CONTENTS:

1 Overview 1

2 Realtime fMRI Cloud Framework 3
  2.1 How it works ................................................................. 3
  2.2 Other Links ................................................................. 5
  2.3 Installation ................................................................. 5
    2.3.1 Step 1: Install Mini-Conda and NodeJS ......................... 5
    2.3.2 Step 2: Install Realtime ProjectInterface on cloud VM (All OS types) ................................................. 6
    2.3.3 Step 3: Install ScannerDataService and/or SubjectService on the Console and Presentation Computers (All OS Types) ................................................. 6
  2.4 Testing the Sample Project ............................................. 6
    2.4.1 Using the Sample Project with synthetic data generated by BrainIAK ................................................. 7
  2.5 Next Steps ................................................................. 7
  2.6 Troubleshooting ........................................................... 8
  2.7 Running the Automated Test Suite .................................... 9
  2.8 Further Reading ........................................................... 9

3 Running a Realtime Experiment 11
  3.1 Running the ProjectInterface ......................................... 11
    3.1.1 Running ProjectInterface in the Cloud ........................ 11
    3.1.2 Running ProjectInterface Locally ................................ 12
  3.2 Connecting with the Browser ......................................... 12
    3.2.1 SSL Certificate Installation ...................................... 12
      3.2.1.1 Install the SSL Certificate in your Web Browser .... 12
    3.2.2 Open Web Page in Browser ....................................... 14
    3.2.3 Notes on Security ................................................... 14
  3.3 Using VNC Server to view an application’s display on the server ................................................. 15
    3.3.1 Install VNC Server on the projectServer Computer ...... 15
  3.4 Definitions ................................................................. 15

4 Making your project cloud enabled 17
  4.1 Project Code ............................................................. 17
    4.1.1 Initialization code .................................................. 17
    4.1.2 Retrieving DICOM Images from the Scanner Computer .... 18
    4.1.3 Send Classification Results for Subject Feedback .......... 18
    4.1.4 Update the User’s Webpage Display ............................ 18
    4.1.5 Read Files from the Console Computer (such as configuration files) ................................................. 18
    4.1.6 Load Project Configurations ...................................... 19
    4.1.7 Timeout Settings .................................................... 19
      4.1.7.1 Setting Global Timeouts: ................................. 20
4.1.7.2 Setting Per-Call Timeouts: .................................................. 20
4.2 Some Alternate Configurations For Your Experiment .......................... 20
  4.2.1 Running everything on the same computer .................................. 20
  4.2.2 Running the subjectService remotely, but read DICOM data from the disk of the projectInterface computer .......................................................... 20
  4.2.3 Reading both remote and local data ............................................. 20

5 Providing Feedback to Subjects ...................................................... 21
  5.1 Using jsPsych ................................................................. 21
    5.1.1 Running the Demo ....................................................... 21

6 Using the BIDS Data Standard with RT-Cloud ..................................... 23
  6.1 BIDS Introduction ......................................................... 23
    6.1.1 The BIDS Archive ..................................................... 23
    6.1.2 BIDS Apps ........................................................... 24
  6.2 Why Use BIDS with RT-Cloud ............................................... 24
    6.2.1 BIDS Apps ........................................................... 24
    6.2.2 Benefits of BIDS Data Standardization .................................. 24
  6.3 Adapting BIDS for use in Real-Time fMRI Experiments ....................... 24
  6.4 How to Incorporate BIDS into your RT-Cloud project .......................... 25
  6.5 Replaying Data from OpenNeuro .............................................. 26

7 Run Project in a Docker Container .................................................. 29
  7.1 Allocate a VM in the cloud .................................................. 29
    7.1.1 Some notes for Azure VM: ............................................. 29
  7.2 Install Docker .................................................................. 29
  7.3 Install rtcloud for Docker .................................................... 30
  7.4 Run rtcloud projectInterface ................................................. 30
  7.5 Alternate simpler calls using the run-docker.sh script ......................... 31
  7.6 Alternate methods using docker-compose ..................................... 31
  7.7 Docker Image with ANTs, FSL and C3D (brainiak/rtcloudxl) ................ 32
  7.8 Building Docker Images ..................................................... 32

8 API Reference ........................................................................ 33
  8.1 rtCommon ................................................................. 33
    8.1.1 Submodules .............................................................. 33
      8.1.1.1 rtCommon.addLogin ................................................. 33
      8.1.1.2 rtCommon.bidsArchive ............................................ 34
      8.1.1.3 rtCommon.bidsCommon ............................................ 40
      8.1.1.4 rtCommon.bidsIncremental ........................................ 46
      8.1.1.5 rtCommon.bidsInterface ............................................ 50
      8.1.1.6 rtCommon.bidsRun .................................................. 54
      8.1.1.7 rtCommon.certsUtils ............................................... 56
      8.1.1.8 rtCommon.checkDicomNiftiConversion ......................... 56
      8.1.1.9 rtCommon.clientInterface ......................................... 57
      8.1.1.10 rtCommon.dataInterface .......................................... 58
      8.1.1.11 rtCommon.dicomToBidsService .................................. 62
      8.1.1.12 rtCommon.errors .................................................. 62
      8.1.1.13 rtCommon.exampleInterface ..................................... 63
      8.1.1.14 rtCommon.exampleService ....................................... 64
      8.1.1.15 rtCommon.fileWatcher ........................................... 65
      8.1.1.16 rtCommon.imageHandling ......................................... 67
      8.1.1.17 rtCommon.openNeuro ............................................. 70
      8.1.1.18 rtCommon.openNeuroService .................................... 71
      8.1.1.19 rtCommon.projectServer .......................................... 71

ii
9 Indices and tables

Python Module Index

Index
Realtime fMRI Cloud Framework is an open-source software package that makes it easier to build and deploy real-time fMRI experiments. The framework provides a coordination hub between the experimenter’s script, a subject feedback script, the scanner data, and experiment control. It streams scanner data (in real-time) to an experimenter’s script and forwards the results for use in subject feedback (optionally using tools like PsychoPy, jsPsych, or PsychToolbox). It provides a web-based user interface that allows for starting and stopping runs, changing settings, and viewing output. It can be configured to run in the cloud, on a cluster, or in the control room. Development was initially funded by Intel Labs; the framework is under active development with funding from NIMH to further extend its capabilities, including support for standards such as BIDS and OpenNeuro.org.

2.1 How it works

There are four general components:

- **Data Server (ScannerDataService)**
  - Watches for new DICOM images written by the MRI scanner.
  - Sends the DICOM images to the projectInterface in the cloud.
  - Listens for requests from the cloud projectInterface to either read or write files (within restricted directories) on the scanner computer.

- **Project and Web Interface (ProjectServer)**
  - Runs in the cloud.
  - Provides a user interface to start/stop and configure a run.
  - Is the communication link between the scannerDataService and the project-specific experiment classification script that runs in the cloud.

- **Project Specific Classification Script**
  - Classification script specific to the fMRI study being done.
  - Is started / stopped by the projectInterface and runs in the cloud.
  - Waits for DICOM files to be retrieved by the scannerDataService, creates a data model, returns classification results via the subjectService or scannerDataService for feedback purposes.

- **Subject Feedback (SubjectService)**
  - Runs on the presentation computer.
– Listens for classification results sent from the projectServer which were generated by the project-specific classification script.

– Uses the classification results in combination with a feedback toolkit and script, such as using PsychoPy, JsPsych, or PsychToolbox, to provide feedback to the subject in the MRI scanner.

**Fig 1: Overview of Components**

**Fig2: Overview of Where Scripts Run**

A projectInterface is started on the cloud computer (or it can instead be run on a local or cluster computer). The pro-
The projectInterface has a web interface which a browser can connect to and allows configuring and starting a run. The web interface is configured so that the browser 'start button' starts the project specific script. Wherever the projectInterface is installed is where your project specific python script will also run. The projectInterface also serves as the intermediary for communication between the the scannerDataService (running in the control room), the project specific script (running in the cloud) and the subjectService for feedback (running on the presentation computer).

A scannerDataService is started on the scanner computer that can watch for files within specified directories. The scannerDataService connects to the projectInterface using a username and password to connect and login.

A subjectService is started on the presentation computer. The subjectService listens for classification results from the project-specific script running in the cloud. These results can then be used by a presentation script to provide feedback to the subject in the MRI scanner. The subjectService connects to the projectInterface using a username and password to connect and login.

### 2.2 Other Links

- Wrapping Your Experiment Script with the RealTime Framework
- Running a Realtime Experiment
- Providing Subject Feedback
- Using BIDS Data Format in RT-Cloud
- Run Project in a Docker Container

### 2.3 Installation

#### 2.3.1 Step 1: Install Mini-Conda and NodeJS

*On the cloud computer where processing will take place, do these steps*

1. Check if you have mini-conda already installed. In a terminal run conda -V
   - *Mac Specific:* Install Mini-Conda
   - *Linux Specific:* Install Mini-Conda
     - wget https://repo.continuum.io/miniconda/Miniconda3-latest-Linux-x86_64.sh
     - bash Miniconda3-latest-Linux-x86_64.sh -b
2. Check if you have Node.js and NPM installed. In a terminal run node -v and npm -v
   - *Update:* Node.js is now included in the conda environment which will be created in Step 5 below, ‘Create the conda environment’
   - *Mac Specific:* Install Node.js
     - Check if Homebrew ‘brew’ is installed run brew -h
       - *Install Homebrew* if it is not installed
     - Run brew update
     - Run brew install node
   - *Linux Specific* (CentOS): Install Node.js
     - sudo yum install epel-release
2.3.2 Step 2: Install Realtime ProjectInterface on cloud VM (All OS types)

On the cloud computer where processing will take place, do these steps

1. Pull the source code git clone https://github.com/brainiak/rt-cloud.git
2. cd rt-cloud/
3. Get the local ip address `<local_ip_addr>`
   * Mac Specific: Google “what’s my ip address”
   * Linux Specific: hostname -i
4. Make a private key and an ssl certificate or copy an existing one into the certs directory
   * mkdir certs; openssl genrsa -out certs/rtcloud_private.key 2048
   * bash scripts/make-sslcert.sh -ip [local_ip_addr]
5. Create the conda environment
   * conda env create -f environment.yml; conda env update -f environment-synthetic-data.yml
   * conda activate rtcloud
6. Install node module dependencies
   * cd web; npm install; cd ..
7. Create a user:
   * bash scripts/add-user.sh -u [new_username] -p [password]

2.3.3 Step 3: Install ScannerDataService and/or SubjectService on the Console and Presentation Computers (All OS Types)

On the console computer where DICOMs are written, do these steps

1. Repeat Step 1.1 above to install Mini-Conda
2. Clone the rt-cloud code git clone https://github.com/brainiak/rt-cloud.git
3. Copy the ssl certificate created in Step 2.4 above to this computer’s rt-cloud/certs directory
   * Copy rt-cloud/certs/rtcloud.crt and rt-cloud/certs/rtcloud_private.key from the cloud computer
   * Copy it into the rt-cloud/certs directory on the scannerDataService computer and subjectService computer

2.4 Testing the Sample Project

For the sample we will run all services (the projectInterface, scannerDataService, and subjectService) on the same computer. Follow the above installation steps; the three service will be run from the same installation directory on the same computer. In a production deployment the projectInterface would typically run in a cloud VM, the scannerDataService would run on the control room console computer, and the subjectService would run on the presentation computer.

Note: The –test option runs in test mode which doesn’t use SSL encryption and accepts a default username and password, both are ‘test’. Never run with the –test option in production.
1. Open a terminal
   • Start the projectInterface
     – conda activate rtcloud

2. Open another terminal
   • Start the dataScannerService
     – conda activate rtcloud
     – bash scripts/run-scannerDataService.sh -s localhost:8888 -d $PWD,/tmp –test

3. Open a third terminal to start the subjectService (where feedback is received)
   • Start the subjectService to receive subject feedback on the presentation computer
     – conda activate rtcloud
     – python rtCommon/subjectService.py -s localhost:8888 –test

   • If prompted for username and password enter: username ‘test’, password ‘test’

5. Alternate step 4 - Run the sample project from the command line (it will automatically connect to a local projectServer and receive data from the scannerDataService)
   • conda activate rtcloud
   • python projects/sample/sample.py

### 2.4.1 Using the Sample Project with synthetic data generated by BrainIAK

1. Alternate to step 1 above
   • Start the projectInterface using the syntheticDataSample project
     – conda activate rtcloud
     – bash scripts/run-projectInterface.sh -p syntheticDataSample –dataRemote –test

2. through 5. same as above.

### 2.5 Next Steps

1. Run the sample project without the –test options. This will require the following steps. See *Running a Realtime Experiment* for instructions on accomplishing these steps.
   • Add the SSL certificate rtcloud/certs/rtcloud.crt that was created in install step 2 above into your web browser.
   • Include the -ip [local_ip_addr] option when starting the projectInterface.
   • Include the -u [username] and -p [password] options when starting the scannerDataService. Use the username and password created in install step 2 above.
   • Navigate web browser to https://localhost:8888 for a SSL connection
     – i.e. instead of the non-SSL http:// address used above for testing
When prompted for login by Web browser use the username and password created in install step 2 above.

2. Install and run the projectInterface on a remote computer. Run the scannerDataService on the computer where the DICOMs are written.

3. Create your own project python script and wrap it with the real-time framework. See Wrapping Your Experiment Script with the RealTime Framework

2.6 Troubleshooting

1. Python module not found - make sure you have installed and activate the conda environment: 'conda activate rtdcloud'. See installation step 2.5 above.

2. Web page gives a blank blue screen - the web javascript bundle wasn’t built, ‘cd web; npm install; npm run build’. See installation step 2.6 above.

3. ScannerDataService or subjectService can’t connect to the projectInterface.
   a. Try specifying the ‘–test’ option to all components (projectInterface, scannerDataService, subjectService). This will disable ssl and allow login with a test user, username: test, password: test. The web page will now be at http://localhost:8888 (not https://)
   b. Make sure the projectInterface computer’s firewall has port 8888 open. Try using an ssh tunnel if in doubt, ‘ssh -N -L 8888:localhost:8888 [remote-computer]’
   c. Try running the scannerDataService on the same computer as the projectInterface to test the connection.
   d. Try using a different port, specify the ‘–port [new_port]’ option when starting the projectInterface and when starting the scannerDataService specify the appropriate port using -s [remote-computer]:[port].
   e. Make sure the ssl certificate and private key (rtcloud.crt and rtcloud_private.key) that were created on the projectInterface computer have been copied to the rtcloud/certs directory on the scannerDataService and subjectService computers.
   f. Make sure you have created a username and password using the ‘scripts/add-user.sh’ script.

4. ProjectInterface cannot find your experiment script. Make sure your script’s name matches the project directory. Or specify the ‘–mainScript [script-name]’ option when starting the projectInterface. In addition the ‘–initScript [init-script]’ and ‘–finalizeScript [finalize-script]’ options can be used to specify the session initialization and finalization scripts.

5. ProjectInterface or web page indicate ‘RemoteServie: DataService not connected’. This means you started the projectInterface using the –dataRemote option but that a scannerDataService has not established a connection to the projectInterface, so it cannot make remote requests for data. Similarly for ‘SubjectService not connected’ errors.

6. An error in your script. Try running your script without starting the projectInterface. The clientInterface() method called by your script will create an internal version of the data services if there is no projectInterface started on localhost. If you specify yesToPrompts=True when instantiating the clientInterface (ClientInterface(yesToPrompts=True)) it will automatically use local services if there is no projectInterface running.

7. A DICOM error is reported such as, “ValueError: The length of the pixel data in the dataset (287580 bytes) doesn’t match the expected length (294912 bytes). The dataset may be corrupted or there may be an issue with the pixel data handler”. This usually indicates that the DICOM file was read by the FileWatcher before the file was completely written. To handle this, adjust the ‘minFileSize’ parameter that is passed to dataInterface.initWatch() or dataInterface.initScannerStream(), see the projects/sample/sample.py for an example. The minFileSize indicates a lower bound file size (in bytes) below which the FileWatcher will continue waiting before reading a file. Set the minFileSize to slightly below the smallest DICOM file size expected.
2.7 Running the Automated Test Suite

1. Follow the installation instructions detailed above
2. Activate the conda environment
   - conda activate rtcloud
3. Additionally, install bids-validator
   - npm install -g bids-validator
4. Run the test suite
   - python -m pytest -s -v tests/

2.8 Further Reading

- Run Project in a Docker Container
- Details - coming soon
CHAPTER THREE

RUNNING A REALTIME EXPERIMENT

3.1 Running the ProjectInterface

The projectInterface is typically run on a VM in the cloud (i.e. a ‘remote’ computer) which does not have direct access to the DICOM images. The advantage of a cloud VM is that any laptop browser can connect to it and no additional hardware or software installation is needed on the control room computer. However the projectInterface can also be run on a ‘local’ computer, meaning on the same computer in the control room where the DICOM images are written.

3.1.1 Running ProjectInterface in the Cloud

1) Start the ProjectInterface

From the cloud VM computer run the following command. See Definitions Section for description of the parameters.

```bash
cd rtcloud/
conda activate rtcloud
```

Example:

```bash
bash scripts/run-projectInterface.sh -p sample -c projects/sample/conf/sample.toml -ip 125.130.21.34 --dataRemote --subjectRemote
```

The -p option is used to locate your project in the /rt-cloud/projects/ directory, the name specified should match your project directory name.

The -c option points to your project configuration file in toml format.

The -ip option is to update the ssl certificate with the ip address where the projectInterface runs. Use ‘hostname -i’ or Google ‘what’s my ip address’ to get the ip address of that computer.

2) Start the scannerDataService

The scannerDataService is started on the control room computer where the DICOM images are written by the scanner. It can forward those images to the projectInterface when requested by your project code. The [username] and [password] are the login credentials to the projectInterface because the scannerDataService must connect to the projectInterface to be able to serve files to it.

```bash
```

Example:
3) **Start the SubjectService** The subjectService is started on the presentation computer where PsychoPy or similar software will run to provide feedback to the subject in the MRI scanner. The [username] and [password] are the login credentials to the projectInterface because the subjectService must connect to the projectInterface to be able to receive classification results.

```bash
```

Example (run from the rt-cloud directory):

```bash
bash scripts/run-subjectService.sh -s 125.130.21.34:8888 -u user1 -p passwd1
```

### 3.1.2 Running ProjectInterface Locally

The projectInterface can also be run on the control room computer where the DICOM images are written. This is called running it ‘locally’. When run locally the fileServer (scannerDataService) is not needed because the projectInterface can directly read the DICOM images from disk.

1) **Start the projectInterface**: same command as above but without the –dataRemote or –subjectRemote options

```bash
bash scripts/run-projectInterface.sh -p [your_project_name] -c [config_file] -ip [local_ip_addr]
```

Example:

```bash
bash scripts/run-projectInterface.sh -p sample -c projects/sample/conf/sample.toml -ip 125.130.21.34
```

### 3.2 Connecting with the Browser

#### 3.2.1 SSL Certificate Installation

The connection between your web browser and the projectInterface is encrypted using SSL for security. In order for your browser to trust the connection to the projectInterface, the SSL certificate created during the projectInterface installation process must be added to a list of trusted certificates on your browser computer.

Copy the ssl certificate `rtcloud/certs/rtcloud.crt` to your computer running the web browser.

#### 3.2.1.1 Install the SSL Certificate in your Web Browser

**On Mac:**

1. Open application ‘Keychain Access’.
2. Click on ‘Certificates’ in bottom left pane
3. Select File->Import Items... and select the ssl certificate downloaded to your computer
4. In ‘Certificates’ pane, double-click the ‘rtcloud.princeton.edu’ certificate
5. Select the ‘Trust’ drop-down item.

6. In ‘When using the certificate’ selector choose ‘Always Trust’

**On Linux:**

1. Navigate to the projectInterface web URL, e.g. https://localhost:8888

   - You will see a security warning about untrusted certificate
   - Click ‘Add Exception’
• You will see a dialog box
• Click ‘Confirm Security Exception’

3.2.2 Open Web Page in Browser

1. Open a browser and navigate to https://<projectInterface_addr>:8888
2. On the login screen enter the [username] and [password] that were created during installation by the adduser.sh script

3.2.3 Notes on Security

There are several security mechanisms

• Encryption - Using SSL connections means all data is encrypted in transit.
• Password - A username and password mean that only authorized users can connect to the projectInterface.
• Restricted directories - The files server restricts which directories it will return files from.
• Restricted file types - The files server restricts which file types (denoted by the file extension, e.g. .dcm) it will return.
• Direction of connection - The files server doesn’t allow connections to it. The files server always initiates the connection going to the projectInterface.
3.3 Using VNC Server to view an application's display on the server

Sometimes it is necessary to see the GUI (graphical window) output of a program running on the projectServer computer. For example, during session initialization a researcher may want to run FSLview to see if the image registration looks correct. This can be done by running a VNC Server on the computer where the projectServer is running. The VNC Server starts a second virtual display (:1) and any application can then render to that display. The web interface has a tab called ‘VNC Viewer’ which can view and interact with the content on that display. Note: the VNC Server uses a program called websockify which can wrap any TCP/IP program so that it can be accessed from a websocket client, websockify wraps VNC Server for this purpose.

3.3.1 Install VNC Server on the projectServer Computer

1. Install the websockify conda environment:

   ```bash
   conda env create -f websockify.yml
   ```

2. Install VNC on the server:

   ```bash
   sudo yum -y install tigervnc-server
   sudo yum -y install xclock
   sudo yum -y install xdotool
   cat <<EOT >> ~/.vnc/xstartup
   unset SESSION_MANAGER
   unset DBUS_SESSION_BUS_ADDRESS
   #exec /etc/X11/xinit/xinitrc
   xsetroot -solid grey -cursor_name left_ptr
   xeyes
   EOT
   ```

3. Start the VNC Server running with websockify to serve connection requests:

   ```bash
   bash scripts/run-vnc.sh
   ```

4. Applications started on the projectServer, such as from the session initialization script, can redirect their output to the VNC display by prepending “DISPLAY=:1” to the command. For example: DISPLAY=:1 xclock

5. The VNC display can be viewed from the VNC Viewer tab of the web interface. Click ‘reconnect’ if needed to re-initialize the VNC connection.

3.4 Definitions

- **[allowed_dirs]** - This allows restricting which directories the fileServer is allowed to return files from. Specify as a comma separated list with no spaces, e.g. ‘-d /tmp,/data,/home/username’

- **[allowed_file_extensions]** - This allows restricting which file types the fileServer is allowed to return. Specify as a comma separated list with no spaces, e.g. ‘-f .dcm,.txt,.mat’

- **[config_file]** - the location of project specific configurations in a toml file format. This will typically be located in a directory within the project such as rtcloud/projects/your_project/conf/. E.g. the sample project config file is rtcloud/projects/sample/conf/sample.toml

- **[local_ip_addr]** - network address of the computer where a command is issued
- Get the local ip address
  * Mac: Google “what’s my ip address”
  * Linux: hostname -i

- [username] [password] - The username and password to login to the projectInterface. This was created with the adduser.sh script during installation of the projectInterface.

- [projectInterface_addr:port] - The network address and port number that the projectInterface is listening on. The default port is 8888. E.g. `-s 125.130.21.34:8888`

- [your_project_name] - The name of the subdirectory under the rtcloud/projects/ directory which contains your project specific code. Your script should use the same name as the directory, i.e. sample.py, so that the projectInterface can find it.

- [run] - An fMRI scanner acquisition block of images. For example running the scanner to collect a block of 200 scans with a TR image repetition time of 2 seconds; this run will take 400 seconds and generate 200 DICOM images.

- [scan] - The file sequence number corresponding to a run. For example, in the image name ‘001_000014_000005.dcm’, the scan number is 14 and the image volume number (TR id) is 5.
MAKING YOUR PROJECT CLOUD ENABLED

Make a new directory under rt-cloud/projects for your project. Use the sample project in rt-cloud/projects/sample as a template for making your python script cloud enabled. The sample.py script corresponds to the script you will make for your experiment.

4.1 Project Code

You’ll need to copy several blocks of code to your project to get it cloud enabled. These are:

4.1.1 Initialization code

Accept at least the following command line parameters in your project python file:

```python
argParser = argparse.ArgumentParser()
argParser.add_argument('--config', '-c', default=defaultConfig, type=str,
    help='experiment config file (.json or .toml)')
argParser.add_argument('--runs', '-r', default='', type=str,
    help='Comma separated list of run numbers')
argParser.add_argument('--scans', '-s', default='', type=str,
    help='Comma separated list of scan number')
args = argParser.parse_args()
```

Create an clientInterface instance for communicating with the projectInterface. The clientInterface automatically connects to a localhost projectInterface when created.

```python
clientInterface = ClientInterface()
```

The clientInterface provides several interfaces for retrieving data, giving subject feedback, and updating the user's webpage.

```python
dataInterface = clientInterfaces.dataInterface
subjInterface = clientInterfaces.subjInterface
webInterface = clientInterfaces.webInterface
```

Note: The clientInterfaces connect to remote services with the following mapping:

```python
dataInterface --> scannerDataService
subjInterface --> subjectService
webInterface --> user web browser
```
4.1.2 Retrieving DICOM Images from the Scanner Computer

Within your python script, use the dataInterface object to request remote files. For example, to retrieve DICOM images as they are created, init a watch on the appropriate directory and then watch for them.

```python
dataInterface.initWatch('/tmp/dicoms', 'samp*.dcm', minFileSize)
rawData = dataInterface.watchFile('/tmp/samp3.dcm')
```

Or use the readRetryDicom helper function which will retry several times across timeouts to retrieve the DICOM image data:

```python
dataInterface.initWatch('/tmp/dicoms', 'samp*.dcm', minFileSize)
dicomData = readRetryDicomFromDataInterface(dataInterface, 'samp3.dcm', timeout=10)
```

Or use the streaming interface to receive image data:

```python
streamId = dataInterface.initScannerStream('/tmp/dicoms', 'samp*.dcm', minFileSize)
dicomData = dataInterface.getImageData(streamId, int(this_TR), timeout=10)
```

Set the minFileSize parameter to the minimum size expected for DICOM files (in bytes). This can be determined by listing the sizes of a set of previously collected DICOM files and selecting slightly less than the smallest as the minimumFileSize. The FileWatcher will not return a file until its minimum size has been reached, this helps ensure that a file is completely written before being made available. However, if this parameter is set too high (higher than the file size) the file will never be returned.

4.1.3 Send Classification Results for Subject Feedback

Send classification results to the presentation computer using the subjectInterface setResult() command:

```python
subjInterface.setResult(runNum, int(TR_id), float(classification_result))
```

Or send classification results to a file on the scanner computer (where scannerDataService is running) which can be read in by a script (e.g. using a toolkit like PsychToolbox) for subject feedback.

```python
dataInterface.putFile(fullpath_filename_to_save, text_to_save)
```

4.1.4 Update the User’s Webpage Display

Send data values to be graphed in the projectInterface web page

```python
webInterface.plotDataPoint(runNum, int(TR_id), float(classification_result))
```

4.1.5 Read Files from the Console Computer (such as configuration files)

Read files from the console computer using getFile

```python
data = dataInterface.getFile(fullpath_filename)
```

Or read the newest file matching a file pattern such as 'samp*.dcm'

```python
data = dataInterface.getNewestFile(fullpath_filepattern)
```
4.1.6 Load Project Configurations

 RT-Cloud experiments use a TOML file for configuration settings. You can define your own configuration variables just by adding them to the TOML configuration file. Your configuration variables will automatically appear in the web interface ‘settings’ tab and you can adjust the values from that page.

 Use the loadConfigFile function from your experiment script to load your configurations into a structured object

```python
import rtCommon.utils as utils
cfg = utils.loadConfigFile(args.config)
```

Access configurations within your experiment script using the config structure

```python
print(cfg.subjectName, cfg.run)
```

The following fields must be present in the config toml file for the projectInterface to work:

- `runNum = [1]` # an array with one or more run numbers e.g. [1, 2, 3]
- `scanNum = [11]` # an array with one or more scan numbers e.g. [11, 13, 15]
- `subjectName = 'subject01'`

Optional parameters used for plotting:

- `title = 'Project Title'`
- `plotTitle = 'Plot Title'`
- `plotXLabel = 'Sample #'`
- `plotYLabel = 'Value'`
- `plotXRangeLow = 0`
- `plotXRangeHigh = 20`
- `plotYRangeLow = -1`
- `plotYRangeHigh = 1`

Additionally, create any of your own unique parameters that you may need for your experiment.

4.1.7 Timeout Settings

 RT-Cloud uses RPC (Remote Procedure Calls) to send command requests from the researcher’s experiment script to the dataInterface, subjectInterface and webInterface. There are two RPC hops to handle a request. The first uses RPyC (a native Python RPC library) to make a call from the script to the projectServer. The second is using a WebSocket RPC implemented in rtCommon/remoteable.py and invoked from rtCommon/projectServerRPC.py to make the call from the projectServer to the remote service (such as DataService). For each hop a global timeout can be set, and a per-call timeout can also be set.
4.1.7.1 Setting Global Timeouts:

- The RPyC global timeout can be set when the ClientInterface is created in the experiment script, as demonstrated in the sample.py project. Simply include the rpyc_timeout= parameter (e.g. ClientInterface(rpyc_timeout=10) for a 10 second timeout). The default is 120 seconds.

- The Websocket RPC global timeout can be set using the setRPCTimeout() of interface objects (i.e. remote-able objects). For example to increase the timeout of the dataInterface in the experiment script, call dataInterface.setRPCTimeout(10) for a 10 second timeout. The default websocket timeout is 60 seconds.

4.1.7.2 Setting Per-Call Timeouts:

- Per-call timeouts for both RPyC and Websocket RPC are set together using the same parameter. To set a larger timeout for a specific call, include a “rpc_timeout” kwarg in that calls parameters. For example, use dataInterface.getFile(“bigfile.bin”, rpc_timeout=60) to set a 60 second timeout for a large file transfer. Note that before setting an RCP timeout you should check that the interface is connected to the ProjectServer, because sometimes interfaces will run locally. To check that, use an interface’s .isUsingProjectServer() command, such as dataInterface.isUsingProjectServer(), see the openNeuroClient project for an example of this usage.

4.2 Some Alternate Configurations For Your Experiment

4.2.1 Running everything on the same computer

Start the projectInterface without the –dataRemote or –subjectRemote options. No need to start any other services, local versions of them will be created internally by the projectInterface.

```
bash scripts/run-projectInterface.sh -p [your_project_name]
```

4.2.2 Running the subjectService remotely, but read DICOM data from the disk of the projectInterface computer

Start the projectInterface only specifying –subjectRemote. Then start a subjectService on a different computer that will connect to the projectInterface. The scannerDataService will automatically be created internally by the projectInterface to read data from the projectInterface computer.

```
bash scripts/run-projectInterface.sh -p [your_project_name] --subjectRemote
```

4.2.3 Reading both remote and local data

Start the projectInterface with the –dataRemote option and connect to it with a scannerDataService from another computer. In addition create another instance of dataInterface() within your script specifying dataRemote=False. This second dataInterface can watch for DICOM files created locally and the remote dataInterface can get/put files to the remote scannerDataInterface (for example to write a text file with the classification results for use by PsychToolbox).

```
dataInterface2 = DataInterface(dataRemote=False, allowedDirs=['*'], allowedFileTypes=['* →'])
```
Ideally we would like to provide feedback to the subject in the MRI scanner via a web interface. This would allow the researcher to open a web browser and move the browser onto a monitor visible by the subject in the scanner. One convenient toolbox for doing this is jsPsych. We have integrated jsPsych into our project and provide a demo using the DecNef style colored circle feedback.

5.1 Using jsPsych

The source code components of jsPsych live in the web/ directory. File web/jsPsychFeedback.html is the main file that will be edited to adjust the type of feedback displayed. The creating a new draw method different types of feedback can be created.

5.1.1 Running the Demo

1. The projectServer must be started with --remoteSubject options enabled to allow the feedback webpage to connect and receive results from the projectServer.
   - conda activate rtcloud
   - bash ./scripts/run-projectInterface.sh --test-p sample --subjectRemote

2. Connect a web browser to the main page
   - http://localhost:8888/
   - Enter ‘test’ for both the username and password since we are running it in unsecure test mode.

3. Connect a web browser to the jsPsych feedback page
   - http://localhost:8888/jspsych

4. Click the ‘Run’ button on the main page and view the subject feedback shown on the jsPsych page
Note: Some of this documentation is taken from Polcyn, S. (2021) “Efficient Data Structures for Integrating the Brain Imaging Data Structure with RT-Cloud, a Real-Time fMRI Cloud Platform” [Unpublished senior thesis].

6.1 BIDS Introduction

BIDS is the leading data standard for neuroscience data and is supported by a wide variety of data formatting and analysis tools. It is the standard used by OpenNeuro which is a large and growing repository of neuroscience datasets. In addition there are a large set of BIDS Apps, which are container-based applications with a standardized interface that work on BIDS-formatted data. The BIDS Validator is an automated and comprehensive validation tool that analyzes datasets and identifies BIDS compliance issues.

6.1.1 The BIDS Archive

The BIDS standard defines the on-disk layout and format of datasets to form a BIDS archive. A BIDS archive is a collection of brain activity image and metadata files for one study, which may comprise multiple subjects across multiple days. While an in-depth understanding of the BIDS standard can be obtained from the full standard, viewable online at https://bids-specification.readthedocs.io/en/stable/, a few key details are as follows:

1. **Brain imaging data is stored in the Neuroimaging Informatics Technology Initiative (NIfTI) format.** NIfTI is a binary file format that starts with a header holding basic information about the brain data contained in the file. The header is followed by the raw brain data. A NIfTI volume’s data typically has 4 dimensions, $x$, $y$, $z$ and $t$ (time), so a NIfTI file can be thought of as containing a sequence of $t$, 3-D images, each of which has dimensions $x \times y \times z$.

2. **Metadata is stored in files separate from the image data.** Unlike the DICOM image format, the NIfTI image format doesn’t store much metadata about the image it contains. Accordingly, in the BIDS data format, the majority of the metadata about the image and the conditions under which it was collected is stored in separate files, typically in the JavaScript Object Notation (JSON) or Tab-Separated Value (TSV) format.

3. **Files are named using BIDS entities.** The name of a file in a BIDS archive follows a standard format, and it is composed of a set of ‘entities’ (like ‘sub’ or ‘run’, corresponding to ‘subject’ and ‘run’, respectively) that signify what the data in the file corresponds to. For example, the filename “sub-01_task-language_run-1_bold.nii” has 4 BIDS entities, separated by underscores (“_”). The 4 entities and their corresponding values are:
   1. ‘sub’: 01 (this file has data for the subject with ID 01)
   2. ‘run’: 1 (this file has data from the 1st run)
   3. ‘task’: language (this file has data from the ‘language’ task)
   4. ‘bold’: No value (The presence of the entity is enough to state the file holds fMRI brain-oxygen-level-dependent (BOLD) data)
In summary, a BIDS archive is a collection of image and metadata files, all named using BIDS entities that correspond to the conditions under which the data or metadata was collected.

### 6.1.2 BIDS Apps

BIDS Apps are containerized applications that operate on BIDS datasets and provide a consistent command-line interface to users. Since each app operates on a BIDS archive, a full analysis pipeline can potentially be created from independent BIDS App containers, so components can be easily added, removed, or modified as needs evolve over time.

### 6.2 Why Use BIDS with RT-Cloud

#### 6.2.1 BIDS Apps

Using BIDS with RT-Cloud connects you to the BIDS Apps ecosystem, so you can integrate existing and future BIDS Apps with your real-time fMRI analysis pipeline, minimizing time spent on setting up computational infrastructure.

#### 6.2.2 Benefits of BIDS Data Standardization

Storing data in a standardized format brings a host of benefits, the following of which were adapted from here.

One major benefit is you and all lab members or clinical team members, once having learned the standard, know immediately how to navigate both new and old datasets. Without a standardized format, different team members may format their data in different ways, forcing you to waste time learning a myriad of data formats and creating significant problems when a team member leaves the organization and can no longer explain to new or existing team members how their dataset format works. Additionally, external collaborators at other institutions can easily work on your dataset if everyone uses the same standard.

Another major benefit is future software packages are likely to grow around this standard. Thus, you can use any of a wide variety of software packages with your new and existing BIDS datasets that conform to the standard and not spend time learning additional software-specific formats or be locked-in to a particular software package.

Finally, if you are required to publish your datasets as a condition of manuscript publication, having data in a standardized format from the beginning enables a seamless upload and review process.

### 6.3 Adapting BIDS for use in Real-Time fMRI Experiments

Real-time fMRI experiments involve processing image data as it arrives from the scanner and providing immediate subject feedback. In essence, rt-fMRI is a streaming model, whereas BIDS is a data-at-rest standard. To adopt BIDS for rt-fMRI we introduce a new idea, the BIDS Incremental.

A BIDS Incremental packages one brain volume into its own BIDS archive. Thus, we can use this to send a stream of very small BIDS archives (i.e., BIDS Increments) for processing. This allows the processing to be done by any application that can ingest BIDS data, such as BIDS-Apps.
6.4 How to Incorporate BIDS into your RT-Cloud project

There are three primary classes to use to leverage BIDS in your RT-Cloud project: BIDS Incremental, BIDS Run, and BIDS Archive.

1) BIDS Incremental is a single-image data structure, encapsulating a single-volume BIDS Archive.

2) BIDS Run is a data structure that efficiently stores a full run’s worth of BIDS Incrementals in-memory and in a deduplicated fashion. It supports appending BIDS Incrementals to a scanning run and retrieving BIDS Incrementals that have already been added.

3) BIDS Archive is a data structure that provides an API for interacting with on-disk BIDS archives and enables efficient movement between the BIDS Run streaming data structure and the on-disk BIDS archive.

Below is a sample of how your project can receive real-time scanner data in BIDS-incremental format. This assumes you are running the scannerDataService in the control room. This example communicates with the scannerDataService via the clientInterface.bidsInterface. A data stream is initialized, giving the scanner directory that the DICOMs will arrive in and the DICOM filename pattern to watch for.

```python
from rtCommon.clientInterface import ClientInterface
# connect to the remote data service (via projectServer on localhost)
clientInterfaces = ClientInterface()
bidsInterface = clientInterfaces.bidsInterface
# specify the BIDS entities for the run being done
entities = {
    'subject': cfg.subjectName,
    'run': cfg.runNum[0],
    'suffix': 'bold',
    'datatype': 'func',
}
# initialize the stream which will watch for DICOMs created at the scanner
# and then convert them to BIDS-incrementals and stream them to this script.
streamId = bidsInterface.initDicomBidsStream(cfg.dicomDir,
                                            cfg.dicomScanNamePattern,
                                            cfg.minExpectedDicomSize,
                                            **entities)

# loop over the expected number of DICOMs per run
for idx in range(scansPerRun):
    bidsIncremental = bidsInterface.getIncremental(streamId, idx)
    imageData = bidsIncremental.imageData
    avg_niftiData = numpy.mean(imageData)
    if cfg.writeBidsArchive is True:
        # See openNeuroClient project under 'projects' directory for more
        # information on accumulating a BIDS archive from a stream of incrementals.
        newRun.appendIncremental(bidsIncremental)

Below is a simple example that shows the interactions between the various classes.

```python
archive = BidsArchive('/tmp/bidsDataset')
print('Subjects:', archive.getSubjects(), 'Runs:', archive.getRuns())

# Query the run using BIDS Entities (see the tutorial for a deeper introduction)
run = archive.getBidsRun(subject='01', run=1, datatype='func')
newRun = BidsRun()
meanActivationValues = []
```

(continues on next page)
for i in range(run.numIncrementals()):
    incremental = run.getIncremental(i)
    meanActivationValues.append(np.mean(incremental.imageData))
    newRun.appendIncremental(incremental)

newArchive = BidsArchive('~/tmp/newBidsDataset')
newArchive.appendBidsRun(newRun)

An overview of how these classes all fit together for sending data from the MRI scanner to a BIDS Archive is shown here:

Retrieving BIDS data from an archive is simply the reverse of that diagram.

For a more in-depth introduction to the various classes and how to use them, check out the bids_tutorial Jupyter notebook tutorials/bids_tutorial.ipynb.

6.5 Replaying Data from OpenNeuro

One goal of this project is to facilitate collaboration and sharing of code and data. To this end we introduce an OpenNeuro module which can access and stream data from the OpenNeuro.org data repository. In essence this is a ‘NetFlix’ type service for fMRI datasets. Researchers can replay datasets through their processing pipelines to try new models, reproduce results or test and debug experiments.

An example of streaming OpenNeuro data can be seen in the projects/openNeuroClient sample project. The key snippets of code are shown below.

```python
# OpenNeuro accession number for a dataset
dsAccession = 'ds002338'
# The subject and run number to replay
templates = {'subject': 'xp201', 'run': 1}
# Initialize the data stream
```
streamId = bidsInterface.initOpenNeuroStream(dsAccession, **entities)
numVols = bidsInterface.getNumVolumes(streamId)
# Retrieve and process each volume as a BIDS-Incremental
for idx in range(numVols):
    bidsIncremental = bidsInterface.getIncremental(streamId, idx)
    imageData = bidsIncremental.imageData
CHAPTER SEVEN

RUN PROJECT IN A DOCKER CONTAINER

7.1 Allocate a VM in the cloud

If you will run the project on the cloud, the following instructions will help deploy a cloud VM. If you will run the project on local resources skip to the 'Install Docker' section.

- Amazon AWS instructions: https://aws.amazon.com/getting-started/tutorials/launch-a-virtual-machine/

7.1.1 Some notes for Azure VM:

- Choose CentOS 7.5 (which the following instructions are based on)
- Create a resource group ‘rtcloud’ to make it easier to track later
- Choose VM instance type F4s_v2, F8s_v2, or F16s_v2 (depending on number of cores desired)
- Choose premium SSD disk, no need for an extra data disk, but we will extend the main disk to 60 GB after VM creation.
- NIC network security group - choose ‘Advanced’ to create a network security group. This will allow you later to configure port 8888 as allowed for traffic.
- Choose Auto-shutdown and set a time during the night (in case you forget to power down)

7.2 Install Docker

Install Docker Engine

```
sudo yum install -y yum-utils device-mapper-persistent-data lvm2
sudo yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo
sudo yum install -y docker-ce docker-ce-cli containerd.io docker-compose
```

Add your username to the docker group (to avoid using sudo for docker commands)

```
sudo usermod -aG docker <username>
nnewgrp docker
```

Config Docker to start at boot time
sudo systemctl enable docker
sudo systemctl start docker

test docker
docker run hello-world

7.3 Install rtcloud for Docker

Pull rtcloud image
docker pull brainiak/rtcloud:latest

Add the rtgroup Add a new group with GID 5454 to your local system which matches the user and group ID used in the rtcloud Docker container. Add your username to be a member of the rtgroup.
sudo groupadd -g 5454 rtgroup
sudo usermod -a -G rtgroup <your-username>
sudo chgrp -R rtgroup <projects-dir>

Create the rtcloud ssl certificate This will create a self-signed SSL certificate called rtcloud.crt to allow encrypted communication with the projectInterface. You will need to install the rtcloud.crt certificate in your browser for trusted communication. The certificate will be created in location: /var/lib/docker/volumes/certs/_data/rtcloud.crt
IP=`curl https://ifconfig.co/`
docker run -it --rm -v certs:/rt-cloud/certs brainiak/rtcloud:latest scripts/make-sslcert.sh -ip $IP

Add a user for web interface The web connection to the projectInterface requires a user/password to authenticate. You can create a username and password with this command.
docker run -it --rm -v certs:/rt-cloud/certs brainiak/rtcloud:latest scripts/add-user.sh -u <username>

7.4 Run rtcloud projectInterface

The above installation only needs to be run once, then the projectInterface can be started whenever needed with these commands.
IP=`curl https://ifconfig.co/`
PROJ_DIR=<full_path_to_project_dir>
PROJ_NAME=<name>
### 7.5 Alternate simpler calls using the run-docker.sh script

The rt-cloud github repo has a run-docker.sh script that encapsulates the docker specific call parameters in the above calls. This can make it simpler to call the functions you want within the docker image. The following shows the previous commands using the run-docker.sh helper script. Set the PROJ_DIR env variable before calling run-docker.sh so it can map the project directory into the docker container.

```
export PROJ_DIR=[path-to-your-local-project]
scripts/run-docker.sh scripts/make-sslcert.sh -ip $IP
scripts/run-docker.sh scripts/add-user.sh -u <username>
scripts/run-docker.sh scripts/run-projectInterface.sh -p sample -c projects/sample/conf/
    sample.toml -ip $IP
```

Or use the --projDir parameter to specify the project directory to map.

```
scripts/run-docker.sh --projDir [path-to-project] scripts/run-projectInterface.sh -p
    sample -c projects/sample/conf/sample.toml -ip $IP
```

### 7.6 Alternate methods using docker-compose

Docker compose can be used to start a container running with all the appropriate directories and ports mapped, making it easier to issue calls (i.e. run commands) in a continuously running container.

The docker compose file is located at: rt-cloud/docker/docker-compose.yml. Edit the docker-compose.yml file and replace /tmp/myproject with the path to your project, and update the internal container mount point by replacing ‘myproject’ in /rt-cloud/projects/myproject with your project directory name.

Then start the docker compose container running docker-compose up.

```
docker-compose -f docker/docker-compose.yml up &
```

Stop the docker compose container by running docker-compose down

```
docker-compose -f docker/docker-compose.yml down
```

The running container will be named rtserver. You can then issue commands to the running container such as:

```
docker exec -it rtserver ls /rt-cloud/projects

docker exec -it rtserver scripts/run-projectInterface.sh -p myproject -c /rt-cloud/
    projects/myproject/config.toml --test
```

This makes it easier to run commands without specifying volumes and ports to map each time, and is more efficient as it uses a running container rather than starting a new container for each command.
7.7 Docker Image with ANTs, FSL and C3D (brainiak/rtcloudxl)

There is a version of the rtcloud docker image that also has ANTs, FSL and C3D installed in the image along with the RT-Cloud framework. It is available as brainiak/rtcloudxl:[release-tag], such as brainiak/rtcloudxl:1.3. This container is significantly larger (about 30 GB uncompressed) than the basic rtcloud image, and so is not listed as the default release of the image.

7.8 Building Docker Images

The dockerfiles needed to build the images are in the rt-cloud/docker directory. The commands to build the images are as follows:

```
docker build -t brainiak/rtcloud:latest -f docker/Dockerfile.rtcloud .
docker build -t brainiak/rtcloudxl:latest -f docker/Dockerfile.rtcloudXL .
```

And to re-tag them, such as for a release:

```
docker tag brainiak/rtcloud:latest brainiak/rtcloud:1.3
```
This page contains auto-generated API reference documentation.

8.1 rtCommon

8.1.1 Submodules

8.1.1.1 rtCommon.addLogin

A command-line script to add or change a user/password for access to the web portal. The password file is stored in rt-cloud/certs/passwd

Examples

$ python addLogin.py # username and password will be requested at prompt $ python addLogin.py -u <username> -p <password> $ python addLogin.py -username <username> -password <password>

Module Contents

Functions

```
addUserPassword(username, password, pwdFile, re-typePasswd=True)
main(username, password)
```

1 Created with sphinx-autoapi
rtCloud

Attributes

currPath

rootPath

passwordFile

argParser

rtCommon.addLogin.currPath
rtCommon.addLogin.rootPath
rtCommon.addLogin.passwordFile = certs/passwd
rtCommon.addLogin addUserPassword(username, password, pwdFile, retypePasswd=True)
rtCommon.addLogin.main(username, password)
rtCommon.addLogin.argParser

8.1.1.2 rtCommon.bidsArchive

bidsArchive.py
Implements interacting with an on-disk BIDS Archive.

Module Contents

Classes

BidsArchive

Functions

failIfEmpty(func)
rtCloud

Attributes

_logger

rtCommon.bidsArchive._logger

rtCommon.bidsArchive.failIfEmpty(func)

class rtCommon.bidsArchive.BidsArchive(rootPath: str)

__str__(self)

Return str(self).

__getattr__(self, attr)

static _stripLeadingSlash(path: str) → str

Strips a leading / from the path, if it exists. This prevents paths defined relative to dataset root (/sub-01/ses-01) from being interpreted as being relative to the root of the filesystem.

Parameters path – Path to strip leading slash from.

Examples

>>> path = '/sub-01/sub-01/func/sub-01_task-test_bold.nii.gz'
>>> BidsArchive._stripLeadingSlash(path)
'sub-01/sub-01/func/sub-01_task-test_bold.nii.gz'
>>> path = '/sub-01/sub-01/func/sub-01_task-test_bold.nii.gz'
'sub-01/sub-01/func/sub-01_task-test_bold.nii.gz'

absPathFromRelPath(self, relPath: str) → str

Makes an absolute path from the relative path within the dataset.

tryGetFile(self, path: str) → bids.layout.BIDSFile

Tries to get a file from the archive using different interpretations of the target path. Interpretations considered are: 1) Path with leading slash, relative to filesystem root 2) Path with leading slash, relative to archive root 3) Path with no leading slash, assume relative to archive root

Parameters path – Path to the file to attempt to get.

Returns BIDSFile (or subclass) if a matching file was found, None otherwise.

Examples

>>> archive = BidsArchive('/path/to/archive')
>>> filename = 'sub-01_task-test_bold.nii.gz'
>>> archive.tryGetFile('/tmp/archive/sub-01/func/' + filename)
<BIDSImageFile filename=/tmp/archive/sub-01/func/sub-01_task-test_bold.nii.gz>
>>> archive.tryGetFile('/') + filename)
<BIDSImageFile filename=/tmp/archive/sub-01/func/sub-01_task-test_bold.nii.gz>

(continues on next page)
>> archive.tryGetFile(filename)
<BIDSImageFile filename=/tmp/archive/sub-01/func/sub-01_task-test_bold.nii.gz

**dirExistsInArchive**(self, relPath: str) → bool

**getReadme**(self) → bids.layout.BIDSFile

**getImages**(self, matchExact: bool = False, **entities) → List[bids.layout.BIDSImageFile]
Return all images that have the provided entities. If no entities are provided, then all images are returned.

**Parameters**

- **matchExact** – Only return images that have exactly the provided entities, no more and no less.
- **entities** – Entities that returned images must have.

**Returns** A list of images matching the provided entities (empty if there are no matches, and containing at most a single image if an exact match is requested).

**Examples**

```python
>>> archive = BidsArchive('/path/to/archive')

Using a dictionary to provide target entities.

```python
>>> entityDict = {'subject': '01', 'datatype': 'func'}
>>> images = archive.getImages(**entityDict)
```

Using keyword arguments to provide target entities.

```python
>>> images = archive.getImages(subject='01', datatype='func')
```

Accessing properties of the image.

```python
>>> image = images[0]
>>> print(image.get_image())
(64, 64, 27, 3)
>>> print(image.path)
/tmp/archive/func/sub-01_task-test_bold.nii
>>> print(image.filename)
sub-01_task-test_bold.nii
```

An exact match must have exactly the same entities; since images must also have the task entity in their filename, the above entityDict will yield no exact matches in the archive.

```python
>>> images = archive.getImages(entityDict, matchExact=True)
ERROR 'No images were an exact match for: {'subject': '01', 'datatype': 'func'}'
>>> print(len(images))
0
```
_updateLayout(self)

Updates the layout of the dataset so that any new metadata or image files are added to the index.

_addImage(self, img: nibabel.Nifti1Image, path: str, updateLayout: bool = True) → None

Replace the image in the dataset at the provided path, creating the path if it does not exist.

Parameters

- **img** – The image to add to the archive
- **path** – Relative path in archive at which to add image
- **updateLayout** – Update the underlying layout object upon conclusion of the image addition.

_addMetadata(self, metadata: dict, path: str, updateLayout: bool = True) → None

Replace the sidecar metadata in the dataset at the provided path, creating the path if it does not exist.

Parameters

- **metadata** – Metadata key/value pairs to add.
- **path** – Relative path in archive at which to add image
- **updateLayout** – Update the underlying layout object upon conclusion of the metadata addition.

isEmpty(self) → bool

getSidecarMetadata(self, image: Union[str, bids.layout.BIDSImageFile], includeEntities: bool = True) → dict

Get metadata for the file at the provided path in the dataset. Sidecar metadata is always returned, and BIDS entities present in the filename are returned by default (this can be disabled).

Parameters

- **image** – Path or BIDSImageFile pointing to the image file to get metadata for.
- **includeEntities** – False to return only the metadata in the image’s sidecar JSON files. True to additionally include the entities in the filename (e.g., ‘subject’, ‘task’, and ‘session’). Defaults to True.

Raises **TypeError** – If image is not a str or BIDSImageFile.

Returns

Dictionary with sidecar metadata for the file and any metadata that can be extracted from the filename (e.g., subject, session).

Examples

```python
>>> archive = BidsArchive('/path/to/archive')
>>> path = archive.getImages()[0].path
>>> archive.getSidecarMetadata(path)
{'AcquisitionMatrixPE': 320, 'AcquisitionNumber': 1, ... }
```

getEvents(self, matchExact: bool = False, **entities) → List[bids.layout.BIDSDataFile]

Gets data from scanner run event files in the archive. Event files to retrieve can be filtered by entities present in the files’ names.

Parameters
- **matchExact** – Whether to only return events files that have exactly the same entities as provided (no more, no less)
- **entities** – Keyword arguments for entities to filter by. Provide in the format entity='value'.

**Returns** A list of BIDSDataFile objects encapsulating the events files matching the provided entities (empty if there are no matches, and containing at most a single object if an exact match is requested).

**Raises** **ValueError** – If the ‘extension’ entity is provided and not valid for an events file (i.e., not ‘.tsv’ or ‘.tsv.gz’)

### Examples

```python
global archive = BidsArchive('.

>>> archive.getEvents()

[BIDSDataFile filename='/tmp/dataset/sub-01/func/sub-01_task-test_events.tsv'], BIDSDataFile filename='/tmp/dataset/sub-02/func/sub-02_task-test_events.tsv')]

>>> sub1Events = archive.getEvents(subject='01')

[BIDSDataFile filename='/tmp/dataset/sub-01/func/sub-01_task-test_events.tsv']

>>> eventsDataFrame = sub1Events[0].get_df()

>>> print(eventsDataFrame[:][:1])

onset  duration  trial_type
0      0         rest
```

### _appendIncremental_

```python
def _appendIncremental(self, incremental: rtCommon.bidsIncremental.BidsIncremental, makePath: bool = True, validateAppend: bool = True) -> bool
```

Appends a BIDS Incremental’s image data and metadata to the archive, creating new directories if necessary (this behavior can be overridden). For internal use only.

**Parameters**

- **incremental** – BIDS Incremental to append
- **makePath** – Create new directory path for BIDS-I data if needed. (default: True).
- **validateAppend** – Compares image metadata and NIfTI headers to check that the images being appended are part of the same sequence and don’t conflict with each other (default: True).

**Raises**

- **RuntimeError** – If the image to append to in the archive is not either 3D or 4D.
- **StateError** – If the image path within the BIDS-I would result in directory creation and makePath is set to False.
- **ValidationError** – If the data to append is incompatible with existing data in the archive.

**Returns** True if the append succeeded, False otherwise.
Examples

Assume we have a NIfTI image `image` and a metadata dictionary `metadata` with all required metadata for a BIDS Incremental.

```python
>>> archive = BidsArchive('.
>>> incremental = BidsIncremental(image, metadata)
>>> archive._appendIncremental(incremental)
```

If we don’t want to create any new files/directories in the archive, makePath can be set to false.

```python
>>> archive = BidsArchive('/tmp/emptyDirectory')
>>> archive._appendIncremental(incremental, makePath=False)
```

```python
/archive/downloads/rtCloud/rtCloud
 admitted
 custom
```
getBidsRun(self, **entities) → rtCommon.bidsRun.BidsRun
Get a BIDS Run from the archive.

Parameters entities – Entities defining a run in the archive.

Returns A BidsRun containing all the BidsIncrementals in the specified run.

Raises
• NoMatchError – If the entities don’t match any runs in the archive.
• QueryError – If the entities match more than one run in the archive.

Examples

```python
>>> archive = BidsArchive('/tmp/dataset')
>>> run = archive.getBidsRun(subject='01', session='02',
   task='testTask', run=1)
>>> print(run.numIncrementals())
53
```
Module Contents

Classes

- **BidsFileExtension**
  Generic enumeration.

- **BidsEntityKeys**
  Generic enumeration.

Functions

- **loadBidsEntities() → dict**
  Loads all accepted BIDS entities from PyBids into a dictionary.

- **filterEntities(metadata: dict) → dict**
  Returns a new dictionary containing all the elements of the argument that

- **getNiftiData(image: nibabel.Nifti1Image, numpy.ndarray) → Nibabel exposes a get_fdata() method, but this converts all the data to**

- **makeDicomFieldBidsCompatible(dicomField: str) → str**
  Remove non-alphanumeric characters to make a DICOM field name

- **correct3DHeaderTo4D(image: nibabel.Nifti1Image, repetitionTime: int, timeUnitCode: int = 8) → None**
  Makes necessary changes to the NIfTI header to reflect the increase in its

- **adjustTimeUnits(imageMetadata: dict) → None**
  Validates and converts in-place the units of various time-based metadata.

- **metadataFromProtocolName(protocolName: str) → dict**
  Extracts BIDS label-value combinations from a DICOM protocol name, if

- **getDicomMetadata(dicomImg: pydicom.dataset.Dataset, kind=\'ts1\' all \'ts1\') → dict**
  Returns the public (even-numbered tags) and private (odd-numbered tags)

- **symmetricDictDifference(d1: dict, d2: dict, equal: Callable[[Any, Any], bool] = opeq) → dict**
  Returns the symmetric difference of the provided dictionaries. This

- **niftiHeadersAppendCompatible(header1: dict, header2: dict) → None**
  Verifies that two Nifti image headers match in along a defined set of

- **niftiImagesAppendCompatible(img1: nibabel.Nifti1Image, img2: nibabel.Nifti1Image) → Tuple[bool, str]**
  Verifies that two Nifti images have headers matching along a defined set of

- **metadataAppendCompatible(meta1: dict, meta2: dict) → Tuple[bool, str]**
  Verifies two metadata dictionaries match in a set of required fields. If a

- **correctEventsFileDatatypes(df: pandas.DataFrame) → pandas.DataFrame**

- **writeDataFrameToEvents(df: pandas.DataFrame, path: str) → None**

8.1. rtCommon
Attributes

---

**logger**

**BIDS_VERSION**

**DATASET_DESC_REQ_FIELDS**

**DEFAULT_DATASET_DESC**

**DEFAULT_README**

**DEFAULT_EVENTS_HEADERS**

**BIDS_FILE_PATTERN**

**BIDS_DIR_PATH_PATTERN**

**BIDS_FILE_PATH_PATTERN**

**PYBIDS_PSEUDO_ENTITIES**

**BIDS_EVENT_COL_TO_DTYPE**

**BidsAttributesToAnonymize**

**UNIT_TO_CODE**

**CODE_TO_UNIT**

---

rtCommon.bidsCommon.logger

rtCommon.bidsCommon.BIDS_VERSION = 1.4.1

rtCommon.bidsCommon.DATASET_DESC_REQ_FIELDS = ['Name', 'BIDSVersion']

rtCommon.bidsCommon.DEFAULT_DATASET_DESC

rtCommon.bidsCommon.DEFAULT_README = Generated BIDS-Incremental Dataset from RT-Cloud

rtCommon.bidsCommon.DEFAULT_EVENTS_HEADERS = ['onset', 'duration']

rtCommon.bidsCommon.BIDS_FILE_PATTERN = sub-{subject}[_{ses-{session}}]_task-{task}[_{acq-{acquisition}}][_{ce-{ceagent}}][_{dir-{direction}}][_r. ..

rtCommon.bidsCommon.BIDS_DIR_PATH_PATTERN = sub-{subject}[/{ses-{session}}]/{datatype<func>}{func}

rtCommon.bidsCommon.BIDS_FILE_PATH_PATTERN

rtCommon.bidsCommon.PYBIDS_PSEUDO_ENTITIES = ['extension']
rtCommon.bidsCommon.BIDS_EVENT_COL_TO_DTYPE

rtCommon.bidsCommon.BidsAttributesToAnonymize = ['PatientID', 'PatientsAge', 'PatientsBirthDate', 'PatientsName', 'PatientsSex', 'PatientsSize',...

class rtCommon.bidsCommon.BidsFileExtension
    Bases: enum.Enum
    Generic enumeration.
    Derive from this class to define new enumerations.
    IMAGE = .nii
    IMAGE_COMPRESSED = .nii.gz
    METADATA = .json
    EVENTS = .tsv

class rtCommon.bidsCommon.BidsEntityKeys
    Bases: enum.Enum
    Generic enumeration.
    Derive from this class to define new enumerations.
    ENTITY = entity
    FORMAT = format
    DESCRIPTION = description

rtCommon.bidsCommon.loadBidsEntities() → dict
    Loads all accepted BIDS entities from PyBids into a dictionary.

    Returns
    A dictionary mapping the entity names to the PyBids Entity object containing information about that entity.

rtCommon.bidsCommon.filterEntities(metadata: dict) → dict
    Returns a new dictionary containing all the elements of the argument that are valid BIDS entities.

rtCommon.bidsCommon.getNiftiData(image: nibabel.Nifti1Image) → numpy.ndarray
    Nibabel exposes a get_fdata() method, but this converts all the data to float64. Since our Nifti files are often converted from DICOM’s, which store data in signed or unsigned ints, treating the data as float can cause issues when comparing images or re-writing a Nifti read in from disk.

rtCommon.bidsCommon.makeDicomFieldBidsCompatible(dicomField: str) → str
    Remove non-alphanumeric characters to make a DICOM field name BIDS-compatible (CamelCase alphanumeric) metadata field. Note: Multi-word keys like ‘Frame of Reference UID’ become ‘FrameofReferenceUID’, which might be different than the expected behavior

    Parameters
dicomField – Name of the DICOM field to convert to BIDS format

    Returns
    DICOM field name in BIDS-compatible format.
Examples

```python
>>> field = "Repetition Time"
>>> makeDicomFieldBidsCompatible(field)
'REpetitionTime'
```

Makes necessary changes to the NIfTI header to reflect the increase in its corresponding image's data shape from 3D to 4D.

**Parameters**

- `image` – NIfTI image to modify header for
- `repetitionTime` – Repetition time for the scan that produced the image
- `timeUnitCode` – The temporal dimension NIfTI unit code (e.g., millimeters is 2, seconds is 8). Defaults to seconds.

**rtCommon.bidsCommon.adjustTimeUnits**

Validates and converts in-place the units of various time-based metadata, which is stored in seconds in BIDS, but often provided using milliseconds in DICOM.

**rtCommon.bidsCommon.metadataFromProtocolName**

Extracts BIDS label-value combinations from a DICOM protocol name, if any are present.

**Returns** A dictionary containing any valid label-value combinations found.

**rtCommon.bidsCommon.getDicomMetadata**

Returns the public (even-numbered tags) and private (odd-numbered tags) metadata from the provided DICOM image.

**Parameters**

- `dicomImg` – A Pydicom object to read metadata from.
- `kind` – Metadata category to get. ‘public’ for public DICOM tags, ‘private’ for private DICOM tags, ‘all’ for all DICOM tags.

**Returns** Dictionary containing requested metadata from the DICOM image.

**Raises** TypeError – If the image provided is not a pydicom.dataset.Dataset object (e.g., if the image were the raw DICOM data).

**rtCommon.bidsCommon.symmetricDictDifference**

Returns the symmetric difference of the provided dictionaries. This consists of 3 parts: 1) Key-value pairs for which both dictionaries have the key, but have different values for that key. 2) All key-value pairs that only the first dictionary has. 3) All key-value pairs that only the second dictionary has.

**Parameters**

- `d1` – First dictionary
- `d2` – Second dictionary
- `equal` – Function that returns True if two keys are equal, False otherwise
**Returns**  A dictionary with all key-value pair differences between the two dictionaries. ‘None’ is used as the value for a key-value pair if that dictionary lacks a key that the other one has.

**Examples**

```python
d1 = {'a': 1, 'b': 2, 'c': 3}
d2 = {'c': 4, 'd': 5}
print(symmetricDictDifference(d1, d2))
{'a': [1, None], 'b': [2, None], 'c': [3, 4], 'd': [None, 5]}
d2 = {'a': 1, 'b': 2, 'c': 4}
print(symmetricDictDifference(d1, d2))
{'c': [3, 4]}
```

**rtCommon.bidsCommon.niftiHeadersAppendCompatible**

```python
rtCommon.bidsCommon.niftiHeadersAppendCompatible(header1: dict, header2: dict)
```

Verifies that two Nifti image headers match along a defined set of NIfTI header fields which should not change during a continuous fMRI scanning session.

This is primarily intended as a safety check, and does not conclusively determine that two images are valid to append to together or are part of the same scanning session.

**Parameters**

- **header1** – First Nifti header to compare (dict of numpy arrays)
- **header2** – Second Nifti header to compare (dict of numpy arrays)

**Returns**  True if the headers match along the required dimensions, False otherwise.

**rtCommon.bidsCommon.niftiImagesAppendCompatible**

```python
```

Verifies that two Nifti images have headers matching along a defined set of NIfTI header fields which should not change during a continuous fMRI scanning session.

This is primarily intended as a safety check, and does not conclusively determine that two images are valid to append to together or are part of the same scanning session.

**Parameters**

- **img1** – First Nifti image to compare
- **img2** – Second Nifti image to compare

**Returns**  

**True if the image headers match along the required dimensions, False** otherwise.

**rtCommon.bidsCommon.metadataAppendCompatible**

```python
rtCommon.bidsCommon.metadataAppendCompatible(meta1: dict, meta2: dict) → Tuple[bool, str]
```

Verifies two metadata dictionaries match in a set of required fields. If a field is present in only one or neither of the two dictionaries, this is considered a match.

This is primarily intended as a safety check, and does not conclusively determine that two images are valid to append to together or are part of the same series.

**Parameters**

- **meta1** – First metadata dictionary
- **meta2** – Second metadata dictionary

**Returns**  

True if all keys that are present in both dictionaries have equivalent values, false otherwise.

```

rtCommon.bidsCommon.writeDataFrameToEvents(df: pandas.DataFrame, path: str) → None
```

### 8.1.1.4 rtCommon.bidsIncremental

bidsIncremental.py

Implements the BIDS Incremental data type used for streaming BIDS data between different applications.

---

**Module Contents**

**Classes**

**BidsIncremental**

**Attributes**

**logger**

```
rtCommon.bidsIncremental.logger

class rtCommon.bidsIncremental.BidsIncremental(image: nibabel.Nifti1Image, imageMetadata: dict, datasetDescription: dict = None)
```

**ENTITIES**

```python
REQUIRED_IMAGE_METADATA = ['subject', 'task', 'suffix', 'datatype', 'RepetitionTime']
```

BIDS Incremental data format suitable for streaming BIDS Archives

**__str__**

```
__str__(self)
```

Return str(self).

**__eq__**

```
__eq__(self, other)
```

Return self==value.

**__getstate__**

```
__getstate__(self)
```
__setstate__(self, state)

_preprocessMetadata(self, imageMetadata: dict) → dict
Pre-process metadata to extract any additional metadata that might be embedded in the provided metadata, like ProtocolName, and ensure that certain metadata values (e.g., RepetitionTime) are within BIDS-specified ranges.

Parameters imageMetadata – Metadata dictionary provided to BIDS incremental to search for additional, embedded metadata

Returns
Original dictionary with all embedded metadata added explicitly and values within BIDS-specified ranges.

_exceptIfMissingMetadata(self, imageMetadata: dict) → None
Ensure that all required metadata is present.

Parameters imageMetadata – Metadata dictionary to check for missing metadata

Raises MissingMetadataError – If not all required metadata is present.

_postprocessMetadata(self, imageMetadata: dict) → dict
Post-process metadata once all required fields are given (e.g., to create derived fields like ‘TaskName’ from ‘task’).

Parameters imageMetadata – Metadata dictionary to post-process.

Returns Metadata dictionary with derived fields set.

static createImageMetadataDict(subject: str, task: str, suffix: str, datatype: str, repetitionTime: int)
Creates an image metadata dictionary for a BIDS-I with all of the basic required fields using the correct key names.

Parameters
• subject – Subject ID (e.g., ‘01’)
• task – Task ID (e.g., ‘story’)
• suffix – Imaging method (e.g., ‘bold’)
• datatype – Data type (e.g., ‘func’ or ‘anat’)
• repetitionTime – TR time, in seconds, used for the imaging run

Returns Dictionary with the provided information ready for use in a BIDS-I

classmethod findMissingImageMetadata(cls, imageMeta: dict) → list
Creates a list of all required metadata fields that the argument dictionary is missing.

Parameters imageMeta – Metadata dictionary to check for missing fields

Returns List of required fields missing in the provided dictionary.
Examples

```python
>>> meta = {'subject': '01', 'task': 'test', 'suffix': 'bold',
          'datatype': 'func'}
>>> BidsIncremental.findMissingImageMetadata(meta)
['RepetitionTime']

classmethod isCompleteImageMetadata(cls, imageMeta: dict) → bool
Verifies that all required metadata fields for BIDS-I construction are present in the dictionary.

Parameters

- **imageMeta** – The dictionary with the metadata fields

Returns

- True if all required fields are present in the dictionary, False otherwise.

Examples

```python
>>> meta = {'subject': '01', 'task': 'test', 'suffix': 'bold',
          'datatype': 'func'}
>>> BidsIncremental.isCompleteImageMetadata(meta)
False
```

_exceptIfNotBids(self, entityName: str) → None
Raise an exception if the argument is not a valid BIDS entity

getMetadataField(self, field: str, strict: bool = False) → Any
Get value for the field in the incremental’s metadata, if it exists.

Parameters

- **field** – Metadata field to retrieve a value for.
- **default** – Default value to return if field is not present.
- **strict** – Only allow getting fields that are defined as BIDS entities in the standard.

Returns

- Entity’s value, or None if the entity isn’t present in the metadata.

Raises

- **ValueError** – If ‘strict’ is True and ‘field’ is not a BIDS entity.
- **KeyError** – If the field is not present in the Incremental’s metadata and not default value is provided.

Examples

```python
>>> incremental.getMetadataField('task')
'faces'
>>> incremental.getMetadataField('RepetitionTime')
1.5
>>> incremental.getMetadataField('RepetitionTime', strict=True)
ValueError: RepetitionTime is not a valid BIDS entity name
```

setMetadataField(self, field: str, value: Any, strict: bool = False) → None
Set metadata field to provided value in Incremental’s metadata.

Parameters
• **field** – Metadata field to set value for.
• **value** – Value to set for the provided entity.
• **strict** – Only allow setting fields that are defined as BIDS entities in the standard.

**Raises** `ValueError` – If `strict` is True and `field` is not a BIDS entity.

`removeMetadataField(self, field: str, strict: bool = False) → None`
Remove a piece of metadata from the incremental’s metadata.

**Parameters**
• **field** – BIDS entity name to retrieve a value for.
• **strict** – Only allow removing fields that are defined as BIDS entities in the standard.

**Raises**
• `ValueError` – If `strict` is True and `field` is not a BIDS entity.
• `RuntimeError` – If the field to be removed is required by the Incremental.

`getImageMetadata(self)`
`getSuffix(self) → str`
`getDatatype(self) → str`
`func or anat`
`getEntities(self) → dict`
`getImageDimensions(self) → tuple`
`getImageHeader(self)`
`getImageData(self) → numpy.ndarray`

`makeBidsFileName(self, extension: rtCommon.bidsCommon.BidsFileExtension) → str`
Create the a BIDS-compatible file name based on the metadata. General format of the filename, per BIDS standard 1.4.1, is as follows (items in [square brackets] are considered optional):
```
sub-<label>[ses-<label>]_task-<label>[_acq-<label>][_ce-<label>][_dir-<label>][_rec-<label>][_run-
/index][_echo-index][_contrast_label].ext
```

**Parameters** `extension` – The extension for the file, e.g., ‘nii’ for images or ‘json’ for metadata

**Returns** Filename from metadata according to BIDS standard 1.4.1.

`getDatasetName(self) → str`
`getImageFileName(self) → str`
`getMetadataFileName(self) → str`
`getEventsFileName(self) → str`
`getImageFilePath(self) → str`
`getMetadataFilePath(self) → str`
`getEventsFilePath(self) → str`
**getDirPath**

\[ \text{getDirPath}(\text{self}) \rightarrow \text{str} \]

Path to where this incremental’s data would be in a BIDS archive, relative to the archive root.

**Returns** Path string relative to root of the imaginary dataset.

**Examples**

```python
>>> print(bidsi.getDataDirPath())
sub-01/ses-2011/anat
```

**getAcquisitionTime**

\[ \text{getAcquisitionTime}(\text{self}) \rightarrow \text{datetime.datetime.time} \]

Returns the acquisition time as a datetime.time

**getRepetitionTime**

\[ \text{getRepetitionTime}(\text{self}) \rightarrow \text{float} \]

Returns the TR repetition time in seconds

**timeToNextTr**

\[ \text{timeToNextTr}(\text{self}, \text{clockSkew}, \text{now=None}) \rightarrow \text{float} \]

Based on acquisition time returns seconds to next TR start

**writeToDisk**

\[ \text{writeToDisk}(\text{self}, \text{datasetRoot: str}, \text{onlyData=False}) \rightarrow \text{None} \]

Writes the incremental’s data to a directory on disk. NOTE: The directory is assumed to be empty, and no checks are made for data that would be overwritten.

**Parameters**

- **datasetRoot** – Path to the root of the BIDS archive to be written to.
- **onlyData** – Only write out the NIfTI image and sidecar metadata (Default False). Useful if writing an incremental out to an existing archive and you don’t want to overwrite existing README or dataset_description.json files.

**Examples**

```python
>>> from bidsArchive import BidsArchive

incremental = BidsIncremental(image, metadata)
>>> root = '/tmp/emptyDirectory'
>>> incremental.writeToDisk(root)
>>> archive = BidsArchive(root)
>>> print(archive)
Root: /tmp/emptyDirectory | Subjects: 1 | Sessions: 1 | Runs: 1
```

---

**8.1.1.5 rtCommon.bidsInterface**

BidsInterface is a client interface (i.e. for the experiment script running in the cloud) that provides data access to BIDS data.

To support RPC calls from the client, there will be two instances of dataInterface, one at the cloud projectServer which is a stub to forward requests (started with dataRemote=True), and another at the control room computer, run as a service and with dataRemote=False.

When not using RPC, i.e. when the projectServer is run without –dataRemote, there will be only one instance of dataInterface, as part of the projectServer with dataRemote=False.
Module Contents

Classes

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BidsInterface</td>
<td>Provides functions for accessing remote or local BIDS data depending on if dataRemote flag is set true or false.</td>
</tr>
<tr>
<td>DicomToBidsStream</td>
<td>A class that watches for DICOM file creation in a specified directory and with</td>
</tr>
<tr>
<td>BidsStream</td>
<td>A class that opens a BIDS archive and prepares to stream the data as</td>
</tr>
</tbody>
</table>

```python
class rtCommon.bidsInterface.BidsInterface(dataRemote=False, allowedDirs=[], scannerClockSkew=0)
    Bases: rtCommon.remoteable.RemoteableExtensible

    Provides functions for accessing remote or local BIDS data depending on if dataRemote flag is set true or false.
    If dataRemote=True, then the RemoteExtensible parent class takes over and forwards all requests to a remote server via a callback function registered with the RemoteExtensible object. In that case none of the methods below will be locally invoked.
    If dataRemote=False, then the methods below will be invoked locally and the RemoteExtensible parent class is inoperable (i.e. does nothing).

    initDicomBidsStream(self, dicomDir, dicomFilePattern, dicomMinSize, anonymize=True, **entities) → int
    Intialize a data stream that watches a directory for DICOM files to be written that match the given file pattern. When a DICOM is written it will be converted to a BIDS incremental and returned.

    Parameters
    - **dicomDir** – the directory where the images are or will be written from the MRI scanner.
    - **dicomFilePattern** – a pattern of the image file names that has a TR tag which will be used to index the images, for example ‘scan01_{TR:03d}.dcm’. In this example a call to getImageData(imgIndex=6) would look for dicom file ‘scan01_006.dcm’.
    - **minFileSize** – Minimum size of the file to return (continue waiting if below this size)
    - **anonymize** – Whether to remove participant specific fields from the Dicom header
    - **entities** – BIDS entities (subject, session, task, run, suffix, datatype) that will be required to fill in the BIDS metadata in the BIDS Incremental

    Returns An int identifier to be used when calling stream functions, such as getIncremental()

    Return type  streamId

    initBidsStream(self, archivePath, **entities) → int
    Initialize a data stream from an existing BIDS archive.

    Parameters
    - **archivePath** – Full path to the BIDS archive
    - **entities** – BIDS entities (subject, session, task, run, suffix, datatype) that define the particular subject/run of the data to stream

    Returns An int identifier to be used when calling stream functions, such as getIncremental()

    Return type  streamId
```

8.1. rtCommon
**initOpenNeuroStream**\(_(self, dsAccessionNumber, **entities)\) \rightarrow int

Initialize a data stream that replays an OpenNeuro dataset.

**Parameters**
- `dsAccessionNumber` - OpenNeuro accession number of the dataset to replay
- `entities` - BIDS entities (subject, session, task, run, suffix, datatype) that define the particular subject/run of the data to stream

**Returns** An identifier used when calling stream functions, such as `getIncremental()`

**Return type** `streamId`

**getIncremental**\(_(self, streamId, volIdx=-1, timeout=5, demoStep=0)\) \rightarrow rtCommon.bidsIncremental.BidsIncremental

Get a BIDS Incremental from a stream

**Parameters**
- `streamId` - The stream handle returned by the `initXXStream` call
- `volIdx` - The brain volume index of the image to return. If -1 is entered it will return the next volume
- `timeout` - Max number of seconds to wait for incremental when running real-time
- `demoStep` - Simulate x second TR delay

**Returns** A BidsIncremental containing the image volume

**getNumVolumes**\(_(self, streamId)\) \rightarrow int

Return the number of image volumes contained in the stream. This is only defined for Bids/OpenNeuro streams (not for DicomBidsStreams)

**Parameters** `streamId` - The stream handle returned by the `initXXStream` call

**Returns** An int of the number of volumes

**closeStream**\(_(self, streamId)\)

**getClockSkew**\(_(self, callerClockTime: float, roundTripTime: float)\) \rightarrow float

Returns the clock skew between the caller’s computer and the scanner clock. This function is assumed to be running in the scanner room and have adjustments to translate this server’s clock to the scanner clock. Value returned is in seconds. A positive number means the scanner clock is ahead of the caller’s clock. The caller should add the skew to their localtime to get the time in the scanner’s clock.

**Parameters**
- `time` - (callerClockTime - current)
- `caller` - (roundTripTime - measured RTT in seconds to remote)

**Returns** Clock skew - seconds the scanner’s clock is ahead of the caller’s clock

**ping**\(_(self)\) \rightarrow float

Returns seconds since the epoch

**class** rtCommon.bidsInterface.DicomToBidsStream\(_(allowedDirs=[])\)

A class that watches for DICOM file creation in a specified directory and with a specified file pattern. When DICOM files arrive it converts them to BIDS incrementals and returns the BIDS incremental. This lets a real-time classification script process data directly as BIDS as it arrives from the scanner.
**initStream**(*self, dicomDir, dicomFilePattern, dicomMinSize, anonymize=True, **entities*)

Initialize a new DicomToBids stream, watches for Dicom and streams as BIDS

**Parameters**

- **dicomDir** – The directory where the scanner will write new DICOM files
- **dicomFilePattern** – A regex style pattern of the DICOM filenames to watch for. They should include a {TR} tag with optional formatting. For example filenames like ‘001_000013_000005.dcm’ would have a pattern ‘001_000013_{TR:06d}.dcm’ where the volume number (TR) will be filled in by a 6 digit leading zeros value.
- **dicomMinSize** – Minimum size of the file to return (will continue waiting if below this size)
- **anonymize** – Whether to remove participant specific fields from the Dicom header
- **entities** – BIDS entities (subject, session, task, run, suffix, datatype) that define the particular subject/run of the data to stream

**abstract getNumVolumes**(*self*) → int

Return the number of brain volumes in the run, unknowable by this interface ahead of time for a real-time DICOM stream

**getIncremental**(*self, volIdx=-1, timeout=5, demoStep=0, returnNifti=False*) → rtCommon.bidsIncremental.BidsIncremental

Get the BIDS incremental for the corresponding DICOM image indicated by the volIdx, where volIdx is equivalent to TR id.

VolIdx acts similar to a file seek pointer. If a volIdx >= 0 is supplied the volume pointer is advanced to that position. If no volIdx or a volIdx < 0 is supplied, then the next image volume after the previous position is returned and the pointer is incremented.

**Parameters**

- **volIdx** – The volume index (or TR) within the run to retrieve.
- **timeout** – Not used but present to support calling convention of other stream types

**Returns** BidsIncremental of that volume index within this subject/run

**class** rtCommon.bidsInterface.BidsStream(*archivePath, **entities*)

A class that opens a BIDS archive and prepares to stream the data as BIDS incrementals.

**getNumVolumes**(*self*) → int

Return the number of brain volumes in the run

**getIncremental**(*self, volIdx=-1, timeout=5, demoStep=0*) → rtCommon.bidsIncremental.BidsIncremental

Get a BIDS incremental for the indicated index in the current subject/run VolIdx acts similar to a file seek pointer. If a volIdx >= 0 is supplied the volume pointer is advanced to that position. If no volIdx or a volIdx < 0 is supplied, then the next image volume after the previous position is returned and the pointer is incremented.

**Parameters**

- **volIdx** – The volume index (or TR) within the run to retrieve.
- **timeout** – Not used but present to support calling convention of other stream types

**Returns** BidsIncremental of that volume index within this subject/run
bidsRun.py

Implements the BIDS Run data type used for representing full fMRI scanning runs as sequences of BIDS incrementals.

Module Contents

Classes

BidsRun

Attributes

logger

rtCommon.bidsRun.logger
class rtCommon.bidsRun.BidsRun(**entities)

__eq__(self, other)
    Return self==value.

getIncremental(self, index: int) → rtCommon.bidsIncremental.BidsIncremental
    Returns the incremental in the run at the provided index.

    Parameters
    index -- Which image of the run to get (0-indexed)

    Returns
    Incremental at provided index.

    Raises
    IndexError -- If index is out of bounds for this run.

Examples

>>> print(run.numIncrementals())
5
>>> inc = run.getIncremental(1)
>>> inc2 = run.getIncremental(5)
IndexError

appendIncremental(self, incremental: rtCommon.bidsIncremental.BidsIncremental, validateAppend: bool = True) → None

Appends an incremental to this run’s data, setting the run’s entities if the run is empty.

Parameters
• **incremental** – The incremental to add to the run.

• **validateAppend** – Validate the incremental matches the current run’s data (default True). Turning off is useful for efficiently creating a whole run at once from an existing image volume, where all data is known to be match already.

 Raises **MetadataMismatchError** – If either the incremental’s entities, its images’s NIfTI header, or its metadata doesn’t match the existing run’s data.

### Examples

Suppose a NIfTI image and metadata dictionary are available in the environment.

```python
>>> incremental = BidsIncremental(image, metadata)
>>> run = BidsRun()
>>> run.appendIncremental(incremental)
>>> metadata['subject'] = 'new_subject'
>>> incremental2 = BidsIncremental(image, metadata)
>>> run.appendIncremental(incremental2)
MetadataMismatchError
```

**asSingleIncremental**(self) → `rtCommon.bidsIncremental.BidsIncremental`

Coalesces the entire run into a single BIDS-I that can be sent over a network, written to disk, or added to an archive.

**Returns**

**BidsIncremental with all image data and metadata represented by the** incrementals composing the run, or None if the run is empty.

### Examples

```python
>>> incremental = run.asSingleIncremental()
>>> incremental.writeToDisk('/tmp/new_dataset')
```

**numIncrementals**(self) → int

Returns number of incrementals in this run.

**getRunEntities**(self) → dict

Returns dictionary of the BIDS entities associated with this run.

### Examples

```python
>>> print(run.getRunEntities())
{'subject': '01', 'session': '01', 'task': 'test', run: 1, 'datatype': 'func', 'suffix': 'bold'}
```
8.1.1.7 rtCommon.certsUtils

Utility functions for using ssl encrypted web connections

Module Contents

Functions

getSslCertFilePath()

getSslKeyFilePath()

Attributes

currPath

rootPath

certsDir

sslCertFile

sslPrivateKey

rtCommon.certsUtils.currPath
rtCommon.certsUtils.rootPath
rtCommon.certsUtils.certsDir
rtCommon.certsUtils.sslCertFile = rtcloud.crt
rtCommon.certsUtils.sslPrivateKey = rtcloud_private.key
rtCommon.certsUtils.getSslCertFilePath()
rtCommon.certsUtils.getSslKeyFilePath()

8.1.1.8 rtCommon.checkDicomNiftiConversion

checkDicomNiftiConversion.py (Last Updated: 05/26/2020)

The purpose of this script is to check that the nifti file conversion done during real-time in the cloud matches the nifti conversion done during your offline analyses, assuming you used heudiconv or you directly called ‘dcm2nix()’. This script addresses the warning you get when you import pydicom.

To run this script, uncomment the lines in ‘def main()’ below. Complete the sections that need to be filled in, denoted by “[FILL IN]””. Save this file and then run the script “python checkDicomNiftiConversion.py” in the terminal.
Module Contents

Functions

```python
checkingDicomNiftiConversion(cfg)

Purpose: check the nibabel nifti/dicom conversion method BEFORE using it in main()
```

```
rtCommon.checkDicomNiftiConversion.checkingDicomNiftiConversion(cfg)

Purpose: check the nibabel nifti/dicom conversion method BEFORE using it in real-time. Here we’re assuming you have raw dicoms and the corresponding converted nifti that you used in pilot data collection.

STEPS:
0. set up config
1. get number of TRs from dicom path and nifti path
2. go through each TR in dicom, convert them each to niftis and save
3. load in each nifti into new data matrix
4. compare this to loaded offline nifti file

rtCommon.checkDicomNiftiConversion.main()
```

8.1.1.9 rtCommon.clientInterface

This module will be imported by the experiment script (i.e. client) running in the cloud and provide the interfaces for all functionality provided to the client by the rt-cloud projectServer.

The client script instantiates a clientInterface object. It will automatically connect to the projectServer running on the localhost (i.e. same host as the client). If a connection is established the interfaces listed below will be stubs that forward requests to remote servers that will handle the requests. If the connection fails (i.e. there is no projectServer running), then local versions of the services will be instantiated, for example to access local files instead of remote files. The user will be prompted if local versions will be used.

Client Service Interfaces provided (i.e. for the classification script client):
- dataInterface - to read and write files from the remote server
- subjectInterface - to send subject feedback and receive responses
- webInterface - to set browser messages, update plots, send/receive configs

Module Contents

Classes

```python
ClientInterface

This class provides the API that an experiment script can use to communicate with the
```

```
WrapRpycObject

Rpyc commands return a rpyc.core.netref object to as a reference to the remote object.
```

8.1. rtCommon
class rtCommon.clientInterface.ClientInterface(rpyc_timeout=120, yesToPrompts=False)

This class provides the API that an experiment script can use to communicate with the project server. It provides both a DataInterface for reading or writing files, and a SubjectInterface for sending/receiving feedback and response to the subject in the MRI scanner.

isDataRemote(self)

Will return false if either no project server is running, or if a projectServer is running with data being served locally by the projectServer (remember that the projectServer and classification client script always run on the same computer).

isSubjectRemote(self)

Same semantics as isDataRemote above.

isUsingProjectServer(self)

class rtCommon.clientInterface.WrapRpycObject(rpycObject)

Bases: object

Rpyc commands return a rpyc.core.netref object to as a reference to the remote object. This class wraps all calls to the remote in order to dereference the rpyc.core.netref and return the actual object using rpyc.classic.obtain(ref)

__getattribute__(self, name)

Return getattr(self, name).

8.1.1.10 rtCommon.dataInterface

DataInterface is a client interface (i.e. for the experiment script running in the cloud) that provides data access, such as reading and writing files.

To support RPC calls from the client, there will be two instances of dataInterface, one at the cloud projectServer which is a stub to forward requests (started with dataRemote=True), and another at the control room computer, run as a service and with dataRemote=False.

When not using RPC, i.e. when the projectServer is run without –dataRemote, there will be only one instance of dataInterface, as part of the projectServer with dataRemote=False.

Module Contents

Classes

DataInterface Provides functions for accessing remote or local files depending on if dataRemote flag is
Functions

```
uploadFilesFromList(dataInterface, fileList: List[str], outputDir: str, srcDirPrefix=None) -> None
Copies files in fileList from the remote onto the system where this call is being made.

downloadFilesFromList(dataInterface, fileList: List[str], outputDir: str, srcDirPrefix=None) -> None
Copies files in fileList from this computer to the remote.

uploadFolderToCloud(dataInterface, srcDir: str, outputDir: str) -> None
Copies a folder (directory) from the remote to the system where this call is run.

uploadFilesToCloud(dataInterface, srcFilePattern: str, outputDir: str) -> None
Copies files matching (regex) srcFilePattern from the remote onto the system.

downloadFolderFromCloud(dataInterface, srcDir: str, outputDir: str, deleteAfter=False) -> None
Copies a directory from the system where this call is made to the remote system.

downloadFilesFromCloud(dataInterface, srcFilePattern: str, outputDir: str, deleteAfter=False) -> None
Copies files matching srcFilePattern from the system where this call is made.
```


Bases: rtCommon.remoteable.RemoteableExtensible

Provides functions for accessing remote or local files depending on if dataRemote flag is set true or false.

If dataRemote=True, then the RemoteExtensible parent class takes over and forwards all requests to a remote server via a callback function registered with the RemoteExtensible object. In that case none of the methods below will be locally invoked.

If dataRemote=False, then the methods below will be invoked locally and the RemoteExtensible parent class is inoperable (i.e. does nothing).

```
__del__(self)
```

```
initScannerStream(self, imgDir: str, filePattern: str, minFileSize: int, anonymize: bool = True, demoStep: int = 0) -> int
Initialize a data stream context with image directory and filepattern. Once the stream is initialized call getImageData() to retrieve image data. NOTE: currently only one stream at a time is supported.

Parameters
- **imgDir** – the directory where the images are or will be written from the MRI scanner.
- **filePattern** – a pattern of the image file names that has a TR tag which will be used to index the images, for example ‘scan01_{TR:03d}.dcm’. In this example a call to getImageData(imgIndex=6) would look for dicom file ‘scan01_006.dcm’.
- **minFileSize** – Minimum size of the file to return (continue waiting if below this size)
- **anonymize** – Whether to remove participant specific fields from the Dicom header

Returns An identifier used when calling getImageData()

Return type streamId
```

```
getImageData(self, streamId: int, imageIndex: int = None, timeout: int = 5) -> pydicom.dataset.FileDataset
Get data from a stream initialized with initScannerStream

Parameters
- **streamId** – Id of a previously opened stream.
```
• **imageIndex** – Which image from the stream to retrieve. If left blank it will retrieve the next image in the stream (next after either the last request or starting from 0 if no previous requests)

• **timeout** – Max number of seconds to wait for image data to be available

Returns The bytes array representing the image data returns pydicom.dataset.FileDataset

```python
getFile(self, filename: str) → bytes
```
Returns a file’s data immediately or fails if the file doesn’t exist.

```python
getNewestFile(self, filepattern: str) → bytes
```
Searches for files matching filePattern and returns the data from the newest one.

```python
initWatch(self, dir: str, filePattern: str, minFileSize: int, demoStep: int = 0) → None
```
Initialize a watch directory for files matching filePattern.
No data is returned by this function, but a filesystem watch is established. After calling initWatch, use watchFile() to watch for a specific file’s arrival.

Parameters

• **dir** – Directory to watch for arrival (creation) of new files

• **filePattern** – Regex style filename pattern of files to watch for (i.e. *.dcm)

• **minFileSize** – Minimum size of the file to return (continue waiting if below this size)

• **demoStep** – Minimum interval (in seconds) to wait before returning files. Useful for demos replaying existing files while mimicking original timing.

```python
watchFile(self, filename: str, timeout: int = 5) → bytes
```
Watches for a specific file to be created and returns the file data.
InitWatch() must be called first, before watching for specific files. If filename includes the full path, the path must match that used in initWatch().

Parameters

• **filename** – Filename to watch for

• **timeout** – Max number of seconds to wait for file to be available

Returns The file data

```python
putFile(self, filename: str, data: Union[str, bytes], compress: bool = False) → None
```
Create a file (filename) and write the bytes or text to it. In remote mode the file is written at the remote.

Parameters

• **filename** – Name of file to create

• **data** – data to write to the file

• **compress** – Whether to compress the data in transit (not within the file), only has affect in remote mode.

```python
listFiles(self, filepattern: str) → List[str]
```
Lists files matching the regex filePattern

```python
listDirs(self, dirpattern: str) → List[str]
```
Lists directories matching the regex filePattern
getAllowedFileTypes(self) → List[str]
   Returns the list of file extensions which are allowed for read and write

getClockSkew(self, callerClockTime: float, roundTripTime: float) → float
   Returns the clock skew between the caller's computer and the scanner clock. This function is assumed to be running in the scanner room and have adjustments to translate this server's clock to the scanner clock. Value returned is in seconds. A positive number means the scanner clock is ahead of the caller's clock. The caller should add the skew to their localtime to get the time in the scanner’s clock. 
   :param callerClockTime - current time: 
   :type callerClockTime - current time: secs since epoch  
   :param roundTripTime - measured RTT in seconds to remote caller: 
   
   Returns  Clockskew - seconds the scanner’s clock is ahead of the caller’s clock

ping(self) → float
   Returns seconds since the epoch

_checkAllowedDirs(self, dir: str) → bool

_checkAllowedFileTypes(self, filename: str) → bool
   Class-private function for checking if a file is allowed.

_filterFileList(self, fileList: List[str]) → List[str]
   Class-private function to filter a list of files to include only allowed ones. 
   Args: fileList - list of files to filter
   Returns: filtered fileList - containing only the allowed files

rtCommon.dataInterface.uploadFilesFromList(dataInterface, fileList: List[str], outputDir: str, srcDirPrefix=None) → None
   Copies files in fileList from the remote onto the system where this call is being made.

rtCommon.dataInterface.downloadFilesFromList(dataInterface, fileList: List[str], outputDir: str, srcDirPrefix=None) → None
   Copies files in fileList from this computer to the remote.

rtCommon.dataInterface.uploadFolderToCloud(dataInterface, srcDir: str, outputDir: str) → None
   Copies a folder (directory) from the remote to the system where this call is run

rtCommon.dataInterface.uploadFilesToCloud(dataInterface, srcFilePattern: str, outputDir: str)
   Copies files matching (regex) srcFilePattern from the remote onto the system where this call is being made.

rtCommon.dataInterface.downloadFolderFromCloud(dataInterface, srcDir: str, outputDir: str, deleteAfter=False) → None
   Copies a directory from the system where this call is made to the remote system.

rtCommon.dataInterface.downloadFilesFromCloud(dataInterface, srcFilePattern: str, outputDir: str, deleteAfter=False) → None
   Copies files matching srcFilePattern from the system where this call is made to the remote system.
**8.1.1.11 rtCommon.dicomToBidsService**

dicomToBidsService.py
A very basic DICOM to BIDS-I converter.

---

**Module Contents**

**Functions**

```python

dicomToBidsInc(dicomImg: pydicom.dataset.Dataset, extraMetadata: dict = {}) → rtCommon.bidsIncremental.BidsIncremental
```

---

**8.1.1.12 rtCommon.errors**

Exception definitions for rtfMRI

---

**Module Contents**

**exception rtCommon.errors.RTError**
Bases: Exception
Top level general error

**exception rtCommon.errors.ValidationError**
Bases: RTError
Invalid information supplied in a call

**exception rtCommon.errors.StateError**
Bases: RTError
System is not in a valid state relative to the request

**exception rtCommon.errors.RequestError**
Bases: RTError
Error in the request

**exception rtCommon.errors.MessageError**
Bases: RTError
Invalid message
exception rtCommon.errors.InvocationError
    Bases: RTError
    program arguments incorrect

exception rtCommon.errors.VersionError
    Bases: RTError
    Client/Server code versions don’t agree

exception rtCommon.errors.MissedDeadlineError
    Bases: RTError
    Server missed a deadline

exception rtCommon.errors.MissedMultipleDeadlines
    Bases: RTError
    Server missed two or more deadlines

exception rtCommon.errors.NotImplementedError
    Bases: RTError
    Functionality is not implemented yet

exception rtCommon.errors.MissingMetadataError
    Bases: RTError
    Required BIDS metadata missing

exception rtCommon.errors.MetadataMismatchError
    Bases: RTError
    Mismatch in metadata (e.g., BIDS or NIfTI)

exception rtCommon.errors.DimensionError
    Bases: RTError
    Invalid image dimensions for requested operation

exception rtCommon.errors.QueryError
    Bases: RTError
    A query failed or returned unexpected results

8.1.1.13 rtCommon.exampleInterface

An example remote interface. Copy this file as a starting point for a new service.

Remote interfaces can be easily created by subclassing the RemoteExtensible class.

All methods in the subclass will be callable through an RPC interface which the experiment script, running in the cloud, can access through the client object.

In addition to creating a subclassed RemoteExtensible, this class (object) must be instantiated within the projectServer-RPC class, e.g. exposed_ExampleInterface. The instantiated object is what will be invoked when RPC calls are made to the exampleInterface.

For the remote case, there must be a remote end-point to handle the request. This can be within a service (such as scannerDataService) that instantiates the classes (such as dataInterface, bidsInterface, exampleInterface etc.) that will handle the remote forwarded requests. Add the object to a data service, or create a new one, that will run at the control room computer and is initialized with dataRemote=False.

8.1. rtCommon
The control room instance will be the actual instance and the projectServerRPC instance is a stub instance (dataRemote=True) that forwards request to the control room instance.

**Module Contents**

**Classes**

**ExampleInterface**  
Provides functions for experimenter scripts  

```python
class rtCommon.exampleInterface.ExampleInterface(dataRemote=False):
    Bases: rtCommon.remoteable.RemoteableExtensible
    Provides functions for experimenter scripts
    echo(self, val)
    testMethod(self, *args, **kwargs)
```

**8.1.1.14 rtCommon.exampleService**

An example remote command-line service, for example as would be run at the scanner computer or the presentation computer to receive requests from the the classification script.

This service instantiates an ExampleInterface for sending/receiving example requests to the projectServer in the cloud. It connects to the remote projectServer. Once a connection is established it waits for requests and invokes the ExampleInterface functions to handle them.

**Module Contents**

**Classes**

**ExampleService**

**Attributes**

```python
currPath
rootPath
connectionArgs
```

rtCommon.exampleService.currPath  
rtCommon.exampleService.rootPath
class rtCommon.exampleService.ExampleService(args, webSocketChannelName='wsData'):
    runDetached(self)
        Starts the receiver in its own thread.
rtCommon.exampleService.connectionArgs

8.1.1.15 rtCommon.fileWatcher

FileWatcher implements a class that watches for files to be created in a directory and then returns the notification that the files is now available.

The FileWatcher class is a virtual class of sorts with two underlying implementations, one for Mac and Windows (WatchdogFileWatcher) and one for Linux (InotifyFileWatcher).

Module Contents

Classes

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileWatcher</td>
<td>Virtual class to watch for the arrival of new files and notify.</td>
</tr>
<tr>
<td>FileNotifyHandler</td>
<td>Handler class that will receive the watchdog notifications. It will queue the notifications</td>
</tr>
<tr>
<td>InotifyFileWatcher</td>
<td>Version of FileWatcher for Linux using Inotify interface.</td>
</tr>
</tbody>
</table>

class rtCommon.fileWatcher.FileWatcher
    Virtual class to watch for the arrival of new files and notify.
    __del__(self)
    initFileNotifier(self, dir, filePattern, minFileSize, demoStep=0) → None
    waitForFile(self, filename, timeout=0, timeCheckIncrement=1)

class rtCommon.fileWatcher.WatchdogFileWatcher
    Version of FileWatcher for Mac and Windows using Watchdog toolkit.
    __del__(self)
    initFileNotifier(self, dir: str, filePattern: str, minFileSize: int, demoStep: int = 0) → None
        Initialize the file watcher to watch in the specified directory for the specified regex-based filepattern.

Parameters

- dir (str) – Directory to watch in
- filePattern (str) – Regex-based filepattern to watch for
- minFileSize (int) – Minimum file size necessary to consider the file is wholly written. Below this size the filewatcher will assume file is paritally written and continue to wait.
- demoStep (int) – If non-zero then it will space out file notifications by demoStep seconds. This is used when the image files are pre-existing but we want to simulate as if the arrive from the scanner every few seconds (demoStep seconds).
waitForFile(self, filename: str, timeout: int = 0, timeCheckIncrement: int = 1) → Optional[str]

Wait for a specific filename to be created in the directory specified in initFileNotifier.

Parameters

- **filename** – Name of File to watch for creation. If filename includes a path it must match that specified in initFileNotifier.
- **timeout** – Max number of seconds to watch for the file creation. If timeout expires before the file is created then None will be returned
- **timeCheckIncrement** – Time interval (secs) to check if file exists in case file creation events are somehow missed.

Returns

The filename of the created file (same as input arg) or None if timeout expires

class rtCommon.fileWatcher.FileNotifyHandler(q, patterns)

Bases: watchdog.events.PatternMatchingEventHandler

Handler class that will receive the watchdog notifications. It will queue the notifications into the queue provided during the init function.

```python
on_created(self, event)
```

```python
on_modified(self, event)
```

class rtCommon.fileWatcher.InotifyFileWatcher

Version of FileWatcher for Linux using Inotify interface.

```python
__del__(self)
```

initFileNotifier(self, dir: str, filePattern: str, minFileSize: int, demoStep: int = 0) → None

Initialize the file watcher to watch for files in the specified directory. Note: inotify doesn’t use filepatterns

Parameters

- **dir** (str) – Directory to watch in
- **filePattern** (str) – ignored by inotify implementation
- **minFileSize** (int) – Minimum file size necessary to consider the file is wholly written. Below this size the filewatcher will assume file is partially written and continue to wait.
- **demoStep** (int) – If non-zero then it will space out file notifications by demoStep seconds. This is used when the image files are pre-existing but we want to simulate as if the arrive from the scanner every few seconds (demoStep seconds).

waitForFile(self, filename: str, timeout: int = 0, timeCheckIncrement: int = 1) → Optional[str]

Wait for a specific filename to be created in the directory specified in initFileNotifier.

Parameters

- **filename** – Name of File to watch for creation. If filename includes a path it must match that specified in initFileNotifier.
- **timeout** – Max number of seconds to watch for the file creation. If timeout expires before the file is created then None will be returned
- **timeCheckIncrement** – Time interval (secs) to check if file exists in case file creation events are somehow missed.

Returns

The filename of the created file (same as input arg) or None if timeout expires
**notifyEventLoop**(self)

Thread function which gets notifications and queues them in the fileNotifyQ

### 8.1.1.16 rtCommon.imageHandling

This script includes all of the functions that are needed (1) to transfer dicom files back and forth from the cloud and (2) to convert the dicom files to nifti files, which is a file format that is better for data analyses.

**Module Contents**

**Functions**

- `getDicomFileName(cfg, scanNum, fileNum)`
  
  This function takes in different variables (which are both specific to the specific

- `anonymizeDicom(dicomImg)`
  
  This function takes in the dicom image that you read in and deletes

- `readDicomFromFile(filename)`
  
  This function takes the path/name of the dicom file of interest and reads it.

- `writeDicomFile(dicomImg, filename)`
  
  This function writes a dicomImg and the path/name of the file to write to.

- `writeDicomToBuffer(dicomImg)`
  
  This function writes dicom data to binary mode so that it can be transferred

- `readDicomFromBuffer(data) → pydicom.dataset.FileDataset`
  
  This function reads data that is in binary mode and then converts it into a

- `readRetryDicomFromDataInterface(dataInterface, filename, timeout=5)`
  
  This function is waiting and watching for a dicom file to be sent to the cloud

- `parseDicomVolume(dicomImg, sliceDim)`
  
  The raw dicom file coming from the scanner will be a 2-dimensional picture

- `getDicomAcquisitionTime(dicomImg) → datetime.datetime.time`
  
  Returns the acquisition time as a datetime.time

- `getDicomRepetitionTime(dicomImg) → float`
  
  Returns the TR repetition time in seconds

- `dicomTimeToNextTr(dicomImg, clockSkew, now=None)`
  
  Based on Dicom header. Returns seconds to next TR start

- `bidsIncrementalTimeToNextTr(bidsIncremental, clockSkew, now=None)`
  
  Based on BidsIncremental header. Returns seconds to next TR start

- `getAxesForTransform(startingDicomFile, cfg)`
  
  This function takes a single dicom file (which can be the first file) and

- `getTransform(target_orientation, dicom_orientation)`
  
  This function calculates the right transformation needed to go from the original

- `saveAsNiftiImage(dicomDataObject, fullNiftiFilename, cfg, reference)`
  
  This function takes in a dicom data object written in bytes, what you expect

- `convertDicomFileToNifti(dicomFilename, niftiFilename)`
  
  Given an in-memory dicomImg, convert it to an in-memory niftiImg.

- `niftiToMem(niftiImg)`
  
  Fully load Nifti image into memory and remove any file-backing.

- `readNifti(niftiFilename, memCached=True)`

8.1. rtCommon
rtCloud

Attributes

**binPath**

**attributesToAnonymize**

rtCommon.imageHandling.binPath

rtCommon.imageHandling.getDicomFileName(cfg, scanNum, fileNum)

This function takes in different variables (which are both specific to the specific scan and the general setup for the entire experiment) to produce the full filename for the dicom file of interest.

Used externally.

rtCommon.imageHandling.attributesToAnonymize = ['PatientID', 'PatientAge', 'PatientBirthDate', 'PatientName', 'PatientSex', 'PatientSize',...]

rtCommon.imageHandling.anonymizeDicom(dicomImg)

This function takes in the dicom image that you read in and deletes lots of different attributes. The purpose of this is to anonymize the dicom data before transferring it to the cloud.

Used externally.

rtCommon.imageHandling.readDicomFromFile(filename)

This function takes the path/name of the dicom file of interest and reads it.

Used internally.

rtCommon.imageHandling.writeDicomFile(dicomImg, filename)

This function takes a dicomImg and the path/name of the file to write to.

Used internally.

rtCommon.imageHandling.writeDicomToBuffer(dicomImg)

This function write dicom data to binary mode so that it can be transferred to the cloud, where it again becomes a dicom. This is needed because files are transferred to the cloud in the following manner: dicom from scanner –> binary file –> transfer to cloud –> dicom file

Used internally.

rtCommon.imageHandling.readDicomFromBuffer(data) → pydicom.dataset.FileDataset

This function reads data that is in binary mode and then converts it into a structure that can be read as a dicom file. This is necessary because files are transferred to the cloud in the following manner: dicom from scanner –> binary file –> transfer to cloud –> dicom file

Used internally.

rtCommon.imageHandling.readRetryDicomFromDataInterface(dataInterface, filename, timeout=5)

This function is waiting and watching for a dicom file to be sent to the cloud from the scanner. It does this by calling the ‘watchFile()’ function in the ‘dataInterface.py’

Used externally (and internally).

:param dataInterface: A dataInterface to make calls on
:param filename: Dicom filename to watch for and read when available
:param timeout: Max number of seconds to wait for file to be available

Returns  The dicom image
rtCommon.imageHandling.parseDicomVolume(dicomImg, sliceDim)

The raw dicom file coming from the scanner will be a 2-dimensional picture made of up multiple image slices that are tiled together. This function separates the image slices to form a single volume.

Used externally.

rtCommon.imageHandling.getDicomAcquisitionTime(dicomImg) \rightarrow \text{datetime.datetime.time}

Returns the acquisition time as a \text{datetime.time}. Note: day, month and year are not specified

rtCommon.imageHandling.getDicomRepetitionTime(dicomImg) \rightarrow \text{float}

Returns the TR repetition time in seconds

rtCommon.imageHandling.dicomTimeToNextTr(dicomImg, clockSkew, now=None)

Based on Dicom header. Returns seconds to next TR start

rtCommon.imageHandling.bidsIncrementalTimeToNextTr(bidsIncremental, clockSkew, now=None)

Based on BidsIncremental header. Returns seconds to next TR start

rtCommon.imageHandling.getAxesForTransform(startingDicomFile, cfg)

This function takes a single dicom file (which can be the first file) and the config file to obtain the target_orientation (in nifti space) and the dicom_orientation (in the original space).

NOTE: You only need to run this function once to obtain the target and dicom orientations. You can save and load these variables so that ‘getTransform()’ is hard coded.

Used externally.

rtCommon.imageHandling.getTransform(target_orientation, dicom_orientation)

This function calculates the right transformation needed to go from the original axis space (dicom_orientation) to the target axis space in nifti space (target_orientation).

Used externally.

rtCommon.imageHandling.saveAsNiftiImage(dicomDataObject, fullNiftiFilename, cfg, reference)

This function takes in a dicom data object written in bytes, what you expect the dicom file to be called (we will use the same name format for the nifti file), and the config file while will have (1) the axes transformation for the dicom file and (2) the header information from a reference scan.

Used externally.

rtCommon.imageHandling.convertDicomFileToNifti(dicomFilename, niftiFilename)

rtCommon.imageHandling.niftiToMem(niftiImg)

Fully load Nifti image into memory and remove any file-backing. NiftiImage by default contains a pointer to the image file for the data.

rtCommon.imageHandling.readNifti(niftiFilename, memCached=True)

rtCommon.imageHandling.convertDicomImgToNifti(dicomImg, dicomFilename=None)

Given an in-memory dicomImg, convert it to an in-memory niftiImg. Note: due to how nibabel niftiImage works, it is just a pointer to a file on disk, so we can’t delete the niftiFile while niftiImage is in use.
8.1.1.17 rtCommon.openNeuro

An interface to access OpenNeuro data and metadata. It can download and cache OpenNeuro data for playback.

Module Contents

Classes

class rtCommon.openNeuro.OpenNeuroCache(cachePath='/tmp/openneuro/')

getCachePath(self)

getS3Client(self)
   Returns an s3 client in order to reuse the same s3 client without always creating a new one. Not thread safe currently.

getDatasetList(self, refresh=False)
   Returns a list of all datasets available in OpenNeuro S3 storage. See https://openneuro.org/public/datasets for datasets info. Alternate method to access from a command line call: [aws s3 –no-sign-request ls s3://openneuro.org/]

isValidAccessionNumber(self, dsAccessionNum)

getSubjectList(self, dsAccessionNum)
   Returns a list of all the subjects in a dataset

   Parameters dsAccessionNum – accession number of dataset to lookup

   Returns  list of subjects in that dataset

getAddression(self, dsAccessionNum)
   Returns the dataset description file as a python dictionary

getReadme(self, dsAccessionNum)
   Return the contents of the dataset README file. Downloads toplevel dataset files if needed.

getArchivePath(self, dsAccessionNum)
   Returns the directory path to the cached dataset files

downloadData(self, dsAccessionNum, downloadWholeDataset=False, **entities)
   This command will sync the specified portion of the dataset to the cache directory. Note: if only the accessionNum is supplied then it will just sync the top-level files. Sync doesn’t re-download files that are already present in the directory. Consider using –delete which removes local cache files no longer on the remote.

   Parameters

   • dsAccessionNum – accession number of the dataset to download data for.

   • downloadWholeDataset – boolean, if true all files in the dataset will be downloaded.

   • entities – BIDS entities (subject, session, task, run, suffix) that define the particular subject/run of the data to download.

   Returns  Path to the directory containing the downloaded dataset data.
**8.1.1.18 rtCommon.openNeuroService**

A command-line service to be run where the OpenNeuro data is downloaded and cached. This service instantiates a BidsInterface object for serving the data back to the client running in the cloud. It connects to the remote projectServer. Once a connection is established it waits for requests and invokes the BidsInterface functions to handle them.

**Module Contents**

**Classes**

<table>
<thead>
<tr>
<th>OpenNeuroService</th>
<th>A class that implements the OpenNeuroService by instantiating a BidsInterface, connecting</th>
</tr>
</thead>
</table>

**Attributes**

<table>
<thead>
<tr>
<th>currPath</th>
</tr>
</thead>
<tbody>
<tr>
<td>rootPath</td>
</tr>
<tr>
<td>connectionArgs</td>
</tr>
</tbody>
</table>

rtCommon.openNeuroService.currPath

rtCommon.openNeuroService.rootPath

class rtCommon.openNeuroService.OpenNeuroService(args, webSocketChannelName='wsData')

A class that implements the OpenNeuroService by instantiating a BidsInterface, connecting to the remote projectServer and servicing requests to the BidsInterface.

runDetached(self)

Starts the receiver in its own thread.

rtCommon.openNeuroService.connectionArgs

**8.1.1.19 rtCommon.projectServer**

Main (command-line) program for running the projectServer. Instantiates both the web interface and an RPC server for handling client script commands.
Classes

- **ProjectServer**
The main server for running a project. This server starts both the web server and an RPC server.

Attributes

- **currPath**
- **rootPath**
- **argParser**

8.1.1.20 *rtCommon.projectServerRPC*

This module provides the RPC server that provides communication services to the experiment script. Note: When using services local to the projectServer, RPCs call do one hop, client -> rpyc server (method) When using remote services RPC calls traverse two links, client -> rpyc server -> (via websockets) remote service
Functions

```python
startRPCThread(rpcService, hostname=None, port=12345)
```
This function starts the Project RPC server for communication between

```python
class rtCommon.projectServerRPC.ProjectRPCService(dataRemote=False, subjectRemote=False, webUI=None)
    Bases: rpyc.Service
    Provides Remote Procedure Call service for the experimenter’s script. This service runs in the projectServer to receive and handle RPC requests from the experimenter script. It makes available to the client a DataInterface, SubjectInterface and WebInterface.
    exposed_DataInterface
    exposed_SubjectInterface
    exposed_BidsInterface
    exposed_WildDisplayInterface
    exposed_ExampleInterface
    exposed_isDataRemote(self)
    exposed_isSubjectRemote(self)
    static registerDataCommFunction(commFunction)
        Register the function call to forward an RPC data requests over websockets. This is the communication for the second hop (described above) to the remote service.
    static registerSubjectCommFunction(commFunction)
        Register the function call to forward an RPC subject requests over websockets. This is the communication for the second hop (described above) to the remote service.
    on_connect(self, conn)
    on_disconnect(self, conn)
```

```python
rtCommon.projectServerRPC.startRPCThread(rpcService, hostname=None, port=12345)
```
This function starts the Project RPC server for communication between the projectServer and the experiment script. IT DOES NOT RETURN.

```python
class rtCommon.projectServerRPC.RPCHandlers(ioLoopInst, webDisplayInterface)
    Class for websocket RPC handlers. This class handles the second hop described in note below, namely from rpyc server to the remote service via websockets.
    Note: When using local services, RPC call do one hop, client -> rpyc server object/method When using remote services RPC calls traverse two links, client -> rpyc server -> (via websockets) remote service
    dataWsCallback(self, client, message)
        Callback for requests sent to remote service over the wsData channel
    subjectWsCallback(self, client, message)
        Callback for requests sent to remote service over the wsSubject channel
```

8.1. rtCommon
rtCloud

dataRequest(self, cmd, timeout=60)
    Function to initiate an outgoing data request from the RPC server to a remote service

subjectRequest(self, cmd, timeout=60)
    Function to initiate an outgoing subject request from the RPC server to a remote service

close_pending_requests(self, channelName)
    Close out all pending RPC requests when a connection is disconnected

setError(self, errStr)
    Set an error message in the user’s browser window

handleRPCRequest(self, channelName, cmd, timeout=60)
    Process RPC requests using websocket RequestHandler to send the request

8.1.1.21 rtCommon.projectUtils

This module contains utility functions used internally by the rtcloud services

Module Contents

Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>watchForExit()</td>
<td>Create a thread which will detect if the parent process exited by</td>
</tr>
<tr>
<td>processShouldExitThread()</td>
<td>If this client was spawned by a parent process, then by listening on</td>
</tr>
<tr>
<td>formatFileData(filename, data)</td>
<td>Convert raw bytes to a specific memory format such as dicom or matlab data</td>
</tr>
<tr>
<td>login(serverAddr, username, password, testMode=False)</td>
<td>Logs in to a web service, prompting user for username/password as needed,</td>
</tr>
<tr>
<td>checkSSLCertAltName(certFilename, altName)</td>
<td>Check if altName is list as an alternate server name in the ssl certificate</td>
</tr>
<tr>
<td>makeSSLCertFile(serverName)</td>
<td></td>
</tr>
</tbody>
</table>

rtCommon.projectUtils.watchForExit()
    Create a thread which will detect if the parent process exited by reading from stdin, when stdin is closed exit this process.

rtCommon.projectUtils.processShouldExitThread()
    If this client was spawned by a parent process, then by listening on stdin we can tell that the parent process exited when stdin is closed. When stdin is closed we can exit this process as well.

rtCommon.projectUtils.formatFileData(filename, data)
    Convert raw bytes to a specific memory format such as dicom or matlab data

rtCommon.projectUtils.login(serverAddr, username, password, testMode=False)
    Logs in to a web service, prompting user for username/password as needed, and returns a session_cookie to allow future requests without logging in.
rtCommon.projectUtils.checkSSLCertAltName(certFilename, altName)
Check if altName is list as an alternate server name in the ssl certificate
rtCommon.projectUtils.makeSSLCertFile(serverName)

8.1.1.22 rtCommon.remoteable

A set of classes that can be subclassed or extended to allow for automatically forwarding methods calls on the subclass to a remote RPC handler.

On cloud side we will have a remoteInstance (with remoteCall stub) that calls the networking crossbar to send the request to the remote. On the remote side we will have a RemoteHandler instance and when messages are received will dispatch them to the handler.

Module Contents

Classes

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remoteable</td>
<td>A class that can be subclassed to allow remote invocation.</td>
</tr>
<tr>
<td>RemoteStub</td>
<td>A remote stub class where none of the attributes of the original class are defined.</td>
</tr>
<tr>
<td>RemoteableExtensible</td>
<td>A class that can be subclassed to allow remote invocation. The remote and local versions</td>
</tr>
<tr>
<td>RemoteHandler</td>
<td>Class that runs at the remote and as message requests are received they are dispatched</td>
</tr>
</tbody>
</table>

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>defaultRpcTimeout</td>
<td></td>
</tr>
</tbody>
</table>

rtCommon.remoteable.defaultRpcTimeout = 60

class rtCommon.remoteable.Remoteable(isRemote=False)
Bases: object
A class that can be subclassed to allow remote invocation. When isRemote is True it returns a remote stub instance, when false it returns the real instance

class rtCommon.remoteable.RemoteStub(classType, isRemote=True)
Bases: object
A remote stub class where none of the attributes of the original class are defined. Therefore __getattr__ will be called for all attributes (i.e. intercepting normal calls) and this class overrides __getattr__ to forward the call request to a remote instance via the registered communication channel function.

setRPCTimeout(self, timeout)

registerCommFunction(self, commFunction)
remoteCall(self, attribute, *args, **kwargs) → any

__getattr__(self, name)

class rtCommon.remoteable.RemoteableExtensible(isRemote=False)
    Bases: object
    A class that can be subclassed to allow remote invocation. The remote and local versions are the same class type (not a stub) and in the remote instance case attributes can be registerd as ‘local’ meaning calls to them will be handled local, all other calls would be sent to the remote instance.
    isRunningRemote(self)
    setRPCTimeout(self, timeout)
    registerCommFunction(self, commFunction)
    remoteCall(self, attribute, *args, **kwargs) → any
    addLocalAttributes(self, methods)
    __getattr__(self, name)
        Return getattr(self, name).

class rtCommon.remoteable.RemoteHandler
    Class that runs at the remote and as message requests are received they are dispatched to this class for processing.

    registerClassInstance(self, classType, classInstance)
    registerClassNameInstance(self, className, classInstance)
    runRemoteCall(self, callDict)

8.1.1.23 rtCommon.resample

Module Contents

rtCommon.resample.image_to_resample
rtCommon.resample.image_reference
rtCommon.resample.resampled_image
rtCommon.resample.base_name

8.1.1.24 rtCommon.scannerDataService

A command-line service to be run where the scanner data is generated (i.e. the control room). This service instantiates a DataInterface and BidsInterface object for serving the data back to the client running in the cloud. It connects to the remote projectServer. Once a connection is established it waits for requets and invokes the DataInterface or BidsInterface functions to handle them.
Module Contents

Classes

ScannerDataService

Attributes

currPath

rootPath

defaultAllowedDirs

defaultAllowedTypes

connectionArgs

rtCommon.scannerDataService.currPath
rtCommon.scannerDataService.rootPath
rtCommon.scannerDataService.defaultAllowedDirs = ['/tmp', '/data']
rtCommon.scannerDataService.defaultAllowedTypes = ['.dcm', '.mat', '.txt']
class rtCommon.scannerDataService.ScannerDataService(args, webSocketChannelName='wsData')
rtCommon.scannerDataService.connectionArgs

8.1.1.25 rtCommon.serialization

Module Contents
## Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>encodeByteTypeArgs(cmd) → dict</code></td>
<td>Check if any args are of type 'bytes' and if so base64 encode them.</td>
</tr>
<tr>
<td><code>decodeByteTypeArgs(cmd) → dict</code></td>
<td>Decodes rpc args that were previously encoded with encodeByteTypeArgs.</td>
</tr>
<tr>
<td><code>npToPy(data)</code></td>
<td>Converts components in data that are numpy types to regular python types.</td>
</tr>
<tr>
<td><code>encodeMessageData(message, data, compress)</code></td>
<td>b64 encode binary data in preparation for sending. Updates the message header as needed.</td>
</tr>
<tr>
<td><code>decodeMessageData(message)</code></td>
<td>Given a message encoded with encodeMessageData (above), decode that message.</td>
</tr>
<tr>
<td><code>generateDataParts(data, msg, compress)</code></td>
<td>A python &quot;generator&quot; that, for data &gt; 10 MB, will create multipart data messages and returns the data bytes.</td>
</tr>
<tr>
<td><code>unpackDataMessage(msg)</code></td>
<td>Handles receiving multipart (an singlepart) data messages and returns the data bytes.</td>
</tr>
</tbody>
</table>

## Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>multiPartDataCache</code></td>
<td></td>
</tr>
<tr>
<td><code>dataPartSize</code></td>
<td></td>
</tr>
</tbody>
</table>

## rtCommon.serialization

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>multiPartDataCache</code></td>
<td></td>
</tr>
<tr>
<td><code>dataPartSize</code></td>
<td></td>
</tr>
</tbody>
</table>

The `npToPy(data)` function converts components in data that are numpy types to regular python types. It uses recursive calls to convert nested data structures.

The `encodeMessageData(message, data, compress)` function b64 encodes binary data in preparation for sending. It updates the message header as needed.

The `decodeMessageData(message)` function decodes a message that was previously encoded with `encodeMessageData` (above).

The `generateDataParts(data, msg, compress)` function is a python "generator" that, for data > 10 MB, will create multipart data messages.

The `unpackDataMessage(msg)` function handles receiving multipart (or single part) data messages and returns the data bytes.
rtCommon.serialization.decodeMessageData(message)

Given a message encoded with encodeMessageData (above), decode that message. Validate and retrieve original bytes. :param message: encoded message to decode :type message: dict

Returns The byte data of the original message from the sender

rtCommon.serialization.generateDataParts(data, msg, compress)

A python “generator” that, for data > 10 MB, will create multi-part messages of 10MB each to send the data incrementally. :param data: data to send :type data: bytes :param msg: message header for the request :type msg: dict :param compress: whether to compress the data before sending :type compress: bool

Returns Repeated calls return the next partial message to be sent until None is returned

rtCommon.serialization.unpackDataMessage(msg)

Handles receiving multipart (an singlepart) data messages and returns the data bytes. In the case of multipart messages a data cache is used to store intermediate parts until all parts are received and the final data can be reconstructed. :param msg: Potentially on part of a multipart message to unpack :type msg: dict

Returns None if not all multipart messages have been received yet, or Data bytes if all multipart messages have been received.

8.1.1.26 rtCommon.structDict

StructDictClass - contains classes StructDict and MatlabStructDict to make it possible to access a dictionary with syntax struct.field.

Module Contents

Classes

<table>
<thead>
<tr>
<th>StructDict</th>
<th>Class that adds a structure type syntax to dictionaries.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MatlabStructDict</td>
<td>Subclass dictionary so that elements can be accessed either as dict['key']</td>
</tr>
</tbody>
</table>

Functions

<table>
<thead>
<tr>
<th>copy_toplevel(data)</th>
</tr>
</thead>
<tbody>
<tr>
<td>recurseCreateStructDict(data)</td>
</tr>
<tr>
<td>recurseSDtoDict(data)</td>
</tr>
<tr>
<td>convertStructuredArrayToDict(sArray)</td>
</tr>
<tr>
<td>isStructuredArray(var) → bool</td>
</tr>
</tbody>
</table>

8.1. rtCommon
Attributes

```python
class rtCommon.structDict.StructDict
    Bases: dict
    Class that adds a structure type syntax to dictionaries, i.e. ‘dict.field’ will invoke dict[‘field’]
    __getattr__(self, key)
        Implement getattr to support syntax “data.field”
    __setattr__(self, key, val)
        Implement setattr to support syntax “data.field=x”
    __delattr__(self, key)
        Implement delattr to support syntax “del data.field”
    __getstate__(self)
        Needed for pickling, return the underlying dictionary
    __setstate__(self, dict_entries)
        Needed for pickling, set the underlying dictionary
    copy(self)
        D.copy() -> a shallow copy of D
rtCommon.structDict.copy_toplevel(data)
rtCommon.structDict.recurseCreateStructDict(data)
    Given a recursive dictionary, i.e. a dictionary that has child dictionaries or lists of dictionaries, convert each child dictionary to a StructDict.
rtCommon.structDict.structDictType
rtCommon.structDict.recurseSDtoDict(data)
    Given a recursive StructDict, i.e. that has child StructDicts or lists of StructDict, convert each child StructDict to a Dictionary.
class rtCommon.structDict.MatlabStructDict(dictionary, name=None)
    Bases: StructDict
    Subclass dictionary so that elements can be accessed either as dict[‘key’] of dict.key. If elements are of type NumPy structured arrays, convert them to dictionaries and then to MatlabStructDict also.
    __getattr__(self, key)
        Implement getattr to support syntax x=data.field
    __setattr__(self, key, val)
        Implement setattr to support syntax data.field=x
    copy(self)
        D.copy() -> a shallow copy of D
    fields(self)
        list all fields including subfields of the special ‘name’ field
```
rtCloud

rtCommon.structDict.convertStructuredArrayToDict(sArray)
Convert a NumPy structured array to a dictionary.

rtCommon.structDict.isStructuredArray(var) → bool
Return True if var is a numpy structured array

8.1.1.27 rtCommon.subjectInterface

SubjectInterface is a client interface (i.e. for the experiment script running in the cloud) that facilitates interaction with the subject in the MRI scanner, such as sending classification results to drive the subject display, or receiving their responses (e.g. button-box).

To support RPC calls from the client, there will be two instances of SubjectInterface, one at the cloud projectServer which is a stub to forward requests (started with subjectRemote=True), and another at the presentation computer, run as a service and with subjectRemote=False.

The subjectInterface instance can also be instantiated within the projectServer if the projectServer and presentation computer run on the same system.

Module Contents

Classes

<table>
<thead>
<tr>
<th>SubjectInterface</th>
<th>Provides functions for sending feedback and receiving responses from the subject in the scanner.</th>
</tr>
</thead>
</table>

class rtCommon.subjectInterface.SubjectInterface(subjectRemote=False)

Bases: rtCommon.remoteable.RemoteableExtensible

Provides functions for sending feedback and receiving responses from the subject in the scanner.

If subjectRemote=True, then the RemoteExtensible parent class takes over and forwards all requests to a remote server via a callback function registered with the RemoteExtensible object. In that case none of the methods below will be locally invoked.

If subjectRemote=False, then the methods below will be invoked locally and the RemoteExtensible parent class is inoperable (i.e. does nothing).

When classification results are received from the experiment script they are placed in a queue which the presentation script can then access. The presentation script can wait on the queue for new results to arrive.

setResult(self, runId: int, trId: int, value: float, onsetTimeDelayMs: int = 0) → None

When setResult is called by the experiment script it queues the result for the presentation script to later read and use to provide subject feedback. :param runId: experiment specific identifier of the run :param trId: volume number of the dicom within a run :param value: the classification result from processing the dicom image for this TR :param onsetTimeDelayMs: time in milliseconds to wait before presenting the feedback stimulus

setResultDict(self, values: dict, onsetTimeDelayMs: int = 0) → None

Same as setResult except the caller can provide a dictionary with whatever entries are desired to be used at the subjectService. :param values: a dictionary with the desired values to send for this TR :param onsetTimeDelayMs: time in milliseconds to wait before presenting the feedback stimulus
**setMessage** *(self, message: str) → None*
 Updates the message displayed to the subject

**getResponse** *(self, runId: int, trld: int)*
 Retrieve the subject response, used by the classification script. See *note* above - these local versions of the function are just for testing or as a place holder when no external subjectInterface is used.

**getAllResponses** *(self)*
 Retrieve all subject responses since the last time this call was made

**dequeueResult** *(self, block: bool = False, timeout: int = None) → float*
 Return the next result value sent by the experiment script. Used by the presentation script. See *note* above - these local versions of the function are just for testing or as a place holder when no external version is used.

### 8.1.1.28 rtCommon.subjectService

An example command-line service to be run at the presentation computer to receive classification results from the classification script.

This service instantiates a SubjectInterface for serving sending/receiving subject feedback to the projectServer in the cloud. It connects to the remote projectServer. Once a connection is established it waits for requests and invokes the SubjectInterface functions to handle them.

Note: This service is intended as an example. In practice this subjectInterface would likely be instantiated within the presentation script and there it would use WsRemoteService to connect this instance to the remote projectServer where the classification is script running.

### Module Contents

#### Classes

**SubjectService**

**Attributes**

- **currPath**
- **rootPath**
- **connectionArgs**

**rtCommon.subjectService.currPath**

**rtCommon.subjectService.rootPath**

**class rtCommon.subjectService.SubjectService** *(args, WebSocketChannelName='wsSubject')*

**runDetached** *(self)*

Starts the receiver in it's own thread.
rtCommon.subjectService.connectionArgs

8.1.1.29 rtCommon.utils

Utils - various utilities for rtfMRI

Module Contents

Classes

DebugLevels
### Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>parseMatlabStruct(top_struct)</code></td>
<td>Load matlab data file and convert it to a MatlabStructDict object for</td>
</tr>
<tr>
<td><code>loadMatFile(filename)</code></td>
<td><code>rtCommon.structDict.MatlabStructDict</code></td>
</tr>
<tr>
<td><code>loadMatFileFromBuffer(data)</code></td>
<td><code>rtCommon.structDict.MatlabStructDict</code></td>
</tr>
<tr>
<td><code>find(A: numpy.ndarray)</code></td>
<td><code>numpy.ndarray</code> Find nonzero elements of A in flat &quot;C&quot; row-major indexing order</td>
</tr>
<tr>
<td><code>loadConfigFile(filename)</code></td>
<td></td>
</tr>
<tr>
<td><code>flatten_1Ds(M)</code></td>
<td></td>
</tr>
<tr>
<td><code>dateStr30(timeval)</code></td>
<td></td>
</tr>
<tr>
<td><code>md5SumFile(filePath)</code></td>
<td></td>
</tr>
<tr>
<td><code>findNewestFile(filepath, filepattern)</code></td>
<td>Find newest file matching pattern according to filesystem creation time.</td>
</tr>
<tr>
<td><code>copyFileWildcard(src, dst)</code></td>
<td></td>
</tr>
<tr>
<td><code>fileCount(dir, pattern)</code></td>
<td></td>
</tr>
<tr>
<td><code>writeFile(filename, data, binary=True)</code></td>
<td></td>
</tr>
<tr>
<td><code>readFile(filename, binary=True)</code></td>
<td></td>
</tr>
<tr>
<td><code>deleteFilesFromList(fileList)</code></td>
<td></td>
</tr>
<tr>
<td><code>deleteFolder(dir)</code></td>
<td></td>
</tr>
<tr>
<td><code>deleteFolderFiles(dir, recursive=True)</code></td>
<td></td>
</tr>
<tr>
<td><code>runCmdCheckOutput(cmd, outputRegex)</code></td>
<td></td>
</tr>
<tr>
<td><code>demoDelay(demoStep, prevEventTime=None)</code></td>
<td>Provides a delay of demoStep seconds from the previous event time (i.e. sleeps)</td>
</tr>
<tr>
<td><code>installLoggers(consoleLevel, fileLevel, fileName=None)</code></td>
<td></td>
</tr>
<tr>
<td><code>getGitCodeId()</code></td>
<td></td>
</tr>
<tr>
<td><code>stringPartialFormat(text, tag, val)</code></td>
<td><code>str</code></td>
</tr>
<tr>
<td><code>trimDictBytes(msg, trimSize=64)</code></td>
<td></td>
</tr>
<tr>
<td><code>getTimeToNextTR(lastTrTime, trRepSec, nowTime, clockSkew)</code></td>
<td>Returns seconds to next TR start time</td>
</tr>
<tr>
<td><code>dtimeToSeconds(valTime)</code></td>
<td><code>float</code></td>
</tr>
<tr>
<td><code>calcAvgRoundTripTime(pingFunc)</code></td>
<td>Returns average round trip time in seconds</td>
</tr>
</tbody>
</table>
Attributes

**gitCodeId**

```python
rtCommon.utils.parseMatlabStruct(top_struct) → rtCommon.structDict.MatlabStructDict
```

Load matlab data file and convert it to a MatlabStructDict object for easier python access. Expect only one
substructure array, and use that one as the name variable in MatlabStructDict. Return the MatlabStructDict
object.

```python
rtCommon.utils.loadMatFile(filename: str) → rtCommon.structDict.MatlabStructDict
```

```python
rtCommon.utils.loadMatFileFromFileBuffer(data) → rtCommon.structDict.MatlabStructDict
```

```python
rtCommon.utils.find(A: numpy.ndarray) → numpy.ndarray
```

Find nonzero elements of A in flat “C” row-major indexing order but sorted as in “F” column indexing order.

```python
rtCommon.utils.loadConfigFile(filename)
```

```python
rtCommon.utils.flatten_1Ds(M)
```

```python
rtCommon.utils.dateStr30(timeval)
```

```python
rtCommon.utils.md5SumFile(filePath)
```

```python
rtCommon.utils.findNewestFile(filepath, filepattern)
```

Find newest file matching pattern according to filesystem creation time. Return the filename.

```python
rtCommon.utils.copyFileWildcard(src, dst)
```

```python
rtCommon.utils.fileCount(dir, pattern)
```

```python
rtCommon.utils.writeFile(filename, data, binary=True)
```

```python
rtCommon.utils.readFile(filename, binary=True)
```

```python
rtCommon.utils.deleteFilesFromList(fileList)
```

```python
rtCommon.utils.deleteFolder(dir)
```

```python
rtCommon.utils.deleteFolderFiles(dir, recursive=True)
```

```python
rtCommon.utils.runCmdCheckOutput(cmd, outputRegex)
```

```python
rtCommon.utils.demoDelay(demoStep, prevEventTime=None)
```

Provides a delay of demoStep seconds from the previous event time (i.e. sleeps) Given demoStep in seconds,
calculate how long to sleep until the next clock cycle will be reached that is an even value of demoStep. Then
sleep that amount of time. If prevEventTime is specified and we are more than 1 demo step since the prevEvent
then don’t sleep.

```python
class rtCommon.utils.DebugLevels
```

```python
L1 = 19
```

```python
L2 = 18
```

```python
L3 = 17
```
rtCloud

L4 = 16
L5 = 15
L6 = 14
L7 = 13
L8 = 12
L9 = 11
L10 = 10

rtCommon.utils.installLoggers(consoleLevel, fileLevel, filename=None)

rtCommon.utils.gitCodeId

rtCommon.utils.getGitCodeId()

rtCommon.utils.stringPartialFormat(text, tag, val) → str

rtCommon.utils.trimDictBytes(msg, trimSize=64)

rtCommon.utils.getTimeToNextTR(lastTrTime, trRepSec, nowTime, clockSkew) → float

  Returns seconds to next TR start time
  :param lastTrTime - datetime.time of the start of last TR:
  :param trRepSec - repetition time in seconds between TRs:
  :param nowTime - current time as datetime.time struct:
  :param clockSkew - seconds that scanner clock is ahead of caller clock:

  Returns seconds to next TR start (as float)

rtCommon.utils.dtimeToSeconds(valTime) → float

  Given a datetime.time return seconds.fraction since day beginning

rtCommon.utils.calcAvgRoundTripTime(pingFunc)

  Returns average round trip time in seconds

8.1.1.30 rtCommon.validationUtils

ValidationUtils - utils to help validate that arrays and data structures match. For example in testing and comparing to a known-good run from matlab.

Module Contents
### Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>compareArrays</code></td>
<td><code>A: numpy.ndarray, B: numpy.ndarray</code> → dict</td>
<td>Compute element-wise percent difference between <code>A</code> and <code>B</code></td>
</tr>
<tr>
<td><code>areArraysClose</code></td>
<td><code>A: numpy.ndarray, B: numpy.ndarray, mean_limit=0.01, stddev_limit=1.0</code> → bool</td>
<td>Compare to arrays element-wise and compute the percent difference.</td>
</tr>
<tr>
<td><code>compareMatStructs</code></td>
<td><code>A: rtCommon.structDict.MatlabStructDict, B: rtCommon.structDict.MatlabStructDict, field_list=None</code> → dict</td>
<td>For each field, not like <code>__*__</code>, walk the fields and compare the values.</td>
</tr>
<tr>
<td><code>isMeanWithinThreshold</code></td>
<td><code>cmpStats: dict, threshold: float</code> → bool</td>
<td>Examine all <code>mean</code> stats in dictionary and compare to threshold value</td>
</tr>
<tr>
<td><code>compareMatFiles</code></td>
<td><code>filename1: str, filename2: str</code> → dict</td>
<td>Load both matlab files and call <code>compareMatStructs</code>.</td>
</tr>
<tr>
<td><code>pearsons_mean_corr</code></td>
<td><code>A: numpy.ndarray, B: numpy.ndarray</code></td>
<td></td>
</tr>
</tbody>
</table>

### Attributes

- `numpyAllNumCodes`
- `StatsEqual`
- `StatsNotEqual`

```python
rtCommon.validationUtils.numpyAllNumCodes
rtCommon.validationUtils.StatsEqual
rtCommon.validationUtils.StatsNotEqual
```

#### rtCommon.validationUtils.compareArrays

```python
rtCommon.validationUtils.compareArrays(A: numpy.ndarray, B: numpy.ndarray) → dict
```

Compute element-wise percent difference between `A` and `B`. Return the mean, max, stddev, histocounts, histobins in a Dict.

#### rtCommon.validationUtils.areArraysClose

```python
rtCommon.validationUtils.areArraysClose(A: numpy.ndarray, B: numpy.ndarray, mean_limit=0.01, stddev_limit=1.0) → bool
```

Compare to arrays element-wise and compute the percent difference. Return True if the mean and stddev are within the supplied limits. Default limits: `{mean: .01, stddev: 1.0}`, i.e. no stddev limit by default.

#### exception

- `rtCommon.validationUtils.StructureMismatchError`

  Bases: `ValueError`

  Inappropriate argument value (of correct type).

```python
rtCommon.validationUtils.compareMatStructs(A: rtCommon.structDict.MatlabStructDict, B: rtCommon.structDict.MatlabStructDict, field_list=None) → dict
```

For each field, not like `__*__`, walk the fields and compare the values. If a field is missing from one of the structs raise an exception. If `field_list` is supplied, then only compare those fields. Return a dict with `{fieldname: stat_results}`.
rtCloud

rtCommon.validationUtils.isMeanWithinThreshold(cmpStats: dict, threshold: float) → bool
Examine all mean stats in dictionary and compare to threshold value

rtCommon.validationUtils.compareMatFiles(filename1: str, filename2: str) → dict
Load both matlab files and call compareMatStructs. Inspect the resulting stats_result to see if any mean is beyond some threshold. Also print out the stats results. Return the result stats.

rtCommon.validationUtils.pearsons_mean_corr(A: numpy.ndarray, B: numpy.ndarray)

8.1.1.31 rtCommon.webDisplayInterface

WebDisplayInterface is a client interface (i.e. for the experiment script running in the cloud) that provides calls that affect what is displayed in the users web browser. It is also used internally within projectServer for setting log and error messages within the web browser.

Module Contents

Classes

WebDisplayInterface

class rtCommon.webDisplayInterface.WebDisplayInterface(ioLoopInst=None)

userLog(self, logStr)
Set a log message in the user log area of the web page

sessionLog(self, logStr)
Set a log message in the session log area of the web page

debugLog(self, logStr)
Set a log message in the debug log area of the web page

setUserError(self, errStr)
Set an error message in the error display area of the web page

setDebugError(self, errStr)
Set an error message in the debug display area of the web page

sendRunStatus(self, statusStr)
Indicate run status in the web page

sendUploadStatus(self, fileStr)

sendConfig(self, config, filename="")
Send the project configurations to the web page

sendPreviousDataPoints(self)
Send previously plotted data points to the web page

plotDataPoint(self, runId, trId, value)
Add a new data point to the web page plots
rtCloud

clearAllPlots(self)
   Clear all data plots in the web page

clearRunPlot(self, runId)
   Clear the data plot for the specified run

getPreviousDataPoints(self)
   Local command to retrieve previously plotted points (doesn’t send to web page)

sendConnStatus(self)
   Send the number of data and subject connections to the web page

wsConnCallback(self, endpoint, cmd)
   Callback function for whenever a new websocket connection is established. Will track the number of connections in order to provide the connection status on the web page interface.

_addResultValue(self, runId, trId, value)
   Track classification result values, used to plot the results in the web browser.

_sendMessageToWeb(self, msg)
   Helper function used by the other methods to send a message to the web page

8.1.1.32 rtCommon.webHttpHandlers

This module provides the callback handlers that the web server will utilize when handling and rendering html page requests.

Module Contents

Classes

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HttpHandler</td>
<td>Generic web handler object that is initialized with the page name to render when called.</td>
</tr>
<tr>
<td>LoginHandler</td>
<td>Renders a login page and authenticates users. Sets a secure-cookie to remember authenticated users.</td>
</tr>
<tr>
<td>LogoutHandler</td>
<td>Clears the secure-cookie so that users will need to re-authenticate.</td>
</tr>
</tbody>
</table>

Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>loadPasswdFile(filename)</td>
<td></td>
</tr>
</tbody>
</table>

storePasswdFile(filename, passwdDict)
Attributes

maxDaysLoginCookieValid

rtCommon.webHttpHandlers.maxDaysLoginCookieValid = 0.5

class rtCommon.webHttpHandlers.HttpHandler
    Bases: tornado.web.RequestHandler
    Generic web handler object that is initialized with the page name to render when called.

    initialize(self, htmlDir, page)
    get_current_user(self)
    get(self)
        Handle a web GET request by returning the appropriate web content to render

class rtCommon.webHttpHandlers.LoginHandler
    Bases: tornado.web.RequestHandler
    Renders a login page and authenticates users. Sets a secure-cookie to remember authenticated users.

    loginAttempts
    loginRetryDelay = 10

    initialize(self, htmlDir, page, testMode)
    get(self)
    post(self)
    checkRetry(self, user)
        Keep a dictionary with one entry per username. Any user not in the passwd file will be entered as ‘invalid_user’. Record login failure count and timestamp for when the next retry is allowed. Reset failed retry count on successful login. Return message with how many seconds until next login attempt is allowed.

class rtCommon.webHttpHandlers.LogoutHandler
    Bases: tornado.web.RequestHandler
    Clears the secure-cookie so that users will need to re-authenticate.

    initialize(self)
    get(self)

rtCommon.webHttpHandlers.loadPasswdFile(filename)
rtCommon.webHttpHandlers.storePasswdFile(filename, passwdDict)
8.1.33 rtCommon.webServer

Web Server module which provides the web user interface to control and monitor experiments

Module Contents

Classes

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web</td>
<td>Cloud service web-interface that is the front-end to the data processing.</td>
</tr>
<tr>
<td>WsBrowserRequestHandler</td>
<td>Command handler for commands that the javascript running in the web browser can call</td>
</tr>
</tbody>
</table>

Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>procOutputReader(proc, lineQueue)</td>
<td>Read output from runSession process and queue into lineQueue for logging</td>
</tr>
<tr>
<td>getCookieSecret(dir)</td>
<td>Used to remember users who are currently logged in.</td>
</tr>
</tbody>
</table>

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>CommonOutputDir</td>
</tr>
<tr>
<td>moduleDir</td>
</tr>
<tr>
<td>rootDir</td>
</tr>
<tr>
<td>logger</td>
</tr>
</tbody>
</table>

rtCommon.webServer.CommonOutputDir = /rtfmriData/
rtCommon.webServer.moduleDir
rtCommon.webServer.rootDir
rtCommon.webServer.logger

class rtCommon.webServer.Web
    Cloud service web-interface that is the front-end to the data processing.
    app
    httpServer
    started = False
    httpPort = 8888
webDir
htmlDir
ioLoopInst
testMode = False
webDisplayInterface

static start(params, cfg, testMode=False)
    Start the web server running. Function does not return.
static addHandlers(handlers)
static stop()
    Stop the web server.
static close()

class rtCommon.webServer.WsBrowserRequestHandler(webDisplayInterface, params, cfg)
Command handler for commands that the javascript running in the web browser can call

_addScript(self, name, path, type)
    Add the experiment script to be connected to the various run button of the web page. These include ‘main-
    Script’ for classification processing, ‘initScript’ for session initialization, and ‘finalizeScript’ for any final
    processing at the end of a session.

on_getDefaultConfig(self)
    Return default configuration settings for the project

on_getDataPoints(self)
    Return data points that have been plotted

on_getRunStatus(self)
    Return run status from the project server

on_clearDataPoints(self)
    Clear all plot datapoints

on_runScript(self, name)
    Run one of the project scripts in a separate process

on_stop(self)
    Stop execution of the currently running project script (only one can run at a time)

on_uploadFiles(self, request)
    Upload files from the dataServer to the cloud computer

_wsBrowserCallback(self, client, message)
    The main message handler for messages received over web sockets from the web page javascript. It will
    parse the message and call the corresponding function above to handle the request.

_runSession(self, cfg, pyScript, tag, logType='run')
    Run the experimenter provided python script as a separate process. Forward the script’s printed output to
    the web page’s log message area.

_uploadFilesHandler(self, request)
    Handle requests from the web interface to upload files to this computer.
_setError(self, errStr)

rtCommon.webServer.procOutputReader(proc, lineQueue)
Read output from runSession process and queue into lineQueue for logging

rtCommon.webServer.getCookieSecret(dir)
Used to remember users who are currently logged in.

8.1.1.34 rtCommon.webSocketHandlers

This module provides classes for handling web socket communication in the web interface.

Module Contents

Classes

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>websocketState</td>
<td>A global static class (really a struct) for maintaining connection and callback information.</td>
</tr>
<tr>
<td>BaseWebSocketHandler</td>
<td>Generic web socket handler. Establishes and maintains a ws connection. Initialized with a callback function that gets called when messages are received on this socket instance.</td>
</tr>
<tr>
<td>DataWebSocketHandler</td>
<td>Sub-class the base handler in order to clean up any outstanding requests on close.</td>
</tr>
<tr>
<td>RejectWebSocketHandler</td>
<td>A web socket handler that rejects connections on the web socket and returns a</td>
</tr>
<tr>
<td>RequestHandler</td>
<td>Class for handling remote requests (such with a remote DataInterface). Each data requests is</td>
</tr>
</tbody>
</table>

Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sendWebSocketMessage(wsName, msg, conn=None)</td>
<td>Send messages from the web server to all clients connected on the specified wsName socket.</td>
</tr>
<tr>
<td>closeAllConnections()</td>
<td></td>
</tr>
<tr>
<td>defaultWebSocketCallback(client, message)</td>
<td></td>
</tr>
</tbody>
</table>

8.1. rtCommon
initialize\(\text{(self, name, callback=\text{None, connNotify=\text{None}})}\)

initialize method is called by Tornado with args provided to the addHandler call

**Parameters**

- **name** – the websocket endpoint name, such as wsData
- **callback** – the handler function to call whenever a client message is received over the connection.
- **connNotify** – the function to call when a connection is opened or closed.

open\(\text{(self)}\)

Called when a new client connection is established

on_close\(\text{(self)}\)

Called when the client connection is closed

on_message\(\text{(self, message)}\)

Called when a message is received from a client connection

class rtCommon.webSocketHandlers.DataWebSocketHandler

Bases: BaseWebSocketHandler

Sub-class the base handler in order to clean up any outstanding requests on close.

on_close\(\text{(self)}\)

Called when the client connection is closed

class rtCommon.webSocketHandlers.RejectWebSocketHandler

Bases: tornado.websocket.WebSocketHandler

A web socket handler that rejects connections on the web socket and returns a pre-configured error with the rejection reason.

initialize\(\text{(self, rejectMsg)}\)

open\(\text{(self)}\)

rtCommon.webSocketHandlers.sendWebSocketMessage\(\text{(wsName, msg, conn=\text{None})}\)

Send messages from the web server to all clients connected on the specified wsName socket.

rtCommon.webSocketHandlers.closeAllConnections()

rtCommon.webSocketHandlers.defaultWebsocketCallback\(\text{(client, message)}\)

class rtCommon.webSocketHandlers.RequestHandler\(\text{(name, ioLoopInst)}\)

Class for handling remote requests (such with a remote DataInterface). Each data requests is given a unique ID and callbacks from the client are matched to the original request and results returned to the corresponding caller.

doRequest\(\text{(self, msg, timeout=\text{None})}\)

Send a request over the web socket, i.e. to the remote FileWatcher. This is typically the only call that a user of this class would make. It is the highest level call of this class, it uses the other methods to complete the request.

prepare_request\(\text{(self, msg)}\)

Prepare a request to be sent, including creating a callback structure and unique ID.
callback(self, client, message)
    Recieve a callback from the client and match it to the original request that was sent.

get_response(self, callId, timeout=None)
    Client calls get_response() to wait for the callback results to be returned.

close_pending_requests(self)
    Close requests and signal any threads waiting for responses.

pruneCallbacks(self)
    Remove any orphaned callback structures that never got a response back.

8.1.1.35 rtCommon.wsRemoteService

An RPC server base class for encapsulating a service class and receiving requests that will call that encapsulated class. This is part of a remote service that communicates with the projectServer.

Module Contents

Classes

WsRemoteService

Functions

isNativeType(var)

parseConnectionArgs()

class rtCommon.wsRemoteService.WsRemoteService(args, channelName)
    remoteHandler
    commLock
    shouldExit = False
    addHandlerClass(self, classType, classInstance)
        Register the class that will handle the received requests via the class type
    addHandlerClassName(self, className, classInstance)
        Register the class that will handle the received requests via the class name
    runForever(self)
        Run the receiver loop. This function doesn’t return.
    static stop()
    static send_response(client, response)
static handle_request(client, message)
  Handle requests from the projectServer. It will call the registered handler to process the request and then return the result back to the projectServer.

static on_message(client, message)
  Main message dispatcher that will get a request from projectServer and start a thread to handle the request.

static on_error(client, error)

static on_close(client, code, reason)

rtCommon.wsRemoteService.isNativeType(var)

rtCommon.wsRemoteService.parseConnectionArgs()
CHAPTER
NINE

INDICES AND TABLES

• genindex
• modindex
• search
<table>
<thead>
<tr>
<th>rtCommon</th>
<th>33</th>
</tr>
</thead>
<tbody>
<tr>
<td>rtCommon.addLogin</td>
<td>33</td>
</tr>
<tr>
<td>rtCommon.bidsArchive</td>
<td>34</td>
</tr>
<tr>
<td>rtCommon.bidsCommon</td>
<td>40</td>
</tr>
<tr>
<td>rtCommon.bidsIncremental</td>
<td>46</td>
</tr>
<tr>
<td>rtCommon.bidsInterface</td>
<td>50</td>
</tr>
<tr>
<td>rtCommon.bidsRun</td>
<td>54</td>
</tr>
<tr>
<td>rtCommon.certsUtils</td>
<td>56</td>
</tr>
<tr>
<td>rtCommon.checkDicomNiftiConversion</td>
<td>56</td>
</tr>
<tr>
<td>rtCommon.clientInterface</td>
<td>57</td>
</tr>
<tr>
<td>rtCommon.dataInterface</td>
<td>58</td>
</tr>
<tr>
<td>rtCommon.dicomToBidsService</td>
<td>62</td>
</tr>
<tr>
<td>rtCommon.errors</td>
<td>62</td>
</tr>
<tr>
<td>rtCommon.exampleInterface</td>
<td>63</td>
</tr>
<tr>
<td>rtCommon.exampleService</td>
<td>64</td>
</tr>
<tr>
<td>rtCommon.fileWatcher</td>
<td>65</td>
</tr>
<tr>
<td>rtCommon.imageHandling</td>
<td>67</td>
</tr>
<tr>
<td>rtCommon.openNeuro</td>
<td>70</td>
</tr>
<tr>
<td>rtCommon.openNeuroService</td>
<td>71</td>
</tr>
<tr>
<td>rtCommon.projectServer</td>
<td>71</td>
</tr>
<tr>
<td>rtCommon.projectServerRPC</td>
<td>72</td>
</tr>
<tr>
<td>rtCommon.projectUtils</td>
<td>74</td>
</tr>
<tr>
<td>rtCommon.remoteable</td>
<td>75</td>
</tr>
<tr>
<td>rtCommon.resample</td>
<td>76</td>
</tr>
<tr>
<td>rtCommon.scannerDataService</td>
<td>76</td>
</tr>
<tr>
<td>rtCommon.serialization</td>
<td>77</td>
</tr>
<tr>
<td>rtCommon.structDict</td>
<td>79</td>
</tr>
<tr>
<td>rtCommon.subjectInterface</td>
<td>81</td>
</tr>
<tr>
<td>rtCommon.subjectService</td>
<td>82</td>
</tr>
<tr>
<td>rtCommon.utils</td>
<td>83</td>
</tr>
<tr>
<td>rtCommon.validationUtils</td>
<td>86</td>
</tr>
<tr>
<td>rtCommon.webDisplayInterface</td>
<td>88</td>
</tr>
<tr>
<td>rtCommon.webHttpHandlers</td>
<td>89</td>
</tr>
<tr>
<td>rtCommon.webServer</td>
<td>91</td>
</tr>
<tr>
<td>rtCommon.websocketHandlers</td>
<td>93</td>
</tr>
<tr>
<td>rtCommon.wsRemoteService</td>
<td>95</td>
</tr>
</tbody>
</table>
Symbols

__del__() (rtCommon.dataInterface.DataInterface method), 59
__del__() (rtCommon.fileWatcher.FileWatcher method), 65
__del__() (rtCommon.fileWatcher.InotifyFileWatcher method), 66
__del__() (rtCommon.fileWatcher.WatchdogFileWatcher method), 65
_delattr__() (rtCommon.structDict.StructDict method), 80
__eq__() (rtCommon.bidsIncremental.BidsIncremental method), 46
__eq__() (rtCommon.bidsRun.BidsRun method), 54
getattr__() (rtCommon.bidsArchive.BidsArchive method), 35
getattr__() (rtCommon.remoteable.RemoteStub method), 76
getattr__() (rtCommon.structDict.MatlabStructDict method), 80
getattr__() (rtCommon.structDict.StructDict method), 80
getattribute__() (rtCommon.clientInterface.WrapRpycObject method), 58
getattribute__() (rtCommon.remoteable.RemoteableExtensible method), 76
_getstate__() (rtCommon.bidsIncremental.BidsIncremental method), 46
_getstate__() (rtCommon.structDict.StructDict method), 80
_setattr__() (rtCommon.structDict.MatlabStructDict method), 80
_setattr__() (rtCommon.structDict.StructDict method), 80
_setstate__() (rtCommon.structDict.StructDict method), 80
__str__() (rtCommon.bidsArchive.BidsArchive method), 35
__str__() (rtCommon.bidsIncremental.BidsIncremental method), 46
addImage() (rtCommon.bidsArchive.BidsArchive method), 37
addMetadata() (rtCommon.bidsArchive.BidsArchive method), 37
addResultValue() (rtCommon.webDisplayInterface.WebDisplayInterface method), 89
addScript() (rtCommon.webServer.WsBrowserRequestHandler method), 92
appendIncremental() (rtCommon.bidsArchive.BidsArchive method), 38
checkAllowedDirs() (rtCommon.dataInterface.DataInterface method), 61
checkAllowedFileTypes() (rtCommon.dataInterface.DataInterface method), 61
exceptIfMissingMetadata() (rtCommon.bidsIncremental.BidsIncremental method), 47
exceptIfNotBids() (rtCommon.bidsIncremental.BidsIncremental method), 48
filterFileList() (rtCommon.dataInterface.DataInterface method), 61
getIncremental() (rtCommon.bidsArchive.BidsArchive method), 39
__imeta() (rtCommon.bidsIncremental.BidsIncremental attribute), 46
postprocessMetadata() (rtCommon.bidsIncremental.BidsIncremental method), 47
preprocessMetadata() (rtCommon.
A

absPathFromRelPath() (rtCommon.bidsArchive.BidsArchive method), 35
addHandlerClass() (rtCommon.wsRemoteService.WsRemoteService method), 95
addHandlerClassName() (rtCommon.wsRemoteService.WsRemoteService method), 95
addHandlers() (rtCommon.webServer.Web static method), 92
addLocalAttributes() (rtCommon.remoteable.RemoteableExtensible method), 76
addUserPassword() (in module rtCommon.addLogin), 34
adjustTimeUnits() (in module rtCommon.bidsCommon), 44
anonymizeDicom() (in module rtCommon.imageHandling), 68
app (rtCommon.webServer.Web attribute), 91
appendBidsRun() (rtCommon.bidsArchive.BidsArchive method), 40
appendIncremental() (rtCommon.bidsRun.BidsRun method), 54
areArraysClose() (in module rtCommon.validationUtils), 87
argParser (in module rtCommon.addLogin), 34
argParser (in module rtCommon.projectServer), 72
asSingleIncremental() (rtCommon.bidsRun.BidsRun method), 55
attributesToAnonymize (in module rtCommon.imageHandling), 68

B

base_name (in module rtCommon.resample), 76
BaseWebSocketHandler (class in rtCommon.websocketHandlers), 93
BIDS_DIR_PATH_PATTERN (in module rtCommon.bidsCommon), 42
BIDS_EVENT_COL_TO_DTYPE (in module rtCommon.bidsCommon), 42
BIDS_FILE_PATH_PATTERN (in module rtCommon.bidsCommon), 42
BIDS_FILE_PATTERN (in module rtCommon.bidsCommon), 42
BIDS_VERSION (in module rtCommon.bidsCommon), 42
BidsArchive (class in rtCommon.bidsArchive), 35
BidsAttributesToAnonymize (in module rtCommon.bidsCommon), 43
BidsEntityKeys (class in rtCommon.bidsCommon), 43
BidsFileExtension (class in rtCommon.bidsCommon), 43
BidsIncremental (class in rtCommon.bidsIncremental), 46
bidsIncrementalTimeToNextTr() (in module rtCommon.imageHandling), 69
BidsInterface (class in rtCommon.bidsInterface), 51
BidsRun (class in rtCommon.bidsRun), 54
BidsStream (class in rtCommon.bidsInterface), 53
binPath (in module rtCommon.imageHandling), 68

calcAvgRoundTripTime() (in module rtCommon.utils), 86
callback() (rtCommon.websocketHandlers.RequestHandler method), 94
certsDir (in module rtCommon.certsUtils), 56
checkingDicomNiftiConversion() (in module rtCommon.checkDicomNiftiConversion), 57
checkRetry() (rtCommon.webHttpHandlers.LoginHandler method), 90
checkSSLCertAltName() (in module rtCommon.projectUtils), 74
clearAllPlots() (rtCommon.webDisplayInterface.WebDisplayInterface method), 88
clearRunPlot() (rtCommon.webDisplayInterface.WebDisplayInterface method), 89
ClientInterface (class in rtCommon.clientInterface), 57
close() (rtCommon.webServer.Web static method), 92
doRequest() (rtCommon.webSocketHandlers.RequestHandler method), 94

downloadData() (rtCommon.openNeuro.OpenNeuroCache method), 70

downloadFilesFromCloud() (in module rtCommon.dataInterface), 61
downloadFilesFromList() (in module rtCommon.dataInterface), 61
downloadFolderFromCloud() (in module rtCommon.dataInterface), 61
dtimeToSeconds() (in module rtCommon.utils), 86

echo() (rtCommon.exampleInterface.ExampleInterface method), 64

encodeByteTypeArgs() (in module rtCommon.serialization), 78
encodeMessageData() (in module rtCommon.serialization), 78

ENTITIES (rtCommon.bidsIncremental.BidsIncremental attribute), 46
ENTITY (rtCommon.bidsCommon.BidsEntityKeys attribute), 43
EVENTS (rtCommon.bidsCommon.BidsFileExtension attribute), 43
ExampleInterface (class in rtCommon.exampleInterface), 64
ExampleService (class in rtCommon.exampleService), 64

exposed_BidsInterface (rtCommon.projectServerRPC.ProjectRPCService attribute), 73
exposed_DataInterface (rtCommon.projectServerRPC.ProjectRPCService attribute), 73
exposed_ExampleInterface (rtCommon.projectServerRPC.ProjectRPCService attribute), 73
exposed_isDataRemote() (rtCommon.projectServerRPC.ProjectRPCService method), 73
exposed_isSubjectRemote() (rtCommon.projectServerRPC.ProjectRPCService method), 73
exposed_SubjectInterface (rtCommon.projectServerRPC.ProjectRPCService attribute), 73
exposed_WebDisplayInterface (rtCommon.projectServerRPC.ProjectRPCService attribute), 73

failIfEmpty() (in module rtCommon.bidsArchive), 35

fileCount() (in module rtCommon.utils), 85
FileNotifyHandler (class in rtCommon.fileWatcher), 66
FileWatcher (class in rtCommon.fileWatcher), 65
filterEntities() (in module rtCommon.bidsCommon), 43
find() (in module rtCommon.utils), 85
findMissingImageMetadata() (rtCommon.bidsIncremental.BidsIncremental class method), 47
findNewestFile() (in module rtCommon.utils), 85
flatten_1Ds() (in module rtCommon.utils), 85
FORMAT (rtCommon.bidsCommon.BidsEntityKeys attribute), 43
formatFileData() (in module rtCommon.projectUtils), 74

generateDataParts() (in module rtCommon.serialization), 79
get() (rtCommon.webHttpHandlers.HttpHandler method), 90
get() (rtCommon.webHttpHandlers.LoginHandler method), 90
get() (rtCommon.webHttpHandlers.LogoutHandler method), 90
get_current_user() (rtCommon.webHttpHandlers.HttpHandler method), 90

get_response() (rtCommon.webSocketHandlers.RequestHandler method), 95
getAcquisitionTime() (rtCommon.bidsIncremental.BidsIncremental method), 50
getAllowedFileTypes() (rtCommon.dataInterface.DataInterface method), 60
getAllResponses() (rtCommon.subjectInterface.SubjectInterface method), 82
getAddress() (rtCommon.openNeuro.OpenNeuroCache method), 70
getAxesForTransform() (in module rtCommon.imageHandling), 69
getBidsRun() (rtCommon.bidsArchive.BidsArchive method), 40
cachePath() (rtCommon.openNeuro.OpenNeuroCache method), 70
rtClockSkew() (rtCommon.bidsInterface.BidsInterface method), 52
getClockSkew() (rtCommon.dataInterface.DataInterface method), 61
getCookieSecret() (in module rtCommon.webServer), 61
getDataDirPath() (rtCommon.bidsIncremental.BidsIncremental method), 49
getDatasetList() (rtCommon.openNeuro.OpenNeuroCache method), 70
getDatasetName() (rtCommon.bidsIncremental.BidsIncremental method), 49
getDatatype() (rtCommon.bidsIncremental.BidsIncremental method), 49
description() (rtCommon.openNeuro.OpenNeuroCache method), 70
getDicomAcquisitionTime() (in module rtCommon.imageHandling), 69
getDicomFileName() (in module rtCommon.imageHandling), 68
getDicomMetadata() (in module rtCommon.bidsCommon), 44
getDicomRepetitionTime() (in module rtCommon.imageHandling), 69
getEntities() (rtCommon.bidsIncremental.BidsIncremental method), 49
getEvents() (rtCommon.bidsArchive.BidsArchive method), 37
getEventsFileName() (rtCommon.bidsIncremental.BidsIncremental method), 49
getEventsFilePath() (rtCommon.bidsIncremental.BidsIncremental method), 49
getImageData() (rtCommon.bidsIncremental.BidsIncremental method), 49
gImageData() (rtCommon.dataInterface.DataInterface method), 59
gImageDimensions() (rtCommon.bidsIncremental.BidsIncremental method), 49
getImageFileName() (rtCommon.bidsIncremental.BidsIncremental method), 49
getImagePath() (rtCommon.bidsIncremental.BidsIncremental method), 49
getImageHeader() (rtCommon.bidsIncremental.BidsIncremental method), 49
getImageMetadata() (rtCommon.bidsIncremental.BidsIncremental method), 49
getImages() (rtCommon.bidsArchive.BidsArchive method), 36
getIncremental() (rtCommon.bidsInterface.BidsInterface method), 52
getIncremental() (rtCommon.bidsInterface.BidsStream method), 53
getIncremental() (rtCommon.bidsInterface.DicomToBidsStream method), 53
getIncremental() (rtCommon.bidsRun.BidsRun method), 54
getMetadataField() (rtCommon.bidsIncremental.BidsIncremental method), 48
getMetadataFileName() (rtCommon.bidsIncremental.BidsIncremental method), 49
getMetadataFilePath() (rtCommon.bidsIncremental.BidsIncremental method), 49
getNewestFile() (rtCommon.dataInterface.DataInterface method), 60
getNiftiData() (in module rtCommon.bidsCommon), 43
getNumVolumes() (rtCommon.bidsInterface.BidsInterface method), 52
getNumVolumes() (rtCommon.bidsInterface.BidsStream method), 53
getNumVolumes() (rtCommon.bidsInterface.DicomToBidsStream method), 53
getPreviousDataPoints() (rtCommon.webDisplayInterface.WebDisplayInterface method), 89
getReadme() (rtCommon.bidsArchive.BidsArchive method), 36
getReadme() (rtCommon.openNeuro.OpenNeuroCache method), 70
getRepetitionTime() (rtCommon.bidsIncremental.BidsIncremental method), 49
getRuntime() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVersion() (rtCommon.bidsRun.BidsRun method), 54
getSystemVer
handle_request() (rtCommon.wsRemoteService.WsRemoteService static method), 95
handleRPCRequest() (rtCommon.projectServerRPC.RPCHandlers method), 74
htmlDir (rtCommon.webServer.Web attribute), 92
HttpHandler (class in rtCommon.webHttpHandlers), 90
httpPort (rtCommon.webServer.Web attribute), 91
httpServer (rtCommon.webServer.Web attribute), 91

H

HANDLE

image_reference (in module rtCommon.resample), 76
image_to_resample (in module rtCommon.resample), 76
initBidsStream() (rtCommon.bidsInterface.BidsInterface method), 51
initDicomBidsStream() (rtCommon.bidsInterface.BidsInterface method), 51
initFileNotifier() (rtCommon.fileWatcher.FileWatcher method), 65
initFileNotifier() (rtCommon.fileWatcher.InotifyFileWatcher method), 66
initFileNotifier() (rtCommon.fileWatcher.WatchdogFileWatcher method), 65
initialize() (rtCommon.webHttpHandlers.HttpHandler method), 90
initialize() (rtCommon.webHttpHandlers.LogoutHandler method), 90
initialize() (rtCommon.webHttpHandlers.LoginHandler method), 90
initialize() (rtCommon.websocketHandlers.BaseWebSocketHandler method), 93
initialize() (rtCommon.websocketHandlers.RejectWebSocketHandler method), 94
initOpenNeuroStream() (rtCommon.bidsInterface.BidsInterface method), 51
initScannerStream() (rtCommon.dataInterface.DataInterface method), 59
initStream() (rtCommon.bidsInterface.DicomToBidsStream method), 52
initWatch() (rtCommon.dataInterface.DataInterface method), 51
InotifyFileWatcher (class in rtCommon.fileWatcher), 60
installLoggers() (in module rtCommon.utils), 86
InvocationError, 62
ioLoopInst (rtCommon.webServer.Web attribute), 92
isCompleteImageMetadata() (rtCommon.bidsIncremental.BidsIncremental class method), 48
isDataRemote() (rtCommon.clientInterface.ClientInterface method), 58
isEmpty() (rtCommon.bidsArchive.BidsArchive method), 37
isMeanWithinThreshold() (in module rtCommon.validationUtils), 88
isNativeType() (in module rtCommon.validationUtils), 88
isRunningRemote() (rtCommon.remoteable.RemoteableExtensible method), 76
rtCommon.utils, 83
rtCommon.validationUtils, 86
rtCommon.webDisplayInterface, 88
rtCommon.webHttpHandlers, 89
rtCommon.webServer, 91
rtCommon.websocketHandlers, 93
rtCommon.wsRemoteService, 95
moduleDir (in module rtCommon.webServer), 91
multiPartDataCache (in module rtCommon.serialization), 78

niftiHeadersAppendCompatible() (in module rtCommon.bidsCommon), 45
niftiImagesAppendCompatible() (in module rtCommon.bidsCommon), 45
niftiToMem() (in module rtCommon.imageHandling), 69
notifyEventLoop() (rtCommon.fileWatcher.InotifyFileWatcher method), 66
NotImplementedError, 63
npToPy() (in module rtCommon.serialization), 78
numIncrementals() (rtCommon.bidsRun.BidsRun method), 55
numpyAllNumCodes (in module rtCommon.validationUtils), 87

on_clearDataPoints() (rtCommon.webServer.WsBrowserRequestHandler method), 92
on_close() (rtCommon.websocketHandlers.BaseWebSocketHandler method), 94
on_close() (rtCommon.websocketHandlers.DataWebSocketHandler method), 94
on_close() (rtCommon.wsRemoteService.WsRemoteService static method), 96
on_connect() (rtCommon.projectServerRPC.ProjectRPCService method), 73
on_created() (rtCommon.fileWatcher.FileNotifyHandler method), 66
on_disconnect() (rtCommon.projectServerRPC.ProjectRPCService method), 73
on_error() (rtCommon.wsRemoteService.WsRemoteService static method), 96
on_getDataPoints() (rtCommon.webServer.WsBrowserRequestHandler method), 92
on_getDefaultConfig() (rtCommon.webServer.WsBrowserRequestHandler method), 92
on_getRunStatus() (rtCommon.webServer.WsBrowserRequestHandler method), 92
on_message() (rtCommon.websocketHandlers.BaseWebSocketHandler method), 94
on_message() (rtCommon.wsRemoteService.WsRemoteService static method), 96
on_modified() (rtCommon.fileWatcher.FileNotifyHandler method), 66
on_runScript() (rtCommon.webServer.WsBrowserRequestHandler method), 92
on_stop() (rtCommon.webServer.WsBrowserRequestHandler method), 92
on_uploadFiles() (rtCommon.webServer.WsBrowserRequestHandler method), 92
open() (rtCommon.websocketHandlers.BaseWebSocketHandler method), 94
open() (rtCommon.websocketHandlers.RejectWebSocketHandler method), 94
OpenNeuroCache (class in rtCommon.openNeuro), 70
OpenNeuroService (class in rtCommon.openNeuroService), 71

parseConnectionArgs() (in module rtCommon.wsRemoteService), 96
parseDicomVolume() (in module rtCommon.imageHandling), 68
parseMatlabStruct() (in module rtCommon.utils), 85
passwordFile (in module rtCommon.addLogin), 34
pearsons_mean_corr() (in module rtCommon.validationUtils), 88
ping() (rtCommon.bidsInterface.BidsInterface method), 52
ping() (rtCommon.dataInterface.DataInterface method), 61
plotDataPoint() (rtCommon.webDisplayInterface.WebDisplayInterface method), 88
post() (rtCommon.webHttpHandlers.LoginHandler method), 90
prepare_request() (rtCommon.websocketHandlers.RequestHandler method), 94
processShouldExitThread() (in module rtCommon.projectUtils), 74
procOutputReader() (in module rtCommon.webServer), 93
ProjectRPCService (class in rtCommon.projectServerRPC), 73
ProjectServer (class in rtCommon.projectServer), 72
pruneCallbacks() (rtCommon.websocketHandlers.RequestHandler method), 95
putFile() (rtCommon.dataInterface.DataInterface method), 60
PYBIDS_PSEUDO_ENTITIES (in module rtCommon.bidsCommon), 42
QueryError, 63
readDicomFromBuffer() (in module rtCommon.imageHandling), 68
readDicomFromFile() (in module rtCommon.imageHandling), 68
readFile() (in module rtCommon.utils), 85
readNifti() (in module rtCommon.imageHandling), 69
readRetryDicomFromDataInterface() (in module rtCommon.imageHandling), 68
recurseCreateStructDict() (in module rtCommon.structDict), 80
recurseSDtoDict() (in module rtCommon.structDict), 80
registerClassInstance() (rtCommon.remoteable.RemoteHandler method), 76
registerClassNameInstance() (rtCommon.remoteable.RemoteHandler method), 76
registerCommFunction() (rtCommon.remoteable.RemoteableExtensible method), 76
registerCommFunction() (rtCommon.remoteable.RemoteStub method), 75
registerDataCommFunction() (rtCommon.projectServerRPC.ProjectRPCService static method), 73
registerSubjectCommFunction() (rtCommon.projectServerRPC.ProjectRPCService static method), 73
RejectWebSocketHandler (class in rtCommon.websocketHandlers), 94
Remoteable (class in rtCommon.remoteable), 75
RemoteableExtensible (class in rtCommon.remoteable), 76
remoteCall() (rtCommon.remoteable.RemoteableExtensible method), 76
remoteCall() (rtCommon.remoteable.RemoteStub method), 75
RemoteHandler (class in rtCommon.remoteable), 76
remoteHandler (rtCommon.wsRemoteService.WsRemoteService attribute), 95
RemoteStub (class in rtCommon.remoteable), 75
removeMetadataField() (rtCommon.bidsIncremental.BidsIncremental method), 49
RequestError, 62
RequestHandler (class in rtCommon.websocketHandlers), 94
REQUIRED_IMAGE_METADATA (rtCommon.bidsIncremental.BidsIncremental attribute), 46
resampled_image (in module rtCommon.resample), 76
rootDir (in module rtCommon.webServer), 91
rootPath (in module rtCommon.addLogin), 34
rootPath (in module rtCommon.certsUtils), 56
rootPath (in module rtCommon.openNeuroService), 71
rootPath (in module rtCommon.projectServer), 72
rootPath (in module rtCommon.scannerDataService), 77
rootPath (in module rtCommon.subjectService), 82
RPCHandlers (class in rtCommon.projectServerRPC), 73
rtCommon module, 33
rtCommon.addLogin module, 33
rtCommon.bidsArchive module, 34
rtCommon.bidsCommon module, 40
rtCommon.bidsIncremental module, 46
rtCommon.bidsInterface module, 50
rtCommon.bidsRun module, 54
rtCommon.certsUtils module, 56
rtCommon.checkDicomNiftiConversion module, 56
rtCommon.clientInterface module, 57
rtCommon.dataInterface module, 58
rtCommon.dicomToBidsService module, 62
rtCommon.errors module, 62
rtCommon.exampleInterface module, 63
rtCommon.exampleService module, 64
rtCommon.fileWatcher module, 65
rtCommon.imageHandling module, 67
rtCommon.openNeuro module, 70
rtCommon.openNeuroService module, 71
rtCommon.projectServer module, 71
rtCommon.projectServerRPC module, 72
rtCommon.projectUtils module, 74
rtCommon.remoteable module, 75
rtCommon.resample module, 76
rtCommon.scannerDataService module, 76
rtCommon.serialization module, 77
rtCommon.structDict module, 79
rtCommon.subjectInterface module, 81
rtCommon.subjectService module, 82
rtCommon.utils module, 83
rtCommon.validationUtils module, 86
rtCommon.webDisplayInterface module, 88
rtCommon.webHttpHandlers module, 89
rtCommon.webServer module, 91
rtCommon.wsSocketHandlers module, 93
rtCommon.wsRemoteService module, 95
RTError, 62
runCmdCheckOutput() (in module rtCommon.utils), 85
runDetached() (rtCommon.exampleService.ExampleService method), 65
runDetached() (rtCommon.openNeuroService.OpenNeuroService method), 71
runDetached() (rtCommon.subjectService.SubjectService method), 82
runForever() (rtCommon.wsRemoteService.WsRemoteService method), 95
runRemoteCall() (rtCommon.remoteable.RemoteHandler method), 76
S
saveAsNiftiImage() (in module rtCommon.imageHandling), 69
ScannerDataService (class in rtCommon.scannerDataService), 77
send_response() (rtCommon.wsRemoteService.WsRemoteService static method), 95
sendConfig() (rtCommon.webDisplayInterface.WebDisplayInterface method), 88
sendConnStatus() (rtCommon.webDisplayInterface.WebDisplayInterface method), 89
sendPreviousDataPoints() (rtCommon.webDisplayInterface.WebDisplayInterface method), 88
sendRunStatus() (rtCommon.webDisplayInterface.WebDisplayInterface method), 88
sendUploadStatus() (rtCommon.webDisplayInterface.WebDisplayInterface method), 88
sendWebSocketMessage() (in module rtCommon.wsSocketHandlers), 94
sessionLog() (rtCommon.webDisplayInterface.WebDisplayInterface method), 88
setDebugError() (rtCommon.webDisplayInterface.WebDisplayInterface method), 88
setError() (rtCommon.projectServerRPC.RPCHandlers method), 74
setMessage() (rtCommon.subjectInterface.SubjectInterface method), 81
setMetadataField() (rtCommon.bidsIncremental.BidsIncremental method), 48
setResult() (rtCommon.subjectInterface.SubjectInterface method), 81
setRPCTimeout() (rtCommon.remoteable.RemoteableExtensible method), 76
<table>
<thead>
<tr>
<th>Method/Function</th>
<th>Module/Class</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>setRPCTimeout()</td>
<td>(rtCommon.remoteable.RemoteStub method)</td>
<td>76</td>
</tr>
<tr>
<td>setUserError()</td>
<td>(rtCommon.webDisplayInterface.WebDisplayInterface method)</td>
<td>88</td>
</tr>
<tr>
<td>shouldExit()</td>
<td>(rtCommon.wsRemoteService.WsRemoteService attribute)</td>
<td>95</td>
</tr>
<tr>
<td>sslCertFile</td>
<td>(in module rtCommon.certsUtils)</td>
<td>56</td>
</tr>
<tr>
<td>sslPrivateKey</td>
<td>(in module rtCommon.certsUtils)</td>
<td>56</td>
</tr>
<tr>
<td>start()</td>
<td>(rtCommon.projectServer.ProjectServer method)</td>
<td>72</td>
</tr>
<tr>
<td>start()</td>
<td>(rtCommon.webServer.Web static method)</td>
<td>92</td>
</tr>
<tr>
<td>started</td>
<td>(rtCommon.webServer.Web attribute)</td>
<td>91</td>
</tr>
<tr>
<td>startRPCThread()</td>
<td>(in module rtCommon.projectServerRPC)</td>
<td>73</td>
</tr>
<tr>
<td>StateError</td>
<td></td>
<td>62</td>
</tr>
<tr>
<td>StatsEqual</td>
<td>(in module rtCommon.validationUtils)</td>
<td>87</td>
</tr>
<tr>
<td>StatsNotEqual</td>
<td>(in module rtCommon.validationUtils)</td>
<td>87</td>
</tr>
<tr>
<td>stop()</td>
<td>(rtCommon.projectServer.ProjectServer method)</td>
<td>72</td>
</tr>
<tr>
<td>storePasswdFile()</td>
<td>(in module rtCommon.webHttpHandlers)</td>
<td>90</td>
</tr>
<tr>
<td>stringPartialFormat()</td>
<td>(in module rtCommon.utils)</td>
<td>86</td>
</tr>
<tr>
<td>StructDict</td>
<td>(class in rtCommon.structDict)</td>
<td>80</td>
</tr>
<tr>
<td>structDictType</td>
<td>(module rtCommon.structDict)</td>
<td>80</td>
</tr>
<tr>
<td>SubjectInterface</td>
<td>(class in rtCommon.subjectInterface)</td>
<td>81</td>
</tr>
<tr>
<td>subjectRequest()</td>
<td>(rtCommon.projectServerRPC.RPCHandlers method)</td>
<td>74</td>
</tr>
<tr>
<td>SubjectService</td>
<td>(class in rtCommon.subjectService)</td>
<td>82</td>
</tr>
<tr>
<td>subjectWsCallback()</td>
<td>(rtCommon.projectServerRPC.RPCHandlers method)</td>
<td>73</td>
</tr>
<tr>
<td>symmetricDictDifference()</td>
<td>(module rtCommon.bidsCommon)</td>
<td>44</td>
</tr>
<tr>
<td>testMethod()</td>
<td>(rtCommon.exampleInterface.ExampleInterface method)</td>
<td>64</td>
</tr>
<tr>
<td>testMode</td>
<td>(rtCommon.webServer.Web attribute)</td>
<td>92</td>
</tr>
<tr>
<td>timeToNextTr()</td>
<td>(rtCommon.bidsIncremental.BidsIncremental method)</td>
<td>50</td>
</tr>
<tr>
<td>trimDictBytes()</td>
<td>(module rtCommon.utils)</td>
<td>86</td>
</tr>
<tr>
<td>tryGetFile()</td>
<td>(rtCommon.bidsArchive.BidsArchive method)</td>
<td>35</td>
</tr>
<tr>
<td>unpackDataMessage()</td>
<td>(module rtCommon.serialization)</td>
<td>79</td>
</tr>
<tr>
<td>uploadFilesFromList()</td>
<td>(module rtCommon.dataInterface)</td>
<td>61</td>
</tr>
<tr>
<td>uploadFilesToCloud()</td>
<td>(module rtCommon.dataInterface)</td>
<td>61</td>
</tr>
<tr>
<td>uploadFolderToCloud()</td>
<td>(module rtCommon.dataInterface)</td>
<td>61</td>
</tr>
<tr>
<td>userLog()</td>
<td>(rtCommon.webDisplayInterface.WebDisplayInterface method)</td>
<td>88</td>
</tr>
<tr>
<td>waitForFile()</td>
<td>(rtCommon.fileWatcher.FileWatcher method)</td>
<td>65</td>
</tr>
<tr>
<td>waitForFile()</td>
<td>(rtCommon.fileWatcher.InotifyFileWatcher method)</td>
<td>66</td>
</tr>
<tr>
<td>waitFile()</td>
<td>(rtCommon.fileWatcher.WatchdogFileWatcher method)</td>
<td>66</td>
</tr>
<tr>
<td>watchdogFileWatcher</td>
<td>(class in rtCommon.fileWatcher)</td>
<td>65</td>
</tr>
<tr>
<td>watchFile()</td>
<td>(rtCommon.dataInterface.DataInterface method)</td>
<td>60</td>
</tr>
<tr>
<td>watchForExit()</td>
<td>(module rtCommon.projectUtils)</td>
<td>74</td>
</tr>
<tr>
<td>Web</td>
<td>(class in rtCommon.webServer)</td>
<td>91</td>
</tr>
<tr>
<td>WebDisplayInterface</td>
<td>(class in rtCommon.webDisplayInterface)</td>
<td>88</td>
</tr>
<tr>
<td>websocketState</td>
<td>(class in rtCommon.webSocketHandlers)</td>
<td>93</td>
</tr>
<tr>
<td>WrapRpycObject</td>
<td>(class in rtCommon.clientInterface)</td>
<td>58</td>
</tr>
<tr>
<td>writeDataFrameToEvents()</td>
<td>(module rtCommon.bidsCommon)</td>
<td>46</td>
</tr>
<tr>
<td>writeDicomFile()</td>
<td>(module rtCommon.imageHandling)</td>
<td>68</td>
</tr>
<tr>
<td>writeDicomToBuffer()</td>
<td>(module rtCommon.imageHandling)</td>
<td>68</td>
</tr>
<tr>
<td>writeFile()</td>
<td>(module rtCommon.utils)</td>
<td>85</td>
</tr>
<tr>
<td>writeToFile()</td>
<td>(rtCommon.bidsIncremental.BidsIncremental method)</td>
<td>50</td>
</tr>
</tbody>
</table>

**Index**
WsBrowserRequestHandler (class in rtCommon.webServer), 92
wsCallbacks (rtCommon.websocketHandlers.websocketState attribute), 93
wsConnCallback() (rtCommon.webDisplayInterface.WebDisplayInterface method), 89
wsConnectionLists (rtCommon.websocketHandlers.websocketState attribute), 93
wsConnLock (rtCommon.websocketHandlers.websocketState attribute), 93
WsRemoteService (class in rtCommon.wsRemoteService), 95