qipipe

Release

Aug 02, 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

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/latest/qip**
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

interfaces

interfaces Package

compress

convert_bolero_mask

copy

dce_to_r1

fastfit

fix_dicom

group_dicom

interface_error

lookup

map_ctp

move

mri_volcluster

preview

reorder_bolero_mask

sticky_identity

touch

uncompress

unpack

update_qiprofile

xnat_copy

xnat_download

xnat_find

xnat_upload

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

modeling

pipeline_error

exception qipipe.pipeline.pipeline_error.**PipelineError**
Bases: exceptions.Exception

The common pipeline error class.

qipipeline

registration

roi

staging

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

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.StringField'>: <type 'str'>, <class 'mongoengin**
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

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 0x426bf40>**
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/la**
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

Q

R

S

T

U

V

W