pydas Documentation

Release 0.3.6

Kitware, Inc.

January 28, 2016

Contents

Contents:

CHAPTER 1

Introduction To pydas

pydas is a Python client library for Midas Server 3.

1.1 Requirements

- A working Midas Server 3 instance
- An enabled web API plugin for your Midas Server 3 instance
- Python version 2.6 or later
- requests Python package

1.2 Installing and Upgrading pydas

The easiest way to install pydas is through pip, as this will automatically install all Python dependencies.

Install pydas:

```
pip install pydas
```

Upgrade pydas:

```
pip install --upgrade pydas
```

1.2.1 Checking your Installation

- The version of your Midas Server 3 instance is displayed at the bottom of any Midas Server 3 page
- The version of pydas you have installed is available via Python interpreter (as of pydas version 0.2.2)

```
>>> import pydas
>>> pydas.version
'0.3.4'
```

You can check the version of your Midas Server 3 instance via pydas, which will ensure that pydas can communicate with your server instance and that your server's web API plugin is active.

```
>>> import pydas
>>> core_driver = pydas.drivers.CoreDriver('http://domain/midas3')
>>> core_driver.get_server_version()
'3.4.1'
```

CHAPTER 2

pydas Quick Start

2.1 Module level method examples

The easiest way to use pydas is by using high level module level methods.

2.1.1 Login to your Midas Server instance

```
>>> import pydas
>>> pydas.login()
Server URL: http://domain/midas3
Email: email@email.com
Password: PASSWORD
'FK0uEYfciJcjBbaAadfTC123213s12810Favf0PJPULX'
```

For reference, this last string is your session token, but you will probably not need to use it.

2.1.2 Simple upload example

Let's say you have a subdirectory under your current location called myfiles, with 2 files in it:

```
myfiles
    file1.txt
    file2.txt
```

After we have logged in to pydas, we want to upload this directory to our Midas Server 3 instance.

```
>>> pydas.upload('myfiles')
Creating Folder from myfiles
Uploading Item from myfiles/file1.txt
Uploading Item from myfiles/file2.txt
```

By default, this will create a folder under your Midas Server 3 user's private folder called myfiles, with 2 items, one for each of the files.

This upload method will upload recursively, so if you have subdirectories under your uploaded directory, they will have corresponding folders created on the Midas Server to mirror the directory structure in your local machine.

2.1.3 Upload example treating leaf folders as items

Let's say you have some folders with only files in them (call them leaf folders), and you want each of the leaf folders to have all files in them uploaded to the same item.

Here we have a top level folder folders_as_items, which has two subfolders. Each of the subfolders has two files:

```
folders_as_items
    item1
        bitstream1_1.txt
        bitstream1_2.txt
    item2
        bitstream2_1.txt
        bitstream2_2.txt
```

```
>>> pydas.upload('folders_as_items',  leaf_folders_as_items=True)
Creating Folder from folders_as_items
Creating Item from folders_as_items/item1.
Uploading Bitstream from folders_as_items/item1/bitstream1_1.txt (1 of 2)
Uploading Bitstream from folders_as_items/item1/bitstream1_2.txt (2 of 2)
Creating Item from folders_as_items/item2.
Uploading Bitstream from folders_as_items/item2/bitstream2_1.txt (1 of 2)
Uploading Bitstream from folders_as_items/item2/bitstream2_2.txt (2 of 2)
```

This upload will create a folder in your Midas Server private folder called folders_as_items, and in that folder will create two items, item1 and item2. item1 will have 2 bitstreams, and item2 will have two bitstreams, corresponding to the files that are in each of the leaf folders.

2.1.4 Upload example for DICOM data

Let's say you have a folder that contains subfolders, each of the subfolders is a DICOM series. You would like to upload these subfolders such that all of the files in the subfolders are combined into one item, and once the item is created, DICOM Metadata is extracted from the item. In this case we will treat the subfolders as leaf folders, and we will also add a callback after the upload of the item to extract DICOM Metadata:

```
dicom_data
    series1
        00380001.dcm
        00380002.dcm
        00380003.dcm
        00380004.dcm
    series2
        00500001.dcm
        00500002.dcm
        00500003.dcm
        00500004.dcm
```

```
>>> extract_dicom_callback = lambda communicator, token, item_id: communicator.extract_dicommetadata
>>> pydas.add_item_upload_callback(extract_dicom_callback)
>>> pydas.upload('dicom_data', leaf_folders_as_items=True)
Creating Folder from dicom_data
Creating Item from dicom_data/series2.
Uploading Bitstream from dicom_data/series2/00500002.dcm (1 of 4)
Uploading Bitstream from dicom_data/series2/00500003.dcm (2 of 4)
Uploading Bitstream from dicom_data/series2/00500004.dcm (3 of 4)
Uploading Bitstream from dicom_data/series2/00500001.dcm (4 of 4)
Creating Item from dicom_data/series1.
```

```
Uploading Bitstream from dicom_data/series1/00380001.dcm (1 of 4)
Uploading Bitstream from dicom_data/series1/00380003.dcm (2 of 4)
Uploading Bitstream from dicom_data/series1/00380002.dcm (3 of 4)
Uploading Bitstream from dicom_data/series1/00380004.dcm (4 of 4)
```

Import high-level pydas functions. Module for the main user classes for pydas.

class pydas.core.**Communicator**(*url*, *drivers=None*)

Class for communicating with Midas Server through its drivers.

debug

Return whether the debug state of every driver is True.

Returns True if the debug state of every driver is True

Return type bool

drivers

Get the list of drivers attached to this communicator.

Returns list of drivers

Return type list[T <= pydas.drivers.BaseDriver]

set_auth(*value*)

Set the authentication in all drivers attached to this communicator.

Parameters **value** (*None | tuple*) – authentication tuple to be passed to requests.request()

url

Return the URL of the server.

Returns URL of the server

Return type string

verify_ssl_certificate

Return whether the SSL certificate will be verified for all drivers attached to this communicator.

Returns True if the SSL certificate will be verified for every driver

Return type bool

This module is for the drivers that actually do the work of communication with the Midas Server instance. Any drivers that are implemented should use the utility functions provided in pydas.drivers.BaseDriver by inheriting from that class.

class pydas.drivers.**BaseDriver**(*url=''*)

Base class for the Midas Server API drivers.

debug

Return the debug state of this driver.

Returns debug state

Return type bool

full_url

Return the full URL of the server including the API suffix.

Returns Full URL of the server

Return type string

login_with_api_key(*email*, *api_key*, *application='Default'*)

Login and get a token. If you do not specify a specific application, 'Default' will be used.

Parameters

- **email** (*string*) – Email address of the user
- **api_key** (*string*) – API key assigned to the user
- **application** (*string*) – (optional) Application designated for this API key

Returns Token to be used for interaction with the API until expiration

Return type string

request(**args*, ***kw*)

Do the generic processing of a request to the server.

If file_payload is specified, it will be PUT to the server.

Parameters

- **method** (*string*) – Desired API method
- **parameters** (*None | dict[string, string]*) – (optional) Parameters to pass in the HTTP body
- **file_payload** (*None | file | FileIO*) – (optional) File-like object to be sent with the HTTP request

Returns Dictionary representing the JSON response to the request

Return type dict

Raises **pydas.exceptions.PydasException** – if the request failed

url

Return the URL of the server.

Returns URL of the server

Return type string

verify_ssl_certificate

Return whether the SSL certificate will be verified.

Returns True if the SSL certificate will be verified

Return type bool

class pydas.drivers.**BatchmakeDriver**(*url=''*)

Driver for the batchmake module API methods.

add_condor_dag(*token*, *batchmaketaskid*, *dagfilename*, *dagmanoutfilename*)

Add a Condor DAG to the given Batchmake task.

Parameters

- **token** (*string*) – A valid token for the user in question.
- **batchmaketaskid** (*int | long*) – id of the Batchmake task for this DAG
- **dagfilename** (*string*) – Filename of the DAG file
- **dagmanoutfilename** (*string*) – Filename of the DAG processing output

Returns The created Condor DAG DAO

Return type dict

add_condor_job(*token*, *batchmaketaskid*, *jobdefinitionfilename*, *outputfilename*, *errorfilename*, *logfilename*, *postfilename*)
Add a Condor DAG job to the Condor DAG associated with this Batchmake task

Parameters

- **token** (*string*) – A valid token for the user in question.
- **batchmaketaskid** (*int | long*) – id of the Batchmake task for this DAG
- **jobdefinitionfilename** (*string*) – Filename of the definition file for the job
- **outputfilename** (*string*) – Filename of the output file for the job
- **errorfilename** (*string*) – Filename of the error file for the job
- **logfilename** (*string*) – Filename of the log file for the job
- **postfilename** (*string*) – Filename of the post script log file for the job

Returns The created Condor job DAO.

Return type dict

class pydas.drivers.**CoreDriver**(*url=''*)
Driver for the core API methods of Midas Server. This contains all of the calls necessary to interact with a Midas Server instance that has no plugins enabled (other than the web API).

create_community(*token*, *name*, ***kwargs*)
Create a new community or update an existing one using the uuid.

Parameters

- **token** (*string*) – A valid token for the user in question.
- **name** (*string*) – The community name.
- **description** (*string*) – (optional) The community description.
- **uuid** (*string*) – (optional) uuid of the community. If none is passed, will generate one.
- **privacy** (*string*) – (optional) Default 'Public', possible values [Public|Private].
- **can_join** (*string*) – (optional) Default 'Everyone', possible values [Everyone|Invitation].

Returns The community dao that was created.

Return type dict

create_folder(*token*, *name*, *parent_id*, ***kwargs*)
Create a folder at the destination specified.

Parameters

- **token** (*string*) – A valid token for the user in question.
- **name** (*string*) – The name of the folder to be created.
- **parent_id** (*int | long*) – The id of the targeted parent folder.
- **description** (*string*) – (optional) The description text of the folder.
- **uuid** (*string*) – (optional) The UUID for the folder. It will be generated if not given.
- **privacy** – (optional) The privacy state of the folder ('Public' or 'Private').
- **reuse_existing** (*bool*) – (optional) If true, will just return the existing folder if there is one with the name provided.

Returns Dictionary containing the details of the created folder.

Return type dict

create_item(*token*, *name*, *parent_id*, ***kwargs*)
Create an item to the server.

Parameters

- **token** (*string*) – A valid token for the user in question.
- **name** (*string*) – The name of the item to be created.
- **parent_id** (*int | long*) – The id of the destination folder.
- **description** (*string*) – (optional) The description text of the item.
- **uuid** (*string*) – (optional) The UUID for the item. It will be generated if not given.
- **privacy** (*string*) – (optional) The privacy state of the item ('Public' or 'Private').

Returns Dictionary containing the details of the created item.

Return type dict

create_link(*token*, *folder_id*, *url*, ***kwargs*)
Create a link bitstream.

Parameters

- **token** (*string*) – A valid token for the user in question.
- **folder_id** (*int | long*) – The id of the folder in which to create a new item that will contain the link. The new item will have the same name as the URL unless an item name is supplied.
- **url** (*string*) – The URL of the link you will create, will be used as the name of the bitstream and of the item unless an item name is supplied.
- **item_name** (*string*) – (optional) The name of the newly created item, if not supplied, the item will have the same name as the URL.
- **length** (*int | long*) – (optional) The length in bytes of the file to which the link points.
- **checksum** (*string*) – (optional) The MD5 checksum of the file to which the link points.

Returns The item information of the item created.

Return type dict

delete_folder(*token*, *folder_id*)
Delete the folder with the passed in folder_id.

Parameters

- **token** (*string*) – A valid token for the user in question.
- **folder_id** (*int | long*) – The id of the folder to be deleted.

Returns None.

Return type None

delete_item(*token*, *item_id*)
Delete the item with the passed in item_id.

Parameters

- **token** (*string*) – A valid token for the user in question.
- **item_id** (*int | long*) – The id of the item to be deleted.

Returns None.

Return type None

download_item(*item_id*, *token=None*, *revision=None*)
Download an item to disk.

Parameters

- **item_id** (*int | long*) – The id of the item to be downloaded.
- **token** (*None | string*) – (optional) The authentication token of the user requesting the download.
- **revision** (*None | int | long*) – (optional) The revision of the item to download, this defaults to HEAD.

Returns A tuple of the filename and the content iterator.

Return type (string, unknown)

folder_children(*token*, *folder_id*)
Get the non-recursive children of the passed in folder_id.

Parameters

- **token** (*string*) – A valid token for the user in question.
- **folder_id** (*int | long*) – The id of the requested folder.

Returns Dictionary of two lists: 'folders' and 'items'.

Return type dict[string, list]

folder_get(*token*, *folder_id*)
Get the attributes of the specified folder.

Parameters

- **token** (*string*) – A valid token for the user in question.
- **folder_id** (*int | long*) – The id of the requested folder.

Returns Dictionary of the folder attributes.

Return type dict

generate_upload_token(*token*, *item_id*, *filename*, *checksum=None*)
Generate a token to use for upload.

Midas Server uses a individual token for each upload. The token corresponds to the file specified and that file only. Passing the MD5 checksum allows the server to determine if the file is already in the asset store.

If **:param:'checksum'** is passed and the token returned is blank, the server already has this file and there is no need to follow this call with a call to *perform_upload*, as the passed in file will have been added as a bitstream to the item's latest revision, creating a new revision if one doesn't exist.

Parameters

- **token** (*string*) – A valid token for the user in question.
- **item_id** (*int | long*) – The id of the item in which to upload the file as a bitstream.

- **filename** (*string*) – The name of the file to generate the upload token for.
- **checksum** (*None | string*) – (optional) The checksum of the file to upload.

Returns String of the upload token.

Return type string

get_community_by_id(*community_id*, *token=None*)
Get a community based on its id.

Parameters

- **community_id** (*int | long*) – The id of the target community.
- **token** (*None | string*) – (optional) A valid token for the user in question.

Returns The requested community.

Return type dict

get_community_by_name(*name*, *token=None*)
Get a community based on its name.

Parameters

- **name** (*string*) – The name of the target community.
- **token** (*None | string*) – (optional) A valid token for the user in question.

Returns The requested community.

Return type dict

get_community_children(*community_id*, *token=None*)
Get the non-recursive children of the passed in community_id.

Parameters

- **community_id** (*int | long*) – The id of the requested community.
- **token** (*None | string*) – (optional) A valid token for the user in question.

Returns List of the folders in the community.

Return type dict[string, list]

get_default_api_key(*email*, *password*)
Get the default API key for a user.

Parameters

- **email** (*string*) – The email of the user.
- **password** (*string*) – The user's password.

Returns API key to confirm that it was fetched successfully.

Return type string

get_item_metadata(*item_id*, *token=None*, *revision=None*)
Get the metadata associated with an item.

Parameters

- **item_id** (*int | long*) – The id of the item for which metadata will be returned
- **token** (*None | string*) – (optional) A valid token for the user in question.
- **revision** (*int | long*) – (optional) Revision of the item. Defaults to latest revision.

Returns List of dictionaries containing item metadata.

Return type list[dict]

get_server_info()
Get general server information.

The information provided includes enabled modules as well as enabled web API functions.

Returns Module and web API information.

Return type dict

get_server_version()
Get the version from the server.

Returns version code from the server

Return type string

get_user_by_email(*email*)
Get a user by the email of that user.

Parameters **email** (*string*) – The email of the desired user.

Returns The user requested.

Return type dict

get_user_by_id(*user_id*)
Get a user by the first and last name of that user.

Parameters **user_id** (*int | long*) – The id of the desired user.

Returns The user requested.

Return type dict

get_user_by_name(*firstname*, *lastname*)
Get a user by the first and last name of that user.

Parameters

- **firstname** (*string*) – The first name of the user.
- **lastname** (*string*) – The last name of the user.

Returns The user requested.

Return type dict

item_get(*token*, *item_id*)
Get the attributes of the specified item.

Parameters

- **token** (*string*) – A valid token for the user in question.
- **item_id** (*int | string*) – The id of the requested item.

Returns Dictionary of the item attributes.

Return type dict

list_communities(*token=None*)
List all communities visible to a user.

Parameters **token** (*None | string*) – (optional) A valid token for the user in question.

Returns The list of communities.

Return type list[dict]

list_modules()
List the enabled modules on the server.

Returns List of names of the enabled modules.

Return type list[string]

list_user_folders(*token*)
List the folders in the users home area.

Parameters **token** (*string*) – A valid token for the user in question.

Returns List of dictionaries containing folder information.

Return type list[dict]

list_users(*limit=20*)
List the public users in the system.

Parameters **limit** (*int | long*) – (optional) The number of users to fetch.

Returns The list of users.

Return type list[dict]

move_folder(*token*, *folder_id*, *dest_folder_id*)
Move a folder to the destination folder.

Parameters

- **token** (*string*) – A valid token for the user in question.
- **folder_id** (*int | long*) – The id of the folder to be moved.
- **dest_folder_id** (*int | long*) – The id of destination (new parent) folder.

Returns Dictionary containing the details of the moved folder.

Return type dict

move_item(*token*, *item_id*, *src_folder_id*, *dest_folder_id*)
Move an item from the source folder to the destination folder.

Parameters

- **token** (*string*) – A valid token for the user in question.
- **item_id** (*int | long*) – The id of the item to be moved
- **src_folder_id** (*int | long*) – The id of source folder where the item is located
- **dest_folder_id** (*int | long*) – The id of destination folder where the item is moved to

Returns Dictionary containing the details of the moved item

Return type dict

perform_upload(*upload_token*, *filename*, ***kwargs*)
Upload a file into a given item (or just to the public folder if the item is not specified.

Parameters

- **upload_token** (*string*) – The upload token (returned by generate_upload_token)

- **filename** (*string*) – The upload filename. Also used as the path to the file, if 'filepath' is not set.
- **mode** (*string*) – (optional) Stream or multipart. Default is stream.
- **folder_id** (*int | long*) – (optional) The id of the folder to upload into.
- **item_id** (*int | long*) – (optional) If set, will append item `bitstreams` to the latest revision (or the one set using **:param:'revision'**) of the existing item.
- **revision** (*string | int | long*) – (optional) If set, will add a new file into an existing revision. Set this to 'head' to add to the most recent revision.
- **filepath** (*string*) – (optional) The path to the file.
- **create_additional_revision** (*bool*) – (optional) If set, will create a new revision in the existing item.

Returns Dictionary containing the details of the item created or changed.

Return type dict

search (*search*, *token=None*)
Get the resources corresponding to a given query.

Parameters

- **search** (*string*) – The search criterion.
- **token** (*None | string*) – (optional) The credentials to use when searching.

Returns Dictionary containing the search result. Notable is the dictionary item 'results', which is a list of item details.

Return type dict

search_item_by_name (*name*, *token=None*)
Return all items.

Parameters

- **name** (*string*) – The name of the item to search by.
- **token** (*None | string*) – (optional) A valid token for the user in question.

Returns A list of all items with the given name.

Return type list[dict]

search_item_by_name_and_folder (*name*, *folder_id*, *token=None*)
Return all items with a given name and parent folder id.

Parameters

- **name** (*string*) – The name of the item to search by.
- **folder_id** (*int | long*) – The id of the parent folder to search by.
- **token** (*None | string*) – (optional) A valid token for the user in question.

Returns A list of all items with the given name and parent folder id.

Return type list[dict]

search_item_by_name_and_folder_name (*name*, *folder_name*, *token=None*)
Return all items with a given name and parent folder name.

Parameters

- **name** (*string*) – The name of the item to search by.
- **folder_name** (*string*) – The name of the parent folder to search by.
- **token** (*None | string*) – (optional) A valid token for the user in question.

Returns A list of all items with the given name and parent folder name.

Return type list[dict]

set_item_metadata(*token*, *item_id*, *element*, *value*, *qualifier=None*)
Set the metadata associated with an item.

Parameters

- **token** (*string*) – A valid token for the user in question.
- **item_id** (*int | long*) – The id of the item for which metadata will be set.
- **element** (*string*) – The metadata element name.
- **value** (*string*) – The metadata value for the field.
- **qualifier** (*None | string*) – (optional) The metadata qualifier. Defaults to empty string.

Returns None.

Return type None

share_item(*token*, *item_id*, *dest_folder_id*)
Share an item to the destination folder.

Parameters

- **token** (*string*) – A valid token for the user in question.
- **item_id** (*int | long*) – The id of the item to be shared.
- **dest_folder_id** (*int | long*) – The id of destination folder where the item is shared to.

Returns Dictionary containing the details of the shared item.

Return type dict

class pydas.drivers.**DicomextractorDriver**(*url=''*)
Driver for the dicomextractor module API methods.

extract_dicommetadata(*token*, *item_id*)
Extract DICOM metadata from the given item

Parameters

- **token** (*string*) – A valid token for the user in question.
- **item_id** (*int | long*) – id of the item to be extracted

Returns the item revision DAO

Return type dict

class pydas.drivers.**MultiFactorAuthenticationDriver**(*url=''*)
Driver for the multi-factor authentication module API methods.

mfa_otp_login(*temp_token*, *one_time_pass*)
Log in to get the real token using the temporary token and otp.

Parameters

- **temp_token** (*string*) – The temporary token or id returned from normal login
- **one_time_pass** (*string*) – The one-time pass to be sent to the underlying multi-factor engine.

Returns A standard token for interacting with the web api.

Return type string

class pydas.drivers.**SolrDriver**(*url=''*)
Driver for the solr module API methods.

solr_advanced_search(*query*, *token=None*, *limit=20*)
Search item metadata using Apache Solr.

Parameters

- **query** (*string*) – The Apache Lucene search query.
- **token** (*None | string*) – (optional) A valid token for the user in question.
- **limit** (*int | long*) – (optional) The limit of the search.

Returns The list of items that match the search query.

Return type list[dict]

class pydas.drivers.**ThumbnailCreatorDriver**(*url=''*)
Driver for the thumbnailcreator module API methods.

create_big_thumbnail(*token*, *bitstream_id*, *item_id*, *width=575*)
Create a big thumbnail for the given bitstream with the given width. It is used as the main image of the given item and shown in the item view page.

Parameters

- **token** (*string*) – A valid token for the user in question.
- **bitstream_id** (*int | long*) – The bitstream from which to create the thumbnail.
- **item_id** (*int | long*) – The item on which to set the thumbnail.
- **width** (*int | long*) – (optional) The width in pixels to which to resize (aspect ratio will be preserved). Defaults to 575.

Returns The ItemthumbnailDao object that was created.

Return type dict

create_small_thumbnail(*token*, *item_id*)
Create a 100x100 small thumbnail for the given item. It is used for preview purpose and displayed in the 'preview' and 'thumbnails' sidebar sections.

Parameters

- **token** (*string*) – A valid token for the user in question.
- **item_id** (*int | long*) – The item on which to set the thumbnail.

Returns The item object (with the new thumbnail id) and the path where the newly created thumbnail is stored.

Return type dict

class pydas.drivers.**TrackerDriver**(*url=''*)
Driver for the tracker module API methods.

add_scalar_data(*token*, *community_id*, *producer_display_name*, *metric_name*, *producer_revision*, *submit_time*, *value*, ***kwargs*)

Create a new scalar data point.

Parameters

- **token** (*string*) – A valid token for the user in question.
- **community_id** (*int | long*) – The id of the community that owns the producer.
- **producer_display_name** (*string*) – The display name of the producer.
- **metric_name** (*string*) – The metric name that identifies which trend this point belongs to.
- **producer_revision** (*int | long | string*) – The repository revision of the producer that produced this value.
- **submit_time** (*string*) – The submit timestamp. Must be parsable with PHP strtotime().
- **value** (*float*) – The value of the scalar.
- **config_item_id** (*int | long*) – (optional) If this value pertains to a specific configuration item, pass its id here.
- **test_dataset_id** (*int | long*) – (optional) If this value pertains to a specific test dataset, pass its id here.
- **truth_dataset_id** (*int | long*) – (optional) If this value pertains to a specific ground truth dataset, pass its id here.
- **silent** (*bool*) – (optional) If true, do not perform threshold-based email notifications for this scalar.
- **unofficial** (*bool*) – (optional) If true, creates an unofficial scalar visible only to the user performing the submission.
- **build_results_url** (*string*) – (optional) A URL for linking to build results for this submission.
- **branch** (*string*) – (optional) The branch name in the source repository for this submission.
- **submission_id** (*int | long*) – (optional) The id of the submission.
- **submission_uuid** (*string*) – (optional) The uuid of the submission. If one does not exist, it will be created.
- **params** (*dict*) – (optional) Any key/value pairs that should be displayed with this scalar result.
- **extra_urls** (*list[dict]*) – (optional) Other URL's that should be displayed with with this scalar result. Each element of the list should be a dict with the following keys: label, text, href
- **unit** (*string*) – (optional) The unit of the scalar value.
- **reproduction_command** (*string*) – (optional) The command to reproduce this scalar.

Returns The scalar object that was created.

Return type dict

associate_item_with_scalar_data(*token*, *item_id*, *scalar_id*, *label*)
Associate a result item with a particular scalar value.

Parameters

- **token** (*string*) – A valid token for the user in question.
- **item_id** (*int | long*) – The id of the item to associate with the scalar.
- **scalar_id** (*int | long*) – Scalar id with which to associate the item.
- **label** (*string*) – The label describing the nature of the association.

create_submission(*token*, ***kwargs*)
Associate a result item with a particular scalar value.

Parameters **token** (*string*) – A valid token for the user in question.

:param uuid (optional) The uuid of the submission (must be unique) :type uuid: string :param name (optional) The name of the submission :type name: string :returns: The submission object that was created. :rtype: dict

upload_json_results(*token*, *filepath*, *community_id*, *producer_display_name*, *metric_name*, *producer_revision*, *submit_time*, ***kwargs*)
Upload a JSON file containing numeric scoring results to be added as scalars. File is parsed and then deleted from the server.

Parameters

- **token** – A valid token for the user in question.
- **filepath** – The path to the JSON file.
- **community_id** – The id of the community that owns the producer.
- **producer_display_name** – The display name of the producer.
- **producer_revision** – The repository revision of the producer that produced this value.
- **submit_time** – The submit timestamp. Must be parsable with PHP strtotime().
- **config_item_id** – (optional) If this value pertains to a specific configuration item, pass its id here.
- **test_dataset_id** – (optional) If this value pertains to a specific test dataset, pass its id here.
- **truth_dataset_id** – (optional) If this value pertains to a specific ground truth dataset, pass its id here.
- **parent_keys** – (optional) Semicolon-separated list of parent keys to look for numeric results under. Use '.' to denote nesting, like in normal javascript syntax.
- **silent** – (optional) If true, do not perform threshold-based email notifications for this scalar.
- **unofficial** – (optional) If true, creates an unofficial scalar visible only to the user performing the submission.
- **build_results_url** – (optional) A URL for linking to build results for this submission.
- **branch** – (optional) The branch name in the source repository for this submission.
- **params** (*dict*) – (optional) Any key/value pairs that should be displayed with this scalar result.

- **extra_urls** (*list of dicts*) – (optional) Other URL's that should be displayed with with this scalar result. Each element of the list should be a dict with the following keys: label, text, href

Returns The list of scalars that were created.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

p

A

B

C

D

E

F

G

I

L