# Ingest API endpoint

* [Introduction](#PROD-6121-IngestAPIendpoint-Introductio)
  + [API Operation](#PROD-6121-IngestAPIendpoint-APIOperatio)
* [Ingest data from a ingest location](#PROD-6121-IngestAPIendpoint-Ingestdataf)
  + [Request](#PROD-6121-IngestAPIendpoint-Request)
    - [Parameters](#PROD-6121-IngestAPIendpoint-Parameters)
      * [Query parameters](#PROD-6121-IngestAPIendpoint-Queryparame)
    - [Notes](#PROD-6121-IngestAPIendpoint-Notes)
      * [locationId](#PROD-6121-IngestAPIendpoint-locationId)
    - [Example Request](#PROD-6121-IngestAPIendpoint-ExampleRequ)
  + [Responses](#PROD-6121-IngestAPIendpoint-Responses)
    - [Status: 202 - Accepted](#PROD-6121-IngestAPIendpoint-Status:202-)
      * [Example Response](#PROD-6121-IngestAPIendpoint-ExampleResp)
    - [Status: 4xx - Client errors](#PROD-6121-IngestAPIendpoint-Status:4xx-)
      * [Example Response](#PROD-6121-IngestAPIendpoint-ExampleResp)
* [Track the status of an ingest](#PROD-6121-IngestAPIendpoint-Trackthesta)
  + [Request](#PROD-6121-IngestAPIendpoint-Request.1)
    - [Parameters](#PROD-6121-IngestAPIendpoint-Parameters.)
      * [Path parameters](#PROD-6121-IngestAPIendpoint-Pathparamet)
  + [Example Request](#PROD-6121-IngestAPIendpoint-ExampleRequ)
  + [Responses](#PROD-6121-IngestAPIendpoint-Responses.1)
* [More detail on ingest behaviour](#PROD-6121-IngestAPIendpoint-Moredetailo)
  + [Implicit Preservation](#PROD-6121-IngestAPIendpoint-ImplicitPre)
  + [Usage of metadataPath](#PROD-6121-IngestAPIendpoint-Usageofmeta)
    - [Location of the metadata file](#PROD-6121-IngestAPIendpoint-Locationoft)
  + [Identifiers, folderPath and Datapool path](#PROD-6121-IngestAPIendpoint-Identifiers)
  + [Bagit Ingests](#PROD-6121-IngestAPIendpoint-BagitIngest)
    - [Example Bagit Format:](#PROD-6121-IngestAPIendpoint-ExampleBagi)
  + [Bagit Validation](#PROD-6121-IngestAPIendpoint-BagitValida)
    - [Bag Declaration: bagit.txt](#PROD-6121-IngestAPIendpoint-BagDeclarat)
    - [Payload Manifest: manifest-algorithm.txt](#PROD-6121-IngestAPIendpoint-PayloadMani)
  + [Usage of isArchive](#PROD-6121-IngestAPIendpoint-UsageofisAr)
  + [Usage of unpack](#PROD-6121-IngestAPIendpoint-Usageofunpa)

# Introduction

The following describes how to use the Ingest APIs to initiate an ingest of content and associated metadata from an ingest location and track the progress of the ingest.

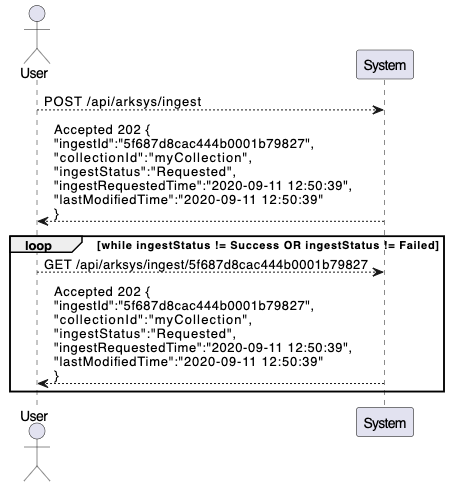
The following APIs exist:

1. Ingest data from an ingest location.
2. Track the status of an ingest.

## API Operation

Ingests can be long lived operations and therefore they operate asynchronously with respect to the http request that initiates them. Therefore, the typical pattern of operation will be to call an ingest endpoint and then monitor it for completion using the ingest status api.

All ingest initiated via the API will be visible on the UI with the 'Ingest Report' screen.



# Ingest data from a ingest location

Ingest data from an ingest location. The ingested files go through a number of stages of processing to check their integrity, extract technical metadata and unpack any contained archives before they are placed into the archive locations. If a metadata file is defined it will determine the overall structure of the collections, objects and files and once the ingest is complete the structure will be visible via the UI.

## Request

POST /api/arksys/ingest

### Parameters

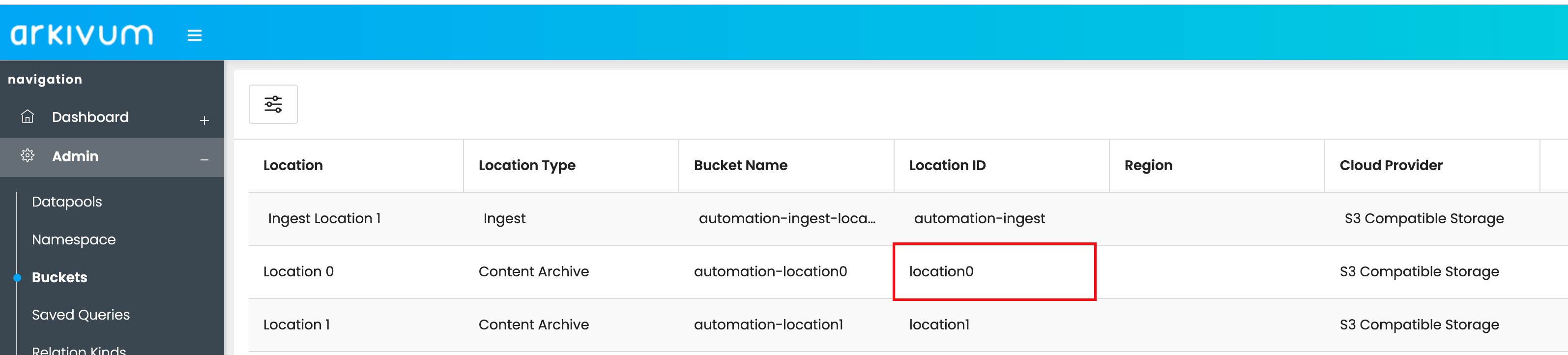
#### Query parameters

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Description** | **Required** |
| ingestPath | String | The path within the ingest location to files (or file) to be ingested, should be URL encoded | Yes |
| metadataPath | String | The path within the ingest location to a metadata file, should be URL encoded | No |
| datapool | String | The name of the datapool into which files are to be ingested, should be URL encoded.  Defaults to the default datapool | No |
| folderPath | String | The folder path used to prefix files paths being ingested, should be URL encoded | No |
| jobTag | String | A tag to be used to link related ingests, should be URL encoded | No |
| unpack | Boolean | Whether archives detected in the ingest location are unpacked or ingested without unpacking.  Defaults to false | No |
| isArchive | Boolean | Whether archives detected in the ingest location are unpacked before the ingest begins or ingested without unpacking.  Defaults to false | No |
| splitterChildren | Integer | This only applies when the ingest is into a datapool which has preservation enabled.  The maximum number of files under a collection or object which can be preserved in one job. Above the threshold files are allocated to separate preservation jobs.  Defaults to 10,000 | No |
| locationId | String | The identifier of an ingest location.  Defaults to the first ingest location. | No |

### Notes

#### locationId

The locationId is visible on the UI as follows:



### Example Request

curl -X POST \

-H "Accept: \*/\*" \

-H "Authorization: Bearer + ACCESS\_TOKEN" \

"https://host.example.com/api/arksys/ingest?ingestPath=ingestPath\_example&metadataPath=metadataPath\_example&datapool=datapool\_example&folderPath=folderPath\_example&jobTag=jobTag\_example&unpack=true&isArchive=true&splitterChildren=56&locationId=locationId\_example"

## Responses

### Status: 202 - Accepted

The Request has been accepted by the system, the ingest can be determined using the ingestId status API with the ingestId.

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| ingestId | String | The ID of the ingest |
| datapool | See the description of the corresponding request parameters above. | |
| ingestPath |
| metadataPath |
| folderPath |
| status | String | The ingest status |
| errorMessage | String | The error message of a failed ingest. |
| collectionId | String | Identifier of the root collection which contains all the ingested entities. |

#### Example Response

The json response should look like this:

{

"ingestId": "60143b7ee5310a740459c599",

"datapool": "Default",

"ingestPath": "dataset1",

"metadataPath": "dataset1/ark-manifest.json",

"folderPath": "myfolder",

"status": "IN\_PROGRESS",

"errorMessage": null,

"collectionId": "myCollection"

}

### Status: 4xx - Client errors

Typical values will be 404 ‘Resource Not Found’ or 400 ‘Bad Request’, but any status code in the 400 → 499 range will have the following form:

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| errorMessage | String | An error message given the reason for failure |
| errorDetails | array[String] | A list of specific error conditions which have contributed to the overall failure. |

#### Example Response

 The json response should look like this:

{

    "errorMessage":"Something has gone wrong",

    "errorDetails": [

"This thing went wrong",

"This other thing also went wrong"

]

}

# Track the status of an ingest

Ingest status. Get the overall status of the ingest.

## Request

GET /api/arksys/ingest/{ingestId}

### Parameters

#### Path parameters

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Description** | **Required** |
| ingestId | String | The identifier of the ingest job. | Yes |

## Example Request

curl -X GET \

-H "Accept: \*/\*" \

-H "Authorization: Bearer + ACCESS\_TOKEN" \

"https://host.example.com/api/arksys/ingest/{ingestId}"

## Responses

See [Ingest from an ingest location](https://arkivum.atlassian.net/wiki/spaces/EN/pages/3423109148/PROD-6121+-+Ingest+API+endpoint#Responses) above.

# More detail on ingest behaviour

## Implicit Preservation

If the ingest is into a datapool which has preservation enabled then once the ingest is mostly complete an automatic preservation is triggered which might result in 1 or more preservation jobs, each preservation job is tracked under it’s own identifier, but they are associated to the original ingest.

## Usage of metadataPath

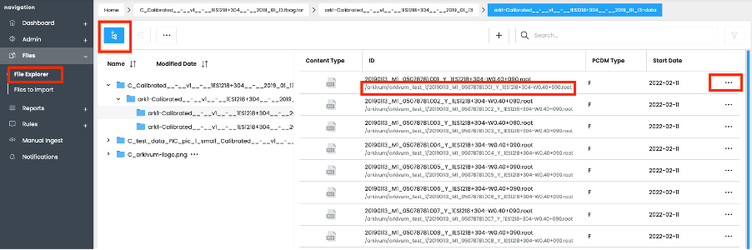
The metadataPath allows the user to specify a metadata file which performs two main functions:

1. Create a [PCDM](https://pcdm.org/2016/04/18/models) structure for the files being ingested. This allows the ingest to be structured in a hierarchy in terms of collection, objects and files, where files are the files within the ingestPath.
2. Define the metadata associated with each collection, objects or file. Metadata has to adhere to formats preconfigured in the system against the datapool. These can be pre-defined formats e.g. ([dublin core](https://www.dublincore.org/), [isad(g)](https://www.ica.org/en/isadg-general-international-standard-archival-description-second-edition)) or created by the user.

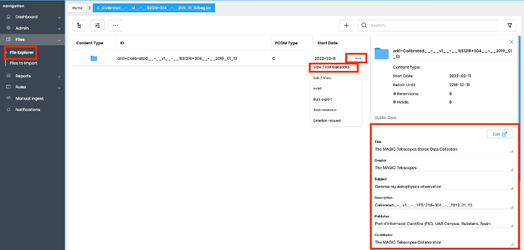
There are two types of metadata file supported:

1. ark-file-meta.csv which is an Arkivum proprietary format expressed in a CSV format which is compatible with Microsoft Excel.
2. ark-manifest.json which is an Arkivum proprietary format expressed in a Json format.

The PCDM Collections, Objects and Files can be viewed using the File Explorer.   The ‘tree view’ icon can be used to show the hierarchical structure of the dataset.



The ellipsis next to a Collection, Object or File can be used to pull up a menu that allows actions such as viewing metadata.



### Location of the metadata file

Whilst it is typical to have the metadata file located next to the files within the ingest and hence be contained within the ingestPath, this is not absolutely required and it is possible to locate the metadata file anywhere within the ingest location. This can be particularly useful for archives which have come from some other system already packed (see isArchive below) or anywhere where it is desirable not to disturb or pollute the files in the ingest location with a proprietary Arkivum format.

## Identifiers, folderPath and Datapool path

Within an metadata file identifiers are used to relate the collection, objects and files and for fieldBindings to extract metadata from other files.

* For files the identifiers are defined by the location of the file within the ingestPath, that is they are relative to the ingestPath and this is how they are used within the metadata file, but once ingested the file identifiers will be prefixed by the datapool path and folderPath. This is too allow files to be unique within the system and to group them into the datapool in which they reside.
* For collections and objects the identifiers are defined in the metadata file and are unchanged by the datapool path or folderPath.

For example, if the path to a dataset in the ingest location is:

test\_data/ARK/planets

And within this path there are files and subfolders like this:

test\_data/ARK/planets/

└── objects

└── milky\_way

└── solar\_system

└── planets

├── earth

├── jupiter

│ ├── data

│ │ └── planetary\_data.xls

│ ├── docs

│ │ ├── 11\_Jupiter\_FC.pdf

│ │ ├── 62211main\_Jupiter\_Lithograph.pdf

│ │ ├── Jupiter.docx

│ │ ├── Jupiter.msg

Then, when an ingest is triggered with ingestPath= test\_data/ARK/planets/ into the default datapool which always has path /, then the files ingested will have IDs that look like this:

/objects/milky\_way/solar\_system/planets/Jupiter/data/planetary\_data.xls

/objects/milky\_way/solar\_system/planets/Jupiter/docs/11\_Jupiter\_FC.pdf

Then, when an ingest is triggered with ingestPath= test\_data/ARK/planets/ into a datapool with path /dp1, then the files ingested will have IDs that look like this:

/dp1/objects/milky\_way/solar\_system/planets/Jupiter/data/planetary\_data.xls

/dp1/objects/milky\_way/solar\_system/planets/Jupiter/docs/11\_Jupiter\_FC.pdf

When an ingest is triggered with ingestPath= test\_data/ARK/planets/ into a datapool with path /dp1 and folderPath= folder1/folder2 then the files ingested will have IDs that look like this:

/dp1/folder1/folder2/objects/milky\_way/solar\_system/planets/Jupiter/data/planetary\_data.xls

/dp1/folder1/folder2/objects/milky\_way/solar\_system/planets/Jupiter/docs/11\_Jupiter\_FC.pdf

If an attempt is made to ingest a dataset that overlaps with data already within the system then you have these options:

1. Delete the dataset from the system before ingesting it again.  See the delete cookbook for more details.
2. Change the subfolder and filenames within the ingestPath so they don’t clash with previous ingests.
3. Use the folderPath parameter to change the path of the files within the system.

## Bagit Ingests

The [Bagit format](https://datatracker.ietf.org/doc/html/rfc8493) is a set of hierarchical file layout conventions designed to support storage and transfer of arbitrary digital content.

If the ingestPath is structured as a Bagit format the system will automatically detect it based on whether the ingestPath  contains a bagit.txt file and a data directory. If so the system will assume that the ingest is a bagit bag.  It will then look for a checksum manifest and will validate the contents of the data directory.

Bagit bags are used purely as a ‘transport wrapper’ for supplying checksums that can be used to validate the payload of a bag.  None of the other features of bagit are supported, e.g. tag manifests or bag-level metadata. As such the following should be noted:

* Only the content of the payload directory will be ingested after a bag has been checksum validated.
* None of the top-level bagit files (bagit.txt, bag-info.txt, manifest-sha256.txt etc.) will be ingested.
* No metadata will be extracted/processed from bag-info.txt or any tag files.  Tag files will not be ingested.
* The system supports the mandatory parts of the bagit 1.0 spec, e.g. SHA256 and SHA512 checksum manifests, in addition, Adler32 checksum manifests are also supported.  The system does not support optional parts of the spec, e.g. MD5 or SHA1 manifests.
* Bagit bag ingest using uncompressed bags, i.e. a bag in folder form.  Zipped bags are only supported by usage of the isArchive parameter.
* Bagit fetch files are not currently supported.
* When referencing a metadata file using the metadataPath the metadata file should be put inside the payload ('data') directory of the bag.   Do NOT put the metadata file at the top level of the bag, i.e. next to the manifest or bagit.txt files, because this will invalidate the bag and the ingest will fail.
* Given the above indentifiers of files are therefore relative to the data directory and not to the the ingestPath.

### Example Bagit Format:

Suppose the contents of the blogs folder looks like this:

test\_data/ARK/bagit\_datasets/blogs

├── bag-info.txt

├── bagit.txt

├── data

│ ├── Blogs

│ │ ├── Arkivum\_Press\_Release\_FINAL.docx

│ │ ├── Arkivum\_J\_Press\_Release\_v1.docx

│ │ └── eTMF\_blog\_for\_website.docx

│ └── ark-file-meta.csv

├── manifest-sha256.txt

├── manifest-sha512.txt

├── tagmanifest-sha256.txt

└── tagmanifest-sha512.txt

To ingest the above the above following parameters would be used

ingestPath = test\_data/ARK/bagit\_datasets/blogs/

metadataPath = test\_data/ARK/bagit\_datasets/blogs/data/ark-file-meta.csv

If the files were ingest into a datapool with path /dp1 and folderPath=folder1/folder2 then the files ingested will have IDs that look like this:

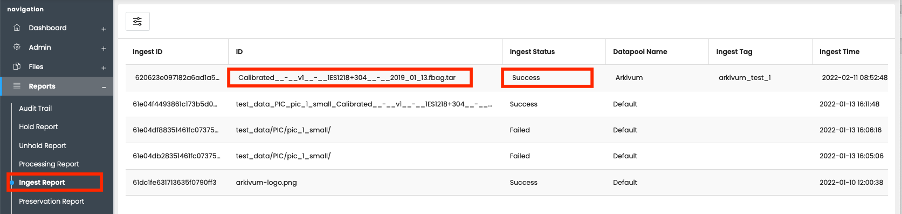
/dp1/folder1/folder2/Blogs/Arkivum\_Press\_Release\_FINAL.docx

/dp1/folder1/folder2/Blogs/Arkivum\_J\_Press\_Release\_v1.docx

/dp1/folder1/folder2/Blogs/eTMF\_blog\_for\_website.docx

## Bagit Validation

Bagit validation can be checked using the Audit Trail.  For example, use the filter option (small funnel icon) and filter for events that contain ‘BAGIT’.



Please note that the following files in the root of the bagit bag are mandatory:

### Bag Declaration: bagit.txt

The bagit.txt tag file MUST consist of exactly two lines in this order:

 BagIt-Version: M.N

Tag-File-Character-Encoding: ENCODING

**M.N** identifies the BagIt major (**M**) and minor (**N**) version numbers. **ENCODING** identifies the character set encoding used by the renaming tag files.  **ENCODING** should be **UTF-8**.

### Payload Manifest: manifest-algorithm.txt

 A payload manifest file lists each payload file along with a corresponding checksum to permit data integrity checking.   Supported checksum algorithms are MD5, SHA256, SHA512 and Adler32.  This supported manifest file names are:

manifest-alder32.txt

manifest-md5.txt

manifest-sha256.txt

manifest-sha512.txt

Each line of a payload manifest file MUST be of the form:

Checksum filepath

One or more manifests can be provided.  It is not mandatory to use multiple manifests.

## Usage of isArchive

When the files to be ingested have come from some other system as an archive which is already packaged, but there is no desire to represent the archive itself with the system then it is possible to request that the archive is unpacked as part of the ingest. The following points are note worthy:

* Extraction of the archive file is done asynchronously as part of the ingest.
* If the archive is in Bagit format the system will extract the archive, validate the bag, and ingest the contents of the payload (data/) directory.  Everything above the payload directory will be discarded.  It is only the contents of the payload that will be ingested.
* After the contents of the archive file (zip, tar etc.) have been extracted, the archive file will be discarded and not ingested.  Only the contents of the archive file are ingested, not the archive file itself.  The archive file is considered as a container for data during transit and not something that should be retained.
* If the archive file contains a single folder at the root level of the archive, then the contents of this folder will be ingested.  The folder itself will not be included in the filepaths of the files that are ingested.
* If a metadata file is supplied by using the metadataPath parameter, then the metadata file cannot be inside the archive file, since metadataPath is always a path within the ingest location.

## Usage of unpack

When a ingestPath contains archives and those archives also contain archives then the option exists to request the system to unpack them as part of the ingest. The unpacking is fully recursive so all archive formats encountered will be unpacked. The following points are note worthy:

1. If an archive is encountered which is corrupted or unsupported or has a password it will be ingested as is with.
2. The format of archives (e.g. tar, zip, 7z) is detected by looking at the [file magic](https://en.wikipedia.org/wiki/List_of_file_signatures) within the file and possibly the extension, therefore detection is best effort.
3. If the archive is in root of the ingestPath, it will be ingested it addition to also unpacking it. This is to ensure there is no data loss should the system incorrectly unpack due to different feature sets between what was used to create the archive and what is used to unpack it.