qipipe

Release

Jul 27, 2017

Contents

CHAPTER 1

Synopsis

qipipe processes DICOM study images.

API http://qipipe.readthedocs.org/en/latest/api/index.html

Git https://github.com/ohsu-qin/qipipe

CHAPTER 2

Feature List

1. Recognizes new study images.
2. Stages images for submission to The Cancer Imaging Archive (TCIA) QIN collection.
3. Masks images to subtract extraneous image content.
4. Corrects motion artifacts.
5. Performs pharmokinetic modeling.
6. Imports the input scans and processing results into the XNAT image database.

CHAPTER 3

Installation

The following instructions assume that you start in your home directory. We recommend the Anaconda environment for scientific packages and pip for the remaining Python packages. Install `qipipe` using the following procedure:

1. Install and activate a qixnat Anaconda environment as described in the qixnat Installation Instructions.
2. Install the `qipipe` dependencies hosted by Anaconda:

```
wget -q --no-check-certificate -O \
  - https://www.github.com/ohsu-qin/qipipe/raw/master/requirements_conda.txt \
  | xargs conda install --yes
```

3. Download the `qipipe` constraints file:

```
wget -q --no-check-certificate -O \
  - https://www.github.com/ohsu-qin/qipipe/raw/master/constraints.txt \
  > /tmp/constraints.txt
```

4. Install the `qipipe` package using pip:

```
pip install qipipe --constraint /tmp/constraints.txt && rm /tmp/constraints.txt
```

5. For ANTS registration, build the ants package from source using the ANTS Compile Instructions:

```
pushd ~/workspace
git clone git://github.com/stnava/ANTs.git
mkdir $HOME/ants
cd $HOME/ants
ccmake ../workspace/ANTs
cmake
#=> Enter "c"
#=> Enter "g"
#=> Exit back to the terminal
make -j 4
popd
```

Then, prepend ANTS to your shell login script. E.g., for Linux or Mac OS X, open an editor on `$HOME/.bashrc` or `$HOME/.bash_profile` and add the following lines:

```
# Prepend ANTS to the path.
ANTS_HOME=$HOME/ants
export PATH=$ANTS_HOME/bin
```

and refresh your environment:

```
. $HOME/.bash_profile
```

CHAPTER 4

Usage

Run the following command for the pipeline options:

```
qipipe --help
```

CHAPTER 5

Development

Download

Download the source by cloning the source repository, e.g.:

```
cd ~/workspace
git clone https://github.com/ohsu-qin/qipipe.git
cd qipipe
```

Installing from a local qipipe clone requires the constraints option:

```
pip install -c constraints.txt -e .
```

Testing

Testing is performed with the nose package, which can be installed separately as follows:

```
conda install nose
```

The unit tests are then run as follows:

```
nosetests test/unit/
```

Documentation

API documentation is built automatically by ReadTheDocs when the project is pushed to GitHub. The modules documented are defined in `doc/api`. If you add a new Python file to a package directory *pkg*, then include it in the `doc/api`*pkg* ‘‘.rst‘‘ file.

Documentation can be generated locally as follows:

- Install Sphinx and `docutils`, if necessary:

```
conda install Sphinx docutils
```

- Run the following in the `doc` subdirectory:

```
make html
```

Read The Docs builds occur in a limited context that sometimes fails on dependencies, e.g. when an install a requires C extension. In that case, the project has a `requirements_read_the_doc.txt` that eliminates the problematic dependency and specify the requirements file in the Read The Docs project Advance Settings.

Release

Building a release requires a PyPI account and the twine package, which can be installed separately as follows:

```
pip install twine
```

The release is then published as follows:

- Confirm that the unit tests run without failure.
- Add a one-line summary release theme to `History.rst`.
- Update the `__init__.py` version.
- Commit these changes:

```
git add --message 'Bump version.' History.rst qipipe/__init__.py
```

- Merge changes on a branch to the current master branch, e.g.:

```
git checkout master
git pull
git merge --no-ff <branch>
```

- Reconfirm that the unit tests run without failure.
- Build the release:

```
python setup.py sdist
```

- Upload the release:

```
twine upload dist/qipipe-<version>.tar.gz
```

- Tag the release:

```
git tag v<n.n.n>
```

- Update the remote master branch:

```
git push
git push --tags
```

API Documentation

helpers

pipeline helpers

The helpers module includes convenience utilities.

bolus_arrival

exception qipipe.helpers.bolus_arrival.**BolusArrivalError**
Bases: exceptions.Exception

Error calculating the bolus arrival.

qipipe.helpers.bolus_arrival.**bolus_arrival_index**(*time_series*)
Determines the DCE bolus arrival time point index. The bolus arrival is the first occurence of a difference in average signal larger than double the difference from first two points.

Parameters **time_series** – the 4D NIfTI scan image file path

Returns the bolus arrival time point index

Raises ***BolusArrivalError*** – if the bolus arrival could not be determined

colors

qipipe.helpers.colors.**colorize**(*lut_file*, **inputs*, ***opts*)
Transforms each input voxel value to a color lookup table reference.

The input voxel values are uniformly partitioned for the given colormap LUT. For example, if the voxel values range from 0.0 to 3.0, then the a voxel value of 0 is transformed to the first LUT color, 3.0 is transformed to the last LUT color, and the intermediate values are transformed to intermediate colors.

The voxel -> reference output file name appends _color to the input file basename and preserves the input file extensions, e.g. the input file k_trans_map.nii.gz is transformed to k_trans_map_color.nii.gz in the output directory.

Parameters

- **lut_file** – the color lookup table file location
- **inputs** – the image files to transform
- **opts** – the following keyword arguments:

Option dest the destination directory (default current working directory)

Option threshold the threshold in the range 0 to nvalues (default 0)

qipipe.helpers.colors.**create_lookup_table**(*ncolors*, *colormap='jet'*, *out_file=None*)
Generates a colormap lookup table with the given number of colors.

Parameters

- **ncolors** – the number of colors to generate
- **colormap** – the matplotlib colormap name

- **out_file** – the output file path (default is the colormap name followed by `_colors.txt` in the current directory)

qipipe.helpers.colors.**label_map_basename**(*location*)

Parameters **location** – the input file path

Returns the corresponding color file name

command

Command qipipe command options.

qipipe.helpers.command.**configure_log**(***opts*)
Configure the logger for the qi* modules.

constants

qipipe.helpers.constants.**CONF_DIR = '/home/docs/checkouts/readthedocs.org/user_builds/qipipe/checkouts/stable/qi**
The common configuration directory.

qipipe.helpers.constants.**MASK_FILE = 'mask.nii.gz'**
The XNAT mask file name with extension.

qipipe.helpers.constants.**MASK_RESOURCE = 'mask'**
The XNAT mask resource name.

qipipe.helpers.constants.**SCAN_TS_BASE = 'scan_ts'**
The XNAT scan time series file base name without extension.

qipipe.helpers.constants.**SCAN_TS_FILE = 'scan_ts.nii.gz'**
The XNAT scan time series file name with extension.

qipipe.helpers.constants.**SESSION_FMT = 'Session%02d'**
The XNAT session name format with argument session number.

qipipe.helpers.constants.**SUBJECT_FMT = '%s%03d'**
The XNAT subject name format with argument collection and subject number.

qipipe.helpers.constants.**VOLUME_DIR_PAT = <_sre.SRE_Pattern object>**
The volume directory name pattern. The directory name is `volume``*number*, where *number* is the zero-padded, one-based volume number matched as the ``volume_number` group, as determined by the *qipipe.pipeline.staging.volume_format()* function.

qipipe.helpers.constants.**VOLUME_FILE_PAT = <_sre.SRE_Pattern object>**
The volume file name pattern. The image file name is the *VOLUME_DIR_PAT* pattern followed by the extension `.nii.gz`.

distributable

qipipe.helpers.distributable.**DISTRIBUTABLE = False**
Flag indicating whether the workflow can be distributed over a cluster. This flag is True if `qsub` is in the execution path, False otherwise.

image

qipipe.helpers.image.**discretize**(*in_file*, *out_file*, *nvalues*, *start=0*, *threshold=None*, *normalizer=<function normalize>*)

Transforms the given input image file to an integer range with the given number of values. The range starts at the given start value. The input values are uniformly mapped into the output range. For example, if the input values range from 0.0 to 3.0 *nvalues* is 101, and the start is 0, then an input value of 0 is transformed to 0, 3.0 is transformed to 100, and the intermediate input values are proportionately transformed to the output range.

If a threshold is specified, then every input value which maps to an output value less than (*threshold* * *nvalues*) - *start* is transformed to the output start value. For example, if the input values range from 0.0 to 3.0, then:

```
discretize(in_file, out_file, 1001, threshold=0.5)
```

transforms input values as follows:

- If the input value maps to the first half of the output range, then the output value is 0.
- Otherwise, the input value *v* maps to the output value (*v* * 1000) / 3.

Parameters

- **in_file** – the input file path
- **out_file** – the output file path
- **nvalues** – the number of output entries
- **start** – the starting output value (default 0)
- **threshold** – the threshold in the range start to nvalues (default start)
- **normalize** – an optional function to normalize the input value (default *normalize()*)

Raises **IndexError** – if the threshold is not in the color range

qipipe.helpers.image.**normalize**(*value*, *vmin*, *vspan*)

Maps the given input value to [0, 1].

Parameters

- **value** – the input value
- **vmin** – the minimum input range value
- **vspan** – the value range span (maxium - minimum)

Returns (*in_val* - *vmin*) / *vspan*

logging

qipipe.helpers.logging.**NIPYPE_LOG_DIR_ENV_VAR = 'NIPYPE_LOG_DIR'**

The environment variable used by Nipype to set the log directory.

qipipe.helpers.logging.**configure**(***opts*)

Configures the logger as follows:

- If there is a *log* option, then the logger is a conventional `qiutil.logging` logger which writes to the given log file.
- Otherwise, the logger delegates to a mock logger that writes to stdout.

Note: In a cluster environment, Nipype kills the dispatched job log config. Logging falls back to the default. For this reason, the default mock logger level is `DEBUG` rather than `INFO`. The dispatched node's log is the stdout captured in the file *work*`/batch/`*node_name*`.o`*node_id*, where *work* the execution work directory.

Parameters **opts** – the `qiutil.command.configure_log` options

Returns the logger factory

`qipipe.helpers.logging.`**logger**(*name*)

This method overrides `qiutil.logging.logger` to work around the following Nipype bug:

- Nipype stomps on any other application's logging. The work-around is to mock a "logger" that writes to stdout.

Parameters **name** – the caller's context `__name__`

Returns the logger facade

metadata

`qipipe.helpers.metadata.`**EXCLUDED_OPTS = set(['plugin_args', 'run_without_submitting'])**

The config options to exclude in the profile.

exception `qipipe.helpers.metadata.`**MetadataError**

Bases: `exceptions.Exception`

Metadata parsing error.

`qipipe.helpers.metadata.`**create_profile**(*configuration*, *sections*, *dest*, ***opts*)

Creates a metadata profile from the given configuration. The configuration input is a {section: {option: value}} dictionary. The target profile is a Python configuration file which includes the given sections. The section content is determined by the input configuration and the keyword arguments. The keyword item overrides a matching input configuration item. The resulting profile is written to a new file.

Parameters

- **configuration** – the configuration dictionary
- **sections** – the target profile sections
- **dest** – the target profile file location
- **opts** – additional {section: {option: value}} items

Returns the target file location

roi

ROI utility functions.

class `qipipe.helpers.roi.`**Extent**(*points*, *scale=None*)

Bases: `object`

The line segments which span the largest volume or area between a set of points.

Parameters

- **points** – the points array

- **scale** (*tuple*) – the anatomical dimension scaling factors (default unit scale)

__init__ (*points*, *scale=None*)

Parameters

- **points** – the points array
- **scale** (*tuple*) – the anatomical dimension scaling factors (default unit scale)

boundary = None
The convex hull boundary in image space.

bounding_box()
Returns the (least, most) points of a rectangle circumscribing the extent.

Returns the (least, most) rectangle points

Return type tuple

scale = None
The anatomical dimension scaling factors (default unit scale).

segments = None
The orthogonal extent segments.

show()
Displays the ROI boundary points and extent segments.

class qipipe.helpers.roi.**ExtentSegmentFactory**(*points*)
Bases: object

A utility factory class that computes the extent line segments from a set of convex hull vertex points.

Parameters **points** – the convex hull vertex points

__init__ (*points*)

Parameters **points** – the convex hull vertex points

create()
Returns the orthogonal segments end point indexes as the tuple (longest, widest, deepest), where each of the tuple elements is a (from, to) segment end point pair of indexes into the *points*, e.g.:

```
>>> points.shape
(128, 3)
>>> factory = ExtentSegmentFactory(points)
>>> segment_indexes = factory.create()
>>> segment_indexes
((34, 12), (122, 14), (48, 111))
>>> segments = points[segments]
>>> np.all(np.equal(segments[0][0], points[34]))
True
>>> np.all(np.equal(segments[0][1], points[12]))
True
```

The bounding segments procedure is as follows:

•Find the length segment *(r1, r2)* which maximizes the Cartesian distance between points.

•Find the point *r3* furthest from *r1* and *r2*.

•Compute the point *o* orthogonal to *r3* on the segment *(r1, r2)*.

•The width segment is then *(r3, r4)*, where the point *r4* minimizes the angle between the segments *(r3, p)* and *(r3, o)* for all points p.

- Iterate on a generalization of the above algorithm to find the depth segment *(r5, r6)*, where:
  - ***r5* maximizes the distance to the plane formed by the** length segment *(r1, r2)* and the width segment *(r3, r4)*.
  - ***r6* is the point which is most orthogonal to the length** and width segments.

Returns the orthogonal segment end point index tuples

Return type list

distances = None
The N x N point distance array, where N is the number of points and `self.distances[i][j]` is the distance from `self.points[i]` to `self.points[j]`.

points = None
The ndarray of boundary points.

class qipipe.helpers.roi.**ROI**(*points*, *scale=None*)
Bases: object

Summary information for a 3D ROI mask.

Parameters

- **points** – the ROI mask points
- **scale** (*tuple*) – the (x, y, z) scaling factors

__init__(*points*, *scale=None*)

Parameters

- **points** – the ROI mask points
- **scale** (*tuple*) – the (x, y, z) scaling factors

extent = None
The 3D *Extent*.

maximal_slice_index()

Returns the zero-based slice index with maximal planar extent

slices = None
The {z: *Extent*} dictionary for the 2D (x, y) points grouped by z value.

qipipe.helpers.roi.**load**(*location*, *scale=None*)
Loads a ROI mask file.

Parameters

- **location** – the ROI mask file location
- **scale** (*tuple*) – the (x, y, z) scaling factors

Returns the *ROI* encapsulation

Return type *ROI*

qipipe.helpers.roi.**reorder_bolero_mask**(*in_file*, *out_file=None*)
Since the OHSU Bolero ROI is drawn over DICOM slice displays, the converted NIfTI file x and y must be transposed and flipped to match the time series. The mask data shape is assumed to be [x, y, slice].

Parameters

- **in_file** – the input Bolero mask file path

- **out_file** – the optional output file path

Returns the reordered mask ndarray data

interfaces

interfaces Package

The interfaces module includes the custom Nipype interface classes. As a convenience, this interfaces module imports all of the non-proprietary interface classes. The proprietary interface class *qipipe.interfaces.fastfit.Fastfit* must be imported separately from the *qipipe.interfaces.fastfit* module, e.g.:

```
from qipipe.interfaces.fastfit import Fastfit
```

Importing fastfit in an environment that does not provide the fastfit application will raise an ImportError.

compress

convert_bolero_mask

OHSU - This module wraps the proprietary OHSU AIRC bolero_mask_conv utility. bolero_mask_conv converts a proprietary OHSU format mask file into a NIfTI mask file.

class qipipe.interfaces.convert_bolero_mask.**ConvertBoleroMask**(***inputs*)
Bases: nipype.interfaces.base.CommandLine

Interface to the proprietary OHSU AIRC bolero_mask_conv utility.

__init__(***inputs*)

copy

class qipipe.interfaces.copy.**Copy**(*from_file=None*, ***inputs*)
Bases: nipype.interfaces.base.BaseInterface

The Copy interface copies a file to a destination directory.

dce_to_r1

OHSU - This module wraps the proprietary OHSU AIRC dce_to_r1 utility.

class qipipe.interfaces.dce_to_r1.**DceToR1**(***inputs*)
Bases: nipype.interfaces.base.CommandLine

Convert a T1-weighted DCE time series of signal intensities to a series of R1 values.

__init__(***inputs*)

qipipe.interfaces.dce_to_r1.**MASK_FILE_NAME = 'valid_mask.nii.gz'**
The dce_to_r1 output mask file base name.

qipipe.interfaces.dce_to_r1.**OUTPUT_FILE_NAME = 'r1_series.nii.gz'**
The dce_to_r1 output R1 file base name.

fastfit

OHSU - This module wraps the proprietary OHSU AIRC `fastfit` software. `fastfit` optimizes the input pharmacokinetic model.

Note: this interface is adapted from the OHSU AIRC cluster file `/usr/global/scripts/fastfit_iface.py`.

---

class qipipe.interfaces.fastfit.**Fastfit**(***inputs*)
Bases: nipype.interfaces.base.CommandLine

Interface to the `fastfit` software package.

__init__(***inputs*)

output_spec
alias of DynamicTraitedSpec

fix_dicom

class qipipe.interfaces.fix_dicom.**FixDicom**(*from_file=None*, ***inputs*)
Bases: nipype.interfaces.base.BaseInterface

The FixDicom interface wraps the *qipipe.staging.fix_dicom.fix_dicom_headers()* function.

group_dicom

interface_error

lookup

class qipipe.interfaces.lookup.**Lookup**(*from_file=None*, ***inputs*)
Bases: nipype.interfaces.io.IOBase

The Lookup Interface wraps a dictionary look-up.

Example:

```
>>> from qipipe.interfaces import Lookup
>>> lookup = Lookup(key='a', dictionary=dict(a=1, b=2))
>>> result = lookup.run()
>>> result.outputs.value
1
```

map_ctp

Maps the DICOM Patient IDs to the CTP Patient IDs.

class qipipe.interfaces.map_ctp.**MapCTP**(*from_file=None*, ***inputs*)
Bases: nipype.interfaces.base.BaseInterface

The MapCTP interface wraps the `qipipe.interfaces.map_ctp.map_ctp()` method.

move

class qipipe.interfaces.move.**Move**(*from_file=None*, ***inputs*)
Bases: nipype.interfaces.base.BaseInterface

The Move interface moves a file to a destination using shutil.move. Unlike shutil.move, the *dest* parent directory is created if it does not yet exist (like mkdir -p).

mri_volcluster

class qipipe.interfaces.mri_volcluster.**MriVolCluster**(*command=None*, ***inputs*)
Bases: nipype.interfaces.base.CommandLine

MriVolCluster encapsulates the FSL mri_volcluster command.

preview

class qipipe.interfaces.preview.**Preview**(*from_file=None*, ***inputs*)
Bases: nipype.interfaces.base.BaseInterface

Preview creates a JPEG image from an input DICOM image.

reorder_bolero_mask

This module reorders the OHSU AIRC bolero_mask_conv result to conform with the time series x and y order.

class qipipe.interfaces.reorder_bolero_mask.**ReorderBoleroMask**(*from_file=None*, ***inputs*)
Bases: nipype.interfaces.base.BaseInterface

Interface to the ROI reordering utility.

sticky_identity

class qipipe.interfaces.sticky_identity.**StickyIdentityInterface**(*fields=None*, *mandatory_inputs=True*, ***inputs*)
Bases: nipype.interfaces.io.IOBase

The StickyIdentityInterface interface class is a copy of the Nipype IdentityInterface. Since Nipype overzealously elides IdentityInterface nodes from the execution graph, IdentityInterface cannot be used to capture workflow output nor to constrain execution order, as in the examples below. StickyIdentityInterface is an IdentityInterface look-alike that preserves the node connection in the execution graph.

Example:

```
>> from qipipe.interfaces import StickyIdentityInterface
>> gate = Node(StickyIdentityInterface(fields=['a', 'b']))
>> workflow.connect(upstream1, 'a', gate, 'a')
>> workflow.connect(upstream2, 'b', gate, 'b')
>> workflow.connect(gate, 'a', downstream, 'a')
```

In this example, the `gate` node starts after both `upstream1` and `upstream2` finish. Consequently, the `downstream` node starts only after both `upstream1` and `upstream2` finish. This execution precedence constraint does not hold if gate were an IdentityInterface.

Example:

```
>> from qipipe.interfaces import StickyIdentityInterface
>> output_spec = Node(StickyIdentityInterface(fields=['a']))
>> workflow.connect(upstream, 'a', output_spec, 'a')
>> upstream.inputs.a = 1
>> # The magic incantation to get a Nipype workflow output.
>> wf_res = workflow.run()
>> output_res = next(n for n in wf_res.nodes()
...                   if n.name == 'output_spec')
>> output_res.inputs.get()['a']
1
>> # But, oddly:
>> output_res.outputs.get()['a']  # bad!
<undefined>
```

Note: a better solution is to set a *preserve* flag on IdentityInterface. If this solution is implemented by Nipype, then this *StickyIdentityInterface* class will be deprecated.

__init__ (*fields=None*, *mandatory_inputs=True*, ***inputs*)

input_spec
alias of DynamicTraitedSpec

output_spec
alias of DynamicTraitedSpec

touch

class qipipe.interfaces.touch.**Touch** (*from_file=None*, ***inputs*)
Bases: nipype.interfaces.base.BaseInterface

The Touch interface emulates the Unix `touch` command. This interface is useful for stubbing out processing nodes during workflow development.

uncompress

unpack

class qipipe.interfaces.unpack.**Unpack** (*input_name*, *output_names*, *mandatory_inputs=True*, ***inputs*)
Bases: nipype.interfaces.io.IOBase

The Unpack Interface converts a list input field to one output field per list item.

Example:

```
>>> from qipipe.interfaces.unpack import Unpack
>>> unpack = Unpack(input_name='list', output_names=['a', 'b'], list=[1, 2])
>>> result = unpack.run()
>>> result.outputs.a
```

```
1
>>> result.outputs.b
2
```

Parameters

- **input_name** – the input list field name
- **output_names** – the output field names
- **mandatory_inputs** – a flag indicating whether every input field is required
- **inputs** – the input field name => value bindings

__init__(*input_name*, *output_names*, *mandatory_inputs=True*, ***inputs*)

Parameters

- **input_name** – the input list field name
- **output_names** – the output field names
- **mandatory_inputs** – a flag indicating whether every input field is required
- **inputs** – the input field name => value bindings

input_spec
alias of DynamicTraitedSpec

output_spec
alias of DynamicTraitedSpec

update_qiprofile

class qipipe.interfaces.update_qiprofile.**UpdateQIProfile**(*from_file=None*, ***inputs*)
Bases: nipype.interfaces.base.BaseInterface

The UpdateQIProfile Nipype interface updates the Imaging Profile database.

xnat_copy

class qipipe.interfaces.xnat_copy.**XNATCopy**(*command=None*, ***inputs*)
Bases: nipype.interfaces.base.CommandLine

The XNATCopy Nipype interface wraps the cpxnat command.

input_spec
alias of *XNATCopyInputSpec*

class qipipe.interfaces.xnat_copy.**XNATCopyInputSpec**(***kwargs*)
Bases: nipype.interfaces.base.CommandLineInputSpec

The input spec with arguments in the following order: * options * the input files, for an upload * the XNAT object path * the destination directory, for a download

Initialize handlers and inputs

xnat_download

qipipe.interfaces.xnat_download.**CONTAINER_OPTS = ['container_type', 'scan', 'reconstruction', 'assessor']**
The download input container options.

class qipipe.interfaces.xnat_download.**XNATDownload**(*from_file=None*, ***inputs*)
Bases: nipype.interfaces.base.BaseInterface

The XNATDownload Nipype interface wraps the qixnat.facade.XNAT.download() method.

Note: only one XNAT operation can run at a time.

Examples:

```
>>> # Download the scan NIfTI files.
>>> from qipipe.interfaces import XNATDownload
>>> XNATDownload(project='QIN', subject='Breast003',
...     session='Session02', scan=1, resource='NIFTI',
...     dest='data').run()
```

```
>>> # Download the scan DICOM files.
>>> from qipipe.interfaces import XNATDownload
>>> XNATDownload(project='QIN', subject='Breast003',
...     session='Session02', scan=1, resource='DICOM',
...     dest='data').run()
```

```
>>> # Download the registration reg_H3pIz4s images.
>>> from qipipe.interfaces import XNATDownload
>>> XNATDownload(project='QIN', subject='Breast003',
...     session='Session02', resource='reg_H3pIz4',
...     dest='data').run()
```

xnat_find

class qipipe.interfaces.xnat_find.**XNATFind**(***inputs*)
Bases: nipype.interfaces.base.BaseInterface

The XNATFind Nipype interface wraps the qixnat.facade.XNAT find_one and find_or_create methods.

Note: concurrent XNAT operations can fail. See the qipipe.pipeline.staging.StagingWorkflow note.

__init__(***inputs*)

xnat_upload

class qipipe.interfaces.xnat_upload.**XNATUpload**(*from_file=None*, ***inputs*)
Bases: nipype.interfaces.base.BaseInterface

The XNATUpload Nipype interface wraps the qixnat.facade.XNAT.upload() method.

pipeline

pipeline Package

This pipeline module includes the following workflows:

- *qipipe.pipeline.qipipeline*: the soup-to-nuts pipeline to stage, mask, register and model new images
- *qipipe.pipeline.staging*: executes the staging workflow to detect new images, group them by volume, import them into XNAT and prepare them for TCIA import
- *qipipe.pipeline.mask*: creates a mask to subtract extraneous tissue from the input images
- *qipipe.pipeline.registration*: masks the target tissue and corrects motion artifacts
- *qipipe.pipeline.modeling*: performs pharmokinetic modeling

mask

class qipipe.pipeline.mask.**MaskWorkflow**(***opts*)
Bases: *qipipe.pipeline.workflow_base.WorkflowBase*

The MaskWorkflow class builds and executes the mask workflow.

The workflow creates a mask to subtract extraneous tissue for a given input session 4D NIfTI time series. The new mask is uploaded to XNAT as a session resource named mask.

The mask workflow input is the *input_spec* node consisting of the following input fields:

•subject: the XNAT subject name

•session: the XNAT session name

•scan: the XNAT scan number

•time_series: the 4D NIfTI series image file

The mask workflow output is the *output_spec* node consisting of the following output field:

•*mask*: the mask file

The optional workflow configuration file can contain the following sections:

•**fsl.MriVolCluster: the** *qipipe.interfaces.mri_volcluster.MriVolCluster* interface options

If the optional configuration file is specified, then the workflow settings in that file override the default settings.

Parameters **opts** – the *qipipe.pipeline.workflow_base.WorkflowBase* initializer keyword arguments, as well as the following keyword arguments:

Option crop_posterior crop posterior to the center of gravity, e.g. for a breast tumor

__init__(***opts*)
If the optional configuration file is specified, then the workflow settings in that file override the default settings.

Parameters **opts** – the *qipipe.pipeline.workflow_base.WorkflowBase* initializer keyword arguments, as well as the following keyword arguments:

Option crop_posterior crop posterior to the center of gravity, e.g. for a breast tumor

run(*subject*, *session*, *scan*, *time_series*)
Runs the mask workflow on the scan NIfTI files for the given time series.

Parameters

- **subject** – the input subject
- **session** – the input session
- **scan** – the input scan number
- **time_series** – the input 3D NIfTI time series to mask

Returns the mask file location

workflow = None
The mask creation workflow.

qipipe.pipeline.mask.**run**(*subject*, *session*, *scan*, *time_series*, ***opts*)
Creates a *qipipe.pipeline.mask.MaskWorkflow* and runs it on the given inputs.

Parameters

- **subject** – the input subject
- **session** – the input session
- **scan** – the input scan number
- **time_series** – the input 4D NIfTI time series to mask
- **opts** – additional *MaskWorkflow* initialization parameters

Returns the mask file location

modeling

qipipe.pipeline.modeling.**FASTFIT_CONF_PROPS = ['model_name', 'optimization_params', 'optional_outs']**
The Fastfit configuration property names.

qipipe.pipeline.modeling.**FASTFIT_PARAMS_FILE = 'params.csv'**
The Fastfit parameters CSV file name.

qipipe.pipeline.modeling.**FXL_MODEL_PREFIX = 'ext_tofts.'**
The Fastfit Standard TOFTS model prefix.

qipipe.pipeline.modeling.**MODELING_CONF_FILE = 'modeling.cfg'**
The modeling workflow configuration.

qipipe.pipeline.modeling.**MODELING_PREFIX = 'pk_'**
The modeling XNAT object label prefix.

class qipipe.pipeline.modeling.**ModelingWorkflow**(***opts*)
Bases: *qipipe.pipeline.workflow_base.WorkflowBase*

The ModelingWorkflow builds and executes the Nipype pharmacokinetic mapping workflow.

The workflow calculates the modeling parameters for an input 4D time series NIfTI image file as follows:

•Compute the $R1_0$ value, if it is not given in the options

•Convert the DCE time series to a R1 map series

•Determine the AIF and R1 fit parameters from the time series

•Optimize the OHSU pharmacokinetic model

•Upload the modeling result to XNAT

The modeling workflow input is the *input_spec* node consisting of the following input fields:

- *subject*: the subject name
- *session*: the session name
- *mask*: the mask to apply to the images
- *time_series*: the 4D time series NIfTI file to model
- *bolus_arrival_index*: the bolus uptake volume index
- the R1 modeling parameters described below

If an input field is defined in the configuration file `R1` section, then the input field is set to that value.

If the $R1_0$ option is not set, then it is computed from the proton density weighted scans and DCE series baseline image.

The outputs are collected in the *output_spec* node for the FXL (Tofts standard) model and the FXR (shutter speed) model with the following fields:

- *r1_series*: the R1 series files
- *pk_params*: the AIF and R1 parameter CSV file
- ***fxr_k_trans*, *fxl_k_trans*: the K^{trans} vascular permeability** transfer constant
- *delta_k_trans*: the FXR-FXL K^{trans} difference
- ***fxr_v_e*, *fxl_v_e*: the v_e extravascular extracellular volume** fraction
- *fxr_tau_i*: the τ_i intracellular H_2O mean lifetime
- *fxr_chi_sq*, *fxl_chi_sq*: the χ^2 intensity goodness of fit

In addition, if $R1_0$ is computed, then the output includes the following fields:

- *pdw_file*: the proton density weighted image
- *dce_baseline*: the DCE series baseline image
- *r1_0*: the computed $R1_0$ value

This workflow is adapted from the AIRC DCE implementation.

Note: This workflow uses proprietary OHSU AIRC software, notably the OHSU implementation of the shutter speed model.

The modeling parameters can be defined in either the options or the configuration as follows:

- The parameters can be defined in the configuration `R1` section.
- The keyword arguments take precedence over the configuration settings.
- The *r1_0_val* takes precedence over the R1_0 computation fields *pd_dir* and *max_r1_0*. If *r1_0_val* is set in the input options, then *pd_dir* and *max_r1_0* are not included from the result.
- If *pd_dir* and *max_r1_0* are set in the input options and *r1_0_val* is not set in the input options, then a *r1_0_val* configuration setting is ignored.
- The *base_end* defaults to 1 if it is not set in either the input options or the configuration.

Parameters

- **opts** – the *qipipe.pipeline.workflow_base.WorkflowBase* initializer keyword arguments, as well as the following keyword arguments:
- **r1_0_val** – the optional fixed $R1_0$ value
- **max_r1_0** – the maximum computed $R1_0$ value, if the fixed $R1_0$ option is not set
- **pd_dir** – the proton density files parent directory, if the fixed $R1_0$ option is not set
- **base_end** – the number of volumes to merge into a R1 series baseline image (default is 1)

__init__ (***opts*)

The modeling parameters can be defined in either the options or the configuration as follows:

•The parameters can be defined in the configuration `R1` section.

•The keyword arguments take precedence over the configuration settings.

•The *r1_0_val* takes precedence over the R1_0 computation fields *pd_dir* and *max_r1_0*. If *r1_0_val* is set in the input options, then *pd_dir* and *max_r1_0* are not included from the result.

•If *pd_dir* and *max_r1_0* are set in the input options and *r1_0_val* is not set in the input options, then a *r1_0_val* configuration setting is ignored.

•The *base_end* defaults to 1 if it is not set in either the input options or the configuration.

Parameters

- **opts** – the *qipipe.pipeline.workflow_base.WorkflowBase* initializer keyword arguments, as well as the following keyword arguments:
- **r1_0_val** – the optional fixed $R1_0$ value
- **max_r1_0** – the maximum computed $R1_0$ value, if the fixed $R1_0$ option is not set
- **pd_dir** – the proton density files parent directory, if the fixed $R1_0$ option is not set
- **base_end** – the number of volumes to merge into a R1 series baseline image (default is 1)

resource = None

The XNAT resource name for all executions of this *qipipe.pipeline.modeling.ModelingWorkflow* instance. The name is unique, which permits more than one model to be stored for each input volume without a name conflict.

run (*subject*, *session*, *scan*, *time_series*, ***opts*)

Executes the modeling workflow described in *qipipe.pipeline.modeling.ModelingWorkflow* on the given input time series resource. The time series can be the merged scan NIFTI files or merged registration files.

This run method connects the given inputs to the modeling workflow inputs. The execution workflow is then executed, resulting in a new uploaded XNAT resource.

Parameters

- **subject** – the subject name
- **session** – the session name
- **scan** – the scan number
- **time_series** – the 4D modeling input time series file location
- **opts** – the following keyword parameters:

Option bolus_arrival_index the bolus uptake volume index

Option mask the XNAT mask resource name

Returns the modeling result dictionary

technique = None
The modeling technique. Built-in techniques include `mock`.

workflow = None
The modeling workflow described in *qipipe.pipeline.modeling.ModelingWorkflow*.

qipipe.pipeline.modeling.**OHSU_CONF_SECTIONS = ['Fastfit', 'R1', 'AIF']**
The OHSU AIRC modeling configuration sections.

qipipe.pipeline.modeling.**associate**(*names*, *values*)
Captures the synchronized *names* and *values* in a dictionary.

Parameters

- **names** – the field names
- **values** – the field values

Returns the target {name: value} dictionary

qipipe.pipeline.modeling.**create_profile**(*technique*, *time_series*, *configuration*, *sections*, *dest*)
qipipe.helpers.metadata.create_profile() wrapper.

Parameters

- **technique** – the modeling technique
- **time_series** – the modeling input time series file path
- **configuration** – the modeling workflow interface settings
- **sections** – the profile sections
- **dest** – the output profile file path

qipipe.pipeline.modeling.**get_aif_shift**(*time_series*, *bolus_arrival_index*)
Calculates the arterial input function offset as:

$t_{\text{arrival}} - t_0$

where t_0 is the first slice acquisition time and t_{arrival} averages the acquisition times at and immediately following bolus arrival.

Parameters

- **time_series** – the modeling input 4D NIfTI image file path
- **bolus_arrival_index** – the bolus uptake series index

Returns the parameter CSV file path

qipipe.pipeline.modeling.**get_fit_params**(*cfg_file*, *aif_shift*)
Makes the CSV file containing the following modeling fit parameters:

•*aif_shift*: arterial input function parameter array

•*aif_delta_t*: acquisition time deltas

•*aif_shift*: acquisition time shift

•*r1_cr*: contrast R1

•*r1_b_pre*: pre-contrast R1

The *aif_shift* is calculated by *get_aif_shift()* and passed to this function. The remaining parameters are read from the *MODELING_CONF_FILE*.

Parameters **cfg_file** – the modeling configuration file

Returns the parameter CSV file path

qipipe.pipeline.modeling.**get_r1_0**(*pdw_file*, *t1w_file*, *max_r1_0*, *mask=None*)

Returns the R1_0 map NIfTI file from the given proton density and T1-weighted images. The R1_0 map is computed using the pdw_t1w_to_r1 function. The pdw_t1w_to_r1 module must be in the Python path.

Parameters

- **pdw_file** – the proton density NIfTI image file path
- **t1w_file** – the T1-weighted image file path
- **max_r1_0** – the R1_0 range maximum
- **mask** – the optional mask image file path to use

Returns the R1_0 map NIfTI image file path

qipipe.pipeline.modeling.**make_baseline**(*time_series*, *base_end*)

Makes the R1_0 computation baseline NIfTI file.

Parameters

- **time_series** – the modeling input 4D NIfTI image file path
- **base_end** – the exclusive limit of the baseline computation input series

Returns the baseline NIfTI file name

Raises **ModelingError** – if the end index is a negative number

qipipe.pipeline.modeling.**run**(*subject*, *session*, *scan*, *time_series*, ***opts*)

Creates a *qipipe.pipeline.modeling.ModelingWorkflow* and runs it on the given inputs.

Parameters

- **subject** – the input subject
- **session** – the input session
- **scan** – input scan
- **time_series** – the input 4D NIfTI time series
- **opts** – the *qipipe.pipeline.modeling.ModelingWorkflow* initializer and run options

Returns the *qipipe.pipeline.modeling.ModelingWorkflow.run()* result

pipeline_error

exception qipipe.pipeline.pipeline_error.**PipelineError**

Bases: exceptions.Exception

The common pipeline error class.

qipipeline

qipipe.pipeline.qipipeline.**MULTI_VOLUME_ACTIONS = ['stage', 'roi', 'register', 'model']**
The workflow actions which apply to a multi-volume scan.

class qipipe.pipeline.qipipeline.**QIPipelineWorkflow**(*project*, *scan_input*, *actions*, ***opts*)
Bases: *qipipe.pipeline.workflow_base.WorkflowBase*

QIPipeline builds and executes the imaging workflows. The pipeline builds a composite workflow which stitches together the following constituent workflows:

- staging: Prepare the new DICOM visits, as described in qipipe.pipeline.staging.StagingWorkflow
- mask: Create the mask from the staged images, as described in *qipipe.pipeline.mask.MaskWorkflow*
- registration: Mask, register and realign the staged images, as described in qipipe.pipeline.registration.RegistrationWorkflow
- modeling: Perform PK modeling as described in *qipipe.pipeline.modeling.ModelingWorkflow*

The constituent workflows are determined by the initialization options stage, register and model. The default is to perform each of these subworkflows.

The workflow steps are determined by the input options as follows:

- If staging is enabled, then the DICOM files are staged for the subject directory inputs. Otherwise, staging is not performed. In that case, if registration is enabled as described below, then the previously staged volume scan stack images are downloaded.
- If modeling is enabled and the registration resource option is set, then the previously realigned images with the given resource name are downloaded.
- If registration or modeling is enabled and the XNAT mask resource is found, then that resource file is downloaded. Otherwise, the mask is created from the staged images.

The workflow input node is *input_spec* with the following fields:

- *subject*: the subject name
- *session*: the session name
- *scan*: the scan number

The constituent workflows are combined as follows:

- The staging workflow input is the workflow input.
- The mask workflow input is the newly created or previously staged scan NIfTI image files.
- The modeling workflow input is the combination of the previously uploaded and newly realigned image files.

The pipeline workflow is available as the *qipipe.pipeline.qipipeline.QIPipelineWorkflow.workflow* instance variable.

Parameters

- **project** – the XNAT project name
- **scan_input** – the *qipipe.staging.iterator.iter_stage()* scan input
- **actions** – the actions to perform

- **opts** – the `qipipe.staging.WorkflowBase` initialization options as well as the following keyword arguments:
- **dest** – the staging destination directory
- **collection** – the image collection name
- **registration_resource** – the XNAT registration resource name
- **registration_technique** – the class:*qipipe.pipeline.registration.RegistrationWorkflow* technique
- **modeling_resource** – the modeling resource name
- **modeling_technique** – the class:*qipipe.pipeline.modeling.ModelingWorkflow* technique
- **scan_time_series** – the scan time series resource name
- **realigned_time_series** – the registered time series resource name

__init__(*project*, *scan_input*, *actions*, ***opts*)

Parameters

- **project** – the XNAT project name
- **scan_input** – the *qipipe.staging.iterator.iter_stage()* scan input
- **actions** – the actions to perform
- **opts** – the `qipipe.staging.WorkflowBase` initialization options as well as the following keyword arguments:
- **dest** – the staging destination directory
- **collection** – the image collection name
- **registration_resource** – the XNAT registration resource name
- **registration_technique** – the class:*qipipe.pipeline.registration.RegistrationWorkflow* technique
- **modeling_resource** – the modeling resource name
- **modeling_technique** – the class:*qipipe.pipeline.modeling.ModelingWorkflow* technique
- **scan_time_series** – the scan time series resource name
- **realigned_time_series** – the registered time series resource name

modeling_resource = None
The modeling XNAT resource name.

modeling_technique = None
The modeling technique.

registration_resource = None
The registration resource name.

registration_technique = None
The registration technique.

run_with_dicom_input(*actions*, *scan_input*)

Parameters

- **actions** – the workflow actions to perform
- **scan_input** – the *qipipe.staging.iterator.iter_stage()* scan input
- **dest** – the TCIA staging destination directory (default is the current working directory)

run_with_scan_download(*project*, *scan_input*, *actions*)
Runs the execution workflow on downloaded scan image files.

Parameters

- **project** – the project name
- **scan_input** – the {project, subject, session} object
- **actions** – the workflow actions

workflow = None
The pipeline execution workflow. The execution workflow is executed by calling the *run_with_dicom_input()* or *run_with_scan_download()* method.

qipipe.pipeline.qipipeline.**SINGLE_VOLUME_ACTIONS = ['stage']**
The workflow actions which apply to a single-volume scan.

qipipe.pipeline.qipipeline.**exclude_files**(*in_files*, *exclusions*)

Parameters

- **in_files** – the input file paths
- **exclusions** – the file names to exclude

Returns the filtered input file paths

qipipe.pipeline.qipipeline.**run**(**inputs*, ***opts*)
Creates a *qipipe.pipeline.qipipeline.QIPipelineWorkflow* and runs it on the given inputs. The pipeline execution depends on the *actions* option, as follows:

•If the workflow actions includes `stage` or `roi`, then the input is the *QIPipelineWorkflow.run_with_dicom_input()* DICOM subject directories input.

•Otherwise, the input is the *QIPipelineWorkflow.run_with_scan_download()* XNAT session labels input.

Parameters

- **inputs** – the input directories or XNAT session labels to process
- **opts** – the *qipipe.staging.iterator.iter_stage()* and *QIPipelineWorkflow* initializer options, as well as the following keyword options:
- **project** – the XNAT project name
- **collection** – the image collection name
- **actions** – the workflow actions to perform (default *MULTI_VOLUME_ACTIONS*)

registration

qipipe.pipeline.registration.**ANTS_CONF_SECTIONS = ['ants.Registration']**
The common ANTs registration configuration sections.

qipipe.pipeline.registration.**ANTS_INITIALIZER_CONF_SECTION = 'ants.AffineInitializer'**
The initializer ANTs registration configuration sections.

qipipe.pipeline.registration.**FSL_CONF_SECTIONS = ['fsl.FLIRT', 'fsl.FNIRT']**
The FSL registration configuration sections.

qipipe.pipeline.registration.**REG_PREFIX = 'reg_'**
The XNAT registration resource name prefix.

class qipipe.pipeline.registration.**RegisterImageWorkflow**(*technique*, ***opts*)
Bases: *qipipe.pipeline.workflow_base.WorkflowBase*

The RegisterImageWorkflow registers an input NIfTI scan image against a reference image.

Three registration techniques are supported:

- `ants`: ANTS SyN symmetric normalization diffeomorphic registration (default)
- `fsl`: FSL FNIRT non-linear registration
- `mock`: Test technique which copies each input scan image to the output image file

The optional workflow configuration file can contain overrides for the Nipype interface inputs in the following sections:

- **AffineInitializer: the** `qipipe.interfaces.ants.utils.AffineInitializer` options
- `ants.Registration`: the ANTs Registration interface options
- `ants.ApplyTransforms`: the ANTs ApplyTransform interface options
- `fsl.FNIRT`: the FSL FNIRT interface options

Note: Since the XNAT *resource* name is unique, a *qipipe.pipeline.registration.RegisterScanWorkflow* instance can be used for only one registration workflow. Different registration inputs require different *qipipe.pipeline.registration.RegisterScanWorkflow* instances.

If the optional configuration file is specified, then the workflow settings in that file override the default settings.

Parameters

- **technique** – the required registration *technique*
- **opts** – the *qipipe.pipeline.workflow_base.WorkflowBase* initializer options, as well as the following keyword arguments:
- **initialize** – flag indicating whether to create an initial affine transform (ANTs only, default false)

__init__(*technique*, ***opts*)
If the optional configuration file is specified, then the workflow settings in that file override the default settings.

Parameters

- **technique** – the required registration *technique*
- **opts** – the *qipipe.pipeline.workflow_base.WorkflowBase* initializer options, as well as the following keyword arguments:
- **initialize** – flag indicating whether to create an initial affine transform (ANTs only, default false)

run(*in_file*, *reference*, ***opts*)
Runs the realignment workflow on the given session scan image.

Parameters

- **reference** – the volume to register against
- **in_file** – the input session scan volume image file
- **opts** – the following keyword arguments:

Option mask the image mask file path

Returns the realigned output file paths

technique = None
The lower-case XNAT registration technique. The built-in techniques include `ants`, *fnirt*‘ and `mock`.

workflow = None
The realignment workflow.

class qipipe.pipeline.registration.**RegisterScanWorkflow**(*reference*, ***opts*)
Bases: *qipipe.pipeline.workflow_base.WorkflowBase*

The RegistrationWorkflow registers input NIfTI scan images against a reference image.

The mask can be obtained by running the *qipipe.pipeline.mask.MaskWorkflow* workflow.

Three registration techniques are supported:

- `ants`: ANTS SyN symmetric normalization diffeomorphic registration (default)
- `fsl`: FSL FNIRT non-linear registration
- `mock`: Test technique which copies each input scan image to the output image file

The optional workflow configuration file can contain overrides for the Nipype interface inputs in the following sections:

- **AffineInitializer: the** `qipipe.interfaces.ants.utils.AffineInitializer` options
- `ants.Registration`: the ANTs Registration interface options
- `ants.ApplyTransforms`: the ANTs ApplyTransform interface options
- `fsl.FNIRT`: the FSL FNIRT interface options

Note: Since the XNAT *resource* name is unique, a *qipipe.pipeline.registration.RegisterScanWorkflow* instance can be used for only one registration workflow. Different registration inputs require different *qipipe.pipeline.registration.RegisterScanWorkflow* instances.

If the optional configuration file is specified, then the workflow settings in that file override the default settings.

Parameters

- **reference** – the volume to register against
- **opts** – the *qipipe.pipeline.workflow_base.WorkflowBase* and *RegisterImageWorkflow* options, as well as the following keyword arguments:
- **technique** – the optional registration *technique* (default `DEF_TECHNIQUE`)

__init__ (*reference*, ***opts*)
If the optional configuration file is specified, then the workflow settings in that file override the default settings.

Parameters

- **reference** – the volume to register against
- **opts** – the *qipipe.pipeline.workflow_base.WorkflowBase* and *RegisterImageWorkflow* options, as well as the following keyword arguments:
- **technique** – the optional registration *technique* (default DEF_TECHNIQUE)

resource = None
The unique XNAT registration resource name. Uniqueness permits more than one registration to be stored for a given session without a name conflict.

run (*subject*, *session*, *scan*, *in_files*, *mask=None*)
Runs the registration workflow on the given session scan images.

Parameters

- **subject** – the subject name
- **session** – the session name
- **scan** – the scan number
- **in_files** – the input session scan volume image files
- **mask** – the optional image mask file path

Returns the realigned 4D time series file path

technique = None
The registration technique (default DEF_TECHNIQUE).

workflow = None
The registration workflow.

qipipe.pipeline.registration.**run** (*subject*, *session*, *scan*, *in_files*, ***opts*)
Runs the registration workflow on the given session scan images.

Parameters

- **subject** – the subject name
- **session** – the session name
- **scan** – the scan number
- **in_files** – the input session scan 3D NIfTI images
- **opts** – the *RegisterScanWorkflow* initializer and *RegisterScanWorkflow.run()* options as well as the following keyword option:
- **reference** – the volume number of the image to register against (default is the first image)

Returns the 4D registration time series

roi

The proprietary OHSU mask conversion workflow.

class qipipe.pipeline.roi.**ROIWorkflow**(**kwargs)
Bases: *qipipe.pipeline.workflow_base.WorkflowBase*

The ROIWorkflow class builds and executes the ROI workflow which converts the BOLERO mask `.bqf` files to NIfTI.

The ROI workflow input consists of the *input_spec* and *iter_slice* nodes. The *input_spec* contains the following input fields:

- *subject*: the subject name
- *session*: the session name
- *scan*: the scan number
- *time_series*: the 4D time series file path
- *lesion*: the lesion number

The *iter_slice* contains the following input fields:

- *slice_sequence_number*: the one-based slice sequence number
- *in_file*: the ROI mask‘‘.bqf‘‘ file to convert

The output is the 3D mask NIfTI file location. The file name is *lesion*`.nii.gz`.

If the optional configuration file is specified, then the workflow settings in that file override the default settings.

Parameters **kwargs** – the *qipipe.pipeline.workflow_base.WorkflowBase* initializer keyword arguments

__init__(**kwargs)
If the optional configuration file is specified, then the workflow settings in that file override the default settings.

Parameters **kwargs** – the *qipipe.pipeline.workflow_base.WorkflowBase* initializer keyword arguments

run(*subject*, *session*, *scan*, *time_series*, **inputs*)
Runs the ROI workflow on the given session scan images.

Parameters

- **subject** – the subject name
- **session** – the session name
- **scan** – the scan number
- **time_series** – the 4D scan time series file path
- **inputs** – the input (lesion number, slice sequence number, in_file) tuples to convert

Returns the XNAT converted ROI resource name, or None if there were no inputs

workflow = None
The ROI workflow.

qipipe.pipeline.roi.**ROI_FNAME_PAT = ‘lesion%d’**
The ROI file name pattern.

qipipe.pipeline.roi.**ROI_RESOURCE = ‘roi’**
The XNAT ROI resource name.

qipipe.pipeline.roi.**base_name**(*lesion*)

Parameters **lesion** – the lesion number

Returns the base name to use

qipipe.pipeline.roi.**run**(*subject*, *session*, *scan*, *time_series*, **inputs*, ***opts*)
Runs the ROI workflow on the given session ROI mask files.

Parameters

- **subject** – the subject name
- **session** – the session name
- **scan** – the scan number
- **time_series** – the 4D scan time series
- **inputs** – the *ROIWorkflow.run()* (lesion number, slice sequence number, in_file) inputs
- **opts** – the *ROIWorkflow* initializer options

Returns the *ROIWorkflow.run()* result

staging

qipipe.pipeline.staging.**SCAN_CONF_FILE = 'scan.cfg'**
The XNAT scan configuration file name.

qipipe.pipeline.staging.**SCAN_METADATA_RESOURCE = 'metadata'**
The label of the XNAT resource holding the scan configuration.

class qipipe.pipeline.staging.**ScanStagingWorkflow**(*is_multi_volume=True*, ***opts*)
Bases: *qipipe.pipeline.workflow_base.WorkflowBase*

The ScanStagingWorkflow class builds and executes the scan staging supervisory Nipype workflow. This workflow delegates to *qipipe.pipeline.staging.stage_volume()* for each iterated scan volume.

The scan staging workflow input is the *input_spec* node consisting of the following input fields:

•*collection*: the collection name

•*subject*: the subject name

•*session*: the session name

•*scan*: the scan number

The scan staging workflow has one iterable:

•the *iter_volume* node with input fields *volume* and *in_files*

This iterable must be set prior to workflow execution.

The staging workflow output is the *output_spec* node consisting of the following output field:

•*out_file*: the 3D volume stack NIfTI image file

Parameters

- **is_multi_volume** – flag indicating whether to include volume merge tasks
- **opts** – the *qipipe.pipeline.workflow_base.WorkflowBase* initializer keyword arguments

__init__(*is_multi_volume=True*, ***opts*)

Parameters

- **is_multi_volume** – flag indicating whether to include volume merge tasks
- **opts** – the *qipipe.pipeline.workflow_base.WorkflowBase* initializer keyword arguments

run(*collection*, *subject*, *session*, *scan*, *vol_dcm_dict*, *dest*)
Executes this scan staging workflow.

Parameters

- **collection** – the collection name
- **subject** – the subject name
- **session** – the session name
- **scan** – the scan number
- **vol_dcm_dict** – the input {volume: DICOM files} dictionary
- **dest** – the destination directory

Returns the (time series, volume files) tuple

workflow = None
The scan staging workflow sequence described in `qipipe.pipeline.staging.StagingWorkflow`.

class `qipipe.pipeline.staging.`**VolumeStagingWorkflow**(***opts*)
Bases: *qipipe.pipeline.workflow_base.WorkflowBase*

The StagingWorkflow class builds and executes the staging Nipype workflow. The staging workflow includes the following steps:

•Group the input DICOM images into volume.

•Fix each input DICOM file header using the *qipipe.interfaces.fix_dicom.FixDicom* interface.

•Compress each corrected DICOM file.

•Upload each compressed DICOM file into XNAT.

•Stack each new volume's 2-D DICOM files into a 3-D volume NIfTI file using the DcmStack interface.

•Upload each new volume stack into XNAT.

•Make the CTP QIN-to-TCIA subject id map.

•Collect the id map and the compressed DICOM images into a target directory in collection/subject/session/volume format for TCIA upload.

The staging workflow input is the *input_spec* node consisting of the following input fields:

•*collection*: the collection name

•*subject*: the subject name

•*session*: the session name

•*scan*: the scan number

The staging workflow has two iterables:

•the *iter_volume* node with input fields *volume* and *dest*

•the *iter_dicom* node with input fields *volume* and *dicom_file*

These iterables must be set prior to workflow execution. The *iter_volume dest* input is the destination directory for the *iter_volume volume*.

The *iter_dicom* node *itersource* is the `iter_volume.volume` field. The `iter_dicom.dicom_file` iterables is set to the {volume: [DICOM files]} dictionary.

The DICOM files to upload to TCIA are placed in the destination directory in the following hierarchy:

/path/to/dest/

subject/

session/

volume*volume number/* *file ...*

where:

- *subject* is the subject name, e.g. `Breast011`
- *session* is the session name, e.g. `Session03`
- *volume number* is determined by the *qipipe.staging.image_collection.Collection.patterns* `volume` DICOM tag
- *file* is the DICOM file name

The staging workflow output is the *output_spec* node consisting of the following output field:

- *image*: the 3D volume stack NIfTI image file

Note: Concurrent XNAT upload fails unpredictably due to one of

---

the causes described in the `qixnat.facade.XNAT.find` method documentation.

The errors are addressed by the following measures:

- setting an isolated `pyxnat` *cache_dir* for each execution node
- serializing the XNAT find-or-create access points with ‘‘JoinNode‘‘s
- increasing the SGE submission resource parameters as shown in the `conf/staging.cfg [upload]` section

If the optional configuration file is specified, then the workflow settings in that file override the default settings.

Parameters **opts** – the *qipipe.pipeline.workflow_base.WorkflowBase* initializer keyword arguments

__init__(***opts*)

If the optional configuration file is specified, then the workflow settings in that file override the default settings.

Parameters **opts** – the *qipipe.pipeline.workflow_base.WorkflowBase* initializer keyword arguments

run(*collection*, *subject*, *session*, *scan*, *volume*, *dest*, **in_files*)

Executes this volume staging workflow.

Parameters

- **collection** – the collection name
- **subject** – the subject name
- **session** – the session name

- **scan** – the scan number
- **volume** – the volume number
- **dest** – the destination directory
- **in_files** – the input DICOM files

Returns the output 3D NIfTI volume file path

workflow = None
The staging workflow sequence described in `qipipe.pipeline.staging.StagingWorkflow`.

`qipipe.pipeline.staging.`**run**(*subject*, *session*, *scan*, **in_dirs*, ***opts*)
Runs the staging workflow on the given DICOM input directory. The return value is a {*volume*: *file*} dictionary, where *volume* is the volume number and *file* is the 3D NIfTI volume file.

Parameters

- **subject** – the subject name
- **session** – the session name
- **scan** – the scan number
- **in_dirs** – the input DICOM file directories
- **opts** – the *ScanStagingWorkflow* initializer options

Returns the *ScanStagingWorkflow.run()* result

`qipipe.pipeline.staging.`**stage_volume**(*collection*, *subject*, *session*, *scan*, *volume*, *in_files*, *dest*, *opts*)
Stages the given volume. The processed DICOM `.dcm.gz` files are placed in the *dest*/*volume* subdirectory. The child *VolumeStagingWorkflow* runs in the _parent_/volume_volume_ directory, where:

- _parent_ is the parent base directory specified in the options (default current directory)
- _volume_ is the volume argument

Parameters

- **collection** – the collection name
- **subject** – the subject name
- **session** – the session name
- **scan** – the scan number
- **volume** – the volume number
- **in_files** – the input DICOM files
- **dest** – the parent destination directory
- **opts** – the *VolumeStagingWorkflow* initializer options

Returns the 3D NIfTI volume file

`qipipe.pipeline.staging.`**upload_dicom**(*project*, *subject*, *session*, *scan*, *dcm_dir*)
Uploads the staged `.dcm.gz` files in *dcm_dir* to the XNAT scan `DICOM` resource

Parameters

- **project** – the project name
- **subject** – the subject name

- **session** – the session name
- **scan** – the scan number
- **dcm_dir** – the input staged directory

qipipe.pipeline.staging.**upload_nifti**(*project*, *subject*, *session*, *scan*, *files*)
Uploads the staged NIfTI files to the XNAT scan NIFTI resource.

Parameters

- **project** – the project name
- **subject** – the subject name
- **session** – the session name
- **scan** – the scan number
- **files** – the NIfTI files to upload

qipipe.pipeline.staging.**volume_format**(*collection*)
The DcmStack format for making a file name from the DICOM volume tag.

Example:

```
>> volume_format('Sarcoma')
"volume%(AcquisitionNumber)03d"
```

Parameters **collection** – the collection name

Returns the volume file name format

workflow_base

class qipipe.pipeline.workflow_base.**WorkflowBase**(*name*, ***opts*)
Bases: object

The WorkflowBase class is the base class for the qipipe workflow wrapper classes.

If the *distributable* flag is set, then the execution is distributed using the Nipype plug-in specified in the configuration *plug_in* parameter.

The workflow plug-in arguments and node inputs can be specified in a qiutil.ast_config.ASTConfig file. The configuration directory order consist of the order consist of the search locations in low-to-high precedence order consist of the following:

1.the qipipe module conf directory

2.the *config_dir* initialization keyword option

The common configuration is loaded from the default.cfg file or in the directory locations. The workflow-specific configuration file name is the lower-case name of the WorkflowBase subclass with .cfg extension, e.g. registration.cfg for qipipe.workflow.registration.RegistrationWorkflow. The configuration settings are then loaded from the common configuration files followed by the workflow-specific configuration files.

Initializes this workflow wrapper object. The *parent* option obviates the other options.

Parameters

- **name** – the module name
- **opts** – the following keyword arguments:

- **project** – the *project*
- **parent** – the parent workflow for a child workflow
- **base_dir** – the *base_dir*
- **config_dir** – the optional workflow node *configuration* file location or dictionary
- **dry_run** – the *dry_run* flag
- **distributable** – the `distributable` flag

Raises ***PipelineError*** – if there is neither a *project* nor a *parent* argument

INTERFACE_PREFIX_PAT = <_sre.SRE_Pattern object>

Regexp matcher for an interface module.

Example:

```
>>> from qipipe.pipeline.workflow_base import WorkflowBase
>>> WorkflowBase.INTERFACE_PREFIX_PAT.match('nipype.interfaces.ants.util.
↪AverageImages').groups()
('nipype.',)
```

MODULE_PREFIX_PAT = <_sre.SRE_Pattern object>

Regexp matcher for a module prefix.

Example:

```
>>> from qipipe.pipeline.workflow_base import WorkflowBase
>>> WorkflowBase.MODULE_PREFIX_PAT.match('ants.util.AverageImages').groups()
('ants.', 'ants.', 'util.', 'AverageImages')
>>> WorkflowBase.MODULE_PREFIX_PAT.match('AverageImages')
None
```

__init__(*name*, ***opts*)

Initializes this workflow wrapper object. The *parent* option obviates the other options.

Parameters

- **name** – the module name
- **opts** – the following keyword arguments:
- **project** – the *project*
- **parent** – the parent workflow for a child workflow
- **base_dir** – the *base_dir*
- **config_dir** – the optional workflow node *configuration* file location or dictionary
- **dry_run** – the *dry_run* flag
- **distributable** – the `distributable` flag

Raises ***PipelineError*** – if there is neither a *project* nor a *parent* argument

base_dir = None

The workflow execution directory (default a new temp directory).

config_dir = None

The workflow node inputs configuration directory.

configuration = None

The workflow node inputs configuration.

depict_workflow(*workflow*)
Diagrams the given workflow graph. The diagram is written to the *name*`.dot.png` in the workflow base directory.

:param workflow the workflow to diagram

dry_run = None
Flag indicating whether to prepare but not run the workflow.

is_distributable = None
Flag indicating whether to submit jobs to a cluster.

logger = None
This workflow's logger.

project = None
The XNAT project name.

qiprofile

qiprofile Package

breast_pathology

This module updates the qiprofile database Subject pathology information from the pathology Excel workbook file.

class qipipe.qiprofile.breast_pathology.**BreastPathologyUpdate**(*subject*)
Bases: *qipipe.qiprofile.pathology.PathologyUpdate*

The Breast pathology update facade.

Parameters **subject** – the Subject Mongo Engine database object to update

__init__(*subject*)

Parameters **subject** – the Subject Mongo Engine database object to update

encounter_type(*row*)
Overrides qipipe.qiprofile.Pathology.encounter_type() to specialize the *intervention_type* to BreastSurgery.

Parameters **row** – the input row

Returns the REST data model Encounter subclass

pathology_content(*row*)
Collects the pathology object from the given input row. This subclass implementation adds the non-empty embedded fields specific to this tumor type.

Parameters **row** – the input row

Returns the {attribute: value} content dictionary

qipipe.qiprofile.breast_pathology.**HORMONES = ['estrogen', 'progesterone']**
The receptor status hormones.

qipipe.qiprofile.breast_pathology.**read**(*workbook*, ***condition*)
This is a convenience method that wraps BreastPathologyWorksheet qipipe.qiprofile.xls.Worksheet.read().

Parameters

- **workbook** – the read-only openpyxl workbook object

- **condition** – the `qipipe.qiprofile.xls.Worksheet.read()` filter condition

Returns the `qipipe.qiprofile.xls.Worksheet.read()` rows

`qipipe.qiprofile.breast_pathology.`**update**(*subject*, *rows*)
Updates the given subject data object from the Breast pathology XLS rows.

Parameters

- **subject** – the `Subject` Mongo Engine database object to update
- **rows** – the input pathology *read()* rows list

chemotherapy

This module updates the qiprofile database Subject chemotherapy protocol information from a Chemotherapy Excel worksheet.

`qipipe.qiprofile.chemotherapy.`**COL_ATTRS = {'Cumulative Amount (mg/m2 BSA)': 'amount'}**
The following non-standard column-attribute associations: * The Cumulative Amount column is the amount attribute.

`qipipe.qiprofile.chemotherapy.`**SHEET = 'Chemotherapy'**
The input XLS sheet name.

`qipipe.qiprofile.chemotherapy.`**read**(*workbook*, ***condition*)
This is a convenience method that wraps `ChemotherapyWorksheet qipipe.qiprofile.xls.Worksheet.read()`.

Parameters

- **workbook** – the read-only `openpyxl` workbook object
- **condition** – the `qipipe.qiprofile.xls.Worksheet.read()` filter condition

Returns the `qipipe.qiprofile.xls.Worksheet.read()` rows

`qipipe.qiprofile.chemotherapy.`**update**(*subject*, *rows*)
Updates the given subject data object from the dosage XLS rows.

Parameters

- **subject** – the `Subject` Mongo Engine database object to update
- **rows** – the input chemotherapy *read()* rows list

clinical

This module updates the qiprofile database clinical information from the clinical Excel workbook file.

`qipipe.qiprofile.clinical.`**update**(*subject*, *in_file*)
Updates the subject clinical database content from the given workbook file.

Parameters

- **subject** – the target qiprofile Subject to update
- **filename** – the input file location

demographics

This module updates the qiprofile database Subject demographics information from the demographics Excel workbook file.

qipipe.qiprofile.demographics.**COL_ATTRS = {'Race': 'races'}**
The following non-standard column-attribute associations: * The Race column is the races attribute.

qipipe.qiprofile.demographics.**SHEET = 'Demographics'**
The input XLS sheet name.

qipipe.qiprofile.demographics.**read**(*workbook*, ***condition*)
Reads the demographics XLS row which matches the given subject.

Parameters **condition** – the row selection filter

Returns the Demographics sheet qipipe.qiprofile.xls.Worksheet.read() row bundle which matches the given subject, or None if no match was found

qipipe.qiprofile.demographics.**update**(*subject*, *rows*)
Updates the given subject data object from the given Demographics sheet rows.

There can be no more than one Demographics update input row for the given subject. The *rows* parameter is an iterable in order to conform to other sheet facade modules.

Parameters

- **subject** – the Subject Mongo Engine database object to update
- **rows** – the input Demographics *read()* rows

Raises **DemographicsError** – if there is more than one input row

dosage

This module updates the qiprofile database Subject drug dosage information from a Chemotherapy Excel worksheet.

class qipipe.qiprofile.dosage.**DosageUpdate**(*subject*, *agent_class*, ***defaults*)
Bases: object

The dosage update abstract class.

Parameters

- **subject** – the Subject Mongo Engine database object to update
- **agent_class** – the dosage agent class
- **defaults** – the {attribute: value} row defaults

DEFAULTS = {'duration': 1}
The default duration is 1 day.

__init__(*subject*, *agent_class*, ***defaults*)

Parameters

- **subject** – the Subject Mongo Engine database object to update
- **agent_class** – the dosage agent class
- **defaults** – the {attribute: value} row defaults

update(*rows*)
Updates the subject data object from the given dosage XLS rows.

Parameters **rows** – the input dosage `qipipe.qiprofile.xls.Worksheet.read()` rows list

class `qipipe.qiprofile.dosage.`**DosageWorksheet**(*workbook*, *sheet*, *agent_class*, ***opts*)
Bases: `qipipe.qiprofile.xls.Worksheet`

The dosage worksheet facade.

Parameters

- **workbook** – the `qipipe.qiprofile.xls.Workbook` object
- **sheet** – the sheet name
- **agent_class** – the agent class
- **opts** – the additional `qipipe.qiprofile.xls.Worksheet` initializer options

__init__(*workbook*, *sheet*, *agent_class*, ***opts*)

Parameters

- **workbook** – the `qipipe.qiprofile.xls.Workbook` object
- **sheet** – the sheet name
- **agent_class** – the agent class
- **opts** – the additional `qipipe.qiprofile.xls.Worksheet` initializer options

imaging

This module updates the qiprofile database imaging information from a XNAT scan.

`qipipe.qiprofile.imaging.`**update**(*subject*, *experiment*, ***opts*)
Updates the imaging content for the qiprofile REST Subject from the XNAT experiment.

Parameters

- **collection** – the `qipipe.staging.image_collection.name`\`
- **subject** – the target qiprofile Subject to update
- **experiment** – the XNAT experiment object
- **opts** – the :class‘Updater‘ keyword arguments

modeling

This module updates the qiprofile database modeling information from a XNAT experiment.

`qipipe.qiprofile.modeling.`**update**(*session*, *resource*)
Updates the modeling content for the given qiprofile session database object from the given XNAT modeling resource object.

Parameters

- **session** – the target qiprofile Session to update
- **resource** – the XNAT modeling resource object

parse

qipipe.qiprofile.parse.**COMMA_DELIM_REGEX = <_sre.SRE_Pattern object>**
Match a comma with optional white space.

qipipe.qiprofile.parse.**FALSE_REGEX = <_sre.SRE_Pattern object>**
The valid False string representations are a case-insensitive match for `F(alse)?`, `Neg(ative)?`, `Absent` or `N(o)?`.

qipipe.qiprofile.parse.**TRAILING_NUM_REGEX = <_sre.SRE_Pattern object>**
A regular expression to extract the trailing number from a string.

qipipe.qiprofile.parse.**TRUE_REGEX = <_sre.SRE_Pattern object>**
The valid True string representations are a case-insensitive match for `T(rue)?`, `Pos(itive)?`, `Present` or `Y(es)?`.

qipipe.qiprofile.parse.**TYPE_PARSERS = {<class 'mongoengine.fields.ListField'>: <function <lambda>>, <class 'mon**
The following type cast conversion parsers: * string field => `str` * integer field => `int` * float field => `float` * boolean field => *parse_boolean()* * list field => *parse_list_string()*

qipipe.qiprofile.parse.**default_parsers**(**classes*)
Associates the data model class fields to a parse function composed as follows:

- The type cast function in *TYPE_PARSERS*, if present
- The controlled value lookup, if the field has controlled values

Parameters **classes** – the data model classes

Returns the {attribute: function} dictionary

qipipe.qiprofile.parse.**extract_trailing_number**(*value*)
Returns the integer at the end of the given input value, as follows:

- If the input value is an integer, then the result is the input value.
- Otherwise, if the input value has a string type, then the result is the trailing digits converted to an integer.
- Any other value is an error.

Parameters **value** – the input integer or string

Returns the trailing integer

Raises **ParseError** – if the input value type is not int or a string type

qipipe.qiprofile.parse.**parse_boolean**(*value*)
Parses the input value as follows:

- If the input value is already a boolean, then return the value
- If the input is None or the empty string, then return None
- **Otherwise, if the input is a string which matches** *TRUE_REGEX*, then return True
- **Otherwise, if the input is a string which matches** *FALSE_REGEX*, then return False
- Any other value is an error.

Parameters **value** – the input value

Returns the value as a boolean

Raises **ParseError** – if the value cannot be converted

qipipe.qiprofile.parse.**parse_list_string**(*value*)
Converts a comma-separated list input string to a list, e.g.:

>> from qipipe.qiprofile.parse import parse_list_string >> parse_list_string('White, Asian') ['White', 'Asian']

Parameters **value** – the input value

Returns the value converted to a list

qipipe.qiprofile.parse.**parse_trailing_number**(*s*)

Parameters **s** – the input string

Returns the trailing number in the string

Raises **ParseError** – if the input string does not have a trailing number

pathology

This module updates the qiprofile database Subject pathology information from the pathology Excel workbook file.

qipipe.qiprofile.pathology.**COL_ATTRS = {'Tumor Width (mm)': 'width', 'Tumor Depth (mm)': 'depth', 'Tumor Siz**
The following non-standard column-attribute associations:

- `Patient Weight (kg)`: *Encounter.weight* attribute
- `Tumor Size Score`: *TNM.size* attribute
- `Tumor Length (mm)`: *TumorExtent.length* attribute
- `Tumor Width (mm)`: *TumorExtent.width* attribute
- `Tumor Depth (mm)`: *TumorExtent.depth* attribute

qipipe.qiprofile.pathology.**ENCOUNTER_TYPES = {'Surgery': <class 'qirest_client.model.clinical.Surgery'>, 'Biopsy**
The encounter {name: class} dictionary.

qipipe.qiprofile.pathology.**PARSERS = {'size': <function <lambda>>, 'body_part': <function <lambda>>, 'lesion_nu**
The following parser associations:

- *subject_number* is an int
- *intervention_type* converts the string to an Encounter subclass
- *body_part* is capitalized
- *size* is a `qirest_client.clinical.TNM.Size` object

class qipipe.qiprofile.pathology.**PathologyUpdate**(*subject*, *tumor_type*, *grade_class*, *pathology_class*)
Bases: `object`

The pathology update abstract class.

Parameters

- **subject** – the `Subject` Mongo Engine database object to update
- **tumor_type** – the subclass tumor type

Option pathology_class the REST data model TumorPathology subclass

Option grade_class the REST data model Grade subclass

__init__(*subject*, *tumor_type*, *grade_class*, *pathology_class*)

Parameters

- **subject** – the `Subject` Mongo Engine database object to update
- **tumor_type** – the subclass tumor type

Option pathology_class the REST data model TumorPathology subclass

Option grade_class the REST data model Grade subclass

encounter_type(*row*)
Infers the encounter type from the given row. This base implementation returns the parsed row *intervention_type* value.

Parameters **row** – the input row

Returns the REST data model Encounter subclass

pathology_content(*row*)
Collects the TumorPathology content from the given input row. This base implementation collects the pathology attribute values from the matching input row attribute value. Other updates are a subclass responsibility.

Parameters **row** – the input row

Returns the {attribute: value} content dictionary

update(*rows*)
Updates the subject data object from the given pathology XLS rows.

Parameters **rows** – the input pathology `read()` rows list

update_encounter(*encounter*, *rows*)
Update the encounter object from the given input row. This base implementation sets the encounter attribute values from the matching input row attribute value and calls `update_pathology()` to update the pathology. Other updates are a subclass responsibility.

Parameters

- **encounter** – the encounter object
- **rows** – the input pathology `read()` rows for the encounter

class `qipipe.qiprofile.pathology.`**PathologyWorksheet**(*workbook*, **classes*, ***opts*)
Bases: `qipipe.qiprofile.xls.Worksheet`

The Pathology worksheet facade.

Parameters

- **workbook** – the `qipipe.qiprofile.xls.Workbook` object
- **classes** – the subclass-specific REST data model subclasses
- **opts** – the following keyword arguments:

Option parsers the non-standard parsers {attribute: function} dictionary

Option column_attributes the non-standard {column name: attribute} dictionary

__init__(*workbook*, **classes*, ***opts*)

Parameters

- **workbook** – the `qipipe.qiprofile.xls.Workbook` object
- **classes** – the subclass-specific REST data model subclasses
- **opts** – the following keyword arguments:

Option parsers the non-standard parsers {attribute: function} dictionary

Option column_attributes the non-standard {column name: attribute} dictionary

qipipe.qiprofile.pathology.**SHEET = 'Pathology'**
The worksheet name.

radiotherapy

This module updates the qiprofile database Subject radiation protocol information from a Radiotherapy Excel worksheet.

qipipe.qiprofile.radiotherapy.**AGENT_DEFAULTS = {'beam_type': 'photon'}**
The default beam type is `photon`.

qipipe.qiprofile.radiotherapy.**COL_ATTRS = {'Cumulative Amount (Gy)': 'amount'}**
The following non-standard column-attribute associations: * The Cumulative Amount column is the amount attribute.

qipipe.qiprofile.radiotherapy.**SHEET = 'Radiotherapy'**
The input XLS sheet name.

qipipe.qiprofile.radiotherapy.**read**(*workbook*, ***condition*)
This is a convenience method that wraps `RadiotherapyWorksheet qipipe.qiprofile.xls.Worksheet.read()`.

Parameters

- **workbook** – the read-only `openpyxl` workbook object
- **condition** – the `qipipe.qiprofile.xls.Worksheet.read()` filter condition

Returns the `qipipe.qiprofile.xls.Worksheet.read()` rows

qipipe.qiprofile.radiotherapy.**update**(*subject*, *rows*)
Updates the given subject data object from the dosage XLS rows.

Parameters

- **subject** – the `Subject` Mongo Engine database object to update
- **rows** – the input radiotherapy *`read()`* rows list

sarcoma_pathology

This module updates the qiprofile database Subject pathology information from the pathology Excel workbook file.

qipipe.qiprofile.sarcoma_pathology.**COL_ATTRS = {'Tumor Location': 'location'}**
The following special column: attribute associations:

•The `Tumor Location` column corresponds to the pathology `location` attribute

qipipe.qiprofile.sarcoma_pathology.**PARSERS = {'necrosis_percent': <function <lambda>>}**
The following special parsers: * The necrosis percent can be an integer or a range, e.g. `80-90`.

class qipipe.qiprofile.sarcoma_pathology.**SarcomaPathologyUpdate**(*subject*)
Bases: *qipipe.qiprofile.pathology.PathologyUpdate*

The Sarcoma pathology update facade.

Parameters **subject** – the `Subject` Mongo Engine database object to update

__init__(*subject*)

Parameters **subject** – the `Subject` Mongo Engine database object to update

pathology_content(*row*)

Collects the pathology object from the given input row. This subclass implementation adds the following items:

- If there are *necrosis_percent* and *tnm* items, then the TNM necrosis_score is inferred from the necrosis percent

Parameters **row** – the input row

Returns the {attribute: value} content dictionary

qipipe.qiprofile.sarcoma_pathology.**read**(*workbook*, ***condition*)

This is a convenience method that wraps `SarcomaPathologyWorksheet` `qipipe.qiprofile.xls.Worksheet.read()`.

Parameters

- **workbook** – the read-only `openpyxl` workbook object
- **condition** – the `qipipe.qiprofile.xls.Worksheet.read()` filter condition

Returns the `qipipe.qiprofile.xls.Worksheet.read()` rows

qipipe.qiprofile.sarcoma_pathology.**update**(*subject*, *rows*)

Updates the given subject data object from the Sarcoma pathology XLS rows.

Parameters

- **subject** – the `Subject` Mongo Engine database object to update
- **rows** – the input pathology *read()* rows list

scan

This module updates the qiprofile database scan information from a XNAT experiment.

qipipe.qiprofile.scan.**update**(*session*, *xscan*)

Updates the scan content for the given qiprofile session database object from the given XNAT scan object.

Parameters

- **session** – the target qiprofile Session to update
- **xscan** – the XNAT scan object

update

qipipe.qiprofile.update.**update**(*project*, *collection*, *subject*, *session*, *in_file*)

Updates the qiprofile database from the clinical spreadsheet and XNAT database for the given session.

Parameters

- **project** – the XNAT project name
- **collection** – the image collection name
- **subject** – the subject number

- **session** – the XNAT session number
- **in_file** – the input spreadsheet file location

xls

class qipipe.qiprofile.xls.**Reader**(*worksheet*, *attributes*, ***condition*)
Bases: object

Reads an Excel worksheet.

Parameters

- **worksheet** – the *worksheet* object
- **conditional** – the optional {attribute: value} row filter condition

__init__(*worksheet*, *attributes*, ***condition*)

Parameters

- **worksheet** – the *worksheet* object
- **conditional** – the optional {attribute: value} row filter condition

read()
Returns a row generator, where each row is a {attribute: value} bunch. This generator yields each row which satisfies the following condition:

1. the row is non-empty, i.e. has at least one cell value, and
2. if this reader has a filter, then the row satisfies the filter condition

Returns the filtered openpyxl row iterator

worksheet = None
The wrapped openpyxl worksheet.

qipipe.qiprofile.xls.**load_workbook**(*filename*)

Parameters **filename** – the XLS workbook file location

Returns the read-only openpyxl workbook object

staging

staging Package

Image processing preparation.

The staging package defines the functions used to prepare the study image files for import into XNAT, submission to the TCIA QIN collections and pipeline processing.

ctp_config

qipipe.staging.ctp_config.**ctp_collection_for_name**(*name*)

Parameters **name** – the QIN collection name

Returns the CTP collection name

fix_dicom

qipipe.staging.fix_dicom.**COMMENT_PREFIX = <_sre.SRE_Pattern object>**
OHSU - the `Image Comments` tag value prefix.

qipipe.staging.fix_dicom.**DATE_FMT = '%Y%m%d'**
The DICOM date format is YYYYMMDD.

qipipe.staging.fix_dicom.**fix_dicom_headers**(*collection*, *subject*, **in_files*, ***opts*)
Fix the given input DICOM files as follows:

- **Replace the `Patient ID` value with the subject number, e.g.** `Sarcoma001`
- Add the `Body Part Examined` tag
- Anonymize the `Patient's Birth Date` tag
- Standardize the file name

OHSU - The `Body Part Examined` tag is set as follows:

- **If the collection is `Sarcoma`, then the body part is the** *qipipe.staging.sarcoma_config.sarcoma_location()*.
- **Otherwise, the body part is the capitalized collection name, e.g.** `BREAST`.

OHSU - Remove extraneous `Image Comments` tag value content which might contain PHI.

The output file name is standardized as follows:

- The file name is lower-case
- The file extension is `.dcm`
- Each non-word character is replaced by an underscore

Parameters

- **collection** – the collection name
- **subject** – the input subject name
- **opts** – the following keyword arguments:
- **dest** – the location in which to write the modified files (default is the current directory)

Returns the files which were created

Raises **StagingError** – if the collection is not supported

image_collection

class qipipe.staging.image_collection.**Collection**(*name*, ***opts*)
Bases: `object`

The image collection.

Parameters

- **name** – the *name*
- **opts** – the following keyword options:

Option subject the subject directory name match regular expression

Option session the session directory name match regular expression

Option scan_types the *scan_types*

Option scan the {scan number: {dicom, roi}} dictionary

Option volume the DICOM tag which identifies a scan volume

Option crop_posterior the *crop_posterior* flag

__init__(*name*, ***opts*)

Parameters

- **name** – the *name*
- **opts** – the following keyword options:

Option subject the subject directory name match regular expression

Option session the session directory name match regular expression

Option scan_types the *scan_types*

Option scan the {scan number: {dicom, roi}} dictionary

Option volume the DICOM tag which identifies a scan volume

Option crop_posterior the *crop_posterior* flag

crop_posterior = None
A flag indicating whether to crop the image posterior in the mask, e.g. for a breast tumor (default False).

instances = {'sarcoma': <qipipe.staging.image_collection.Collection object>, 'breast': <qipipe.staging.image_collectio
The collection {name: object} dictionary.

name = None
The capitalized collection name.

patterns = None
The DICOM and ROI meta-data patterns. This `patterns` attribute consists of the entries `dicom` and `roi`, Each of these fields has a mandatory `glob` entry and an optional `regex` entry. The `glob` entry matches the scan subdirectory containing the DICOM or ROI files. The `regex` entry matches the DICOM or ROI files in the subdirectory. The default in the absence of a `regex` entry is to include all files in the subdirectory.

scan_types = None
The scan {number: type} dictionary.

qipipe.staging.image_collection.**with_name**(*name*)

Returns the *Collection* whose name is a case-insensitive match for the given name, or None if no match is found

iterator

class qipipe.staging.iterator.**VisitIterator**(*project*, *collection*, **session_dirs*, ***opts*)
Bases: object

Scan DICOM generator class .

Parameters

- **project** – the XNAT project name
- **collection** – the image collection name

- **session_dirs** – the session directories over which to iterate
- **opts** – the *iter_stage()* options

__init__(*project*, *collection*, **session_dirs*, ***opts*)

Parameters

- **project** – the XNAT project name
- **collection** – the image collection name
- **session_dirs** – the session directories over which to iterate
- **opts** – the *iter_stage()* options

collection = None
The *iter_stage()* collection name parameter.

project = None
The *iter_stage()* project name parameter.

scan = None
The *iter_stage()* scan number option.

session_dirs = None
The input directories.

skip_existing = None
The *iter_stage()* *skip_existing* flag option.

qipipe.staging.iterator.**iter_stage**(*project*, *collection*, **inputs*, ***opts*)
Iterates over the the scans in the given input directories. This method is a staging generator which yields a tuple consisting of the {subject, session, scan, dicom, roi} object.

The input directories conform to the *qipipe.staging.image_collection.Collection.patterns* `subject` regular expression.

Each iteration {subject, session, scan, dicom, roi} object is formed as follows:

•The *subject* is the XNAT subject name formatted by `SUBJECT_FMT`.

•The *session* is the XNAT experiment name formatted by `SESSION_FMT`.

•The *scan* is the XNAT scan number.

•*dicom* is the DICOM directory.

•*roi* is the ROI directory.

Parameters

- **project** – the XNAT project name
- **collection** – the *qipipe.staging.image_collection.Collection.name*
- **inputs** – the source subject directories to stage
- **opts** – the following keyword option:
- **scan** – the scan number to stage (default stage all detected scans)
- **skip_existing** – flag indicating whether to ignore each existing session, or scan if the *scan* option is set (default True)

Yield the {subject, session, scan, dicom, roi} objects

map_ctp

TCIA CTP preparation utilities.

class qipipe.staging.map_ctp.**CTPPatientIdMap**
Bases: dict

CTPPatientIdMap is a dictionary augmented with a `map_subjects()` input method to build the map and a *write()* output method to print the CTP map properties.

CTP_FMT = '%s-%04d'
The CTP Patient ID format with arguments (CTP collection name, input Patient ID number).

MAP_FMT = 'ptid/%s=%s'
The ID lookup entry format with arguments (input Paitent ID, CTP patient id).

MSG_FMT = 'Mapped the QIN patient id %s to the CTP subject id %s.'
The log message format with arguments (input Paitent ID, CTP patient id).

SOURCE_PAT = <_sre.SRE_Pattern object>
The input Patient ID pattern is the study name followed by a number, e.g. `Breast10`.

add_subjects(*collection*, **patient_ids*)
Adds the input => CTP Patient ID association for the given input DICOM patient ids.

Parameters

- **collection** – the image collection name
- **patient_ids** – the DICOM Patient IDs to map

Raises **StagingError** – if an input patient id format is not the study followed by the patient number

write(*dest=<open file '<stdout>', mode 'w'>*)
Writes this id map in the standard CTP format.

Parameters **dest** – the IO stream on which to write this map (default stdout)

qipipe.staging.map_ctp.**PROP_FMT = 'QIN-%s-OHSU.ID-LOOKUP.properties'**
The format for the Patient ID map file name specified by CTP.

qipipe.staging.map_ctp.**map_ctp**(*collection*, **subjects*, ***opts*)
Creates the TCIA patient id map. The map is written to a property file in the destination directory. The property file name is given by *property_filename()*.

Parameters

- **collection** – the image collection
- **subjects** – the subject names
- **opts** – the following keyword option:
- **dest** – the destination directory

Returns the subject map file path

qipipe.staging.map_ctp.**property_filename**(*collection*)
Returns the CTP id map property file name for the given collection. The Sarcoma collection is capitalized in the file name, Breast is not.

ohsu

This module contains the OHSU-specific image collections.

The following OHSU QIN scan numbers are captured:

- 1: T1
- 2: T2
- 4: DW
- 6: PD

These scans have DICOM files specified by the *qipipe.staging.image_collection.Collection.patterns* `dicom` attribute. The T1 scan has ROI files as well, specified by the patterns `roi.glob` and `roi.regex` attributes.

`qipipe.staging.ohsu.`**BREAST_DW_PAT = '*sorted/*Diffusion'**
The Breast DW DICOM directory match pattern.

`qipipe.staging.ohsu.`**BREAST_PD_PAT = '*sorted/*PD*'**
The Breast pseudo-proton density DICOM directory match pattern.

`qipipe.staging.ohsu.`**BREAST_ROI_PAT = 'processing/R10_0.[456]*/slice*'**
The Breast ROI glob filter. The `.bqf` ROI files are in the following session subdirectory:

processing/<R10 directory>/slice<slice index>/

`qipipe.staging.ohsu.`**BREAST_ROI_REGEX = <_sre.SRE_Pattern object at 0x3ac1060>**
The Breast ROI .bqf ROI file match pattern.

`qipipe.staging.ohsu.`**BREAST_SESSION_REGEX = <_sre.SRE_Pattern object>**
The Sarcoma session directory match pattern. The variations `Visit_3`, `Visit3`, `visit3`, `BC4V3`, `BC4_V3` and `B4V3` all match Breast Session03.

`qipipe.staging.ohsu.`**BREAST_SUBJECT_REGEX = <_sre.SRE_Pattern object>**
The Breast subject directory match pattern.

`qipipe.staging.ohsu.`**BREAST_T2_PAT = '*sorted/2_tirm_tra_bilat'**
The Breast T2 DICOM directory match pattern.

`qipipe.staging.ohsu.`**MULTI_VOLUME_SCAN_NUMBERS = [1]**
Only T1 scans can have more than one volume.

`qipipe.staging.ohsu.`**SARCOMA_DW_PAT = '*Diffusion'**
The Sarcoma DW DICOM directory match pattern.

`qipipe.staging.ohsu.`**SARCOMA_ROI_PAT = 'Breast processing results/multi_slice/slice*'**
The Sarcoma ROI glob filter. The `.bqf` ROI files are in the session subdirectory:

Breast processing results/<ROI directory>/slice<slice index>/

(Yes, the Sarcoma processing results is in the "Breast processing results" subdirectory)!

`qipipe.staging.ohsu.`**SARCOMA_ROI_REGEX = <_sre.SRE_Pattern object>**
The Sarcoma ROI .bqf ROI file match pattern.

Note: The Sarcoma ROI directories are inconsistently named, with several alternatives and duplicates.

TODO - clarify which of the Sarcoma ROI naming variations should be used.

Note: There are no apparent lesion number indicators in the Sarcoma ROI input.

TODO - confirm that there is no Sarcoma lesion indicator.

qipipe.staging.ohsu.**SARCOMA_SESSION_REGEX = <_sre.SRE_Pattern object>**
The Sarcoma session directory match pattern. The variations `Visit_3`, `Visit3`, `visit3` `S4V3`, and `S4_V3` all match Sarcoma Session03.

qipipe.staging.ohsu.**SARCOMA_SUBJECT_REGEX = <_sre.SRE_Pattern object>**
The Sarcoma subject directory match pattern.

qipipe.staging.ohsu.**SARCOMA_T2_PAT = '*T2*'**
The Sarcoma T2 DICOM directory match pattern.

qipipe.staging.ohsu.**SESSION_REGEX_PAT = '\n (?: # Don't capture the prefix\n [vV]isit # The Visit or visit prefix forr**
The session directory match pattern. This pattern must be specialized for each collection by replacing the %s place-holder with a string.

qipipe.staging.ohsu.**T1_PAT = '*concat*'**
The T1 DICOM directory match pattern.

qipipe.staging.ohsu.**VOLUME_TAG = 'AcquisitionNumber'**
The DICOM tag which identifies the volume. The OHSU QIN collections are unusual in that the DICOM images which comprise a 3D volume have the same DICOM Series Number and Acquisition Number tag. The series numbers are consecutive, non-sequential integers, e.g. 9, 11, 13, ..., whereas the acquisition numbers are consecutive, sequential integers starting at 1. The Acquisition Number tag is selected as the volume number identifier.

roi

OHSU - ROI utility functions.

TODO - move this to ohsu-qipipe.

class qipipe.staging.roi.**LesionROI**(*lesion*, *volume_number*, *slice_sequence_number*, *location*)
Bases: `object`

Aggregate with attributes *lesion* *volume*, *slice* and *location*.

Parameters

- **lesion** – the *lesion* value
- **volume_number** – the *volume* value
- **slice_sequence_number** – the *slice* value
- **location** – the *location* value

__init__(*lesion*, *volume_number*, *slice_sequence_number*, *location*)

Parameters

- **lesion** – the *lesion* value
- **volume_number** – the *volume* value
- **slice_sequence_number** – the *slice* value
- **location** – the *location* value

lesion = None
The lesion number.

location = None
The absolute BOLERO ROI .bqf file path.

slice = None
The one-based slice sequence number.

volume = None
The one-based volume number.

qipipe.staging.roi.**PARAM_REGEX = <_sre.SRE_Pattern object>**
The regex to parse a parameter file.

qipipe.staging.roi.**iter_roi**(*regex*, **in_dirs*)
Iterates over the the OHSU ROI `.bqf` mask files in the given input directories. This method is a *LesionROI* generator, e.g.:

```
>>> # Find .bqf files anywhere under /path/to/session/processing.
>>> next(iter_roi('.*/\.bqf', '/path/to/session'))
{lesion: 1, slice: 12, path: '/path/to/session/processing/rois/roi.bqf'}
```

;param regex the file name match regular expression

Parameters **in_dirs** – the ROI directories to search

Yield the *LesionROI* objects

sort

qipipe.staging.sort.**sort**(*collection*, *scan*, **in_dirs*)
Groups the DICOM files in the given location by volume.

Parameters

- **collection** – the collection name
- **scan** – the scan number
- **in_dirs** – the input DICOM directories

Returns the {volume: files} dictionary

sarcoma_config

qipipe.staging.sarcoma_config.**CFG_FILE = '/home/docs/checkouts/readthedocs.org/user_builds/qipipe/checkouts/st**
The Sarcoma Tumor Location configuration file. This file contains properties that associate the subject name to the location, e.g.:

```
Sarcoma004 = SHOULDER
```

The value is the SNOMED anatomy term.

qipipe.staging.sarcoma_config.**sarcoma_config**()

Returns the sarcoma configuration

Return type ConfigParser

qipipe.staging.sarcoma_config.**sarcoma_location**(*subject*)

Parameters **subject** – the XNAT Subject ID

Returns the subject tumor location

staging_error

Python Module Index

q

Index

Symbols

A

B

C

D

E

F

G

Q

R

S

T

U

V

W

X