render-python Documentation

Release 1.0.1

Forrest Collman, Russel Torres, Eric Perlman

Sep 12, 2017

Contents:

CHAPTER 1

API

User Guide

Getting Started

First you must have a render server running and accessible to you. For instructions on getting a render server up and running http://www.github.com/saalfeldlab/render

For the purposes of this tutorial i'm going to assume you have a render-server installed and up and running on localhost and your render dashboard is viewable at http://localhost:8080/render-ws/view/index.html

You also should have available on your local system a path to the render-ws-java-client scripts with a compiled jar file. One way to get these things, with render-python already installed is to use get the render-python Docker image (fcollmanrender-python), where it will be installed at usrlocalrenderrender-ws-java-clientsrcscripts

```
$ docker pull fcollman\render-python
$ docker run -t fcollman\render-python \
    -v .:\scripts \
    python \scripts\my_script.py
```

Making a new stack

First we have to setup our default connection properties for our render server

```
import renderapi

#create a renderapi.connect.Render object
render_connect_params ={
    'host':'localhost',
    'port':8080,
    'owner':'myowner',
    'project':'myproject',
    'client_scripts':'\usr\local\render\render-ws-java-client\src\scripts'
```

```
    'memGB':'2G'
}
render = renderapi.connect(**render_connect_params)
```

You can simplify your call to *renderapi.connect()* if you wish by setting up some or all of the following environment variables DEFAULT_RENDER_HOST, DEFAULT_RENDER_PORT, DEFAULT_RENDER_CLIENT_SCRIPTS, DEFAULT_RENDER_PROJECT, DEFAULT_RENDER_OWNER, RENDER_CLIENT_HEAP.

Now we can create a new stack on the render server

```
#make a new stack
stack = 'mystack'
renderapi.stack.create_stack(stack,render=render)
```

now you should be able to see your stack at http://localhost:8080/render-ws/view/stacks.html. It is presently in the LOADING state as it is awaiting tiles to be loaded. In order to give it some tiles you must fill out some metadata information about the tile. Many fields are technically optional, but it's best to fill them out if you can. In this example we will do it manually, of course most users will likely write code to create this metadata from existing metadata.

```
#define a tile layout
layout = Layout(sectionId='1',
                scopeId='myscope',
                cameraId='mycamera',
                imageRow=0,
                imageCol=0,
                stageX=100.0,
                stageY=300.0,
                rotation=0.0,
                pixelsize=3.0)
```

Next you have to define the set of transformations that should be applied to the raw image. In this example we will use *renderapi.transform.AffineModel*. However, you can use any of the transforms defined in *renderapi.transform* or any transformation in the mpicpg library (https://github.com/axtimwalde/mpicbg) installed with your render server via the *renderapi.transform.Transform* class, provided you know its classname and dataString.

```
#define a simple transformation, here a translation based upon layout
at = AffineTransform(B0=layout.stageX/layout.pixelsize,
                     B1=layout.stageY/layout.pixelsize)
```

Now we can define the actual tile

```
tilespec = TileSpec(tileId='000000000000',
                z=0.0,
                width=2048,
                height=2048,
                imageUrl='/data/images/0_0_0.tif',
                maskUrl=None,
                layout=layout,
                tforms=[at])
```

Note that the path to the imageUrl needs to be a path that is readable by the render server if you want server side rendering of images to function correctly. If you only want to use render to store metadata information then this isn't strictly necessary, but other web-services such as https://github.com/neurodata/ndviz will require this.

Importing tilespecs

Now we can import the tile to our stack using *renderapi.client* which uses the render-ws-java-client scripts to perform client side validation and bounding box estimation of your tiles before uploading them to the server.

```
#use the simple non-parallelized upload option
renderapi.client.import_tilespecs(stack,
                                  [tilespec],
                                  render = render)

#now close the stack
renderapi.stack.set_stack_state(stack, 'COMPLETE', render = render)
```

If you have many tilespecs to import, it often makes sense to parallelize the client side validation and bounding box estimation. So lets simulate the importing of many tiles

```
rows = 10
cols = 20
sections = 500
overlap = .8 #20% overlap
pix = 3.0 #nm
img_width = 2048 #pixels
img_height = img_width

tilespecs = []
for section in range(sections):
    for r in range(rows):
        for c in range(cols):
                layout = Layout(sectionId='%05d'%section,
                                scopeId='myscope',
                                cameraId='mycamera',
                                imageRow=0,
                                imageCol=0,
                                stageX=c*img_width*overlap*pix,
                                stageY=r*img_height*overlap*pix,
                                rotation=0.0,
                                pixelsize=pix)



                #define a simple transformation, here a translation based upon layout
                at = AffineTransform(B0=c*img_width*overlap,
                                     B1=r*img_height*overlap)

                tileId = '%d_%d_%d'%(section,r,c)

                ts = TileSpec(tileId=,
                              z=section,
                              width=img_width,
                              height=img_height,
                              imageUrl='/data/images/%s.tif'%tileId,
                              maskUrl=None,
                              layout=layout,
                              tforms=[at])
                tilespecs.append(ts)
```

This would of course would need to be adapted to suit the needs of your specific situation, but assuming you have a large number of tilespecs, they can be imported more efficently using *renderapi.client.*

import_tilespecs_parallel() which will also close the stack for you if you'd like.

```
renderapi.client.import_tilespecs_parallel(stack,
                                           tilespecs,
                                           close_stack=True,
                                           render = render)
```

When you are done, you should be able to see your stack on the render dashboard.

Transformations

The idea behind render is that it serves as a central place to store the metadata data about image tiles and how they should be tranformed. Central to that concept is what transformations it supports. Render is written in java and uses the mpicbg library (https://github.com/axtimwalde/mpicbg), the same library that backs TrakEM2, to perform all server side image transformation.

Render-python is client side library and can assist you in managing and setting those tranformations, and performing some calculations using the `renderapi.transorm` module. We have focused our initial efforts at supporting the most commonly used types of transformations.

Some transformation types presently support *tform* and 'inverse_tform' methods for calculating where numpy array sets of points map to and from these tranformations. Some presently support *estimate* methods which given a set of source and destination points, allow the estimation of a best fit transformation.

renderapi.transform.estimate_dstpts() simplifies mapping points through an ordered list of transformations. *renderapi.transform.estimate_transformsum()* provides a general way to produce a single *renderapi.transform.Polynomial2DTransform* that approximates a list of tranforms which have implemented *.tform* methods.

One aspect to keep in mind about render is that it supports *renderapi.transform.ReferenceTransform* which allows many tiles to share a common transformation without having to store its parameters directly. This is a conveient way to save on database storage and ensure many tiles are identically manipulated. However, depending on what you want to do, it can make things slightly more complicated.

For example, presently, the java client side scripts used to calculate bounding boxes and validate transformations need to have access to the referenced transformations in order to do their work, even if they already exist in the database. This is what the *sharedTransforms* argument in import methods within *renderapi.client* is for.

The `renderapi.transform.ReferenceTransform.tform()` method does not exist because the *renderapi.transform.ReferenceTransform* transform doesn't have any data about what kind the transform parameters are. Most render-python calls default to returning dereferenced transforms, which avoids this issue. However, this will break the efficency gains if you simply upload those dereferenced transforms. This is why *renderapi.tilespec.get_tile_spec_raw()* exists, in order to give you referenced transforms if you so wish. Also, you can use the *renderapi.client.importTransformChangesClient* to accelerate and simplify many transform modification tasks, as it won't do any client side validation or bounding box calculations.

Pointmatch database

Pointmatchs are locations between two images that correspond. Render has two groups of web services that are both available on the same web interface, the 'tile' based services and the 'pointmatch' services.

They are loosely coupled in the sense that they are stored in distinct databases, and make no assumptions about how the other is structured. This allows them to be deployed and maintained seperately, but can make things confusing if you assume that one knows more about what the other is doing than it does. To make things less confusing there is a set of reccomendations that you can read at *Pointmatch Assumptions*.

Pointmatches are stored by collection, group's and id's (see *reccomendation*) and have a source 'p' and a destination 'q', thus each set of matches in a collection is specified by a pGroupId, pId, qGroupId, qId combination. Functions in the *renderapi.pointmatch* module allow you to make queries on these point matches in various ways, and upload new matches. You might place some point matches between tile 0_0_0 on section 0, and tile 1_0_0 on section 1, using *renderapi.pointmatch.import_matches()* and then retrieve them using *renderapi.pointmatch.get_matches_from_tile_to_tile()*.

```
matches_in={
     'pGroupId':'0',
     'qGroupId':'1',
     'pId':'0_0_0',
     'qId':'1_0_0',
     'matches':{
         'p':[[0,0],[1000,1000],[1000,0],[0,1000]],
         'q':[[0,0],[1000,1000],[1000,0],[0,1000]],
         'w':[1,1,1,1]
     }
 }
 renderapi.pointmatch.import_matches('mycollection',
                                     [matches_in],
                                     render=render)
 matches_out=renderapi.pointmatch.get_matches_from_tile_to_tile('mycollection'
                                                                '0',
                                                                '0_0_0,
                                                                '1',
                                                                '1_0_0',
                                                                render = render)

 print(matches_out)
 >> [{
     'pGroupId':'0',
     'qGroupId':'1',
     'pId':'0_0_0',
     'qId':'1_0_0',
     'matches':{
         'p':[[0,0],[1000,1000],[1000,0],[0,1000]],
         'q':[[0,0],[1000,1000],[1000,0],[0,1000]],
         'w':[1,1,1,1]
     }
 }]
```

Installation

from source

```
$ git clone https://www.github.com/fcollman/render-python
$ cd render-python
$ python setup.py install
```

Pointmatch Assumptions

Generally, I would reccomend adopting a set of conventions that constrain the relationship between data in the 'tile' and 'pointmatch' databases. These conventions are informed in large part based upon how the EMAligner https://github.com/khaledkhairy/EM_aligner assumes they are related, when using them to solve alignment problems.

tile_owners = pointmatch_owners

Particularly on deployments which have only a single render service, this will save you from having to redefine the default owner in the *renderapi.render.Render* or override it at each step. Make all stacks related to a dataset owned by a single owner, and make all pointmatch collections owned by that same owner.

groupId=sectionId and id=tileId

Or more verbosely, groupId/qGroupId/pGroupId = sectionId and pId/qId/id = tileId. Groups thus correspond to what section the point match is from and the id's correspond to what tile they are from. Technically, there is no need to make this association, and none of the render web services strictly require it. However, if you want to use the pointmatch results in combination with the tile services, it will be far easier if there is a strict mapping between these two databases. In addition, tools like EMAligner and ndviz are presently written in a way that assumes this mapping is held, so you need to make the same assumption if you want to use those services.

Know your 'local'

Write all pointmatches between tiles in a consistent 'local' coordinate system and make that local coordinate system the raw image space given by rendering the tile using *renderapi.image.get_tile_image_data()* with normalizeForMatching=True and scale=1.0.

This would be conceptually simple, if the 'local' meant the same as *renderapi.coordinate* module defines local to mean, namely the raw image space, with the upper left hand pixel at 0,0 and positive x to the right and positive y down.

However, in some deployments of render this is not the case, and you might find that rendering a tile using normalizeForMatching=True does not produce a raw image tile. In fact it might render blank data in some circumstances if you have more than 1 transformation.

This is because, at Janelia the EMAligner was developed on TEM images that need a lens correction transformation, and the pointmatches are defined on the 'local' coordinate system after lens correction. This simplifies solving for the non-lens component of the transformation, as the EMAligner only needs to specify the single transformation that brings 'local' pointmatches into 'global' alignment and can safely disregard the non-linear effects of the lens correction.

However, it produces the confusing result that mapping the 'local' point matches coordinates through *renderapi.coordinate.local_to_world_coordinates()* does not give the correct result, and you have to have a second stack with lens correction transformations removed in order to map point match coordinates from local to world coordinates accurately using the coordinate mapping service.

More discussion on this at https://github.com/saalfeldlab/render/issues/13, https://github.com/saalfeldlab/render/issues/31.

I implemented an alternative strategy at https://github.com/saalfeldlab/render/pull/29 which adds a removeAllOption=True to many render calls which simply removes all transformations from the tilespec before returning or rendering. In applications where non-linear lens corrections are minimal, this simplifies things.

However for TEM or other applications with stereotyped non-linear transformations, it will make using the EMAligner to solve alignment problems more difficult, as the EMAligner doesn't know that it should map the pointmatches into the post non-linear correction space before attemptign to solve and isn't presently written to do this.

renderapi package

Submodules

renderapi.client module

render functions relying on render-ws client scripts

class renderapi.client.**WithPool**(**args*, ***kwargs*)
Bases: pathos.multiprocessing.ProcessPool

pathos ProcessingPool with functioning __exit__ call

Parameters

- ***args** – variable length argument list matching input to pathos.multiprocessing.Pool
- ****kwargs** – keyword argument input matching pathos.multiprocessing.Pool

Examples

```
>>> with WithPool(number_processes) as pool:
>>>     pool.map(myfunc, myInput)
```

renderapi.client.**call_run_ws_client**(*className*, *add_args=[]*, *renderclient=None*, *memGB=None*, *client_script=None*, *subprocess_mode=None*, ***kwargs*)
simple call for run_ws_client.sh – all arguments set in add_args

Parameters

- **className** (*str*) – Render java client classname to call as first argv for Render's call_run_ws_client.sh wrapper script
- **add_args** (list of str, optional) – command line arguments
- **renderclient** (*renderapi.render.RenderClient*, optional) – render client connection object
- **memGB** (*str, optional*) – GB memory for this java process (defaults to '1G' or value defined in renderclient)
- **client_script** (*str, optional*) – client script to be used as the Render library's call_run_ws_client.sh wrapper script (this option overrides value in renderclient)
- **subprocess_mode** (*str, optional*) – subprocess mode 'call', 'check_call', 'check_output' (default 'call')

Returns result of subprocess_mode call

Return type obj

renderapi.client.**coordinateClient**(*stack*, *z*, *fromJson=None*, *toJson=None*, *localToWorld=None*, *numberOfThreads=None*, *subprocess_mode=None*, *host=None*, *port=None*, *owner=None*, *project=None*, *client_script=None*, *memGB=None*, *render=None*, ***kwargs*)
run CoordinateClient.java

map coordinates between local and world systems

Parameters

- **stack** (*str*) – stack representing the world coordinates
- **z** (*str*) – z value of the section containing the tiles to map
- **fromJson** (*str*) – input json file in format defined by list of coordinate dictionaries (for world to local) or list of list of coordinate dictionaries (local to world)
- **toJson** (*str*) – json to save results of mapping coordinates
- **localToWorld** (*bool*) – whether to transform form local to world coordinates (False if None)
- **numberOfThreads** (*int*) – number of threads for java process (1 if None)

Returns list representing mapped coordinates

Return type `list` of `dict` for local to world or `list` of `list` of `dict` for world to local

`renderapi.client.`**get_param**(*var*, *flag*)

`renderapi.client.`**importJsonClient**(*stack*, *tileFiles=None*, *transformFile=None*, *subprocess_mode=None*, *host=None*, *port=None*, *owner=None*, *project=None*, *client_script=None*, *memGB=None*, *render=None*, ***kwargs*)

run ImportJsonClient.java see render documentation (add link here)

Parameters

- **stack** (*str*) – stack to which tilespecs in tileFiles will be imported
- **tileFiles** (`list` of `str`) – json files containing tilespecs to import
- **transformFile** (*str, optional*) – json file containing transform specs which are referenced by tilespecs in tileFiles
- **render** (*renderapi.render.Render*) – render connection object

`renderapi.client.`**importTransformChangesClient**(*stack*, *targetStack*, *transformFile*, *targetOwner=None*, *targetProject=None*, *changeMode=None*, *close_stack=True*, *subprocess_mode=None*, *host=None*, *port=None*, *owner=None*, *project=None*, *client_script=None*, *memGB=None*, *render=None*, ***kwargs*)

run ImportTransformChangesClient.java

Parameters

- **stack** (*str*) – stack from which tiles will be transformed
- **targetStack** (*str*) – stack that will hold results of transforms
- **transformFile** (*str*) – locaiton of json file in format defined below

```
[{{"tileId": <tileId>,
   "transform": <transformDict>}},
  {{"tileId": ...}},
  ...
]
```

- **targetOwner** (*str*) – owner of target stack
- **targetProject** (*str*) – project of target stack

- **changeMode** (*str*) – method to apply transform to tiles. Options are: 'APPEND' – add transform to tilespec's list of transforms 'REPLACE_LAST' – change last transform in tilespec's

  list of transforms to match transform

  'REPLACE_ALL' – overwrite tilespec's transforms field to match transform

Raises ClientScriptError – if changeMode is not valid

renderapi.client.**import_jsonfiles**(*stack*, *jsonfiles*, *transformFile=None*, *client_scripts=None*, *host=None*, *port=None*, *owner=None*, *project=None*, *close_stack=True*, *render=None*, ***kwargs*)

import jsons using client script serially

Parameters

- **jsonfiles** (list of str) – iterator of filenames to be uploaded
- **transformFile** (*str*) – path to a jsonfile that contains shared transform references (if necessary)
- **close_stack** (*bool*) – mark render stack as COMPLETE after successful import
- **render** (renderapi.render.Render) – render connect object

renderapi.client.**import_jsonfiles_and_transforms_parallel_by_z**(*stack*, *jsonfiles*, *transformfiles*, *poolsize=20*, *client_scripts=None*, *host=None*, *port=None*, *owner=None*, *project=None*, *close_stack=True*, *render=None*, ***kwargs*)

imports json files and transform files in parallel

Parameters

- **stack** (*str*) – the stack to import within
- **jsonfiles** (list of str) – "list of tilespec" json paths to import
- **transformfiles** (list of str) – "list of transform files" paths which matches in a 1-1 way with jsonfiles, so referenced transforms are shared only within a single element of these matched lists. Useful cases where there is as single z transforms shared by all tiles within a single z, but not across z's
- **poolsize** (*int, optional*) – number of processes for multiprocessing pool
- **close_stack** (*bool, optional*) – whether to mark render stack as COMPLETE after successful import
- **render** (renderapi.render.Render) – render connect object
- ****kwargs** – arbitrary keyword arguments

renderapi.client.**import_jsonfiles_parallel**(*stack*, *jsonfiles*, *poolsize=20*, *transformFile=None*, *client_scripts=None*, *host=None*, *port=None*, *owner=None*, *project=None*, *close_stack=True*, *render=None*, ***kwargs*)

import jsons using client script in parallel

Parameters

- **stack** (*str*) – the stack to upload into
- **jsonfiles** (list of str) – list of jsonfile paths to upload
- **poolsize** (*int*) – number of upload processes spawned by multiprocessing pool
- **transformFile** (*str*) – a single json file path containing transforms referenced in the jsonfiles
- **close_stack** (*bool*) – whether to mark render stack as COMPLETE after successful import
- **render** (renderapi.render.Render) – render connect object
- ****kwargs** – arbitrary keyword arguments

renderapi.client.**import_jsonfiles_validate_client**(*stack*, *jsonfiles*, *transformFile=None*, *client_scripts=None*, *host=None*, *port=None*, *owner=None*, *project=None*, *close_stack=True*, *mem=6*, *validator=None*, *render=None*, ***kwargs*)

Uses java client for parallelization and validation

Parameters

- **stack** (*str*) – stack to which jsonfiles should be uploaded
- **jsonfiles** (list of str) – tilespecs in json files
- **transformFile** (*str, optional*) – json file listing transformspecs with ids which are referenced in tilespecs contained in jsonfiles

renderapi.client.**import_single_json_file**(*stack*, *jsonfile*, *transformFile=None*, *client_scripts=None*, *host=None*, *port=None*, *owner=None*, *project=None*, *render=None*, ***kwargs*)

calls client script to import given jsonfile

Parameters

- **stack** (*str*) – stack to import into
- **jsonfile** (*str*) – path to jsonfile to import
- **transformFile** (*str*) – path to a file that contains shared transform references if necessary
- **render** (renderapi.render.RenderClient) – render connect object

renderapi.client.**import_tilespecs**(*stack*, *tilespecs*, *sharedTransforms=None*, *subprocess_mode=None*, *host=None*, *port=None*, *owner=None*, *project=None*, *client_script=None*, *memGB=None*, *render=None*, ***kwargs*)

method to import tilesepcs directly from *renderapi.tilespec.TileSpec* objects

Parameters

- **stack** (*str*) – stack to which tilespecs will be added
- **tilespecs** (list of *renderapi.tilespec.TileSpec*) – list of tilespecs to import
- **sharedTransforms** (list of *renderapi.transform.Transform* or *renderapi.transform.TransformList* or *renderapi.transform.InterpolatedTransform*, optional) – list of shared referenced transforms to be ingested
- **render** (*renderapi.render.Render*) – render connect object

renderapi.client.**import_tilespecs_parallel**(*stack*, *tilespecs*, *sharedTransforms=None*, *subprocess_mode=None*, *poolsize=20*, *close_stack=True*, *host=None*, *port=None*, *owner=None*, *project=None*, *client_script=None*, *memGB=None*, *render=None*, ***kwargs*)

method to import tilesepcs directly from *renderapi.tilespec.TileSpec* objects using pathos.multiprocessing to parallelize

Parameters

- **stack** (*str*) – stack to which tilespecs will be added
- **tilespecs** (list of *renderapi.tilespec.TileSpec*) – list of tilespecs to import
- **sharedTransforms** (obj:*list* of *renderapi.transform.Transform* or *renderapi.transform.TransformList* or *renderapi.transform.InterpolatedTransform*, optional) – list of shared referenced transforms to be ingested
- **poolsize** (*int*) – degree of parallelism to use
- **subprocess_mode** (*str*) – subprocess mode used when calling client side java
- **close_stack** (*bool*) – mark render stack as COMPLETE after successful import
- **render** (*:class:renderapi.render.Render*) – render connect object

renderapi.client.**local_to_world_array**(*stack*, *points*, *tileId*, *subprocess_mode=None*, *host=None*, *port=None*, *owner=None*, *project=None*, *client_script=None*, *memGB=None*, *render=None*, ***kwargs*)

placeholder function for coordinateClient localtoworld

Parameters

- **stack** (*str*) – stack to which world coordinates are mapped
- **points** (*dict*) – local points to map to world
- **tileId** (*str*) – tileId to which points correspond
- **subprocess_mode** (*str*) – subprocess mode used when calling clientside java client

Returns points in world coordinates corresponding to local points

Return type list

renderapi.client.**renderSectionClient**(*stack*, *rootDirectory*, *zs*, *scale=None*, *maxIntensity=None*, *minIntensity=None*, *format=None*, *doFilter=None*, *fillWithNoise=None*, *subprocess_mode=None*, *host=None*, *port=None*, *owner=None*, *project=None*, *client_script=None*, *memGB=None*, *render=None*, ***kwargs*)

run RenderSectionClient.java

Parameters

- **stack** (*str*) – stack to which zs to render belong
- **rootDirectory** (*str*) – directory to which rendered sections should be generated
- **zs** (list of str) – z indices of sections to render
- **scale** (*float*) – factor by which section image should be scaled (this materialization is 32-bit limited)
- **maxIntensity** (*int*) – value todisplay as white on a linear colormap
- **minIntensity** (*int*) – value to display as black on a linear colormap
- **format** (*str*) – output image format in 'PNG', 'TIFF', 'JPEG'
- **doFilter** (*str*) – string representing java boolean for whether to render image with default filter (varies with render version)
- **fillWithNoise** (*str*) – string representing java boolean for whether to replace saturated image values with uniform noise

renderapi.client.**tilePairClient**(*stack*, *minz*, *maxz*, *outjson=None*, *delete_json=False*, *baseowner=None*, *baseproject=None*, *basestack=None*, *xyNeighborFactor=None*, *zNeighborDistance=None*, *excludeCornerNeighbors=None*, *excludeCompletelyObscuredTiles=None*, *excludeSameLayerNeighbors=None*, *excludeSameSectionNeighbors=None*, *excludePairsInMatchCollection=None*, *minx=None*, *maxx=None*, *miny=None*, *maxy=None*, *subprocess_mode=None*, *host=None*, *port=None*, *owner=None*, *project=None*, *client_script=None*, *memGB=None*, *render=None*, ***kwargs*)

run TilePairClient.java see render documentation (#add link here)

This client selects a set of tiles 'p' based on its position in a stack and then searches for nearby 'q' tiles using geometric parameters

Parameters

- **stack** (*str*) – stack from which tilepairs should be considered
- **minz** (*str*) – minimum z bound from which tile 'p' is selected
- **maxz** (*str*) – maximum z bound from which tile 'p' is selected
- **outjson** (*str or None*) – json to which tile pair file should be written (defaults to using temporary file and deleting after completion)
- **delete_json** (*bool*) – whether to delete outjson on function exit (True if outjson is None)
- **baseowner** (*str*) – owner of stack from which stack was derived
- **baseproject** (*str*) – project of stack from which stack was derived
- **basestack** (*str*) – stack from which stack was derived

- **xyNeighborFactor** (*float*) – factor to multiply by max(width, height) of tile 'p' in order to generate search radius in z (0.9 if None)
- **zNeighborDistance** (*int*) – number of z sections defining the half-height of search cylinder for tile 'p' (2 if None)
- **excludeCornerNeighbors** (*bool*) – whether to exclude potential 'q' tiles based on center points falling outside search (True if None)
- **excludeCompletelyObscuredTiles** (*bool*) – whether to exclude potential 'q' tiles that are obscured by other tiles based on Render's sorting (True if None)
- **excludeSameLayerNeighbors** (*bool*) – whether to exclude potential 'q' tiles in the same z layer as 'p'
- **excludeSameSectionNeighbors** (*bool*) – whether to exclude potential 'q' tiles with the same sectionId as 'p'
- **excludePairsInMatchCollection** (*str*) – a matchCollection whose 'p' and 'q' pairs will be ignored if generated using this client
- **minx** (*float*) – minimum x bound from which tile 'p' is selected
- **maxx** (*float*) – maximum x bound from wich tile 'p' is selected
- **miny** (*float*) – minimum y bound from which tile 'p' is selected
- **maxy** (*float*) – maximum y bound from wich tile 'p' is selected

Returns list of tilepairs

Return type list of dict

renderapi.client.**transformSectionClient**(*stack*, *transformId*, *transformClass*, *transformData*, *zValues*, *targetProject=None*, *targetStack=None*, *replaceLast=None*, *subprocess_mode=None*, *host=None*, *port=None*, *owner=None*, *project=None*, *client_script=None*, *memGB=None*, *render=None*, ***kwargs*)

run TranformSectionClient.java

Parameters

- **stack** (*str*) – stack containing section to transform
- **transformId** (*str*) – unique transform identifier
- **transformClass** (*str*) – transform className defined by the java mpicbg library
- **transformData** (*str*) – mpicbg datastring delimited by "," instead of " "
- **zValues** (*list*) – z values to which transform should be applied
- **targetProject** (*str, optional*) – project to which transformed sections should be added
- **targetStack** (*str, optional*) – stack to which transformed sections should be added
- **replaceLast** (*bool, optional*) – whether to replace the last transform in the section with this transform

renderapi.client.**world_to_local_array**(*stack*, *points*, *subprocess_mode=None*, *host=None*, *port=None*, *owner=None*, *project=None*, *client_script=None*, *memGB=None*, *render=None*, ***kwargs*)

placeholder function for coordinateClient worldtolocal

Parameters

- **stack** (*str*) – stack to which world coordinates are mapped
- **points** (*dict*) – local points to map to world
- **subprocess_mode** (*str*) – subprocess mode used when calling client side java
- **render** (*renderapi.render.Render*) – render connect object

Returns dictionaries defining local coordinates and tileIds corresponding to world point

Return type list of list

renderapi.coordinate module

coordinate mapping functions for render api

renderapi.coordinate.**local_to_world_coordinates**(*stack*, *tileId*, *x*, *y*, *host=None*, *port=None*, *owner=None*, *project=None*, *session=<requests.sessions.Session object>*, *render=None*, ***kwargs*)

convert coordinate from local to world with webservice request

Parameters

- **stack** (*str*) – render stack to map coordinates through
- **z** (*float*) – z coordinate to map
- **x** (*float*) – x coordinate to map
- **y** (*float*) – y coordinate to map
- **session** (*requests.session.Session*) – session object used in request
- **render** (renderapi.render.Render) – render connect object

Returns

dictionary of world coordinates following this pattern

```
{
    "tileId": "string",
    "visible": false,
    "world": [
        [0,0],
        [1,0]...
    ],
    "error": "string"
}
```

Return type dict

renderapi.coordinate.**local_to_world_coordinates_array**(*stack*, *dataarray*, *tileId*, *z*, *render=None*, *host=None*, *port=None*, *owner=None*, *project=None*, *client_script=None*, *doClientSide=False*, *number_of_threads=20*, *session=<requests.sessions.Session object>*, ***kwargs*)

map local to world coordinates using numpy array

Parameters

- **stack** (*str*) – render stack to map
- **dataarray** (*numpy.array*) – Nx2 array of points in local coordinates
- **tileId** (*str*) – tile to map points from
- **z** (*float*) – z position to map
- **render** (renderapi.render.Render) – render connect object
- **doClientSide** (*boolean*) – (Default value = False)
- **number_of_threads** (*int*) – (Default value = 20)
- **session** (*requests.session.Session*) – session object used in request
- **render** – render connect object

Returns Nx2 numpy array in world coordinates

Return type numpy.array

renderapi.coordinate.**local_to_world_coordinates_batch**(*stack*, *d*, *z*, *host=None*, *port=None*, *owner=None*, *project=None*, *session=<requests.sessions.Session object>*, *render=None*, ***kwargs*)

convert coordinate parameters from local to world

Parameters

- **stack** (*str*) –
- **d** (*list[dict]*) – list of dictionary of local coordinates to map

```
[ {
"tileId": "string",
"local": [
    [0,0],
    [1,0]...
],
"error": "string"
}]
```

- **z** (*float*) – z coordinate to map from
- **session** – (Default value = requests.session()
- **render** (renderapi.render.Render) – render connect object

Returns

list of dictionaries containing world coordinates

```
[ {
"tileId": "string",
"world": [
    [0,0],
    [1,0]...
],
"error": "string"
}]
```

Return type list[dict]

renderapi.coordinate.**local_to_world_coordinates_clientside**(*stack*, *jsondata*, *z*, *host=None*, *port=None*, *owner=None*, *project=None*, *client_script=None*, *number_of_threads=20*, *render=None*, ***kwargs*)

map_coordinates_clientside for mapping local to world

Parameters

- **stack** (*str*) – render stack to map
- **jsondata** (*list[dict]*) – local coordinates in dictionary format
- **z** (*float*) – z position to map
- **number_of_threads** (*int*) – threads for java client script to use during mapping

Returns world coordinates in dictionary format

Return type dict

renderapi.coordinate.**map_coordinates_clientside**(*stack*, *jsondata*, *z*, *host*, *port*, *owner*, *project*, *client_script*, *isLocalToWorld=False*, *store_injson=False*, *store_outjson=False*, *number_of_threads=20*, *memGB='1G'*)

map coordinates using the java client library

Parameters

- **stack** (*str*) – stack to map
- **jsondata** (*dict*) – json dictionary to map following the pattern of local>world or world>local
- **z** (*float*) – z position to map
- **isLocalToWorld** (*boolean*) – whether transform is local to world (False implies world to local)
- **store_injson** (*boolean*) – whether to store input json file (created with tempfile)
- **store_outjson** (*boolean*) – whether to store output json file (created with tempfile)

- **number_of_threads** (*int*) – threads to execute clientside computation
- **render** (renderapi.render.Render) – render connect object

Returns json data as would be returned by client calls of local>world or world>local

Return type *json*

renderapi.coordinate.**package_point_match_data_into_json**(*dataarray*, *tileId*, *local_or_world='local'*)

Convert a set of points defined by a numpy array and a tileId to a json for use in the renderapi

Parameters

- **dataarray** (*numpy.array*) – a Nx2 array of points
- **tileId** (*str*) – a tileId to package them into
- **local_or_world** –

  whether this should be represented as a local or world coordinate (Default value = 'local')

Returns

dictionary representation of those points and tileId following

```
{
    "tileId": "string",
    "world": [
        [0,0],
        [1,0]...
    ],
    "error": "string"
}
```

Return type dict

renderapi.coordinate.**unpackage_local_to_world_point_match_from_json**(*json_answer*)

converts a local>world call json response into a numpy array

Parameters **json_answer** (*list[dict]*) – response from a local>world call (N long)

Returns Nx2 numpy array of coordinates

Return type numpy.array

renderapi.coordinate.**unpackage_world_to_local_point_match_from_json**(*json_answer*, *tileId*)

Converts a dictionary answer from a world>local coordinates call from a dictionary to numpy array format

Parameters

- **json_answer** (*list[dict]*) – json reponse from a world>local call (N long)
- **tileId** (*str*) – tileId to extract, usually the world tileId passed in

Returns Nx2 array of local points

Return type numpy.array

renderapi.coordinate.**world_to_local_coordinates**(*stack*, *z*, *x*, *y*, *host=None*, *port=None*, *owner=None*, *project=None*, *session=<requests.sessions.Session object>*, *render=None*, ***kwargs*)

maps an world x,y,z coordinate in stack to a local coordinate :param stack: render stack to map coordinates

through :type stack: str :param z: z coordinate to map :type z: float :param x: x coordinate to map :type x: float :param y: y coordinate to map :type y: float :param session: session object used in request :type session: requests.session.Session :param render: render connect object :type render: renderapi.render.Render

Returns

list of dictionaries of local coordinates following this pattern

```
[
    {
        "tileId": "string",
        "visible": false,
        "local": [
            [0,0],
            [1,0]...
        ],
        "error": "string"
    }
]
```

Return type *json*

renderapi.coordinate.**world_to_local_coordinates_array**(*stack*, *dataarray*, *tileId*, *z*, *render=None*, *host=None*, *port=None*, *owner=None*, *project=None*, *client_script=None*, *doClientSide=False*, *number_of_threads=20*, *session=<requests.sessions.Session object>*, ***kwargs*)

map world to local coordinates using numpy array

Parameters

- **stack** (*str*) – render stack to map
- **dataarray** (*numpy.array*) – Nx2 numpy array of points to world points to map
- **tileId** (*str*) – tileId to map from and to
- **z** (*float*) – z coordinate to map
- **render** (renderapi.render.Render) – render connect object
- **doClientSide** (*boolean*) – (Default value = False)
- **number_of_threads** (*int*) – (Default value = 20)
- **session** (*requests.session.Session*) – session object used in request

Returns Nx2 numpy array of points in local coordinates

Return type numpy.array

renderapi.coordinate.**world_to_local_coordinates_batch**(*stack*, *d*, *z*, *host=None*, *port=None*, *owner=None*, *project=None*, *execute_local=False*, *session=<requests.sessions.Session object>*, *render=None*, ***kwargs*)

convert coordinate parameters from world to local

Parameters

- **stack** (*str*) – stack to map coordinates
- **d** (*list[dict]*) –

  list of dictionary of world coordinates to map following this schema

```
[ {
    "tileId": "string",
    "world": [
        [0,0],
        [1,0]...
    ],
    "error": "string"
    }]
```

- **z** (*float*) – z coordinate to map
- **execute_local** (*boolean*) – (Default value = False)
- **session** (*requests.session.Session*) – session object used in request
- **render** (renderapi.render.Render) – render connect object

Returns

list of lists of dictionaries containing local positions that overlap with this point, (one world point may map to multiple local points) following..

```
[[ {
 "tileId": "string",
 "visible": True,False,
 "local": [
     [0,0],
     [1,0]...
 ],
 "error": "string"
 }]
 ]
```

Return type list[list[dict]]

renderapi.coordinate.**world_to_local_coordinates_clientside**(*stack*, *jsondata*, *z*, *host=None*, *port=None*, *owner=None*, *project=None*, *client_script=None*, *number_of_threads=20*, *render=None*, ***kwargs*)

map_coordinates_clientside for mapping world to local

Parameters

- **stack** (*str*) – render stack to map
- **jsondata** (*dict*) – world coordinates in dictionary format
- **z** (*float*) – z coordinate to map

- **number_of_threads** (*int*) – number of threads to use when doing parallelization
- **render** (renderapi.render.Render) – render connect object

Returns local coordinates in dictionary format

Return type *json*

renderapi.errors module

Custom errors for render api

exception renderapi.errors.**ClientScriptError**
Bases: *renderapi.errors.RenderError*

exception renderapi.errors.**ConversionError**
Bases: *renderapi.errors.RenderError*

exception renderapi.errors.**EstimationError**
Bases: *renderapi.errors.RenderError*

exception renderapi.errors.**RenderError**
Bases: exceptions.Exception

exception renderapi.errors.**SpecError**
Bases: *renderapi.errors.RenderError*

renderapi.image module

renderapi.image.**get_bb_image**(*stack*, *z*, *x*, *y*, *width*, *height*, *scale=1.0*, *minIntensity=None*, *maxIntensity=None*, *binaryMask=None*, *filter=None*, *maxTileSpecsToRender=None*, *host=None*, *port=None*, *owner=None*, *project=None*, *img_format=None*, *session=<requests.sessions.Session object>*, *render=None*, ***kwargs*)

render image from a bounding box defined in xy and return numpy array:

renderapi.render.renderaccess() decorated function

Parameters

- **stack** (*str*) – name of render stack to get image from
- **z** (*float*) – z value to render
- **x** (*int*) – leftmost point of bounding rectangle
- **y** (*int*) – topmost pont of bounding rectangle
- **width** (*int*) – number of units @scale=1.0 to right (+x() of bounding box to render
- **height** (*int*) – number of units @scale=1.0 down (+y) of bounding box to render
- **scale** (*float*) – scale to render image at (default 1.0)
- **binaryMask** (*bool*) – whether to treat maskimage as binary
- **maxTileSpecsToRender** (*int*) – max number of tilespecs to render
- **filter** (*bool*) – whether to use server side filtering
- **render** (*renderapi.render.Render*) – render connect object

- **session** (requests.sessions.Session) – sessions object to connect with

Returns [N,M,:] array of image data from render

Return type numpy.array

Raises RenderError

renderapi.image.**get_section_image**(*stack*, *z*, *scale=1.0*, *filter=False*, *maxTileSpecsToRender=None*, *img_format=None*, *host=None*, *port=None*, *owner=None*, *project=None*, *session=<requests.sessions.Session object>*, *render=None*, ***kwargs*)

render an section of image

renderapi.render.renderaccess() decorated function

Parameters

- **stack** (*str*) – name of render stack to render image from
- **z** (*float*) – layer Z
- **scale** (*float*) – linear scale at which to render image (e.g. 0.5)
- **filter** (*bool*) – whether or not to apply server side filtering
- **maxTileSpecsToRender** (*int*) – maximum number of tile specs in rendering
- **img_format** (*str*) – one of IMAGE_FORMATS 'png','.png','jpg','jpeg', '.jpg','tif','.tif','tiff'
- **render** (*renderapi.render.Render*) – render connect object
- **session** (*requests.sessions.Session*) – sessions object to connect with

Returns [N,M,:] array of image data of section from render

Return type numpy.array

Examples

```
>>> import renderapi
>>> render = renderapi.render.connect('server',8080,'me','myproject')
>>> img = render.run(renderapi.stack.get_section_image,'mystack',3.0)
```

renderapi.image.**get_tile_image_data**(*stack*, *tileId*, *normalizeForMatching=True*, *removeAllOption=False*, *scale=None*, *filter=None*, *host=None*, *port=None*, *owner=None*, *project=None*, *img_format=None*, *session=<requests.sessions.Session object>*, *render=None*, ***kwargs*)

render image from a tile with all transforms and return numpy array

renderapi.render.renderaccess() decorated function

Parameters

- **stack** (*str*) – name of render stack to get tile from
- **tileId** (*str*) – tileId of tile to render
- **normalizeForMatching** (*bool*) – whether to render the tile with transformations removed ('local' coordinates)

- **removeAllOption** (*bool*) – whether to remove all transforms from image when doing normalizeForMatching some versions of render only remove the last transform from list. (or remove till there are max 3 transforms)
- **scale** (*float*) – force scale of image
- **filter** (*bool*) – whether to apply server side filtering to image
- **img_format** (*str*) – image format: one of IMAGE_FORMATS = 'png','.png','jpg', 'jpeg','.jpg','tif','.tif','tiff'
- **render** (*renderapi.render.Render*) – render connect object
- **session** (requests.sessions.Session) – sessions object to connect with

Returns [N,M,:] array of image data from render

Return type numpy.array

Raises RenderError

renderapi.pointmatch module

Point Match APIs

renderapi.pointmatch.**add_merge_collections**(*request_url*, *mcs*)
utility function to add mergeCollections to request_url

Parameters

- **request_url** (*str*) – request url
- **mcs** (list of str) – list of mergeCollections to add

Returns request_url with ?mergeCollection=mc[0]&mergeCollection=mc[1]... appended

Return type str

renderapi.pointmatch.**delete_point_matches_between_groups**(*matchCollection*, *pGroupId*, *qGroupId*, *render=None*, *owner=None*, *host=None*, *port=None*, *session=<requests.sessions.Session object>*, ***kwargs*)
delete all the matches between two specific groups deletes all matches where (pgroup == pGroupId and qgroup == qGroupId) OR (pgroup == qGroupId and qgroup == pGroupId()

renderapi.render.renderaccess() decorated function

Parameters

- **matchCollection** (*str*) – matchCollection name
- **pgroup** (*str*) – first group
- **qgroup** (*str*) – second group
- **mergeCollections** (list of str or None) – other matchCollections to aggregate into answer
- **owner** (*unicode*) – matchCollection owner (fallback to render.DEFAULT_OWNER) (note match owner != stack owner always)

- **render** (Render) – Render connection object
- **session** (*requests.session.Session*) – requests session

Returns list of matches (see matches definition)

Return type list of dict

Raises RenderError – if cannot get a reponse from server

renderapi.pointmatch.**get_match_groupIds**(*matchCollection*, *owner=None*, *host=None*, *port=None*, *session=<requests.sessions.Session object>*, *render=None*, ***kwargs*)

get all the groupIds in a matchCollection

renderapi.render.renderaccess() decorated function

Parameters

- **matchCollection** (*str*) – matchCollection name
- **owner** (*str*) – matchCollection owner (fallback to render.DEFAULT_OWNER) (note match owner != stack owner always)
- **render** (Render) – Render connection object
- **session** (*requests.session.Session*) – requests session

Returns groupIds in matchCollection

Return type list of str

Raises RenderError – if cannot get a reponse from server

renderapi.pointmatch.**get_match_groupIds_from_only**(*matchCollection*, *mergeCollections=None*, *render=None*, *owner=None*, *host=None*, *port=None*, *session=<requests.sessions.Session object>*, ***kwargs*)

get all the source pGroupIds in a matchCollection

renderapi.render.renderaccess() decorated function

Parameters

- **matchCollection** (*str*) – matchCollection name
- **owner** (*unicode*) – matchCollection owner (fallback to render.DEFAULT_OWNER) (note match owner != stack owner always)
- **render** (RenderClient) – RenderClient connection object
- **session** (*requests.session.Session*) – requests session

Returns pGroupIds in matchCollection

Return type list of str

Raises RenderError – if cannot get a reponse from server

renderapi.pointmatch.**get_match_groupIds_to_only**(*matchCollection*, *mergeCollections=None*, *render=None*, *owner=None*, *host=None*, *port=None*, *session=<requests.sessions.Session object>*, ***kwargs*)

get all the destination qGroupIds in a matchCollection

renderapi.render.renderaccess() decorated function

Parameters

- **matchCollection** (*str*) – matchCollection name
- **owner** (*unicode*) – matchCollection owner (fallback to render.DEFAULT_OWNER) (note match owner != stack owner always)
- **render** (Render) – Render connection object
- **session** (*requests.session.Session*) – requests session

Returns qGroupIds in matchCollection

Return type list of str

Raises RenderError – if cannot get a reponse from server

renderapi.pointmatch.**get_matchcollection_owners**(*host=None*, *port=None*, *session=<requests.sessions.Session object>*, *render=None*, ***kwargs*)

get all the matchCollection owners

renderapi.render.renderaccess() decorated function

Parameters

- **render** (renderapi.render.Render) – Render connection object
- **session** (*requests.session.Session*) – requests session

Returns matchCollection owners

Return type list of unicode

Raises RenderError – if cannot get a reponse from server

renderapi.pointmatch.**get_matchcollections**(*owner=None*, *host=None*, *port=None*, *session=<requests.sessions.Session object>*, *render=None*, ***kwargs*)

get all the matchCollections owned by owner

renderapi.render.renderaccess() decorated function

Parameters

- **owner** (*unicode*) – matchCollection owner (fallback to render.DEFAULT_OWNER) (note match owner != stack owner always)
- **render** (Render) – Render connection object
- **session** (*requests.session.Session*) – requests session

Returns matchcollections owned by owner

Return type list of unicode

Raises RenderError – if cannot get a reponse from server

renderapi.pointmatch.**get_matches_from_group_to_group**(*matchCollection*, *pgroup*, *qgroup*, *mergeCollections=None*, *render=None*, *owner=None*, *host=None*, *port=None*, *session=<requests.sessions.Session object>*, ***kwargs*)

get all the matches between two specific groups returns all matches where pgroup == pGroupId and qgroup == qGroupId OR pgroup == qGroupId and qgroup == pGroupId

renderapi.render.renderaccess() decorated function

Parameters

- **matchCollection** (*str*) – matchCollection name
- **pgroup** (*str*) – first group
- **qgroup** (*str*) – second group
- **mergeCollections** (list of str or None) – other matchCollections to aggregate into answer
- **owner** (*unicode*) – matchCollection owner (fallback to render.DEFAULT_OWNER) (note match owner != stack owner always)
- **render** (RenderClient) – RenderClient connection object
- **session** (*requests.session.Session*) – requests session

Returns list of matches (see matches definition)

Return type list of dict

Raises RenderError – if cannot get a reponse from server

renderapi.pointmatch.**get_matches_from_tile_to_tile**(*matchCollection*, *pgroup*, *pid*, *qgroup*, *qid*, *mergeCollections=None*, *render=None*, *owner=None*, *host=None*, *port=None*, *session=<requests.sessions.Session object>*, ***kwargs*)

get all the matches between two specific tiles returns all matches where pgroup == pGroupId and pid=pId and qgroup == qGroupId and qid == qId OR qgroup == pGroupId and Qid=pId and Pgroup == qGroupId and pid == qId

renderapi.render.renderaccess() decorated function

Parameters

- **matchCollection** (*str*) – matchCollection name
- **pgroup** (*str*) – first group
- **pid** (*str*) – first id
- **qgroup** (*str*) – second group
- **qid** (*str*) – second id
- **mergeCollections** (list of str or None) – other matchCollections to aggregate into answer
- **owner** (*unicode*) – matchCollection owner (fallback to render.DEFAULT_OWNER) (note match owner != stack owner always)
- **render** (RenderClient) – RenderClient connection object
- **session** (*requests.session.Session*) – requests session

Returns list of matches (see matches definition)

Return type list of dict

Raises `RenderError` – if cannot get a reponse from server

`renderapi.pointmatch.`**`get_matches_involving_tile`**(*matchCollection*, *groupId*, *id*, *mergeCollections=None*, *owner=None*, *host=None*, *port=None*, *session=<requests.sessions.Session object>*, ***kwargs*)

get all the matches involving a specific tile returns all matches where groupId == pGroupId and id == pId OR groupId == qGroupId and id == qId

renderapi.render.renderaccess() decorated function

Parameters

- **matchCollection** (*str*) – matchCollection name
- **groupId** (*str*) – groupId to query
- **id** (*str*) – id to query
- **mergeCollections** (`list` of `str`, optional) – other matchCollections to aggregate into answer
- **owner** (*unicode*) – matchCollection owner (fallback to render.DEFAULT_OWNER) (note match owner != stack owner always)
- **render** (`Render`) – Render connection object
- **session** (*requests.session.Session*) – requests session

Returns list of matches (see matches definition)

Return type `list` of `dict`

Raises `RenderError` – if cannot get a reponse from server

`renderapi.pointmatch.`**`get_matches_outside_group`**(*matchCollection*, *groupId*, *mergeCollections=None*, *owner=None*, *host=None*, *port=None*, *session=<requests.sessions.Session object>*, *render=None*, ***kwargs*)

get all the matches outside a groupId in a matchCollection returns all matches where pGroupId == groupId and qGroupId != groupId

renderapi.render.renderaccess() decorated function

Parameters

- **matchCollection** (*str*) – matchCollection name
- **groupId** (*str*) – groupId to query
- **mergeCollections** (`list` of `str`) – other matchCollections to aggregate into answer
- **owner** (*unicode*) – matchCollection owner (fallback to render.DEFAULT_OWNER) (note match owner != stack owner always)
- **render** (`Render`) – Render connection object
- **session** (*requests.session.Session*) – requests session

Returns list of matches (see matches definition)

Return type `list` of `dict`

Raises `RenderError` – if cannot get a reponse from server

renderapi.pointmatch.**get_matches_with_group**(*matchCollection*, *pgroup*, *mergeCollections=None*, *render=None*, *owner=None*, *host=None*, *port=None*, *session=<requests.sessions.Session object>*, ***kwargs*)

get all the matches from a specific groups returns all matches where pgroup == pGroupId

renderapi.render.renderaccess() decorated function

Parameters

- **matchCollection** (*str*) – matchCollection name
- **pgroup** (*str*) – source group to query
- **mergeCollections** (list of str or None) – other matchCollections to aggregate into answer
- **owner** (*unicode*) – matchCollection owner (fallback to render.DEFAULT_OWNER) (note match owner != stack owner always)
- **render** (Render) – Render connection object
- **session** (*requests.session.Session*) – requests session

Returns list of matches (see matches definition)

Return type list of dict

Raises RenderError – if cannot get a reponse from server

renderapi.pointmatch.**get_matches_within_group**(*matchCollection*, *groupId*, *mergeCollections=None*, *owner=None*, *host=None*, *port=None*, *session=<requests.sessions.Session object>*, *render=None*, ***kwargs*)

get all the matches within a groupId in a matchCollection returns all matches where pGroupId == groupId and qGroupId == groupId

renderapi.render.renderaccess() decorated function

Parameters

- **matchCollection** (*str*) – matchCollection name
- **groupId** (*str*) – groupId to query
- **mergeCollections** (list of str or None) – other matchCollections to aggregate into answer
- **owner** (*unicode*) – matchCollection owner (fallback to render.DEFAULT_OWNER) (note match owner != stack owner always)
- **render** (RenderClient) – RenderClient connection object
- **session** (*requests.session.Session*) – requests session

Returns list of matches (see matches definition)

Return type list of dict

Raises RenderError – if cannot get a reponse from server

renderapi.pointmatch.**import_matches**(*matchCollection*, *data*, *owner=None*, *host=None*, *port=None*, *session=<requests.sessions.Session object>*, *render=None*, ***kwargs*)

import matches into render database

renderapi.render.renderaccess() decorated function

Parameters

- **matchCollection** (*str*) – matchCollection name
- **data** (list of dict) – list of matches to import (see matches definition)
- **owner** (*unicode*) – matchCollection owner (fallback to render.DEFAULT_OWNER) (note match owner != stack owner always)
- **render** (Render) – Render connection object
- **session** (*requests.session.Session*) – requests session

Returns server response

Return type requests.response.Reponse

renderapi.render module

class renderapi.render.**Render**(*host=None*, *port=None*, *owner=None*, *project=None*, *client_scripts=None*)

Bases: object

Render object to store connection settings for render server. Baseclass that doesn't require client_scripts definition for client side java processing.

See *connect()* for parameter definitions.

DEFAULT_HOST
str – render host to which make_kwargs will default

DEFAULT_PORT
int – render port to which make_kwargs will default

DEFAULT_OWNER
str – render owner to which make_kwargs will default

DEFAULT_PROJECT
str – render project to which make_kwargs will default

DEFAULT_CLIENT_SCRIPTS
str – render client scripts path to which make_kwargs will default

DEFAULT_KWARGS
"kwargs to which the render object falls back. Depends on – self.DEFAULT_HOST, self.DEFAULT_OWNER, self.DEFAULT_PORT, self.DEFAULT_PROJECT, self.DEFAULT_CLIENT_SCRIPTS

Returns default keyword arguments

Return type dict

make_kwargs(*host=None*, *port=None*, *owner=None*, *project=None*, *client_scripts=None*, ***kwargs*)
make kwargs using this render object's defaults and any designated kwargs passed in

Parameters

- **host** (*str or None*) – render webservice host
- **port** (*int or None*) – render webservice port
- **owner** (*str or None*) – render webservice owner
- **project** (*str or None*) – render webservice project
- **client_scripts** (*str or None*) – render java client script location
- ****kwargs** – all other keyword arguments passed through

Returns keyword arguments with missing host,port,owner,project,client_scripts filled in with defaults

Return type dict

run (*f*, **args*, ***kwargs*)

run function from object technically shorter than adding render=Render to kwargs

Parameters

- **f** (*func*) – renderapi function you want to call
- ***args** – args passed to that function
- ****kwargs** – kwargs passed to that function

Returns function with this *Render* instance in keyword arguments as render=

Return type func

Examples

```
>>> render = Render('server',8080)
>>> metadata = render.run(renderapi.render.get_stack_metadata_by_owner,
↪'myowner')
```

class renderapi.render.**RenderClient**(*client_script=None*, *memGB=None*, *validate_client=True*, **args*, ***kwargs*)

Bases: *renderapi.render.Render*

Render object to run java client commands via a wrapped client script. Should use *connect()* to create and for documentation of parameters.

DEFAULT_HOST
str – render host to which make_kwargs will default

DEFAULT_PORT
int – render port to which make_kwargs will default

DEFAULT_OWNER
str – render owner to which make_kwargs will default

DEFAULT_PROJECT
str – render project to which make_kwargs will default

DEFAULT_CLIENT_SCRIPTS
str – render client scripts path to which make_kwargs will default

client_script
str – location of wrapper script for java client with input same as Render java client's run_ws_client.sh

memGB
str – string defining heap in GB to be utilized by java clients (default '1G' for 1 GB)

make_kwargs(**args*, ***kwargs*)
method to fill in default properties of RenderClient object

Parameters

- ***args** – args used to initialize RenderClient
- ****kwargs** – kwargs used to initialize RenderClient

Returns keyword arguments with missing host,port,owner,project,client_scripts,client_script,memGB filled in with defaults

Return type dict

renderapi.render.**connect**(*host=None*, *port=None*, *owner=None*, *project=None*, *client_scripts=None*, *client_script=None*, *memGB=None*, *force_http=True*, *validate_client=True*, *web_only=False*, ***kwargs*)
helper function to create a *Render* instance, or *RenderClient* if sufficent parameters are provided. Will default to using environment variables if not specified in call, and prompt user for any parameters that are not given.

Parameters

- **host** (*str*) – hostname for target render server – will prepend "http://" if host does not begin with 'http' and force_http keyword evaluates True. Can be set by environment variable RENDER_HOST.
- **port** (*str, int, or None*) – port for target render server. Optional as in 'http://hostname{[}:port]'. Can be set by environment variable RENDER_PORT.
- **owner** (*str*) – owner for render-ws. Can be set by environment variable RENDER_OWNER.
- **project** (*str*) – project for render webservice. Can be set by environment variable RENDER_PROJECT.
- **client_scripts** (*str*) – directory path for render-ws-java-client scripts. Can be set by environment variable RENDER_CLIENT_SCRIPTS.
- **client_script** (*str, optional*) – path to a wrapper for java client classes. Used only in RenderClient. Can be set by environment variable RENDER_CLIENT_SCRIPT.
- **memGB** (*str*) – heap size in GB for java client scripts, example for 1 GB: '1G'. Used only in RenderClient. Can be set by environment variable RENDER_CLIENT_HEAP.
- **force_http** (*bool*) – whether to prepend 'http://' to render host if it does not begin with 'http'
- **validate_client** (*bool*) – whether to validate existence of RenderClient run_ws_client.sh script
- **web_only** (*bool*) – whether to check environment variables/prompt user for client_scripts directory if not in arguments

Returns a connect object to simplify specifying what render server to connect to (returns *RenderClient* if sufficent parameters are passed)

Return type *Render*

Raises ValueError – if empty user input is given for required field

renderapi.render.**format_baseurl**(*host*, *port*)
format host and port to a standard template render-ws url

Parameters

- **host** (*str*) – host of render server
- **port** (*int or None*) – port of render server

Returns a url to the render endpoint at that host/port combination (append render-ws/v1)

Return type str

renderapi.render.**format_preamble**(*host*, *port*, *owner*, *project*, *stack*)
format host, port, owner, project, and stack parameters to the access point to stack-based apis

Parameters

- **host** (*str*) – render host
- **port** (*int*) – render host port
- **owner** (*str*) – render owner
- **project** (*str*) – render project
- **stack** (*str*) – render stack

Returns a url to the endpoint for that host, port, owner, project, stack combination

Return type str

renderapi.render.**get_owners**(*host=None*, *port=None*, *session=<requests.sessions.Session object>*, *render=None*, ***kwargs*)
return list of owners across all Projects and Stacks for a render server

renderaccess() decorated function

Parameters

- **host** (*str*) – render host (defaults to host from render)
- **port** (*int*) – render port (default to port from render)
- **session** (*requests.Session*) – requests session
- **render** (RenderClient) – RenderClient connection object

Returns list of strings containing all render owners

Return type list

renderapi.render.**get_projects_by_owner**(*owner=None*, *host=None*, *port=None*, *session=<requests.sessions.Session object>*, *render=None*, ***kwargs*)
return list of projects belonging to a single owner for render stack

renderaccess() decorated function

Parameters

- **owner** (*str*) – render owner
- **render** (RenderClient) – render connect object
- **session** (*requests.sessions.Session*) – http session to use

Returns render projects by this owner

Return type list of unicode

renderapi.render.**get_stack_metadata_by_owner**(*owner=None*, *host=None*, *port=None*, *session=<requests.sessions.Session object>*, *render=None*, ***kwargs*)

return metadata for all stacks belonging to particular owner on render server

renderaccess() decorated function

Parameters

- **owner** (*str*) – render owner
- **render** (RenderClient) – render connect object
- **session** (*requests.sessions.Session*) – http session to use

Returns stackInfo metadata, TODO example

Return type dict

renderapi.render.**get_stacks_by_owner_project**(*owner=None*, *project=None*, *host=None*, *port=None*, *session=<requests.sessions.Session object>*, *render=None*, ***kwargs*)

return list of stacks belonging to an owner's project on render server

renderaccess() decorated function

Parameters

- **owner** (*str*) – render owner
- **project** (*str*) – render project
- **render** (RenderClient) – render connect object
- **session** (*requests.sessions.Session*) – http session to use

Returns render stacks by this owner in this project

Return type list of str

renderapi.render.**renderaccess**(*func*)

Decorator allowing functions asking for host, port, owner, project to default to a connection defined by a *Render* object using its *RenderClient.make_kwargs()* method.

You can if you wish specify any of the arguments, in which case they will not be filled in by the default values, but you don't have to.

As such, the documentation omits describing the parameters which are natural to expect will be filled in by the renderaccess decorator.

Parameters **f** (*func*) – function to decorate

Returns decorated function

Return type func

Examples

```
>>> render = renderapi.render.connect('server',8080,'me','my_project')
>>> stacks = renderapi.render.get_stacks_by_owner_project(render=render)
```

renderapi.stack module

class renderapi.stack.**StackVersion**(*cycleNumber=None*, *cycleStepNumber=None*, *stackResolutionX=None*, *stackResolutionY=None*, *stackResolutionZ=None*, *materializedBoxRootPath=None*, *mipmapPathBuilder=None*, *versionNotes=None*, *createTimestamp=None*, ***kwargs*)

StackVersion, metadata about a stack

cycleNumber
int – cycleNumber, use as you wish to track versions (default None)

cycleStepNumber
int – cycleStepNumber, use as you with to track versions (default None)

stackResolutionX
float – stackResolutionX, resolution of scale = 1.0 in nm

stackResolutionY
float – stackResolutionY, resolution of scale = 1.0 in nm

stackResolutionZ
float – stackResolutionZ, resolution of scale = 1.0 in nm

mipmapPathBuilder
str – path to mipmap builder (?)

materializedBoxRootPath
str – path to materializer (?)

createTimeStamp
str – time stamp of stack creation (default to now)

versionNotes
str – notes about this stack (optional)

from_dict(*d*)
deserialization function

Parameters **d** (*dict*) – dictionary to update the properties of this object

to_dict()
serialization function

Returns json compatible verson of this object

Return type dict

renderapi.stack.**clone_stack**(*inputstack*, *outputstack*, *skipTransforms=False*, *toProject=None*, *zs=None*, *close_stack=True*, *host=None*, *port=None*, *owner=None*, *project=None*, *session=None*, *render=None*, ***kwargs*)

clone a stack

renderapi.render.renderaccess() decorated function

Parameters

- **inputstack** (*str*) – name of input stack to clone
- **outputstack** (*str*) – name of destination stack. if exists, must be LOADING
- **skipTransforms** (*bool*) – boolean whether to strip transformations in new stack (default=False)
- **toProject** (*str*) – string name of destination project (default same as inputstack)

- **zs** (list of float or None) – list of selected z values to clone into stack (optional)
- **close_stack** (*bool*) – whether to set stack to COMPLETE when finished
- **render** (renderapi.render.Render) – render connect object
- **session** (*requests.sessions.Session*) – session object (default start a new one)

Returns server response

Return type requests.session.response

renderapi.stack.**create_stack**(*stack*, *cycleNumber=None*, *cycleStepNumber=None*, *stackResolutionX=None*, *stackResolutionY=None*, *stackResolutionZ=None*, *force_resolution=True*, *host=None*, *port=None*, *owner=None*, *project=None*, *session=<requests.sessions.Session object>*, *render=None*, ***kwargs*)

creates a new stack

renderapi.render.renderaccess() decorated function

Parameters

- **stack** (*str*) – render stack name to create
- **cycleNumber** (*int*) – cycleNumber to use to track stages
- **cycleStepNumber** (*int*) – cycleStepNumber to use to track stages
- **stackResolutionX** (*float*) – resolution of x pixels at scale=1.0
- **stackResolutionY** (*float*) – resolution of y pixels at scale=1.0
- **stackResoluiontZ** (*float*) – resolution of z sections at scale=1.0
- **force_resolution** (*bool*) – fill in resolution of 1.0 for missing resolutions
- **render** (renderapi.render.Render) – render connect object
- **session** (*requests.sessions.Session*) – session object (default start a new one)

Returns server response

Return type requests.session.response

Raises RenderError

renderapi.stack.**delete_section**(*stack*, *z*, *host=None*, *port=None*, *owner=None*, *project=None*, *session=<requests.sessions.Session object>*, *render=None*, ***kwargs*)

removes a single z from a stack

renderapi.render.renderaccess() decorated function

Parameters

- **stack** (*str*) – stack to delete section from
- **z** (*float*) – z value to delete
- **render** (renderapi.render.Render) – render connect object
- **session** (*requests.sessions.Session*) – session object (default start a new one)

Returns server response

Return type requests.session.response

renderapi.stack.**delete_stack**(*stack*, *host=None*, *port=None*, *owner=None*, *project=None*, *session=<requests.sessions.Session object>*, *render=None*, ***kwargs*)

deletes a stack from render server

renderapi.render.renderaccess() decorated function

Parameters

- **stack** (*str*) – stack to delete
- **render** (renderapi.render.Render) – render connect object
- **session** (*requests.sessions.Session*) – session object (default start a new one)

Returns server response

Return type requests.session.response

renderapi.stack.**delete_tile**(*stack*, *tileId*, *host=None*, *port=None*, *owner=None*, *project=None*, *session=<requests.sessions.Session object>*, *render=None*, ***kwargs*)

removes a tile from a stack

renderapi.render.renderaccess() decorated function

Parameters

- **stack** (*str*) – stack to delete tile from
- **tileId** (*str*) – tileId of tilespec to remove from stack
- **render** (renderapi.render.Render) – render connect object
- **session** (*requests.sessions.Session*) – session object (default start a new one)

Returns server response

Return type requests.session.response

renderapi.stack.**get_bounds_from_z**(*stack*, *z*, *host=None*, *port=None*, *owner=None*, *project=None*, *session=<requests.sessions.Session object>*, *render=None*, ***kwargs*)

get a bounds dictionary for a specific z

renderapi.render.renderaccess() decorated function

Parameters

- **stack** (*str*) – stack to get bounds from
- **z** (*float*) – z value to get bounds for
- **render** (renderapi.render.Render) – render connect object
- **session** (*requests.sessions.Session*) – session object (default start a new one)

Returns bounds with keys minY,minY,maxX,maxY,minZ,maxZ

Return type dict

Raises RenderError

renderapi.stack.**get_sectionId_for_z**(*stack*, *z*, *host=None*, *port=None*, *owner=None*, *project=None*, *session=<requests.sessions.Session object>*, *render=None*, ***kwargs*)

returns the sectionId associated with a particular z value

renderapi.render.renderaccess() decorated function

Parameters

- **stack** (*str*) – stack to look within
- **sectionId** (*str*) – sectionId to find z value
- **render** (renderapi.render.Render) – render connect object
- **session** (*requests.sessions.Session*) – session object (default start a new one)

Returns z value of sectionId

Return type float

Raises RenderError

renderapi.stack.**get_section_z_value**(*stack*, *sectionId*, *host=None*, *port=None*, *owner=None*, *project=None*, *session=<requests.sessions.Session object>*, *render=None*, ***kwargs*)

get the z value for a specific sectionId (string)

renderapi.render.renderaccess() decorated function

Parameters

- **stack** (*str*) – render stack string to look within
- **sectionId** (*str*) – sectionId to find z value
- **render** (renderapi.render.Render) – render connect object
- **session** (*requests.sessions.Session*) – session object (default start a new one)

Returns z value of section

Return type float

Raises RenderError

renderapi.stack.**get_stack_bounds**(*stack*, *host=None*, *port=None*, *owner=None*, *project=None*, *session=<requests.sessions.Session object>*, *render=None*, ***kwargs*)

get bounds of a whole stack

renderapi.render.renderaccess() decorated function

Parameters

- **stack** (*str*) – stack to get bounds from
- **render** (renderapi.render.Render) – render connect object
- **session** (*requests.sessions.Session*) – session object (default start a new one)

Returns bounds with keys minY,minY,maxX,maxY,minZ,maxZ

Return type dict

Raises RenderError

renderapi.stack.**get_stack_metadata**(*stack*, *host=None*, *port=None*, *owner=None*, *project=None*, *session=<requests.sessions.Session object>*, *render=None*, ***kwargs*)

get the stack metadata for a stack

renderapi.render.renderaccess() decorated function

Parameters

- **stack** (*str*) – stack to get the metadata for
- **render** (renderapi.render.Render) – render connect object
- **session** (*requests.sessions.Session*) – session object (default start a new one)

Returns metadata of the stack

Return type *StackVersion*

Raises RenderError

renderapi.stack.**get_stack_sectionData**(*stack*, *host=None*, *port=None*, *owner=None*, *project=None*, *session=<requests.sessions.Session object>*, *render=None*, ***kwargs*)

returns information about the sectionIds of each slice in stack

renderapi.render.renderaccess() decorated function

Parameters

- **stack** (*str*) – name of stack to get data about
- **render** (renderapi.render.Render) – render connect object
- **session** (*requests.sessions.Session*) – session object (default start a new one)

Returns

sectionData as below

```
[{
    "sectionId": "string",
    "z": 0,
    "tileCount": 0,
    "minX": 0,
    "maxX": 0,
    "minY": 0,
    "maxY": 0
}]
```

Return type dict

Raises RenderError

renderapi.stack.**get_stack_tileIds**(*stack*, *host=None*, *port=None*, *owner=None*, *project=None*, *session=<requests.sessions.Session object>*, *render=None*, ***kwargs*)

get tileIds for a stack

renderapi.render.renderaccess() decorated function

Parameters

- **stack** (*str*) – stack to get tileIds
- **render** (renderapi.render.Render) – render connect object
- **session** (*requests.sessions.Session*) – session object (default start a new one)

Returns list of tileIds in stack

Return type list of str

Raises RenderError

renderapi.stack.**get_z_value_for_section**(*stack*, *sectionId*, ***kwargs*)
DEPRECATED (use *get_section_z_value()*) instead

renderapi.stack.**get_z_values_for_stack**(*stack*, *project=None*, *host=None*, *port=None*, *owner=None*, *session=<requests.sessions.Session object>*, *render=None*, ***kwargs*)
get a list of z values for which there are tiles in the stack

renderapi.render.renderaccess() decorated function

Parameters

- **stack** (*str*) – stack to get z values for
- **render** (renderapi.render.Render) – render connect object
- **session** (*requests.sessions.Session*) – session object (default start a new one)

Returns z values in stack

Return type list of float

Raises RenderError

renderapi.stack.**likelyUniqueId**(*host=None*, *port=None*, *session=<requests.sessions.Session object>*, *render=None*, ***kwargs*)
return hex-code nearly-unique id from render server

renderapi.render.renderaccess() decorated function

Parameters

- **render** (renderapi.render.Render) – render connect object
- **session** (*requests.sessions.Session*) – session object (default start a new one)

Returns string representation of hex-code

Return type str

renderapi.stack.**make_stack_params**(*host*, *port*, *owner*, *project*, *stack*)
utility function to turn host,port,owner,project,stack combinations to java CLI based argument list for subprocess calling

Parameters

- **host** (*str*) – render server
- **port** (*int*) – render server port
- **owner** (*str*) – render owner
- **project** (*str*) – render project
- **stack** (*str*) – render stack

Returns java CLI list of arguments for subprocess calling

Return type list of str

renderapi.stack.**set_stack_metadata**(*stack*, *sv*, *host=None*, *port=None*, *owner=None*, *project=None*, *session=<requests.sessions.Session object>*, *render=None*, ***kwargs*)
sets the stack metadata for a stack

renderapi.render.renderaccess() decorated function

Parameters

- **stack** (*str*) – stack to set the metadata for
- **sv** (StackVersion) – metadata for the stack
- **render** (renderapi.render.RenderClient) – render connect object
- **session** (*requests.sessions.Session*) – session object (default start a new one)

Returns response from server

Return type requests.response

renderapi.stack.**set_stack_state**(*stack*, *state='LOADING'*, *host=None*, *port=None*, *owner=None*, *project=None*, *session=<requests.sessions.Session object>*, *render=None*, ***kwargs*)

set state of selected stack.

TODO there is a limited direction in which these stack changes can go

renderapi.render.renderaccess() decorated function

Parameters

- **stack** (*str*) – stack to set state for
- **state** (*str*) – state of stack, one of ['LOADING', 'COMPLETE', 'OFFLINE', 'READ_ONLY']
- **render** (renderapi.render.Render) – render connect object
- **session** (*requests.sessions.Session*) – session object (default start a new one)

Returns server response

Return type requests.session.response

Raises RenderError

renderapi.tilespec module

class renderapi.tilespec.**ImagePyramid**(*mipMapLevels=[]*)

Image Pyramid class representing a set of MipMapLevels which correspond to mipmapped (continuously downsmapled by 2x) representations of an image at level 0 Can be put into dictionary formatting using dict(ip) or OrderedDict(ip)

mipMapLevels

list of *MipMapLevel* – list of *MipMapLevel* objects defining image pyramid

append(*mmL*)

appends a MipMapLevel to this ImagePyramid

Parameters **mml** (*MipMapLevel*) – *MipMapLevel* to append

get(*to_get*)

gets a specific mipmap level in dictionary form

Parameters **to_get** (*int*) – level to get

Returns representation of requested MipMapLevel

Return type dict

levels

list of MipMapLevels in this ImagePyramid

to_dict()
return dictionary representation of this object

to_ordered_dict(*key=None*)
returns `OrderedDict` represention of this object, ordered by mipmapLevel

Parameters **key** (*func*) – function to sort ordered dict of `mipMapLevel` dicts (default is by level)

Returns sorted dictionary of *mipMapLevels* in ImagePyramid

Return type OrderedDict

update(*mmL*)
updates the ImagePyramid with this MipMapLevel. will overwrite existing mipMapLevels with same level

Parameters **mml** (`MipMapLevel`) – mipmap level to update in pyramid

class renderapi.tilespec.**Layout**(*sectionId=None*, *scopeId=None*, *cameraId=None*, *imageRow=None*, *imageCol=None*, *stageX=None*, *stageY=None*, *rotation=None*, *pixelsize=None*, *force_pixelsize=True*, ***kwargs*)
Layout class to describe acquisition settings

sectionId
str – sectionId this tile was taken from

scopeId
str – what microscope this came from

cameraId
str – camera this was taken with

imageRow
int – what row from a row,col layout this was taken

imageCol
int – column from a row,col layout this was taken

stageX
float – X stage coordinates for where this was taken

stageY
float – Y stage coordinates for where this taken

rotation
float – angle of camera when this was taken

pixelsize
float – effective size of pixels (in units of choice)

from_dict(*d*)
set this object equal to the fields found in dictionary

Parameters **d** (*dict*) – dictionary to use to update

to_dict()
return a dictionary representation of this object

Returns json compatible dictionary of this object

Return type dict

class renderapi.tilespec.**MipMapLevel**(*level*, *imageUrl=None*, *maskUrl=None*)
MipMapLevel class to represent a level of an image pyramid. Can be put in dictionary formatting using dict(mML)

level
int – level of 2x downsampling represented by mipmaplevel

imageUrl
str or None – uri corresponding to image

maskUrl
str or None – uri corresponding to mask

to_dict()

Returns json compatible dictionary representaton

Return type dict

class renderapi.tilespec.**TileSpec**(*tileId=None*, *z=None*, *width=None*, *height=None*, *imageUrl=None*, *maskUrl=None*, *minint=0*, *maxint=65535*, *layout=None*, *tforms=[]*, *inputfilters=[]*, *scale3Url=None*, *scale2Url=None*, *scale1Url=None*, *json=None*, *mipMapLevels=[]*, ***kwargs*)
Fundamental class of render that store image tiles and their transformations

tileId
str – unique string specifying a tile's identity

z
float – z values this tile exists within

width
int – width in pixels of the raw tile

height
int – height in pixels of the raw tile

imageUrl
str – an image path URI that can be accessed by the render server, with an ImageJ.open command. Preceded by , e.g. 'file://' for files or s3:// for s3

maskUrl
str – an image path that can be accessed by the render server which can be interpreted as an alpha mask for the image (same as spec imageUrl)

minint
int – pixel intensity value to display as black in a linear colormap (default 0)

maxint
int – pixel intensity value to display as white in a linear colormap (default 65535)

layout
Layout – a *Layout* object for this tile

tforms
list of Transform

or :obj:`list` of :obj:`TransformList`

or :obj:`list` of :obj:`InterpolatedTransform`
Transform objects (see *transform.AffineModel*, *transform.TransformList*, *transform.Polynomial2DTransform*, *transform.Transform*, *transform.ReferenceTransform*) to apply to this tile

inputfilters
list – a list of filters to apply to this tile (not yet implemented)

mipMapLevels :obj:`list` of :obj:`MipMapLevel`
MipMapLevel objects for this tile

json
dict or None – dictionary to initialize this object with (if not None overrides and ignores all keyword arguments)

scale3Url
str – uri of a mipmap level 3 image of this tile (DEPRECATED, use mipMapLevels, but will override)

scale2Url
str – uri of a mipmap level 2 image of this tile (DEPRECATED, use mipMapLevels, but will override)

scale1Url
str – uri of a mipmap level 1 image of this tile (DEPRECATED, use mipMapLevels, but will override)

bbox
bbox defined to fit shapely call

from_dict(*d*)
Method to load tilespec from json dictionary

d [dict] dictionary to use to set properties of this object

to_dict()
method to produce a json tilespec for this tile returns a json compatible dictionary

Returns json compatible dictionary representation of this object

Return type dict

renderapi.tilespec.**get_tile_spec**(*stack*, *tile*, *host=None*, *port=None*, *owner=None*, *project=None*, *session=<requests.sessions.Session object>*, *render=None*, ***kwargs*)
renderapi call to get a specific tilespec by tileId note that this will return a tilespec with resolved transform references by accessing the render-parameters version of this tile

renderapi.render.renderaccess() decorated function

Parameters

- **stack** (*str*) – name of render stack to retrieve
- **tile** (*str*) – tileId of tile to retrieve
- **render** (renderapi.render.Render) – render connect object
- **session** (*requests.sessions.Session*) – sessions object to connect with

Returns TileSpec with dereferenced transforms

Return type *TileSpec*

renderapi.tilespec.**get_tile_spec_raw**(*stack*, *tile*, *host=None*, *port=None*, *owner=None*, *project=None*, *session=<requests.sessions.Session object>*, *render=None*, ***kwargs*)
renderapi call to get a specific tilespec by tileId note that this will return a tilespec without resolved transform references

renderapi.render.renderaccess() decorated function

Parameters

- **stack** (*str*) – name of render stack to retrieve
- **tile** (*str*) – tileId of tile to retrieve

- **render** (renderapi.render.Render) – render connect object
- **session** (*requests.sessions.Session*) – sessions object to connect with

Returns TileSpec with referenced transforms intact

Return type *TileSpec*

renderapi.tilespec.**get_tile_spec_renderparameters**(*stack*, *tile*, *host=None*, *port=None*, *owner=None*, *project=None*, *session=<requests.sessions.Session object>*, *render=None*, ***kwargs*)

renderapi call to get the render parameters of a specific tileId

renderapi.render.renderaccess() decorated function TODO provide example of return

Parameters

- **stack** (*str*) – name of render stack to retrieve
- **tile** (*str*) – tileId of tile to retrieve
- **render** (renderapi.render.Render) – render connect object
- **session** (*requests.sessions.Session*) – sessions object to connect with

Returns render-parameters json with the tilespec for that tile and dereferenced transforms

Return type dict

renderapi.tilespec.**get_tile_specs_from_box**(*stack*, *z*, *x*, *y*, *width*, *height*, *scale=1.0*, *host=None*, *port=None*, *owner=None*, *project=None*, *session=<requests.sessions.Session object>*, *render=None*, ***kwargs*)

renderapi call to get all tilespec that exist within a 2d bounding box specified with min x,y values and width, height note that this will return a tilespec with resolved transform references

renderapi.render.renderaccess() decorated function

Parameters

- **stack** (*str*) – name of render stack to retrieve
- **z** (*float*) – z value of bounding box
- **x** (*float*) – minimum x value
- **y** (*float*) – minimum y value
- **width** (*float*) – width of box (in scale=1.0 units)
- **height** (*float*) – height of box (in scale=1.0 units)
- **scale** (*float*) – scale to use when retrieving render parameters (not important)
- **render** (renderapi.render.Render) – render connect object
- **session** (*requests.sessions.Session*) – sessions object to connect with

Returns TileSpec objects with dereferenced tansforms

Return type list of *TileSpec*

renderapi.tilespec.**get_tile_specs_from_minmax_box**(*stack*, *z*, *xmin*, *xmax*, *ymin*, *ymax*, *scale=1.0*, *host=None*, *port=None*, *owner=None*, *project=None*, *session=<requests.sessions.Session object>*, *render=None*, ***kwargs*)

renderapi call to get all tilespec that exist within a 2d bounding box specified with min and max x,y values note that this will return a tilespec with resolved transform references

renderapi.render.renderaccess() decorated function

Parameters

- **stack** (*str*) – name of render stack to retrieve
- **z** (*float*) – z value of bounding box
- **xmin** (*float*) – minimum x value
- **ymin** (*float*) – minimum y value
- **xmax** (*float*) – maximum x value
- **ymax** (*float*) – maximum y value
- **scale** (*float*) – scale to use when retrieving render parameters (not important)
- **render** (renderapi.render.Render) – render connect object
- **session** (*requests.sessions.Session*) – sessions object to connect with

Returns *TileSpec* objects with dereferenced tansforms

Return type list of *TileSpec*

renderapi.tilespec.**get_tile_specs_from_stack**(*stack*, *host=None*, *port=None*, *owner=None*, *project=None*, *session=<requests.sessions.Session object>*, *render=None*, ***kwargs*)

get flat list of tilespecs for stack using i for sl in l for i in sl

renderapi.render.renderaccess() decorated function

Parameters

- **stack** (*str*) – render stack
- **render** (renderapi.render.Render) – render connect object
- **session** (*requests.sessions.Session*) – sessions object to connect with

Returns list of TileSpec objects from that stack

Return type list of *TileSpec*

renderapi.tilespec.**get_tile_specs_from_z**(*stack*, *z*, *host=None*, *port=None*, *owner=None*, *project=None*, *session=<requests.sessions.Session object>*, *render=None*, ***kwargs*)

Get all TileSpecs in a specific z values. Returns referenced transforms.

renderapi.render.renderaccess() decorated function

Parameters

- **stack** (*str*) – render stack
- **z** (*float*) – render z

- **render** (renderapi.render.Render) – render connect object
- **session** (*requests.sessions.Session*) – sessions object to connect with

Returns list of TileSpec objects from that stack at that z

Return type list of *TileSpec*

renderapi.transform module

handling mpicbg transforms in python

Currently only implemented to facilitate Affine and Polynomial2D used in Khaled Khairy's EM aligner workflow

class renderapi.transform.**AffineModel**(*M00=1.0*, *M01=0.0*, *M10=0.0*, *M11=1.0*, *B0=0.0*, *B1=0.0*, *transformId=None*, *json=None*)

Bases: *renderapi.transform.Transform*

Linear 2d Transformation mpicbg classname: mpicbg.trakem2.transform.AffineModel2D implements this simple math x'=M00*x + M01*x + B0 y'=M10*x + M11*y + B1

M00
float – x'+=M00*x

M01
float – x'+=M01*y

M10
float – y'+=M10*x

M11
float – y'+=M11*y

B0
float – x'+=B0

B1
float – y'+=B1

transformId
str, optional – unique transformId for this transform

M
numpy.array – 3x3 numpy array representing 2d Affine with homogeneous coordinates populates with values from M00, M01, M10, M11, B0, B1 with load_M()

className = 'mpicbg.trakem2.transform.AffineModel2D'

concatenate(*model*)

concatenate a model to this model – ported from trakEM2 below:

```
final double a00 = m00 * model.m00 + m01 * model.m10;
final double a01 = m00 * model.m01 + m01 * model.m11;
final double a02 = m00 * model.m02 + m01 * model.m12 + m02;

final double a10 = m10 * model.m00 + m11 * model.m10;
final double a11 = m10 * model.m01 + m11 * model.m11;
final double a12 = m10 * model.m02 + m11 * model.m12 + m12;
```

Parameters **model** (AffineModel) – model to concatenate to this one

Returns model after concatenating model with this model

Return type *AffineModel*

static convert_points_vector_to_array(*points*, *Nd=2*)
method for convertion x,y,K points to x,y vectors

Parameters

- **points** (*numpy.array*) – a Nx3 vector of points after transformation
- **Nd** (*int*) – the number of dimensions to cutoff (should be 2)

Returns **numpy.array**

Return type a Nx2 array of x,y points

static convert_to_point_vector(*points*)
method to help reshape x,y points to x,y,1 vectors

Parameters **points** (*numpy.array*) – a Nx2 array of x,y points

Returns a Nx3 array of x,y,1 points used for transformations

Return type numpy.array

dataString
dataString string for this transform

estimate(*A*, *B*, *return_params=True*, ***kwargs*)
method for setting this transformation with the best fit given the corresponding points A,B

Parameters

- **A** (*numpy.array*) – a Nx2 matrix of source points
- **B** (*numpy.array*) – a Nx2 matrix of destination points
- **return_params** (*boolean*) – whether to return the parameter matrix
- ****kwargs** – keyword arguments to pass to self.fit

Returns a 2x3 matrix of parameters for this matrix, laid out (x,y) x (x,y,offset) (or None if return_params=False)

Return type numpy.array

static fit(*A*, *B*)
function to fit this transform given the corresponding sets of points A & B

Parameters

- **A** (*numpy.array*) – a Nx2 matrix of source points
- **B** (*numpy.array*) – a Nx2 matrix of destination points

Returns a 6x1 matrix with the best fit parameters ordered M00,M01,M10,M11,B0,B1

Return type numpy.array

inverse_tform(*points*)
transform a set of points through the inverse of this transformation

Parameters **points** (*numpy.array*) – a Nx2 array of x,y points

Returns a Nx2 array of x,y points after inverse transformation

Return type numpy.array

invert()
return an inverted version of this transformation

Returns an inverted version of this transformation

Return type *AffineModel*

load_M()
method to take the attribute of self and fill in self.M

rotation
counter-clockwise rotation

scale
tuple of scale for x, y

shear
counter-clockwise shear angle

tform(*points*)
transform a set of points through this transformation

Parameters **points** (*numpy.array*) – a Nx2 array of x,y points

Returns a Nx2 array of x,y points after transformation

Return type numpy.array

translation
tuple of translation in x, y

class renderapi.transform.**InterpolatedTransform**(*a=None*, *b=None*, *lambda_=None*, *json=None*)
Transform spec defined by linear interpolation of two other transform specs

a
Transform or *TransformList* or *InterpolatedTransform* – transform at minimum weight

b
Transform or *TransformList* or *InterpolatedTransform* – transform at maximum weight

lambda_
float – value in interval [0.,1.] which defines evaluation of the linear interpolation between a (at 0) and b (at 1)

from_dict(*d*)
deserialization routine

Parameters **d** (*dict*) – json compatible representation

to_dict()
serialization routine

Returns json compatible representation

Return type dict

class renderapi.transform.**LensCorrection**(*dataString=None*, *json=None*, *transformId=None*)
Bases: *renderapi.transform.NonLinearCoordinateTransform*

a placeholder for the lenscorrection transform, same as NonLinearTransform for now

className = 'lenscorrection.NonLinearTransform'

class renderapi.transform.**NonLinearCoordinateTransform**(*dataString=None*, *json=None*, *transformId=None*)

Bases: *renderapi.transform.Transform*

render-python class that implements the mpicbg.trakem2.transform.NonLinearCoordinateTransform class

Parameters

- **dataString** (*str or None*) – data string of transformation
- **json** (*dict or None*) – json compatible dictionary representation of the transformation

Returns a transform instance

Return type *NonLinearTransform*

className = 'mpicbg.trakem2.transform.NonLinearCoordinateTransform'

dataString

kernelExpand(*src*)

creates an expanded representation of the x,y src points in a polynomial form

Parameters **points** (*numpy.array*) – a Nx2 array of x,y points

Returns a (N x self.length) array of coefficents

Return type numpy.array

tform(*src*)

transform a set of points through this transformation

Parameters **points** (*numpy.array*) – a Nx2 array of x,y points

Returns a Nx2 array of x,y points after transformation

Return type numpy.array

class renderapi.transform.**NonLinearTransform**(*dataString=None*, *json=None*, *transformId=None*)

Bases: *renderapi.transform.NonLinearCoordinateTransform*

className = 'mpicbg.trakem2.transform.nonLinearTransform'

class renderapi.transform.**Polynomial2DTransform**(*dataString=None*, *src=None*, *dst=None*, *order=2*, *force_polynomial=True*, *params=None*, *identity=False*, *json=None*, ***kwargs*)

Bases: *renderapi.transform.Transform*

Polynomial2DTransform implemented as in skimage

params

numpy.array – 2xK matrix of polynomial coefficents up to order K

asorder(*order*)

return polynomial transform appoximation of this transformation with a lower order

Parameters **order** (*int*) – desired order (must have order> current order)

Returns transform of lower order

Return type *Polynomial2DTransform*

Raises ConversionError – if target order < input order

className = 'mpicbg.trakem2.transform.PolynomialTransform2D'

coefficients(*order=None*)
determine number of coefficient terms in transform for a given order

Parameters **order** (*int, optional*) – order of polynomial, defaults to self.order

Returns number of coefficient terms expected in transform

Return type int

dataString
dataString of polynomial

estimate(*src*, *dst*, *order=2*, *test_coords=True*, *max_tries=100*, *return_params=True*, ***kwargs*)
method for setting this transformation with the best fit given the corresponding points src,dst

Parameters

- **src** (*numpy.array*) – a Nx2 matrix of source points
- **dst** (*numpy.array*) – a Nx2 matrix of destination points
- **order** (*int*) – order of polynomial to fit
- **test_coords** (*bool*) – whether to test model after fitting to make sure it is good (see fitgood)
- **max_tries** (*int*) – how many times to attempt to fit the model (see fitgood)
- **return_params** (*bool*) – whether to return the parameter matrix
- ****kwargs** – dictionary of keyword arguments including those that can be passed to fitgood

Returns a (2,(order+1)*(order+2)/2) matrix of parameters for this matrix (or None if return_params=False)

Return type numpy.array

static fit(*src*, *dst*, *order=2*)
function to fit this transform given the corresponding sets of points src & dst polynomial fit

Parameters

- **src** (*numpy.array*) – a Nx2 matrix of source points
- **dst** (*numpy.array*) – a Nx2 matrix of destination points
- **order** (*bool*) – order of polynomial to fit

Returns a [2,(order+1)*(order+2)/2] array with the best fit parameters

Return type numpy.array

static fromAffine(*aff*)
return a polynomial transformation equavalent to a given Affine

Parameters **aff** (AffineModel) – transform to become equivalent to

Returns Order 1 transform equal in effect to aff

Return type *Polynomial2DTransform*

Raises ConversionError – if input model is not AffineModel

is_affine
(boolean) TODO allow default to Affine

order
(int) order of polynomial

tform(*points*)
transform a set of points through this transformation

Parameters **points** (*numpy.array*) – a Nx2 array of x,y points

Returns a Nx2 array of x,y points after transformation

Return type numpy.array

class renderapi.transform.**ReferenceTransform**(*refId=None*, *json=None*)
Transform which is simply a reference to a transform stored elsewhere

refId
str – transformId of the referenced transform

from_dict(*d*)
deserialization routine

Parameters **d** (*dict*) – json compatible representation of this transform

to_dict()
serialization routine

Returns json compatible representation of this transform

Return type dict

class renderapi.transform.**RigidModel**(**args*, ***kwargs*)
Bases: *renderapi.transform.AffineModel*

model for fitting Rigid only transformations (rotation+translation) or (determinate=1, orthonormal eigenvectors) implemented as an *AffineModel*

M00
float – x'+=M00*x

M01
float – x'+=M01*y

M10
float – y'+=M10*x

M11
float – y'+=M11*y

B0
float – x'+=B0

B1
float – y'+=B1

transformId
str, optional – unique transformId for this transform

M
numpy.array – 3x3 numpy array representing 2d Affine with homogeneous coordinates populates with values from M00, M01, M10, M11, B0, B1 with load_M()

className = 'mpicbg.trakem2.transform.RigidModel2D'

estimate(*A*, *B*, *return_params=True*, ***kwargs*)
method for setting this transformation with the best fit given the corresponding points src,dst

Parameters

- **A** (*numpy.array*) – a Nx2 matrix of source points
- **B** (*numpy.array*) – a Nx2 matrix of destination points
- **return_params** (*bool*) – whether to return the parameter matrix

Returns a 2x3 matrix of parameters for this matrix, laid out (x,y) x (x,y,offset) (or None if return_params=False)

Return type numpy.array

static fit(*src*, *dst*, *rigid=True*, ***kwargs*)

function to fit this transform given the corresponding sets of points src & dst Umeyama estimation of similarity transformation

Parameters

- **src** (*numpy.array*) – a Nx2 matrix of source points
- **dst** (*numpy.array*) – a Nx2 matrix of destination points
- **rigid** (*bool*) – whether to constrain this transform to be rigid

Returns a 6x1 matrix with the best fit parameters ordered M00,M01,M10,M11,B0,B1

Return type numpy.array

class renderapi.transform.**SimilarityModel**(**args*, ***kwargs*)

Bases: *renderapi.transform.RigidModel*

class for fitting Similarity transformations (translation+rotation+scaling) or (orthogonal eigen vectors with equal eigenvalues)

implemented as an *AffineModel*

M00

float – x'+=M00*x

M01

float – x'+=M01*y

M10

float – y'+=M10*x

M11

float – y'+=M11*y

B0

float – x'+=B0

B1

float – y'+=B1

transformId

str, optional – unique transformId for this transform

M

numpy.array – 3x3 numpy array representing 2d Affine with homogeneous coordinates populates with values from M00, M01, M10, M11, B0, B1 with load_M()

className = 'mpicbg.trakem2.transform.SimilarityModel2D'

static fit(*src*, *dst*, *rigid=False*, ***kwargs*)
function to fit this transform given the corresponding sets of points src & dst Umeyama estimation of similarity transformation

Parameters

- **src** (*numpy.array*) – a Nx2 matrix of source points
- **dst** (*numpy.array*) – a Nx2 matrix of destination points
- **rigid** (*bool*) – whether to constrain this transform to be rigid

Returns a 6x1 matrix with the best fit parameters ordered M00,M01,M10,M11,B0,B1

Return type numpy.array

class renderapi.transform.**Transform**(*className=None*, *dataString=None*, *transformId=None*, *json=None*)
Bases: object

Base transformation class

className
str – mpicbg java classname of this transform

dataString
str – string reprsentation of this transform as speced by mpicbg java class library

transformId
str, optional – unique Id for this transform (optional)

from_dict(*d*)
deserialization routine

Parameters **d** (*dict*) – json compatible representation of this transform

to_dict()
serialization routine

Returns json compatible representation of this transform

Return type dict

class renderapi.transform.**TransformList**(*tforms=None*, *transformId=None*, *json=None*)
A list of Transforms

tforms
list of *Transform* – transforms to apply

transformId
str, optional – uniqueId for this TransformList

from_dict(*d*)
deserialization function

Parameters **d** (*dict*) – json compatible dictionary representation of this TransformList

to_dict()
serialization function

Returns json & render compatible representation of this TransformList

Return type dict

to_json()
serialization function

Returns string representation of the json & render representation of this TransformList

Return type str

class renderapi.transform.**TranslationModel**(**args*, ***kwargs*)
Bases: *renderapi.transform.AffineModel*

Translation fitting and estimation as an *AffineModel* Linear 2d Transformation mpicbg classname: mpicbg.trakem2.transform.AffineModel2D implements this simple math x'=M00*x + M01*x + B0 y'=M10*x + M11*y + B1

M00
float – x'+=M00*x

M01
float – x'+=M01*y

M10
float – y'+=M10*x

M11
float – y'+=M11*y

B0
float – x'+=B0

B1
float – y'+=B1

transformId
str, optional – unique transformId for this transform

M
numpy.array – 3x3 numpy array representing 2d Affine with homogeneous coordinates populates with values from M00, M01, M10, M11, B0, B1 with load_M()

className = 'mpicbg.trakem2.transform.TranslationModel2D'

estimate(*src*, *dst*, *return_params=True*)
method for setting this transformation with the best fit given the corresponding points src,dst

Parameters

- **src** (*numpy.array*) – a Nx2 matrix of source points
- **dst** (*numpy.array*) – a Nx2 matrix of destination points
- **return_params** (*bool*) – whether to return the parameter matrix

Returns a 2x3 matrix of parameters for this matrix, laid out (x,y) x (x,y,offset) (or None if return_params=False)

Return type numpy.array

static fit(*src*, *dst*)
function to fit Translation transform given the corresponding sets of points src & dst

Parameters

- **src** (*numpy.array*) – a Nx2 matrix of source points
- **dst** (*numpy.array*) – a Nx2 matrix of destination points

Returns a 6x1 matrix with the best fit parameters ordered M00,M01,M10,M11,B0,B1

Return type numpy.array

renderapi.transform.**estimate_dstpts**(*transformlist*, *src=None*)
estimate destination points for list of transforms. Recurses through lists.

Parameters

- **transformlist** (*:obj:list of :obj:Transform*) – transforms that have a tform method implemented
- **src** (*numpy.array*) – a Nx2 array of source points

Returns Nx2 array of destination points

Return type numpy.array

renderapi.transform.**estimate_transformsum**(*transformlist*, *src=None*, *order=2*)
pseudo-composition of transforms in list of transforms using source point transformation and a single estimation. Will produce an Affine Model if all input transforms are Affine, otherwise will produce a Polynomial of specified order

Parameters

- **transformlist** (list of *Transform*) – list of transform objects that implement tform
- **src** (*numpy.array*) – Nx2 array of source points for estimation
- **order** (*int*) – order of Polynomial output if transformlist inputs are non-Affine

Returns best estimate of transformlist in a single transform of this order

Return type *AffineModel* or *Polynomial2DTransform*

renderapi.transform.**load_leaf_json**(*d*)
function to get the proper deserialization function for leaf transforms

Parameters **d** (*dict*) – json compatible representation of leaf transform to deserialize

Returns proper function to deserialize this transformation

Return type func

Raises RenderError – if d['type'] != leaf or is omitted

renderapi.transform.**load_transform_json**(*d*, *default_type='leaf'*)
function to get the proper deserialization function

Parameters

- **d** (*dict*) – json compatible representation of Transform
- **default_type** (*str*) – what kind of transform should we assume this if it is not specified in 'type' ('leaf','list','ref','interpolated')

Returns proper function to deserialize this transformation

Return type func

Raises RenderError – if d['type'] isn't one of ('leaf','list','ref','interpolated')

renderapi.utils module

utilities to make render/java/web/life interfacing easier

class renderapi.utils.**NullHandler**(*level=0*)
Bases: logging.Handler

handler to avoid logging errors for, e.g., missing logger setup

emit(*record*)

class renderapi.utils.**RenderEncoder**(*skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *encoding='utf-8'*, *default=None*)

Bases: json.encoder.JSONEncoder

json Encoder in the following hierarchy for serialization: obj.to_dict() dict(obj) JsonEncoder.default(obj) obj.__dict__

default(*obj*)

default encoder for that handles Render objects

Parameters **obj** (*obj*) – any object that implements to_dict, dict(obj), JsonEncoder.default(obj), or __dict__ (in order)

Returns json encodable datatype

Return type dict or list

renderapi.utils.**defaultifNone**(*val*, *default=None*)

simple default handler

Parameters

- **val** (*obj*) – value to fill in default
- **default** (*obj*) – default value

Returns val if val is not None, else default

Return type obj

renderapi.utils.**fitargspec**(*f*, *oldargs*, *oldkwargs*)

fit function argspec given input args tuple and kwargs dict

Parameters

- **f** (*func*) – function to inspect
- **oldargs** (*tuple*) – arguments passed to func
- **oldkwards** (*dict*) – keyword args passed to func

Returns

- *new_args* – args with values filled in according to f spec
- *new_kwargs* – kwargs with values filled in according to f spec

renderapi.utils.**jbool**(*val*)

return string representing java string values of py booleans

Parameters **val** (*bool*) – boolean to encode

Returns 'true' or 'false'

Return type str

renderapi.utils.**post_json**(*session*, *request_url*, *d*, *params=None*)

POST requests with RenderError handling

Parameters

- **session** (*requests.session.Session*) – requests session
- **request_url** (*str*) – url
- **d** (*dict*) – data payload (will be json dumps-ed)

- **params** (*dict*) – requests parameters

Returns requests.response

Return type server response

Raises RenderError – if cannot post

renderapi.utils.**put_json**(*session*, *request_url*, *d*, *params=None*)
PUT requests with RenderError handling

Parameters

- **session** (*requests.session.Session*) – requests session
- **request_url** (*str*) – url
- **d** (*dict*) – data payload (will be json dumps-ed)
- **params** (*dict*) – requests parameters

Returns server response

Return type requests.response

Raises RenderError – if cannot post

renderapi.utils.**renderdump**(*obj*, **args*, ***kwargs*)
json.dump using the RenderEncoder

Parameters

- **obj** (*obj*) – object to dumps
- ***args** – json.dump args
- ****kwargs** – json.dump kwargs

renderapi.utils.**renderdump_temp**(*obj*, **args*, ***kwargs*)
json.dump into a temporary file renderdump_temp(obj), obj will be dumped through renderdump into a temporary file

Parameters

- **obj** (*obj*) – object to dump
- ***args** – json.dump args
- ****kwargs** – json.dump kwargs

Returns path to location where temporary file was dumped

Return type str

renderapi.utils.**renderdumps**(*obj*, **args*, ***kwargs*)
json.dumps using the RenderEncode

Parameters

- **obj** (*obj*) – object to dumps
- ***args** – json.dumps args
- ****kwargs** – json.dumps kwargs

Returns serialized object

Return type str

renderapi.utils.**stripLogger**(*logger_tostrip*)
remove all handlers from a logger – useful for redefining

Parameters **logger_tostrip** (logging.Logger) – logging logger to strip

Module contents

renderapi.**connect**(*host=None*, *port=None*, *owner=None*, *project=None*, *client_scripts=None*, *client_script=None*, *memGB=None*, *force_http=True*, *validate_client=True*, *web_only=False*, ***kwargs*)
helper function to create a *Render* instance, or RenderClient if sufficent parameters are provided. Will default to using environment variables if not specified in call, and prompt user for any parameters that are not given.

Parameters

- **host** (*str*) – hostname for target render server – will prepend "http://" if host does not begin with 'http' and force_http keyword evaluates True. Can be set by environment variable RENDER_HOST.
- **port** (*str, int, or None*) – port for target render server. Optional as in 'http://hostname{[}:port]'. Can be set by environment variable RENDER_PORT.
- **owner** (*str*) – owner for render-ws. Can be set by environment variable RENDER_OWNER.
- **project** (*str*) – project for render webservice. Can be set by environment variable RENDER_PROJECT.
- **client_scripts** (*str*) – directory path for render-ws-java-client scripts. Can be set by environment variable RENDER_CLIENT_SCRIPTS.
- **client_script** (*str, optional*) – path to a wrapper for java client classes. Used only in RenderClient. Can be set by environment variable RENDER_CLIENT_SCRIPT.
- **memGB** (*str*) – heap size in GB for java client scripts, example for 1 GB: '1G'. Used only in RenderClient. Can be set by environment variable RENDER_CLIENT_HEAP.
- **force_http** (*bool*) – whether to prepend 'http://' to render host if it does not begin with 'http'
- **validate_client** (*bool*) – whether to validate existence of RenderClient run_ws_client.sh script
- **web_only** (*bool*) – whether to check environment variables/prompt user for client_scripts directory if not in arguments

Returns a connect object to simplify specifying what render server to connect to (returns RenderClient if sufficent parameters are passed)

Return type *Render*

Raises ValueError – if empty user input is given for required field

class renderapi.**Render**(*host=None*, *port=None*, *owner=None*, *project=None*, *client_scripts=None*)
Bases: object

Render object to store connection settings for render server. Baseclass that doesn't require client_scripts definition for client side java processing.

See *connect()* for parameter definitions.

DEFAULT_HOST
str – render host to which make_kwargs will default

DEFAULT_PORT
int – render port to which make_kwargs will default

DEFAULT_OWNER
str – render owner to which make_kwargs will default

DEFAULT_PROJECT
str – render project to which make_kwargs will default

DEFAULT_CLIENT_SCRIPTS
str – render client scripts path to which make_kwargs will default

DEFAULT_KWARGS
"kwargs to which the render object falls back. Depends on – self.DEFAULT_HOST, self.DEFAULT_OWNER, self.DEFAULT_PORT, self.DEFAULT_PROJECT, self.DEFAULT_CLIENT_SCRIPTS

Returns default keyword arguments

Return type dict

make_kwargs(*host=None*, *port=None*, *owner=None*, *project=None*, *client_scripts=None*, ***kwargs*)
make kwargs using this render object's defaults and any designated kwargs passed in

Parameters

- **host** (*str or None*) – render webservice host
- **port** (*int or None*) – render webservice port
- **owner** (*str or None*) – render webservice owner
- **project** (*str or None*) – render webservice project
- **client_scripts** (*str or None*) – render java client script location
- ****kwargs** – all other keyword arguments passed through

Returns keyword arguments with missing host,port,owner,project,client_scripts filled in with defaults

Return type dict

run(*f*, **args*, ***kwargs*)

run function from object technically shorter than adding render=Render to kwargs

Parameters

- **f** (*func*) – renderapi function you want to call
- ***args** – args passed to that function
- ****kwargs** – kwargs passed to that function

Returns function with this *Render* instance in keyword arguments as render=

Return type func

Examples

```
>>> render = Render('server',8080)
>>> metadata = render.run(renderapi.render.get_stack_metadata_by_owner,
↪'myowner')
```

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

r

Index

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

Z