HTRC Data API Users Guide

Version	Status	Maturity	Comments
3.0.0	Released	Stable	CQL-based access instead of Hector-based access
1.0	Released	Released	
1.0.1-SNAPSHOT	Under testing	Release candidate	Added token count feature

Table of Contents

Synopsis

The HTRC Data API is a RESTful web service for the retrieval of multiple volumes, pages of volumes, and METS metadata documents. In order to support the efficient retrieval of volumes and pages in bulk, the Data API deviates from the typical RESTful API design out of necessity: Resources are not identified on the URL paths, but instead are sent as request parameters.

Use:

The HTRC Data API pulls full text OCR and METS metadata for specified volumes into the HTRC Data Capsules. You can download the full text of specified volumes using the HTRC Data API, with the volumeIDs of the desired volumes passed as parameters to the APIl. Volume IDs are standard identification numbers for items in the HathiTrust Digital Library. Currently, the HTRC Data API can only access a snapshot of public domain volumes.

API

Note: all parameter values must be URL encoded

Retrieve Volumes

Description	Returns requested volumes
URL	`/volumes`
Supported Response Types	`application/zip` (normal response) `text/plain` (error response)
Method	POST
Request Types	`application/x-www-form-urlencoded`
Request Headers	`` `Authorization: Bearer [JWT_TOKEN]` (replacing `[JWT_TOKEN]` with the valid token) ``
Request Body	Request parameters as body content. See Parameters below

Parameters

Name	Description	Type	Default value	Required	Note
volumeIDs	The list of volumeIDs to be retrieved.	string	N/A	yes	VolumeIDs are separated by the pipe character '|'
concat	The flag to indicate concatenation option.	boolean	false	no	See section on response format for details on its impact on the returned data
mets	The flag to indicate if METS document should be returned	boolean	false	no	
version	A specific version of the Data API to use	string	N/A	no	Not implemented. Place holder only

Responses

HTTP Status Code	Response Body	Response Type	Description
200 (ok)	A binary Zip stream	`application/zip`	Page content and metadata of the requested volumes aggregated as a Zip stream
400 (bad request)	`Missing required parameter volumeIDs`	`text/plain`	The required parameter volumeIDs is missing in the request
400 (bad request)	`Malformed Volume ID List. Offending token: ${token}`	`text/plain`	The value for volumeIDs is malformed and the Data API cannot parse it. `${token}` will be the token that causes the error.

Example

Description	Request for volumes **`inu.3011012`** and **`uc2.ark:/13960/t2qxv15`**, with concatenation option enabled so each volume is a single text file in the returned Zip stream.
Raw volumeIDs	**`inu.3011012`**`|`**`uc2.ark:/13960/t2qxv15`**
URL encoded request body	*`volumeIDs`*`=`**`inu.3011012`**`%7C`**`uc2.ark%3A%2F13960%2Ft2qxv15`***`&concat`***`=true`

Example Request

```
curl -v -X POST -o volumes.zip \
        -d "volumeIDs=uc2.ark%3A%2F13960%2Ft12n5fs57" \
        -H "Content-Type: application/x-www-form-urlencoded" \
        -H "Authorization: Bearer TOKEN" \
        https://sandbox.htrc.illinois.edu:25443/data-api/volumes
```

Note: The **TOKEN** placeholder in the example request needs to have a valid value.

Retrieve Pages

Description	Returns requested pages
URL	`/pages`
Supported Response Types	`application/zip` (normal response) `text/plain` (error response)
Method	POST
Request Types	`application/x-www-form-urlencoded`
Request Headers	`Content-Type: application/x-www-form-urlencoded`
Request Body	Request parameters as body content. See Parameters below

Parameters

Name	Description	Type	Default value	Required	Note
pageIDs	The list of pageIDs to be retrieved	string	N/A	yes	PageIDs are separated by the pipe character '**|**'
concat	The flag to indicate concatenation option	boolean	false	no	See section on response format for details on its impact on the returned data ⚠ "concat" and "mets" cannot be both set
mets	The flag to indicate if METS documents should be returned	boolean	false	no	⚠ "concat" and "mets" cannot be both set
version	A specific version of the Data API to use	string	N/A	no	Not implemented. Place holder only

Responses

HTTP Status Code	Response Body	Response Type	Description
200 (ok)	A binary Zip stream	`application/zip`	Page content and metadata of the requested pages aggregated as a Zip stream
400 (bad request)	`Missing required parameter pageIDs`	`text/plain`	The required parameter pageIDs is missing in the request
400 (bad request)	`Malformed Page ID List. Offending token: ${token}`	`text/plain`	The value for pageIDs is malformed and the Data API cannot parse it. `${token}` will be the token that caused the error.
400 (bad request)	`Conflicting parameters in page retrieval. Offending Parameters: ${param1}, ${param2}`	`text/plain`	Some request parameters have conflict. `${param1}` and `${param2}` will be the names of the parameters that caused the conflict. In the current version of the Data API, this is most likely caused by setting both "mets" and "concat" for page retrieval.

Example

Description	Request for the 1st, 2nd, 20th, and 30th pages of the volume **`inu.3011012`**, and the 11th, 17th, 22th, 30th, 45th, and 55th pages of the volume **`uc2.ark:/13960/t2qxv15`**, with each page being a separate text file along with the corresponding METS document of each volume in the returned Zip stream.
Raw pageIDs	**`inu.3011012[1,2,20,30]|uc2.ark:/13960/t2qxv15[11,45,30,17,22,55]`**
URL encoded request body	*`pageIDs`*`=inu.3011012%5B1%2C2%2C20%2C30%5D%7Cuc2.ark%3A%2F13960%2Ft2qxv15%5B11%2C45%2C30%2C17%2C22%2C55%5D&`*`mets`*`=true`

Token Count (Deprecated)

Description	Returns token counts of requested volumes
URL	`/tokencount`
Supported Response Types	`application/zip` (normal response) `text/plain` (error response)
Method	POST
Request Types	`application/x-www-form-urlencoded`
Request Headers	`Content-Type: application/x-www-form-urlencoded`
Request Body	Request parameters as body content. See Parameters below

Parameters

Name	Description	Type	Default Value	Required	Note
volumeIDs	the list of volumes to be token counted	string	N/A	yes	VolumeIDs are separated by the pipe character '`|`'
level	specifies whether the token counts to be aggregated at volume level or page level. Use "`volume`" for volume level, and "`page`" for page level	string	`volume`	no	
sortBy	specifies the token count output to be sorted on a fields. Use "`token`" for sorting based on the token's UTF-8 order, and "`count`" for sorting based on the token count order. If left unspecified, the results do not guarantee any orders.	string	N/A	no	Token ordering is based on UTF-8 character values, so character "Z" comes before character "a " (if using ascending ordering). For token count ordering, tokens with the same count are ordered by token's UTF-8 values.
sortOrder	specifies whether output to use ascending or descending ordering. Use "`asc`" for ascending ordering, and "`desc`" for descending ordering.	string	`asc`	no	this parameter only has effect when used together with `sortBy`, otherwise it is ignored.
version	A specific version of the Data API to use	string	N/A	no	Not implemented. Place holder only

Responses

HTTP Status Code	Response Body	Response Type	Description
200 (ok)	a binary Zip Stream	`application/zip`	Token count output aggregated as a Zip stream
400 (bad request)	`Missing required parameter volumeIDs`	`text/plain`	The required parameter volumeIDs is missing in the request
400 (bad request)	`Malformed Volume ID List. Offending token: ${token}`	`text/plain`	The value for volumeIDs is malformed and the Data API cannot parse it. `${token}` will be the token that caused the error.

Example

Description	Request for page level token count of the volumes **`inu.3011012`** and **`uc2.ark:/13960/t2qxv15`**, with the token count output to be sorted by the tokens in descending order
Raw volumeIDs	**`inu.3011012|uc2.ark:/13960/t2qxv15`**
URL encoded request body	*`volumeIDs`*=**`inu.3011012%7Cuc2.ark%3A%2F13960%2Ft2qxv15`**&*`level`*=page&*`sortBy`*=token&*`sortOrder`*=desc

Response Format

Zip Structure Layout

The directory structure layout of the Zip stream returned from the Data API may be one of the following patterns depending on the optional parameters:

> ⚠ Strictly speaking, inside a Zip file the structure is flat, so there is no "directories" but only file entries. However, in practice almost all Zip tools give the illusion of directories by leveraging the slash characters '/' in the name of each Zip entry. For the discussion here, we follow such practice and treat the inside of a Zip file as if it were a conventional filesystem.

Suppose there are 2 hypothetical volumes in the corpus: **`foo.001122`**, which has 5 pages, and **`bar.ark:/13960/t123`**, which has 3 pages. Both volumes also have the associated METS xml files. The client tries to request for these 2 volumes, and also tries to request for another volume **`gon.000000`** that no longer exists in the corpus, which would cause the entry `ERROR.err` to be included in the returned Zip strea.

Request Description	Zip Structure Layout	Explanation of Entries

Retrieve volumes
 	
`concat=false`
	`volumes.zip`
	
	`    |-- foo.001122/`
	
	`    |    |--00000001.txt`
	
	`    |    |--00000002.txt`
	
	`    |    |--00000003.txt`
	
	`    |    |--00000004.txt`
	
	`    |    |--00000005.txt`
	
	`    |    \--mets.xml`
	
	`    |-- bar.ark+=13960=t123/`
	
	`    |    |--00000001.txt`
	
	`    |    |--00000002.txt`
	
	`    |    |--00000003.txt`
	
	`    |    \--mets.xml`
	
	`    |--volume-rights.txt`
	
	`    \--ERROR.err`
		Because the request parameter `concat=false`, each volume has its own directory, and the pages and metadata documents of each volume are individual files stored under the volume directory.
		
		`foo.001122/` is a directory named after the first volume, **`foo.001122`**. The directory name underwent a Pairtree clean process, but since it does not contain any filesystem unsafe characters, the cleaned ID looks the same as the original.
		
		Inside `foo.001122/`, files `00000001.txt` through `00000005.txt` are the 5 pages of this volume
		
		`mets.xml` will also be inside of `foo.001122/` if the request parameter `mets=true`
		
		`bar.ark+=13960=t123/` is a directory named after the second volume, **`bar.ark:/13960/t123`**. The directory name underwent a Pairtree clean process so that filesystem-unsafe characters such as colons '**`:`**' and slashes '**`/`**' are escaped and replaced with filesystem-safe characters.
		
		Inside `bar.ark+=13960=t123/`, files `00000001.txt` through `00000003.txt` are the 3 pages of this volume.
		
		mets.xml will also be inside of `bar.ark+=13960=t123/` if the request parameter `mets=true`
		
		volume-rights.txt is a file at the top level. It contains the HTRC Data Protection Level for each volume.
		
		`ERROR.err` is a file at the top level. It is present if the request encountered some errors and the detailed error information is stored in this file. In this example, its presence is caused by the request for a non-existent volume **`gon.000000`**

Retrieve volumes
 	
`concat=true`
	`volumes.zip`
	
	`    |-- foo.001122.txt`
	
	`    |-- foo.001122.mets.xml`
	
	`    |-- bar.ark+=13960=t123.txt`
	
	`    |-- bar.ark+=13960=t123.mets.xml`
	
	`    |--volume-rights.txt`
	
	`    \--ERROR.err`
		Because the request parameter `concat=true`, each volume is a single text file, where the pages of the volume are concatenated into the file in the page order.
		
		`foo.001122.txt` is the text file entry for the volume **`foo.001122`**. The filename underwent a Pairtree clean process, but since it does not contain any filesystem unsafe characters, the cleaned ID looks identical to the original.
		
		`foo.001122.mets.xml` will be present if the request parameter `mets=true`.
		
		`bar.ark+=13960=t123.txt` is the text file entry for the volume `bar.ark:/13960/t123`. The filename underwent a Pairtree clean process, so filesystem-unsafe characters such as colons '`:`' and slashes '`/`' are replaced with filesystem-safe characters.
		
		`bar.ark+=13960=t123.mets.xml` will be present if the request parameter `mets=true`.
		
		volume-rights.txt is a file at the top level. It contains the HTRC Data Protection Level for each volume.
		
		`ERROR.err` is a file at the top level. It is present if the request encountered some errors and the detailed error information is stored in this file. In this example, its presence is caused by the request for a non-existent volume **`gon.000000`**

Retrieve pages
 	
`concat=false`
	`pages.zip`
	
	`    |-- foo.001122/`
	
	`    |    |--00000001.txt`
	
	`    |    |--00000002.txt`
	
	`    |    |--00000003.txt`
	
	`    |    |--00000004.txt`
	
	`    |    |--00000005.txt`
	
	`    |    \\--mets.xml`
	
	`    |-- bar.ark+=13960=t123/`
	
	`    |    |--00000001.txt`
	
	`    |    |--00000002.txt`
	
	`    |    |--00000003.txt`
	
	`    |    \\--mets.xml`
	
	`    |--volume-rights.txt`
	
	`    \\--ERROR.err`
		The Zip stream returned from the Data API for page retrieval with the request parameter `concat=false` is very similar to that returned for volume retrieval with `concat=false`. The difference is that only pages requested for will be included.

Retrieve pages
 	
`concat=true`
	`pages.zip`
	
	`    |--wordseq.txt`
	
	`    \\--ERROR.err`
		Because the request parameter `concat=true`, the returned Zip stream is a "sequence of words" where the content of all pages from all volumes is aggregated into a single text file entry named `wordseq.txt`.
		
		`ERROR.err` is a file at the top level. It is present if the request encountered some errors and the detailed error information is stored in this file. In this example, its presence is caused by the request for a non-existent volume **`gon.000000`**
		
		Note that there is no METS metadata returned because mixing METS metadata and page content into the word sequence could potentially contaminate the information in the word sequence file.

Token count
 	
(Deprecated)
 	
`level=volume`
	`tokencount.zip`
	
	`    |-- foo.001122.count`
	
	`    |-- bar.ark+=13960=t123.count`
	
	`    \\--ERROR.err`
		Because the request parameter `level=volume`, the returned Zip stream contains the token count of each volume as an entry, and the name of the entry is the Pairtree cleaned volumeID with `".count"` as the extension.
		
		`ERROR.err` is a file at the top level. It is present if the request encountered some errors and the detailed error information is stored in this file. In this example, its presence is caused by the request for a non-existent volume **`gon.000000`**

Token count
 	
(Deprecated)
 	
`level=page`
	`tokencount.zip`
	
	`    |-- foo.001122/`
	
	`    |    |--00000001.count`
	
	`    |    |--00000002.count`
	
	`    |    |--00000003.count`
	
	`    |    |--00000004.count`
	
	`    |    \\--00000005.count`
	
	`    |-- bar.ark+=13960=t123/`
	
	`    |    |--00000001.count`
	
	`    |    |--00000002.count`
	
	`    |    \\--00000003.count`
	
	`    \\--ERROR.err`
		Because the request parameter level= page, in the returned Zip stream, each volume is a directory whose entry name is the Pairtree cleaned volumeID, and each page of the volume is an entry under the directory, and the name of the page is the 8-digit zero-padded page sequence number followed by "`.count`" extension.
		
		`ERROR.err` is a file at the top level. It is present if the request encountered some errors and the detailed error information is stored in this file. In this example, its presence is caused by the request for a non-existent volume **`gon.000000`**

Token Count Output Format and Sorting Order (Deprecated)

Each token count output entry is a list of tokens and number of occurrences within the aggregation. The token and its occurrence count is separated by a space character (0x20), and each token-occurrence pair is a line and is separated from other pairs by a new line character (0x0A). However, if an aggregation does not contain any texts (e.g. an empty page), that particular entry will be empty.

sortBy & sortOrder	Token Count Output	Description

unspecified	`orange 1`
	
	`banana 2`
	
	`acorn 2`
	
	`A-team 1`
	
	`Xylophone 3`
	
	`apple 1`
	
	`coconut 1`
		if the parameter **`sortBy`** is not specified, the returned result does not guarantee any ordering of the token-occurrence pairs, nor does it guarantee the same ordering of these pairs between any 2 runs with the exact same parameters

sortBy=token& sortOrder=asc	A-team 1 Xylophone 3 acorn 2 apple 1 banana 2 coconut 1 orange 1	with ascending ordering on the tokens, the returned result is sorted using the UTF-8 value of the tokens in ascending order. In this example, the tokens starting with capital letter "X" come before these starting with lower case letter "a".
sortBy=token& sortOrder=desc	orange 1 coconut 1 banana 2 apple 1 acorn 2 Xylophone 3 A-team 1	this is the exact reverse of the case above
sortBy=count& sortOrder=asc	A-team 1 apple 1 coconut 1 orange 1 acorn 2 banana 2 Xylophone 3	with ascending ordering on the occurrence count, the returned result is sorted using the count value in ascending order; however, when multiple tokens have the same count, the order is determined by the ascending ordering of the tokens.
sortBy=count& sortOrder=desc	Xylophone 3 banana 2 acorn 2 coconut 1 apple 1 A-team 1	this is the exact reverse of the above case, and specifically, when multiple tokens have the same count, the order is determined by the descending ordering of the tokens.

Access Data API with JWT

While the Data API by itself does not enforce any security mechanism for authentication and/or authorization, it is can only be directly called using JWT in Secure Mode while in a Capsule. The Capsules come with fixed JWT saved to the image that you will use to make API calls. The scripts are with the tokens are saved at /home/dcuser/.htrc in each Capsule.