llQTools Documentation

Release 1.0.0

Nicholas Willhite

Jan 23, 2019

Contents

Contents:

CHAPTER 1

Getting Started

1.1 Install and Upgrade

This section provides directions to install or upgrade the IIQTools package for your instance of InsightIQ.

1.1.1 Installing the RPM

This is by far the easiest way to install IIQTools.

1. Download the newest RPM here
2. Copy the RPM file onto the machine running InsightIQ
3. Install with this command, replacing `</path/to/rpm>` with the literal file path:

```
$ sudo yum install --disablerepo=* <path/to/rpm>
```

1.1.2 Installing Python pip

Python pip is the preferred way to install Python packages. I f your instance of InsightIQ doesn't have pip installed, there are some pretty simple directions at:

https://pip.pypa.io/en/stable/installing/

Note: The `--upgrade` flag is automatically ignored if you're performing an initial install

1.1.3 With Internet connection

To install or upgrade the IIQTools package to your instance of InsightIQ run:

```
$ sudo pip install --upgrade iiqtools
```

1.1.4 Without Internet connection

1. Download the package from PyPI
2. Copy the package to the InsightIQ instance
3. Run these commands, replacing `<package>` with the actual package name:

```
$ sudo pip install --upgrade <package>
```

1.2 Setting up a development environment

This section is *only* for settings up a development environment. Only follow these directions if you intend to contribute source code to the project.

If you have the *know-how* to build and test IIQTools on MacOS or Windows, we'd love to get a Pull Request from you updating this section!

1.2.1 Linux

These are non-Python libraries required to build and test IIQTools. Use the builtin packagemanger (i.e. apt-get, yum, dnf) to install these packages.

RHEL/CentOS/Fedora:

- make
- python-devel
- postgresql-devel
- postgresql
- redhat-rpm-config

Ubuntu:

- make
- python-dev
- postgresql
- libpq-dev

Before installing the Python dependencies, it's highly encouraged to configure a Python Virtualenv for working with IIQTools.

Once you've got your virtualenv configured and activated, run this command to install the dependencies for building and testing IIQTools:

```
$ pip install -r requirements-dev.txt
```

Once you've installed those dependencies, you can run all the unit tests like this:

```
$ make test
```

And generate the docs with:

```
$ make docs
```

Or clean up everything with:

```
$ make clean
```

CHAPTER 2

Scripts & Tools

This section goes over the scripts and tools added to your InsightIQ instance when you install the IIQTools package. Each section should have some examples of what running the script looks like, or what the tool does.

2.1 iiqtools_gather_info

The point of this script is to provide a convenient way for users to collect logs and configuration information about InsightIQ so that remote support can investigate any issues that they are having.

2.1.1 Examples (assuming your running as the `root` users)

Printing the *help* message:

```
[root@localhost ~]$ iiqtools_gather_info --help
usage: iiqtools_gather_info [-h] [--output-dir OUTPUT_DIR] --case-number
                      CASE_NUMBER

Generate a .tar file for debugging InsightIQ

optional arguments:
  -h, --help            show this help message and exit
  --output-dir OUTPUT_DIR
                        The directory to write the .tgz file to (default:
                        /home/administrator)
  --case-number CASE_NUMBER
                        The Service Request number. Used in naming tar file.
                        (default: None)
```

Generating a gather:

```
[root@localhost ~]$ iiqtools_gather_info --case-number 1234
2017-09-11 14:53:45,298 - INFO - Collecting config information
2017-09-11 14:53:46,123 - INFO - Collecting log files
2017-09-11 14:53:48,126 - INFO - Log gather complete
2017-09-11 14:53:48,126 - INFO - Created log file /home/administrator/IIQLogs-sr1234-
↪1505166825.tgz
```

Forgetting to provide the `--case-number` argument:

```
[root@localhost ~]$ iiqtools_gather_info
usage: iiqtools_gather_info [-h] [--output-dir OUTPUT_DIR] --case-number
                            CASE_NUMBER
iiqtools_gather_info: error: argument --case-number is required
```

Outputting the tar file to a different directory:

```
[root@localhost ~]$ iiqtools_gather_info --case-number 1234 --output-dir /datastore
2017-09-11 14:55:44,195 - INFO - Collecting config information
2017-09-11 14:55:44,451 - INFO - Collecting log files
2017-09-11 14:55:45,957 - INFO - Log gather complete
2017-09-11 14:55:45,957 - INFO - Created log file /datastore/IIQLogs-sr1234-
↪1505166944.tgz
```

2.2 iiqtools_tar_to_zip

Starting with InsightIQ 3.2, you could export a cluster's database from one instance, then import it later or on another InsightIQ instance. Initially, the exported data was in tar file format, but in InsightIQ 4.1 we switched to using a zip file. The switch was to resolve a bug where importing large exports would time out. The data contained within the tar and the zip files is identical; only the compression format has changed. This means that if we convert an old tar export to zip, we can use that archive in newer versions of InsightIQ.

Use cases for this script:

Migration Upgrades Instead of upgrading an existing deployment, you export the data on your old instance, use this script to convert the format, and then import that data on a new deployment of InsightIQ. This approach is ideal for OVA deployments of InsightIQ because the newer OVAs for InsightIQ have the latest security patches applied, and the root partition is configured with LVM.

Maintain Legacy Exports With the upgrade to 4.1, any datastore exports created on the older version of InsightIQ are no longer compatible. This script will update the format of those older datastore exports so you can continue to use them in newer versions of InsightIQ.

2.2.1 Usage Examples

Obtaining the *help* message:

```
[administrator@localhost ~]$ iiqtools_tar_to_zip --help
usage: iiqtools_tar_to_zip [-h] -s SOURCE_TAR [-o OUTPUT_DIR]
Convert .tar to .zip for IIQ datastore export files

optional arguments:
  -h, --help            show this help message and exit
  -s SOURCE_TAR, --source-tar SOURCE_TAR
                        The source .tar file to convert to .zip (default:
```

(continues on next page)

(continued from previous page)

```
                        None)
  -o OUTPUT_DIR, --output-dir OUTPUT_DIR
                        The (default: /home/administrator)
```

Simple usage (this export was only about 20MB in size):

```
[administrator@localhost ~]$ iiqtools_tar_to_zip --source-tar /datastore/insightiq_
↪export_1505412864.tar.gz
2017-09-15 16:57:02,669 - INFO - Converting /datastore/insightiq_export_1505412864.
↪tar.gz to zip format
2017-09-15 16:57:02,849 - INFO - InsightIQ datastore tar export contained  2 files
2017-09-15 16:57:02,850 - INFO - Converting insightiq_export_1505412864/dog-pools_
↪003048c644105df4124ad80c701933e83eff.dump
2017-09-15 16:57:03,120 - INFO - Converting insightiq_export_1505412864/dog-pools_
↪003048c644105df4124ad80c701933e83eff_config.json
2017-09-15 16:57:03,160 - INFO - New zip formatted file saved to /home/administrator/
↪insightiq_export_1505412864.zip
```

Creating the new zip in a different directory:

```
[administrator@localhost ~]$ iiqtools_tar_to_zip --source-tar /datastore/insightiq_
↪export_1505412864.tar.gz --output-dir /tmp
2017-09-15 17:00:08,897 - INFO - Converting /datastore/insightiq_export_1505412864.
↪tar.gz to zip format
2017-09-15 17:00:09,073 - INFO - InsightIQ datastore tar export contained  2 files
2017-09-15 17:00:09,073 - INFO - Converting insightiq_export_1505412864/dog-pools_
↪003048c644105df4124ad80c701933e83eff.dump
2017-09-15 17:00:09,337 - INFO - Converting insightiq_export_1505412864/dog-pools_
↪003048c644105df4124ad80c701933e83eff_config.json
2017-09-15 17:00:09,374 - INFO - New zip formatted file saved to /tmp/insightiq_
↪export_1505412864.zip
```

2.3 iiqtools_version

A rather straght forward script that prints the version of InsightIQ and IIQTools that's installed.

Example Usage:

```
[administrator@localhost ~]$ iiqtools_version
InsightIQ: 4.1.1.3
IIQTools: 0.1.0
```

2.4 iiqtools_patch

A tool for installing, uninstalling, and displaying patches to IsightIQ source code.

Note: Installing **and** uninstalling requires the InsightIQ application to be restarted. Running the `iiqtools_patch` tool with `sudo` will automatically restart the application.

Display all installed patches:

```
[administrator@localhost ~]$ iiqtools_patch --show

        Patches
        -------
        patch1234

        Count: 1
```

Display details for a specific patch:

```
[administrator@localhost ~]$ iiqtools_patch --show

Here's an example patch details
```

Uninstalling a patch as a non-root user:

```
[administrator@localhost ~]$ iiqtools_patch --uninstall patch1234
2017-10-03 12:49:19,656 - INFO - Successfully uninstalled patch
2017-10-03 12:49:19,657 - INFO - Non-root user detected for patch install. Unable to␣
↪restart InsightIQ.
2017-10-03 12:49:19,657 - INFO - **Patch wont take effect unless you restart␣
↪InsightIQ**

2017-10-03 12:49:19,657 - INFO - Please run the following command to restart␣
↪InsightIQ:
2017-10-03 12:49:19,657 - INFO - sudo service insightiq restart
```

Installing a patch with `sudo`:

```
[administrator@localhost ~]$ sudo iiqtools_patch --install insightiq-patch-1234.tgz
[sudo] password for administrator:
2017-10-03 12:54:26,643 - INFO - Installed IIQ version: 4.1.1.3
2017-10-03 12:54:26,644 - INFO - Patch min version: 4.1.0
2017-10-03 12:54:26,644 - INFO - Patch max version: 4.1.1.3
2017-10-03 12:54:26,645 - INFO - Successfully installed patch
2017-10-03 12:54:26,645 - INFO - Restarting InsightIQ
2017-10-03 12:54:34,098 - INFO - Stopping insightiq:       [  OK  ]
Starting insightiq:                                        [  OK  ]
```

2.5 iiqtools_cluster_backup

The point of this tool is to make automating backups of your cluster data easy; just setup a crontab!

InsightIQ supports exporting/importing cluster data, but it requires a user to click through the UI. This tool calls the same API as the UI, but instead does the API call from the CLI instead of a browser. The API that is called requires a user with elevated privileges for the backup to work. Attempting to use a read-only user will cause your backups to fail. To be clear, this tool needs an admin of InsightIQ, not the host Linux machine running the InsightIQ application.

Note: It's highly recommend to setup a local user instead of using the default `administrator`.

2.5.1 Setting up the `iiq_backup` user account

The default `administrator` account used by InsightIQ has `sudo` power over the host machine running the application. In other words, that account is root by a different name. The iiqtools_cluster_backup tool requires a password to be supplied, either as a CLI argument or interactively. When setting up a crontab, you must use the CLI argument option. This means that the password will be in clear text in the crontab file. **TODO link to stackoverflow** This is the main reason that setting up an alternate account is a great idea! All local users on the host machine running InsightIQ are by default admin account in the application.

To create the `iiq_backup` user account, run the following command:

```
[administrator@localhost ~]$ sudo useradd iiq_backup && sudo passwd iiq_backup
```

Once that user is created, you'll have to give them access to the key file:

```
[administrator@localhost ~]$ sudo chmod 440 /etc/isilon/secret_key
[administrator@localhost ~]$ sudo chown :iiq_backup /etc/isilon/secret_key
```

2.5.2 Usage Examples

Here are some examples of using the iiqtools_backup_cluster tool.

Printing available clusters:

```
[administrator@localhost ~]$ iiqtools_cluster_backup --show-clusters
Clusters monitored by InsightIQ
-------------------------------
  myCluster
  myOtherCluster
  isi-nas-01
```

Interactively supplying the password:

```
[administrator@localhost ~]$ iiqtools_cluster_backup --clusters myOtherCluster --
↪location /mnt/backups --username iiq_backup
Please enter the password for iiq_backup :
Cluster archive underway.
To monitor status you can either follow /var/log/insightiq_export_import.log or
check the Settings page in the InsightIQ UI.
```

Backing up to an NFS export:

```
[administrator@localhost ~]$ iiqtools_cluster_backup --clusters myCluster --location␣
↪10.7.1.2:/ifs/data --username iiq_backup --password a
Cluster archive underway.
To monitor status you can either follow /var/log/insightiq_export_import.log or
check the Settings page in the InsightIQ UI.
```

Backing up multiple clusters:

```
[administrator@localhost ~]$ iiqtools_cluster_backup --clusters myCluster isi-nas-01 -
↪-location /mnt/backups --username iiq_backup --password a
Cluster archive underway.
To monitor status you can either follow /var/log/insightiq_export_import.log or
check the Settings page in the InsightIQ UI.
```

Trying to backup a cluster while the InsightIQ application is offline:

```
[administrator@localhost ~]$ iiqtools_cluster_backup --clusters myCluster --location␣
↪10.7.1.2:/ifs/data --username iiq_backup --password a
***Unable to communicate with the InsightIQ API***
Please verify that the insightiq service is running and try again
```

Limiting the number of historic backup files to 6:

```
[administrator@localhost ~]$ iiqtools_cluster_backup --max-backups 6--clusters␣
↪myCluster isi-nas-01 --location /mnt/backups --username iiq_backup --password a
```

Note: The `--max-backups` argument only deletes an old backup if the existing number of backups *before* starting a new backup is larger than the supplied value. For example, if you set `--max-backups` to 4 and the directory you are backing up to already has 4 previous backups, then (assuming the backup completes successfully) you will have a grand total of 5 backups in that directory.

2.5.3 Crontab Examples

This section assumes you've created the `iiq_backup` user account.

Note: Only one backup can happen at a time.

Backup every Monday at 1:00 AM

```
0 1 * * * mon iiqtools_cluster_backup --clusters myCluster myOtherCluster isi-nas-01 -
↪-location isi-nas.corp:/ifs/iiq/backups --username iiq_backup --password a
```

Backup only the cluster you care about, once a month at 2:00 AM

```
0 2 1 * * * iiqtools_cluster_backup --clusters myCluster --location /mnt/backups --
↪username iiq_backup --password a
```

CHAPTER 3

Contributing

We love contributions from anyone! What's a contribution you ask, well just about anything.

3.1 Feature Requests

Got an idea for a script or tool to add to IIQTools, but need some help making it? File an issue on Github. Please keep in mind that we can't alter InsightIQ source code, so there's literal zero chance of adding something to the InsightIQ UI.

3.2 Bugs

What is a bug? Well, anything that impacts your ability to use IIQTools. This includes things like:

- "the script generated this traceback"
- "your docs are wrong"
- "that script didn't do what I thought it would"

but it's not limited to that. This package exist to make your life easier, so if part of it is confusing, even some part of the process, let us know by filing an issue on Github.

Note: If it's some sort of code bug, please provide the version of InsightIQ, IIQTools, and any tracebacks that were generated. We need that information to fix the bug.

3.3 Source code

Wow! Thanks for wanting to add some source code to IIQTools. The process is pretty standard Github stuff:

1. Fork the repo
2. Add code to your fork
3. Put up a Pull Request to our master branch

There are a few of rules to keep in mind:

1. Write some unit tests. We're not demanding 100% coverage, but the closer the better.
2. Don't incorporate any additional 3rd party libraries. Some of our users cannot access the internet, and adding 3rd party libs complicates the install process.
3. Follow our docstring format (look at existing code), and update the docs directory as necessary.

CHAPTER 4

Source code docs

This is all auto-generated documentation from the IIQTools source code. It defines the different modules, their APIs, and some examples.

4.1 iiqtools.exceptions

Centralized location for all custom Exceptions

exception iiqtools.exceptions.**CliError**(*command*, *stdout*, *stderr*, *exit_code*, *message='Command Failure'*)

Raised when an CLI command has a non-zero exit code.

Parameters

- **command** (*String*) – The CLI command that was ran
- **stdout** (*String*) – The output from the standard out stream
- **stderr** (*String*) – The output from the standard error stream
- **exit_code** – The exit/return code from the command

exception iiqtools.exceptions.**DatabaseError**(*message*, *pgcode*)

Raised when an error occurs when interacting with the InsightIQ database

Attribute pgcode The error code used by PostgreSQL. https://www.postgresql.org/docs/9.3/static/errcodes-appendix.html

Attribute message The error message

4.2 iiqtools.utils

The *utils* module for the IIQTools package. These are common-need/generic functions for making scripts/tools for use with InsightIQ.

4.2.1 cli_parsers

This module contains functions that parse stdout of a CLI command into a usable Python data structure.

iiqtools.utils.cli_parsers.**df_to_dict**(*output*)
Parse the output from the command *df* into a dictionary

Returns Dictionary

Parameters **output** (*String*) – **Required** The pile of stuff outputted by running *df*

iiqtools.utils.cli_parsers.**ifconfig_to_dict**(*ifconfig_output*)
Parse the output from the command *ifconfig* into a dictionary

Returns Dictionary

Parameters **ifconfig_output** (*String*) – **Required** The pile of stuff outputted by running *ifconfig*

iiqtools.utils.cli_parsers.**memory_to_dict**(*output*)
Parse the output from the command *free -m* into a dictionary

Returns Dictionary

Parameters **output** (*String*) – **Required** The pile of stuff outputted by running *free -m*

4.2.2 database

Utilities for interacting with the InsightIQ database

class iiqtools.utils.database.**Column**
A database column consiting of the column's name, and it's type

Type namedtuple

Parameters

- **name** – The column's name
- **type** – The database type for the given column (i.e. int, float, double)

class iiqtools.utils.database.**Database**(*user='postgres'*, *dbname='insightiq'*)
Simplifies communication with the database.

The goal of this object is to make basic interactions with the database simpler than directly using the psycopg2 library. It does this by reducing the number of API methods, providing handy built-in methods for common needs (like listing tables of a database), auto-commit of transactions, and auto-rollback of bad SQL transactions. This object is not indented for power users, or long lived processes (like the InsightIQ application); it's designed for shorter lived "scripts".

Parameters

- **user** (*String, default postgres*) – The username when connection to the databse
- **dbname** (*String, default insightiq*) – The specific database to connection to. InsightIQ utilizes a different database for every monitored cluster, plus one generic database for the application (named "insightiq").

close()
Disconnect from the database

cluster_databases()
Obtain a list of all the cluster databases

Returns List

execute (*sql*, *params=None*)
Run a single SQL command

Returns Generator

Parameters

- **sql** (*String*) – **Required** The SQL syntax to execute
- **params** (*Iterable*) – The values to use in a parameterized SQL query

This method is implemented as a Python Generator: https://wiki.python.org/moin/Generators This means you are suppose to iterate over the results:

```
db = Database()
for row in db.execute("select * from some_table;"):
    print row
```

If you want all the rows as a single thing, just use `list`:

```
db = Database()
data = list(db.execute("select * from some_table;")
```

But **WARNING** that might cause your program to run out of memory and crash! That reason is why this method is a generator by default ;)

To perform a parameterized query (i.e. avoid SQL injection), provided the parameters as an iterable:

```
db = Database()
# passing in "foo_column" alone would try and string format every
# character of "foo_column" into your SQL statement.
# Instead, make "foo_column" a tuple by wrapping it like ("foo_column",)
# Note: the trailing comma is required.
data = list(db.execute("select %s from some_table", ("foo_column",)))
```

executemany (*sql*, *params*)
Run the SQL for every iteration of the supplied params

This method behaves exactly like *execute*, except that it can perform multiple SQL commands in a single transaction. The point of this method is so you can retain Atomicity when you must execute the same SQL with different parameters. This method isn't intended to be faster than looping over the normal *execute* method with the different parameters.

Returns Generator

Parameters

- **sql** (*String*) – **Required** The SQL syntax to execute
- **params** (*Iterable*) – **Required** The parameterized values to iterate

isolation_level
Set the isolation level of your connnection to the database

primary_key (*table*)
Given a table, return the primary key

Note: If you supply a timeseries table that DOES NOT have an EPOC timestamp in the name, you will get zero results. For timeseries tables, supply a table that contains the EPOC timestamps to see the primary

key.

Returns Tuple of namedtuples -> (Column(name, type), Column(name, type))

Parameters **table** (*String*) – **Required** The table to obtain the primary key from

table_schema(*table*)
Given a table, return the schema for that table

Returns Tuple of namedtuples -> (Column(name, type), Column(name, type))

Parameters **table** (*String*) – **Required** The table to obtain the primary key from

tables()
Obtain a list of all the tables for the database you're connected to

Returns List

4.2.3 generic

This module contains miscellaneous utility functions

iiqtools.utils.generic.**check_path**(*cli_value*)
Validate that the supplied path is an actual file system directory.

This function is intended to be used with the argparse lib as an argument type.

Raises argparse.ArgumentTypeError

Returns String

Parameters **cli_value** (*String*) – The value supplied by the end user

iiqtools.utils.generic.**printerr**(*message*)
Just like print(), but outputs to stderr.

Returns None

Parameters **message** (*PyObject*) – The thing to write to stderr

4.2.4 logger

iiqtools.utils.logger.**get_logger**(*log_path=None*, *stream_lvl=0*, *file_lvl=20*)
Factory for making logging objects

The verbosity of the logs are configurable as defined by the official Python documentation: https://docs.python.org/2/library/logging.html#logging-levels

Returns logging.Logger

Raises AssertionError on bad parameter input

Parameters

- **log_path** (*String*) – **Required** The absolute file path to write logs to
- **stream_lvl** (*Integer, default 20*) – Set to print log messages to the terminal.
- **file_lvl** – How verbose the log file messages are. This value cannot be zero.

4.2.5 shell

This module reduces boilerplate when interacting with the command shell, i.e. BASH.

Example A:

```
>>> result = shell.run_cmd('ls')
>>> print result.stdout
README.txt
someOtherFile.txt
>>> print result.stderr

>>> result.exit_code
0
```

Example B:

```
>>> # run_cmd does not support the Unix Pipeline
>>> result = shell.run_cmd('cat foo.txt | grep "not supported"')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    raise CliError(cli_syntax, stdout, stderr, exit_code)
iiqtools.exceptions.CliError: Command Failure: cat foo.txt | grep "not supported"
```

class iiqtools.utils.shell.**CliResult**

The outcome from running a CLI command

Type collections.namedtuple

Parameters

- **command** (*String*) – The CLI command that was ran
- **stdout** (*String*) – The output from the standard out stream
- **stderr** (*String*) – The output from the standard error stream
- **exit_code** – The exit/return code from the command

iiqtools.utils.shell.**run_cmd**(*cli_syntax*)

Execute a simple CLI command.

This function blocks until the CLI command returns and does not support the Unix pipeline.

Returns CliResult - namedtuple

Raises CliError (when exit code is not zero)

Parameters **cli_syntax** (*String*) – The CLI command to run.

4.2.6 versions

This module make obtaining and comparing version strings easy!

class iiqtools.utils.versions.**PatchInfo**

Describes the state of patches for InsightIQ

Parameters

- **iiq_dir** (*String*) – The file system path where InsightIQ source is located.
- **patches_dir** (*String*) – The file system path where patches for InsightIQ are stored.

- **specific_patch** (*String*) – Only populated when a patch is being installed/removed/read.
- **is_installed** – If specific_patch is installed or not.
- **readme** (*String*) – The README.txt for specific_patch if applicable.
- **all_patches** (*Tuple*) – All currently installed patches.

Type is_installed Boolean

class iiqtools.utils.versions.**Version**(*version*, *name*)

Implements comparison operators for common version strings

Only versions strings upwards of 4 digits and consisting only of numbers is supported. Version strings breakdown into major, minor, patch, and build.

Example:

```
>>> version_as_string = '1.2.3'
>>> version = Version(name='myPackage', version=version_as_string)
>>> version.major
1
>>> version.patch
3
>>> type(version.minor)
<type 'int'>
>>> type(version.build)
<type 'NoneType'>
```

Comparing Versions:

```
>>> v1 = Version(name='myPackage', version='1.2.3')
>>> v2 = Version(name='newPackage', version='1.4.0')
>>> v1 < v2
True
>>> v2 >= v1
True
>>> v2 == v1
False
```

The Version object also support comparison of string versions:

```
>>> v1 = Version(name='foo', version='4.5')
>>> v1 > '5.0'
False
```

It's worth noting, more specific versions (that would otherwise be equal), are considered greater:

```
>>> v1 = Version(name='bar', version='1.2.0')
>>> v1 > '1.2'
True
```

This is because a version without a value is None, and zero is greater than None in Python:

```
>>> 0 > None
True
```

iiqtools.utils.versions.**get_iiq_version**()

Obtain the version of InsightIQ installed

Returns iiqtools.utils.versions.Version

iiqtools.utils.versions.**get_iiqtools_version**()
Obtain the version of iiqtools installed

Returns iiqtools.utils.versions.Version

iiqtools.utils.versions.**get_patch_info**(*specific_patch*, *log*)
Obtain the current state of patches for InsightIQ

Returns PatchInfo (namedtuple)

Parameters

- **specific_patch** (*String*) – **Required** The name of a patch that's being installed/removed/read.
- **log** (*logging.Logger*) – **Required** The logging object. This param is really here to make unit testing easier -> https://en.wikipedia.org/wiki/Dependency_injection

4.2.7 insightiq_api

This module is for performing privileged API calls to InsightIQ.

exception iiqtools.utils.insightiq_api.**ConnectionError**
Unable to establish an connection to the OneFS API

class iiqtools.utils.insightiq_api.**InsightiqApi**(*username*, *password*, *verify=False*)
An authenticated connection to the InsightIQ API

This object is a simple wrapper around a requests Session. The point of wrapping the requests Session object is to remove boiler plate code in making API calls to InsightIQ, and to auto-handle authenticating to the API. The most noteworthy changes to this object and how you use the requests Session object is that you *must* provide the username and password when instantiating the object, and when you make a request, you only supply the URI end point (i.e. not the *http://my-host.org:8080* part).

Supports use of `with` statements, which will automatically handle creating and closing the HTTP session with InsightIQ.

Example:

```
>>> with InsightiApi(username='administrator', password='foo') as iiq:
        response = iiq.get('/api/clusters')
```

Parameters

- **username** (*String*) – **Required** The name of the administrative account for InsightIQ
- **password** (*String*) – **Required** The password for the administrative account being used.
- **verify** (*Boolean*) – Perform SSL/TLS cert validation using system certs. Setting to True will likely cause issues when using a self-signed SSL/TLS cert. Default is False.

delete(*endpoint*, *params=None*, *data=None*, *headers=None*, ***kwargs*)
Perform an HTTP DELETE request

Returns PyObject

Parameters

- **endpoint** (*String*) – **Required** The URI end point of the InsightIQ API to call

- **params** (*Dictionary*) – The HTTP parameters to send in the HTTP request
- **data** (*PyObject*) – The HTTP body content to send in the request. The Python object supplied (i.e. list, dict, etc) will be auto-converted to JSON string.
- **headers** (*Dictionary*) – Any additional HTTP headers to send in the request

end_session()
Logout of InsightIQ and close the connection to the server

get(*endpoint*, *params=None*, *data=None*, *headers=None*, ***kwargs*)
Perform an HTTP GET request

Returns PyObject

Parameters

- **endpoint** (*String*) – **Required** The URI end point of the InsightIQ API to call
- **params** (*Dictionary*) – The HTTP parameters to send in the HTTP request
- **data** (*PyObject*) – The HTTP body content to send in the request. The Python object supplied (i.e. list, dict, etc) will be auto-converted to JSON string.
- **headers** (*Dictionary*) – Any additional HTTP headers to send in the request

head(*endpoint*, *params=None*, *data=None*, *headers=None*, ***kwargs*)
Perform an HTTP HEAD request

Returns PyObject

Parameters

- **endpoint** (*String*) – **Required** The URI end point of the InsightIQ API to call
- **params** (*Dictionary*) – The HTTP parameters to send in the HTTP request
- **data** (*PyObject*) – The HTTP body content to send in the request. The Python object supplied (i.e. list, dict, etc) will be auto-converted to JSON string.
- **headers** (*Dictionary*) – Any additional HTTP headers to send in the request

post(*endpoint*, *params=None*, *data=None*, *headers=None*, ***kwargs*)
Perform an HTTP POST request

Returns PyObject

Parameters

- **endpoint** (*String*) – **Required** The URI end point of the InsightIQ API to call
- **params** (*Dictionary*) – The HTTP parameters to send in the HTTP request
- **data** (*PyObject*) – The HTTP body content to send in the request. The Python object supplied (i.e. list, dict, etc) will be auto-converted to JSON string.
- **headers** (*Dictionary*) – Any additional HTTP headers to send in the request

put(*endpoint*, *params=None*, *data=None*, *headers=None*, ***kwargs*)
Perform an HTTP PUT request

Returns PyObject

Parameters

- **endpoint** (*String*) – **Required** The URI end point of the InsightIQ API to call
- **params** (*Dictionary*) – The HTTP parameters to send in the HTTP request

- **data** (*PyObject*) – The HTTP body content to send in the request. The Python object supplied (i.e. list, dict, etc) will be auto-converted to JSON string.
- **headers** (*Dictionary*) – Any additional HTTP headers to send in the request

renew_session()

Create a new session to the InsightIQ API

Returns requests.Session

Raises ConnectionError

The InsightIQ API can be a bit fickle, so this method automatically retries establishing upwards of 3 times.

class iiqtools.utils.insightiq_api.**Parameters**(**args*, ***kwargs*)

Object for working with HTTP query parameters

This object supports the Python dictionary API, and lets you define the same HTTP query parameter more than once. Additional definitions for the same query parameter creates a new entry in the underlying list. This decision makes it simple to iterate Parameters to build up the HTTP query string because you do not have to iterate parameter values. Because you can define the same parameter more than once, using the standard dictionary API will only impact the first occurrence of that parameter. To modify a specific parameter, you must use the methods in this class which extend the dictionary API.

This documentation is specific to how Parameters extends the normal Python dictionary API. For documentation about the Python dictionary API, please checkout their official page here.

Example creating duplicate parameters:

```
>>> params = Parameters()
>>> for value in range(3):
...     params.add('myParam', value)
...
Parameters([['myParam', 0], ['myParam', 1], ['myParam', 2]])
```

What **NOT** to do:

```
>>> params = Parameters()
>>> for doh in range(3):
...     params['homer'] = doh
...
>>> params
Parameters([['homer', 2], ['homer', 2], ['homer', 2]])
```

Iterating Parameters to build a query string:

```
>>> query = []
>>> params = Parameters(one=1, two=2)
>>> for name, value in params.items():
...     query.append('%s=%s' % (name, value))
...
>>> query_str = '&'.join(query)
>>> query_str
'one=1&two=2'
```

Parameters

- **args** (*List, Tuple, or Dictionary*) – Data to initialize the Parameters object with.
- **kwargs** (*Dictionary*) – Data to initialize the Parameters object with.

NAME
The array index for a parameter name; avoids magic numbers

VALUE
The array index for a parameter value; avoids magic numbers

add(*name*, *value*)
Add a duplicate parameter

Returns None

Parameters

- **name** (*String*) – **Required** The name of the parameter
- **value** (*PyObject*) – **Required** The value for the duplicate parameter

delete_parameter(*name*, *occurrence*)
Delete a specific parameter that is defined more than once. Thread safe.

Returns None

Parameters

- **name** (*String*) – **Required** The parameter to delete.
- **occurrence** (*Integer*) – **Required** The N-th instance of a parameter. Zero based numbering.

get_all(*name*)
Return the key/value pairs for a parameter. Order is maintained.

Returns List

Parameters **name** (*String*) – **Required** The name of the query parameter

items()
Iterate Parameters, and return the param/value pairs

Returns Generator

modify_parameter(*name*, *new_value*, *occurrence*)
Change the value of a specific parameter that is defined more than once. Thread safe.

Returns None

Parameters

- **name** (*String*) – **Required** The parameter to delete.
- **new_value** (*PyObject*) – **Required** The value for the parameter
- **occurrence** (*Integer*) – **Required** The N-th instance of a parameter. Zero based numbering.

4.3 iiqtools.iiqtools_gather_info

This module contains all the business logic for collecting logs and configuration information about InsightIQ for remote troubleshooting.

iiqtools.iiqtools_gather_info.**add_from_memory**(*the_tarfile*, *data_name*, *data*)
Simplify adding in-memory information to the tar file

Returns None

Parameters

- **the_tarfile** (*tarfile.open*) – The open tarfile object
- **data_name** (*String*) – The reference to the data; i.e. it's name when you uncompress the file
- **data** (*String*) – The contents of the in-memory information

iiqtools.iiqtools_gather_info.**call_iiq_api**(*uri*)
Make an internal API call to the InsightIQ API

Returns JSON String

Parameters **uri** (*string*) – The API end point to call

iiqtools.iiqtools_gather_info.**cli_cmd_info**(*command*, *parser*)
Standardizes the JSON format for any data collected via a CLI command

Returns JSON String

Parameters **command** (*String*) – The CLI command to execute

iiqtools.iiqtools_gather_info.**clusters_info**()
Obtain a pile of data about all monitored clusters in InsightIQ

Returns JSON String

iiqtools.iiqtools_gather_info.**datastore_info**()
Obtain data about the datastore for InsightIQ

Returns JSON String

iiqtools.iiqtools_gather_info.**get_tarfile**(*output_dir*, *case_number*, *the_time=None*)
Centralizes logic for making tgz file for InsightIQ logs

Returns tarfile.TarFile

Parameters

- **output_dir** (*String*) – **Required** The directory to save the tar file in
- **case_number** (*String or Integer*) – **Required** The SR that the logs are for; used in file name.
- **the_time** (*EPOC time stamp*) – An optional EPOC timestamp to use when naming the file. If not supplied, this function calls time.time().

iiqtools.iiqtools_gather_info.**ifconfig_info**()
Obtain data about the network interfaces on the host OS

Returns JSON String

iiqtools.iiqtools_gather_info.**iiq_version_info**()
Obtain info about the version of InsightIQ installed on the host OS

Returns JSON String

iiqtools.iiqtools_gather_info.**ldap_info**()
Obtain the config for LDAP in InsightIQ

Returns JSON String

iiqtools.iiqtools_gather_info.**main**(*the_cli_args*)
Entry point for running script

iiqtools.iiqtools_gather_info.**memory_info**()
Obtain data about RAM on the host OS

Returns JSON String

iiqtools.iiqtools_gather_info.**mount_info**()
Obtain data about mounted file systems on the host OS

Returns JSON String

iiqtools.iiqtools_gather_info.**parse_cli**(*cli_args*)
Handles parsing the CLI, and gives us –help for (basically) free

Returns argparse.Namespace

Parameters **cli_args** (*List*) – The arguments passed to the script

iiqtools.iiqtools_gather_info.**reports_info**()
Obtain info about any scheduled reports in InsightIQ

Returns JSON String

4.4 iiqtools.iiqtools_patch

This module contains all the business logic for collecting logs and configuration information about InsightIQ for remote troubleshooting.

iiqtools.iiqtools_gather_info.**add_from_memory**(*the_tarfile*, *data_name*, *data*)
Simplify adding in-memory information to the tar file

Returns None

Parameters

- **the_tarfile** (*tarfile.open*) – The open tarfile object
- **data_name** (*String*) – The reference to the data; i.e. it's name when you uncompress the file
- **data** (*String*) – The contents of the in-memory information

iiqtools.iiqtools_gather_info.**call_iiq_api**(*uri*)
Make an internal API call to the InsightIQ API

Returns JSON String

Parameters **uri** (*string*) – The API end point to call

iiqtools.iiqtools_gather_info.**cli_cmd_info**(*command*, *parser*)
Standardizes the JSON format for any data collected via a CLI command

Returns JSON String

Parameters **command** (*String*) – The CLI command to execute

iiqtools.iiqtools_gather_info.**clusters_info**()
Obtain a pile of data about all monitored clusters in InsightIQ

Returns JSON String

iiqtools.iiqtools_gather_info.**datastore_info**()
Obtain data about the datastore for InsightIQ

Returns JSON String

iiqtools.iiqtools_gather_info.**get_tarfile**(*output_dir*, *case_number*, *the_time=None*)
Centralizes logic for making tgz file for InsightIQ logs

Returns tarfile.TarFile

Parameters

- **output_dir** (*String*) – **Required** The directory to save the tar file in
- **case_number** (*String or Integer*) – **Required** The SR that the logs are for; used in file name.
- **the_time** (*EPOC time stamp*) – An optional EPOC timestamp to use when naming the file. If not supplied, this function calls time.time().

iiqtools.iiqtools_gather_info.**ifconfig_info**()
Obtain data about the network interfaces on the host OS

Returns JSON String

iiqtools.iiqtools_gather_info.**iiq_version_info**()
Obtain info about the version of InsightIQ installed on the host OS

Returns JSON String

iiqtools.iiqtools_gather_info.**ldap_info**()
Obtain the config for LDAP in InsightIQ

Returns JSON String

iiqtools.iiqtools_gather_info.**main**(*the_cli_args*)
Entry point for running script

iiqtools.iiqtools_gather_info.**memory_info**()
Obtain data about RAM on the host OS

Returns JSON String

iiqtools.iiqtools_gather_info.**mount_info**()
Obtain data about mounted file systems on the host OS

Returns JSON String

iiqtools.iiqtools_gather_info.**parse_cli**(*cli_args*)
Handles parsing the CLI, and gives us –help for (basically) free

Returns argparse.Namespace

Parameters **cli_args** (*List*) – The arguments passed to the script

iiqtools.iiqtools_gather_info.**reports_info**()
Obtain info about any scheduled reports in InsightIQ

Returns JSON String

4.5 iiqtools.iiqtools_tar_to_zip

This script converts the format of datastore exports (not the CSV exports) from tar to zip. In InsightIQ 4.1, the format was changed to fix bug 162840, in which an attempt to import a large datastore export would time out. The only change to the exported data is the format. So to use an export from an older instance (before 4.1) all you have to convert the format. In other words, the data is still the same, it's just a different compression format in InsightIQ 4.1.

class iiqtools.iiqtools_tar_to_zip.**BufferedZipFile**(*file*, *mode='r'*, *compression=0*, *allowZip64=False*)

A subclass of zipfile.ZipFile that can read from a file-like object and stream the contents into a new zip file.

writebuffered(*filename*, *file_handle*, *file_size*)

Stream write data to the zip archive

Parameters

- **filename** (*String*) – **Required** The name to give the data once added to the zip file
- **file_handle** (Anything that supports the read method) – **Required** The file-like object to read
- **file_size** (*Integer*) – **Required** The size of the file in bytes

iiqtools.iiqtools_tar_to_zip.**check_tar**(*value*)

Validate that the supplied tar file is an InsightIQ datastore export file.

Raises argparse.ArgumentTypeError

Returns String

Parameters **value** (*String*) – **Required** The CLI value to validate

iiqtools.iiqtools_tar_to_zip.**get_timestamp_from_export**(*source_tar*)

Allows us to create the new zip archive with the correct timestamp

Returns String

Parameters **source_tar** (*String*) – **Required** The tar that's being converted to a zip

iiqtools.iiqtools_tar_to_zip.**joinname**(*export_dir*, *file_name*)

The tar/zip used by InsightIQ expects the data nested in a directory. This function handles absolute and relative paths for file_name.

Returns String

Parameters

- **export_dir** (*String*) – **Required** The directory name to nest the file under
- **file_name** (*String*) – **Required** The name of the filed nested in the directory

iiqtools.iiqtools_tar_to_zip.**main**(*the_cli_args*)

Entry point for the iiq_tar_to_zip script

iiqtools.iiqtools_tar_to_zip.**parse_cli**(*the_cli_args*)

Handles parsing the CLI, and gives us –help for (basically) free

Returns argparse.Namespace

Parameters **cli_args** (*List*) – **Required** The arguments passed to the script

4.6 iiqtools.iiqtools_version

This module contains the business logic for print the versions of InsightIQ and IIQTool that's installed.

iiqtools.iiqtools_version.**main**(*the_cli_args*)

Entry point for iiq_version script

iiqtools.iiqtools_version.**parse_cli**(*the_cli_args*)

Defines the CLI interface for `iiq_version`

CHAPTER 5

Patching InsightIQ

This section explains how the patching mechanism works and how to create new patches for InsightIQ. This section is indented for source code contributors.

If you're looking for documentation about install/uninstall patches, please checkout the Scripts section.

5.1 How patching works

5.1.1 Where patches are stored

The patching mechanism creates a directory under the installation path of the InsightIQ source code. For example, if you deploy a new OVA of InsightIQ 4.1.1, the installation path is `/usr/share/isilon/lib/python2.7/site-packages/insightiq`. The directory created is named `patches`, for obvious reasons. The `patches` directory contains subdirectories; one for each patch currently installed.

The reason for putting the `patches` directory under the InsightIQ source code is it simplifies upgrades. When you ask the patching mechanism "what patches are currently installed" after an upgrade, it'll always report that no patches are installed. In otherwords, upgrading InsightIQ removes all patches. There's no way to avoid this; Python will overwrite the installation directory when upgrading the InsightIQ package.

Example file structure before InsightIQ upgrade:

```
insightiq
  \patches
    \patch-1234
    \patch-598
  \config
  \core
  features.py
```

Example file structure after InsightIQ upgrade:

```
insightiq
  \config
  \core
  features.py
```

This simplifies upgrades because we A) don't have to try an "re-apply" a patch after an upgrade, or B) update our list of installed patches after an upgrade.

Specific patch directory contents

Lets say we have patch-1234 installed. The directory created (which is used to reference the patch) contains the following:

```
\patch-1234
  meta.ini
  README.txt
  \originals
```

The `meta.ini` and `README.txt` are defined in the next section, and come from the tar patch file. The directory `originals` contains backup copies of the original source files. When you install a patch, the patching tool creates these backups to enable us to uninstall the patch. Within that directory, you'll see some rather long file names:

```
\originals
  insightiq___controllers___security.py
```

The triple-underbars ___ are used to replace the forward slash normally associated with a file system path. A triple-underbar is used instead of a single-underbar because it's common for a Python source file to contain a single-underbar.

5.1.2 What's in a patch

A patch for InsightIQ is a tar file consisting of the following files:

README.txt A description of the patch. This must always include the bug number for the issue that requires patching.

meta.ini A config file that defines what source files are getting patched, and what versions of InsightIQ the patch works with.

PATCHED_FILE The patched source file, where PATCHED_FILE is actually the name of the source file. There can be as many as these files as needed for the patch.

Within the tar file, these files must be stored in a directory that names the patch following the convention of patch-PATCH_NUMBER, where PATCH_NUMBER is the actual number used to identify the patch for Isilon. In other words, if you unfurl the patch tar file, a directory with the patch files is written to your file system.

Example of patch file contents:

```
README.txt
meta.ini
insightiq/controllers/security.py
```

The meta.ini

The format is your standard INI file, with the headings `info`, `version` and `files`.

The `info` heading has two keys, `name` which is the name of the patch, and `bug` which is the number for the associated bug that is being patched.

The `versions` heading has two keys, `minimum` and `maximum`, and as you've likely guessed, define the oldest and newest versions of InsightIQ that the patch applies to.

The `files` heading defines what files are being patched, and can have as many keys as necessary. The key names should be the location of the original source file (relative to the installation directory), and the key values are the md5 hash of that original source file. Why are we including md5 hashes? To avoid installing patches that clobber (i.e. patch B overwrites the same file as patch A).

Example meta.ini:

```
[info]
name = patch-1234
bug = 156986
[version]
minimum = 4.1.0
maximum = 4.1.1
[files]
insightiq/controllers/security.py = 56266031a30cab220f56ee43b4159ded
```

Note: Version values are inclusive. If your patch *only* applies to one specific release, both `minimum` and `maximum` should have the same value.

5.2 Making a patch

There are the steps to create a patch for a single source file. In this example, we are patching insightiq/controllers/security.py

1. Create a directory for the patch:

   ```
   $ mkdir patch-1234
   ```

2. Copy your README.txt and meta.ini files into the directory made in step 1:

   ```
   $ cp README.txt meta.ini patch-1234
   ```

3. Create all subdirectories for the source file paths:

   ```
   $ mkdir -p patch-1234/insightiq/controllers
   ```

Copy the patched files unto their respective locations::

```
$ cp security.py patch-1234/insightiq/controllers/
```

1. Create the tgz file:

   ```
   $ cd patch-1234 && tar -zcvf insightiq-patch-1234.tgz *
   ```

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

i

Index

J

L

M

N

P

R

T

V

W