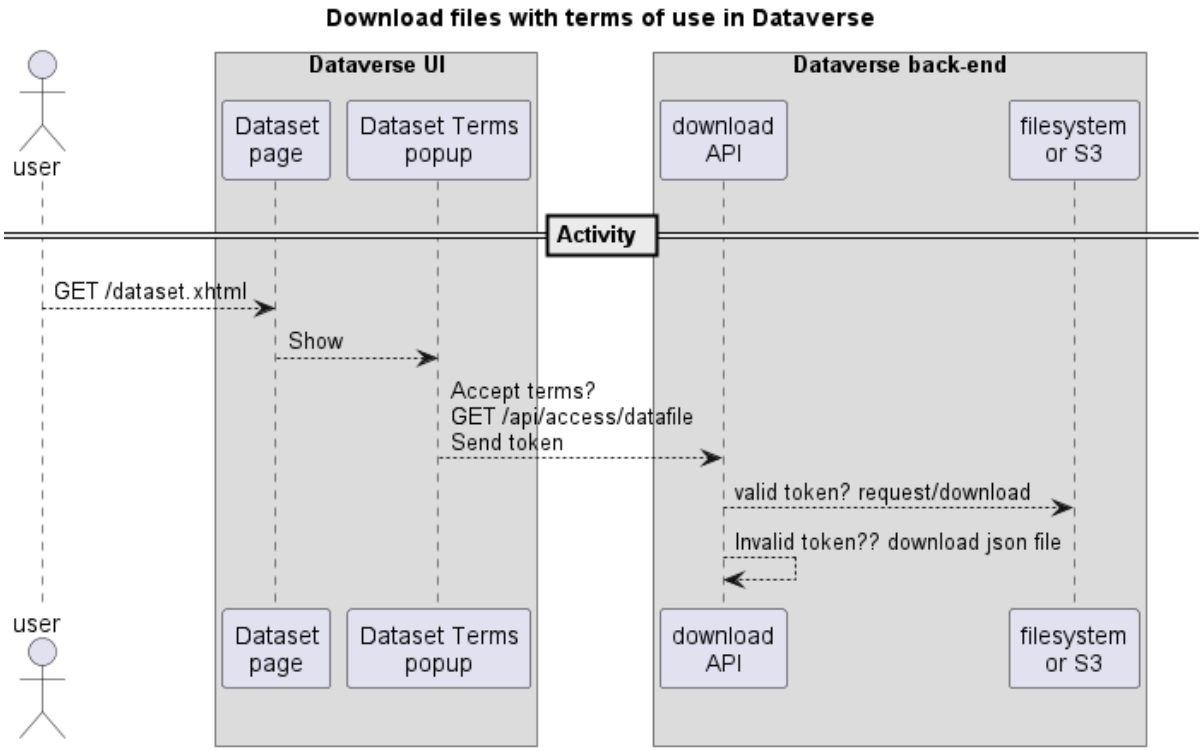

Assumptions:

- You can create an API token only if you have a user on a Dataverse instance. At least we have each user's last name, first name and email address. Desirably the Affiliation.
- Using the API, anyone can download files with no access restrictions.
- If someone uses an API Token, we could know the user associated to that token, do not?
- As a user, when you request access to a restricted datafile you must accept the Terms of Access for Restricted Files.

CIMMYT Proposal - File download:

- Proposed changes are highlighted in italics.

Terms of use:

a. As a user, when you request access to a restricted datafile you must accept the Terms of Access for Restricted Files and its Terms of use.

b. Since not all institutions may require these restrictions, we propose adding a global setting to disable this new functionality.

File download flow

- If its dataset has no Terms of use & the datafile has no access restrictions:
    - No changes.
- If its dataset has no Terms of use & the datafile has access restrictions:
    - No changes, an API token is required.

- If its dataset has Terms of use & the datafile has no access restrictions:
    - When a bot or user attempts to download a datafile directly from the API, they will not download the datafile itself; instead, they will download a PDF or TXT containing all the metadata of the datafile and the data from user or bot attempting the download: User agent, IP, Date and Time.
    - Additionally, a message will be added to the file similar to: "If you wish to download the datafile [datafile name], please go to [Persistent Datafile URL]."
    - At the end of the file, a text will also be added mentioning that the datafile is subject to usage restrictions and explicit approval is required.

- If its dataset has Terms of use & the datafile has access restrictions:
    - An API token is required.
    - *No changes, see section A of this block.*

Notes from 2024-05-14 meeting:

- acceptTerms=true is not sufficient
- signed URLs?
- What should the SPA do? Should it use signed URLs or just redirect to the (currently open) download API.
- CIMMYT wants all users to go to the UI to accept terms of use.
- CIMMYT wants guest users to continue to download data
- CIMMYT is not sure how they feel about a future Python script set to auto-agree to all terms of use. (This could be done now using Selenium to drive the UI.)
-

Requirements:

- When UI users are shown the license/terms and must accept before downloading, it should not be possible to copy/share the raw download URL which could then be used to access the file without being aware of the terms

- Users trying to access a file via URL/API call (same thing, but API users are potentially making other API calls), there should be some mechanism to assure that the license/terms have been presented to them and that they have agreed - at a level equivalent to the click-through process in the UI
- It should be clear when a user tries to access a file whether they have succeeded (and now have a copy of the file) or have been denied/redirected to accept terms.
- UIs, such as the new SPA, that interact with the Dataverse backend solely through the API, should be able to mimic the click-through process in the current UI (i.e. the API has to support this)
- The mechanism should cover Dataverse's multiple file download mechanism as well (correct?)
- Current UI and SPA, which do/will provide a click-through agreement prior to download should be minimally impacted, i.e. the user should not have to go through additional steps.

Questions

- The current UI does not show a click through agreement when the default license (often CC-0) is used. Is this acceptable for URL/API access as well?
- Should this requirement apply to restricted files as well? (Nominally for a restricted file, some has requested and been granted access, but that doesn't mean they've seen the terms, e.g. if they've used the API for requesting access. That said, they are more clearly aware of the source and, prior to granting access, the admin/curator could make sure they've agreed.)

Technical Considerations:

- Designs that essentially add an ?accept=true flag to the URL are not sufficient as it is too easy to just copy/share the URL with that added flag
- None of this can stop a user from uploading the file to some other service and pointing to that URL instead.
- Robots will be unaware of any mechanism we create which would mean they will not have access to file contents. (We do advertise download URLs in headers right now although they aren't visible in the page.) That could be desirable, or a problem.
- We include download URLs in metadata exports (which nominally include license/terms info)

Design Options:
There are ~3 phases to address and options for any given phase might be usable with several options in another phase:

Step 1: What happens when a user requests a file and hasn't seen/accepted the terms:

- Nominally they should receive an error message (otherwise they could assume they retrieved the file). For an HTTP call, that could be:
    - 401 - Unauthorized, though this usually means you need to send some form of authorization (username/password) rather than not meeting some other condition
    - 403 - Forbidden, though this often means that nothing can be done/the user has authenticated and just doesn't have permission.

    - 400/409 - Bad Request/Conflict - these both indicate something is wrong with the request in some way: 400 is mostly for malformed requests but 'hasn't indicated that terms are accepted' might be a valid 400 error. Alternately 409 is more about a conflict with the state of the server, in this case the server doesn't know if you've accepted terms.
    - 451- Unavailable For Legal Reasons - usually used for censored material but with an explanation might be relevant for not having accepted the terms.
- Alternately, could there be a /requestDownloadURL API call that, if you've signed terms, just redirects to the file, and if not, gives you an HTML or JSON response depending on the Accept header? This might avoid the confusion of not getting the file in some cases while possibly making it easier to provide a human or machine readable format (as you could/would send an appropriate Accept header whereas calling a file download URL you'd expect the Accept to be whatever type the file is). The mechanisms in step 3 could be used to decide whether the response includes the redirect or not? (Could also not send an HTTP redirect but just provide a protected download url, but this would complicate current use from the UI.)
- In the terms-not-yet-accepted case, The response message should contain information about the issue and how to resolve it. That could include:
    - Explanatory text
    - The License name/URI or a link to the terms pane of the dataset page
        - Providing a link could allow the user to use an "Accept" header when accessing it, which could allow a user to go to web version or text/pdf etc. while a script could request "application/json", etc.
    - The license text and terms text in some format(s)
    - A working link to get the file, or
    - A link to some other resource/API the user should visit to accept the terms and download the file(s)
- Nominally the response message should be human and machine readable or configurable (as in the alternative with /requestDownloadURL above)
- To be consistent with the UI/Dataverse's internal logic, a dataset will either have a license (from the configured list for a given Dataverse instance) or will have 'Custom Terms' which consist of some entry in the "Terms of Use" field and optional entries in the following fields. The response should include all of this information (or the link to the terms pane of the dataset where they will all be displayed):
    - Confidentiality Declaration
    - Special Permissions
    - Restrictions
    - Citation Requirements
    - Depositor Requirements
    - Conditions
    - Disclaimer

Step 2: How does the user accept the license/terms?

- The user could be forced to use a browser to view the license/terms and click accept
- The user could be required to make a separate api call to some form of /api/dataset/<i>/acceptTerms?answer=true endpoint

- The message could include a URL, protected by some means (see Step 3), that allows a file(s) download

Step 3: How do they then retrieve the file?
The primary goal of the whole process is to assure that the user making the final request to get the file(s) (i.e. one that will succeed) is the same person that made the earlier request and went through steps 1 & 2). There are a few options that could enforce this:

- A session cookie - sessions are a standard mechanism to track history across multiple HTTP calls. In this case, the error response in step 1 could include a session cookie that indicates the user has made that HTTP call and received the response with the license/terms info.
    - The session could directly authorize the user to retrieve the file(s), e.g. including the session cookie when using the file download URL would authorize access because the session includes some termsSeen=true statement.
    - Alternately, the user could also be required to go to another acceptTerms endpoint which would add an 'accepted=true' statement to the session, which would then be sufficient to authorize file download.
- SignedUrls could be used. Signed URLs in general are a standard concept. Dataverse has a specific implementation. In essence, signing adds a goodUntil date/time to a URL along with a cryptographic hash that allows Dataverse to be sure that the URL has not been changed by the user. When the user goes to a signedURL (in the browser or using curl or some other tool to call the API), Dataverse checks both the hash and that the URL has not timed out. Nominally one can consider this a token that is good only for the specific file download request and only for a few minutes (hence it is only sharable for that short time period). (The advantage of signedURLs for Dataverse is that they are scalable - there is no memory/database entry per signedURL that has to be kept (and discarded when it is used/after some time if it is never used)).
    - A signedURL could be delivered to the user in the initial error message
    - A signed URL could be delivered in response to the user calling an /acceptTerms endpoint
    - SignedURLs could be sent at both steps, assuring the the user had been sent the terms in step 1 before they called the acceptTerms API and retrieved the signed download URL
- A new token mechanism could be developed. Similar to the existing API token, this mechanism would generate tokens and manage them in memory/database. However, unlike API tokens, these would be short-lived or one-time-use, specific to a given dataset, and not associated with a given user. FWIW: There are some examples in our code where we use in memory caches that automatically remove tokens after a given time period or first-use, which simplifies getting rid of tokens after use/after they timeout. The options would essentially be the same as with signedURLs - tokens could be minted in response to the initial blocked download request and/or upon calling an /acceptTerms API call.

Discussion

- With any of these mechanisms, the current UI could generate the appropriate credential (statements in the session, signedURL, token) to send with the download URL, so the process would not change for existing users.
- Nominally the SPA can work with any of these mechanisms as well. The initial error response could be what indicates to the SPA that it should show the user a click-through agreement. (E.g. the SPA initiates download and for a CC-0/default license case, it works. For other cases, the error response is sent and the SPA uses the (hopefully JSON) contents to create a click-through dialog.)
- Sessions are nominally the most standard mechanism for keeping state between HTTP calls, but they are not used as much with APIs. Tools like curl don't make session info obvious and it may be harder to explain to users what they must do. (Dataverse currently ignores sessions in the API unless that feature is turned on. With it on, users who log in can then use the API as an authenticated user. If we use it here, their use for authentication would probably end up being turned on as well.)
- SignedURLs and tokens would be similar. SignedURLs exist and are used elsewhere. Tokens would be new but probably not hard to build. They would allow one-time-use rather than just a short timeout (at the expense of requiring per token memory use). Both would be things that could be cut/pasted easily (signedURLs would be longer overall)
- Forcing the user to go to the browser could be problematic for pure API use. (It can be done. OIDC login with API use, e.g. for Globus does this - you make an API call, get a URL, go to the browser, login, and get a token to add to your future API calls. It's a similar mechanism to register your TV for Netflix, etc.). The idea of having a /requestDownloadURL call could help in that with Accept:text/html it could display a form to click through which Accept:application/json could provide a machine readable format with an appropriate URL to do a non-browser acceptance. That seems harder to do if we return an error code on the existing download URL.

  Proposal 1:
  Add an /api/datafiles/<id>/requestDownloadURL method that:
  - if signed (and file is downloadable by the user), replies with a redirect to a signed version of the download URL
  - if not signed, responds with an HTML form or JSON structure showing the terms and asking the user to accept. Clicking yes on the form sends you to a signed version of the downloadURL
    - the HTML form includes Javascript to handle the onclick to send the user to the signed download URL (a way to avoid having to implement form handling in the server)
    - JSON would be similar - includes either the license URL or the terms field contents and a signed download URL in some "IAcceptTerms" element.
  - The current download API succeeds if signed and fails with a 403 and a URL to the /requestDownloadURL endpoint for the file
  - Current UI changes to redirect user to the signed /requestDownloadUrl. This design adds one redirect beyond the current setup but otherwise looks the same

- SPA would call /requestDownloadUrl w/o signature and decide whether/how to show terms based on the response.
- Include signatures on download URLs in dataset/file page headers (allowing robots/crawlers to download the files for a short time?)
- Don't include signatures in exported metadata files (as it is now, but causing any use of those URLs to fail.)
- Include a setting to allow this to be enabled/disabled on a given Dataverse instance.

Add a similar mechanism to protect the view-dataset-files-and-folders-as-a-directory-index, downloading-all-files-in-a-dataset, and multiple-file-bundle-download calls.

- These are needed to fully protect files but perhaps could be future PRs if the major problem today is with single file downloads.

Proposal 2:
Handle this through user education? Add a general Terms of Use for the site that prohibits sharing download URLs? Scan for posted URLs and request take-downs?

Proposal 3: