hpycc Documentation

Author

Nov 02, 2020

Contents:

1 hpycc Readme **1**
- 1.1 Docker Announcement . . . 1
- 1.2 Documentation . . . 1
- 1.3 Installation . . . 1
- 1.4 Current Status . . . 2
- 1.5 Dependencies . . . 2
- 1.6 Main Functions . . . 2
  - 1.6.1 connection(username, server="localhost", port=8010, repo=None, password="password", legacy=False, test_conn=True) . . . 2
  - 1.6.2 get_output(connection, script, ...) & save_output(connection, script, path, ...) . . . 2
  - 1.6.3 get_outputs(connection, script, ...) . . . 2
  - 1.6.4 get_thor_file(connection, logical_file, path, ...) & save_thor_file(connection, logical_file, path, ...) . . . 2
  - 1.6.5 run_script(connection, script, ...) . . . 3
  - 1.6.6 spray_file(connection, source_file, logical_file, ...) . . . 3
  - 1.6.7 docker_tools.HPCCContainer(tag="6.4.26-1", ...) . . . 3
- 1.7 Examples . . . 3
- 1.8 Issues, Bugs, Comments? . . . 4

2 hpycc package **5**
- 2.1 Subpackages . . . 5
  - 2.1.1 hpycc.utils package . . . 5
    - 2.1.1.1 Submodules . . . 5
    - 2.1.1.2 hpycc.utils.docker_tools module . . . 5
    - 2.1.1.3 hpycc.utils.filechunker module . . . 5
    - 2.1.1.4 hpycc.utils.parsers module . . . 6
    - 2.1.1.5 Module contents . . . 7
- 2.2 Submodules . . . 7
- 2.3 hpycc.connection module . . . 7
  - 2.3.1 Classes . . . 7
- 2.4 hpycc.delete module . . . 9
  - 2.4.1 functions . . . 9
- 2.5 hpycc.get module . . . 10
  - 2.5.1 Functions . . . 10
- 2.6 hpycc.run module . . . 14
  - 2.6.1 Functions . . . 14

2.7 hpycc.save module 15
2.7.1 Functions 15
2.8 hpycc.spray module 16
2.8.1 Functions 16

3 Indices and tables **19**

Python Module Index **21**

Index **23**

CHAPTER 1

hpycc Readme

The hpycc package is intended to simplify the use of data stored on HPCC and make it easily available to both users and other servers through basic Python calls. Its long-term goal is to make access to and manipulation of HPCC data as quick and easy as any other type system.

1.1 Docker Announcement

HPCC seem to have pulled the images that this is based on. I've created one for the test environment I developed against but if you want ot dev against a different one you will need to build the relevent container. I've forked the old HPCC docker repo in case you need it or it gets removed: https://github.com/datamacgyver/docker-hpcc

Note that in the docker files I've tried the wget for HPCC is broken. You can get the new ones from the HPCC website but there's a modified one in this repoo I used for the build.

1.2 Documentation

The below readme and package documentation is available at https://hpycc.readthedocs.io/en/latest/

The package's github is available at: https://github.com/OdinProAgrica/hpycc

This package is released under GNU GPLv3 Licence: https://www.gnu.org/licenses/gpl-3.0.en.html

Want to use this in R? Then the reticulate package is your friend! Just save as a CSV and read back in. That or you can use an R notebook with a Python chunk.

1.3 Installation

Install with:

pip install hpycc

Or, if you are still a bit old school:

python -m pip install hpycc

1.4 Current Status

Tested and working on HPCC v6.4.2 and python 3.5.2 under windows 10. Has been used on Linux systems but not extensively tested.

1.5 Dependencies

The package itself mainly uses core Python, Pandas is needed for outputting dataframes.

There is a dependency for client tools to run ECL scripts (you need ecl.exe and eclcc.exe). Make sure you install the right client tools for your HPCC version and add the dir to your system path, e.g. C:\Program Files (x86)\HPCCSystems\X.X.X\clienttools\bin.

Tests and docker container functions require docker to spin up HPCC environments.

1.6 Main Functions

Below summarises the key functions and non-optional parameters. For specific arguments see the relevant function's documentation. Note that while retrieving a file is a multi-thread process, running a script and getting the results is not. Therefore if your file is quite big you may be better off saving the results of a script using run.run_script() with a thor file output then downloading the file with get.get_thor_file().

1.6.1 connection(username, server="localhost", port=8010, repo=None, password="password", legacy=False, test_conn=True)

Create a connection to a new HPCC instance. This is then passed to any interface functions.

1.6.2 get_output(connection, script, ...) & save_output(connection, script, path, ...)

Run a given ECL script and either return the first result as a pandas dataframe or save it to file.

1.6.3 get_outputs(connection, script, ...)

Run a given ECL script and return all results as a dict of pandas dataframes or save them to files.

1.6.4 get_thor_file(connection, logical_file, path, ...) & save_thor_file(connection, logical_file, path, ...)

Get a logical file and either return as a pandas dataframe or save it to file.

1.6.5 run_script(connection, script, . . .)

Run a given ECL script. 10 rows will be returned but they will be dumped, no output is given.

1.6.6 spray_file(connection, source_file, logical_file, . . .)

Spray a csv or pandas DataFrame into HPCC.

1.6.7 docker_tools.HPCCContainer(tag="6.4.26-1", . . .)

Designed for our testing but made available generally, a collection of functions for running and managing HPCC docker containers is also available. The above function starts a container, see help file for shutting down and other management tasks.

1.7 Examples

The below code gives an example of functionality:

```
import hpycc
import pandas as pd
from hpycc.utils import docker_tools
from os import remove

# Start an HPCC docker image for testing
docker_tools.HPCCContainer(tag="6.4.26-1")

# Setup stuff
username = 'HPCC_dev'
test_file = 'test.csv'
f_hpcc_1 = '~temp::testfile1'
f_hpcc_2 = '~temp::testfile2'
ecl_script = 'ecl_script.ecl'

# Let's create a connection object so we can interface with HPCC.
# up with Docker
conn = hpycc.Connection(username, server="localhost")
try:
    # So, let's spray up some data:
    pd.DataFrame({'col1': [1, 2, 3, 4], 'col2': ['a', 'b', 'c', 'd']}).to_csv(test_
↪file, index=False)
    hpycc.spray_file(conn, test_file, f_hpcc_1, expire=7)

    # Lovely, we can now extract that as a Thor file:
    df = hpycc.get_thor_file(conn, f_hpcc_1)
    print(df)
    # Note __fileposition__ column. This will be drop-able in future versions.

    ################################
    #   col1 col2  \__fileposition__#
    # 0    1    a                 0 #
    # 1    3    c                20 #
    # 2    2    b                10 #
    # 3    4    d                30 #
```

(continues on next page)

(continued from previous page)

```
    ##################################

    # If preferred data can also be extracted using an ECL script.
    with open(ecl_script, 'w') as f:
        f.writelines("DATASET('%s', {STRING col1; STRING col2;}, THOR);" % f_hpcc_1)
        # Note, all columns are currently string-ified by default
    df = hpycc.get_output(conn, ecl_script)
    print(df)

    ################
    #   col1 col2  #
    # 0    1    a  #
    # 1    3    c  #
    # 2    2    b  #
    # 3    4    d  #
    ############## #


    # get_thor_file() is optimised for large files, get_output is not (yet). To run a␣
↪script and
    # download a large result you should therefore save a thor file and grab that.

    with open(ecl_script, 'w') as f:
        f.writelines("a := DATASET('%s', {STRING col1; STRING col2;}, THOR);"
                     "OUTPUT(a, , '%s');" % (f_hpcc_1, f_hpcc_2))
    hpycc.run_script(conn, ecl_script)
    df = hpycc.get_thor_file(conn, f_hpcc_2)
    print(df)

    ################################
    #   col1 col2  \__fileposition__#
    # 0    1    a                 0 #
    # 1    3    c                20 #
    # 2    2    b                10 #
    # 3    4    d                30 #
    ################################

finally:
    # Shutdown our docker container
    docker_tools.HPCCContainer(pull=False, start=False).stop_container()
    remove(ecl_script)
    remove(test_file)
```

1.8 Issues, Bugs, Comments?

Please use the package's github: https://github.com/OdinProAgrica/hpycc

Any contributions are also welcome.

CHAPTER 2

hpycc package

2.1 Subpackages

2.1.1 hpycc.utils package

2.1.1.1 Submodules

2.1.1.2 hpycc.utils.docker_tools module

Functions to create and control HPCC docker images. Requires Docker to be installed and running!!!!!!!

class hpycc.utils.docker_tools.**HPCCContainer**(*tag='latest'*, *name='hpycc_test_img'*, *users=None*, *pull=True*, *start=True*)

Bases: object

create_passwords()

pull_image()

put_archive(*b*, *name*, *path*)

setup_hpcc()

start_container()

start_hpcc()

stop_container()

2.1.1.3 hpycc.utils.filechunker module

functions that chunk an iterable.

Functions

- *make_chunks* – Return tuples of start index and chunk size.

hpycc.utils.filechunker.**make_chunks**(*num*, *chunk_size=10000*)
Return tuples of start index and chunk size.

Parameters

- **num** (*int*) – Total number of items.
- **chunk_size** (*int, optional*) – Max chunk size, 10,000 by default.

Returns **chs** – List of chunks in the form [(start_index, num_items)]

Return type list of tuples

2.1.1.4 hpycc.utils.parsers module

hpycc.utils.parsers.**apply_custom_dtypes**(*schema*, *dtypes*)

hpycc.utils.parsers.**get_python_type_from_ecl_type**(*child*)
Get the python type from an hpcc schema node

Parameters **child** (*XML node*) – Node of schema xml. See *parse_schema_from_xml*

Returns **type** – Pythonic type. If the HPCC type cannot be mapped, is str.

Return type type

hpycc.utils.parsers.**parse_schema_from_xml**(*xml*)
Parse an ECL schema into python types.

Parameters **xml** (*str*) – xml string returned by ecl run. This is located in the json as ["WUResultResponse]["Result"]["XmlSchema"]["xml"].

Returns

- *OrderedDict* – dict of column stats, in the form {name: Str, type: Str, is_a_set: Bool}.
- *list* – Column names in order of occurrence.

hpycc.utils.parsers.**parse_wuid_from_failed_response**(*result*)

hpycc.utils.parsers.**parse_wuid_from_xml**(*result*)
Function retrieves a WUID for a script that has run. This retrieves it only in the cases where the request response was in XML format.

Parameters **result** ('*XML*') – The XML response for the script that has run.

Returns **wuid** – The Workunit ID from the XML.

Return type str

hpycc.utils.parsers.**parse_xml**(*xml*)
Return a DataFrame from a nested XML.

Parameters **xml** (*str*) – xml to be parsed.

Returns **df** – Parsed xml.

Return type pd.DataFrame

2.1.1.5 Module contents

2.2 Submodules

2.3 hpycc.connection module

Object for connecting to a HPCC instance.

This module provides a *Connection* class to connect to a HPCC instance. This connection is used as the first input to the majority of public functions in the *hpycc* package.

2.3.1 Classes

- *Connection* – HPCC connection class.

class hpycc.connection.**Connection**(*username*, *server='localhost'*, *port=8010*, *repo=None*, *password='password'*, *legacy=False*, *test_conn=True*)

Bases: object

check_syntax(*script*)

Run an ECL syntax check on an ECL script.

Uses eclcc to run a syntax check on *script*. If the syntax check fails, ie. an error is present, a SyntaxError will be raised. Note that this requires that *eclcc.exe* is on the path. Attributes *legacy* and *repo* are also used.

Parameters **script** (*str*) – path to ECL script.

Returns

Return type None

Raises SyntaxError – If the script fails the syntax check.

get_chunk_from_hpcc(*logical_file*, *start_row*, *n_rows*, *max_attempts*, *max_sleep*)

Using the HPCC instance at *server*:*port* and the credentials *username* and *password*, return the JSON response to a request for a part of a *logical_file*. Starting at *start row* and *n_rows* long.

Parameters

- **logical_file** (*str*) – Name of logical file.
- **start_row** (*int*) – First row to return where 0 is the first row of the dataset.
- **n_rows** (*int*) – Number of rows to return.
- **max_attempts** (*int*) – Maximum number of times url should be queried in the case of an exception being raised.
- **max_sleep** (*int*) – Maximum time, in seconds, to sleep between attempts. The true sleep time is a random int between *max_sleep* and *max_sleep* * 0.75.

Returns **resp** – JSON formatted response containing rows and all associated metadata.

Return type json

get_logical_file_chunk(*logical_file*, *start_row*, *n_rows*, *max_attempts*, *max_sleep*)

Return a chunk of a logical file from an HPCC instance.

Using the HPCC instance at *server*:*port* and the credentials *username* and *password*, return a chunk of *logical_file* which starts at row *start_row* and is *n_rows* long.

Parameters

- **logical_file** (*str*) – Name of logical file.
- **start_row** (*int*) – First row to return where 0 is the first row of the dataset.
- **n_rows** (*int*) – Number of rows to return.
- **max_attempts** (*int*) – Maximum number of times url should be queried in the case of an exception being raised.
- **max_sleep** (*int*) – Maximum time, in seconds, to sleep between attempts. The true sleep time is a random int between *max_sleep* and *max_sleep* * 0.75.

Returns **result_response** – Rows of logical file as list of dicts. In the form [{"col1": 1, "col2": 2}, {"col1": 1, "col2": 2}, ...].

Return type pd.DataFrame, str

run_ecl_script(*script*, *syntax_check*, *delete_workunit*, *stored*)

Run an ECL script and return the stdout and stderr.

Run the ECL script *script* on the HPCC instance at *server*:*port*, using the credentials *username* and *password*. If *syntax_check*, run a syntax check before execution. Attributes *legacy* and *repo* are also used.

Parameters

- **script** (*str*) – path to ECL script.
- **syntax_check** (*bool*) – If a syntax check should be ran before the script is executed.
- **delete_workunit** (*bool*) – Delete workunit once completed.
- **stored** (*dict or None*) – Key value pairs to replace stored variables within the script. Values should be str, int or bool.

Returns **result** – NamedTuple in the form (stdout, stderr).

Return type namedtuple

Raises subprocess.CalledProcessError: – If script fails syntax check.

See also:

`syntax_check()`, *run_ecl_string()*

run_ecl_string(*string*, *syntax_check*, *delete_workunit*, *stored*)

Run an ECL string and return the stdout and stderr.

Run the ECL string *string* on the HPCC instance at *server*:*port*, using the credentials *username* and *password*. If *syntax_check*, run a syntax check before execution. Attributes *legacy* and *repo* are also used.

Parameters

- **string** (*str*) – ECL script as a string.
- **syntax_check** (*bool*) – If a syntax check should be ran before the script is executed.
- **delete_workunit** (*bool*) – Delete workunit once completed.
- **stored** (*dict or None*) – Key value pairs to replace stored variables within the script. Values should be str, int or bool.

Returns **result** – NamedTuple in the form (stdout, stderr).

Return type namedtuple

Raises SyntaxError: – If script fails syntax check.

See also:

`syntax_check()`, *run_ecl_script()*

run_url_request(*url*, *max_attempts*, *max_sleep*)
Return the contents of a url.

Use attributes *username* and *password* to return the contents of *url*. Parameter *max_attempts* can be used to retry if an exception is raised. Each attempt is delayed by up to *max_sleep* seconds, so a large number of retries may be slow.

Parameters

- **url** (*str*) – URL to query.
- **max_attempts** (*int*) – Maximum number of times url should be queried in the case of an exception being raised.
- **max_sleep** (*int*) – Maximum time, in seconds, to sleep between attempts. The true sleep time is a random int between *max_sleep* and *max_sleep* * 0.75.

Returns **r** – Response object from *url*

Return type requests.models.Response

Raises requests.exceptions.RetryError: – If max_attempts is exceeded.

test_connection()
Assert that the *Connection* can connect to the HPCC instance.

This method attempts to connect to ECL Watch using its *server* and *port* attributes. The credentials provided are its *username* and *password*.

Returns

Return type True

Raises Exception: – If the connection fails, the relevant exception is raised.

2.4 hpycc.delete module

Functions to delete things in HPCC. The first input to all functions is an instance of *Connection*.

2.4.1 functions

- *delete_logical_file* – delete given logical file
- *delete_workunit* – delete given workunit (based on WUID)

hpycc.delete.**delete_logical_file**(*connection*, *logical_file*, *delete_workunit=True*)
Delete a logical file.

Parameters

- **connection** (*Connection*) – HPCC Connection instance, see also *Connection*.
- **logical_file** (*str*) – Logical file to be downloaded.
- **delete_workunit** (*bool, optional*) – Delete workunit once completed. True by default.

Returns

Return type None

hpycc.delete.**delete_workunit**(*connection*, *wuid*, *max_attempts=3*, *max_sleep=15*)
Delete a workunit

Parameters

- **connection** (*Connection*) – HPCC Connection instance, see also *Connection*.
- **wuid** (*string*) – Workunit ID
- **max_attempts** (*int, optional*) – Maximum number of times url should be queried in the case of an exception being raised. 3 by default.
- **max_sleep** (*int, optional*) – Maximum time, in seconds, to sleep between attempts. The true sleep time is a random int between *max_sleep* and *max_sleep* * 0.75. 5 by default.

Returns If the workunit is deleted successfully.

Return type True

Raises ValueError: – If the workunit could not be deleted.

2.5 hpycc.get module

Functions to get data out of a HPCC instance.

This module contains functions to get either the output(s) of an ECL script, or the contents of a logical file. The first input to all functions is an instance of *Connection*.

2.5.1 Functions

- *get_output* – Return the first output of an ECL script.
- *get_outputs* – Return all outputs of an ECL script.
- *get_thor_file* – Return the contents of a thor file.

hpycc.get.**get_output**(*connection*, *script*, *syntax_check=True*, *delete_workunit=True*, *stored=None*)
Return the first output of an ECL script as a pandas.DataFrame.

Note that whilst attempts are made to preserve the datatypes of the result, anything with an ambiguous type will revert to a string. If the output of the ECL string is an empty dataset (or if the script does not output anything), an empty pandas.DataFrame is returned.

Parameters

- **connection** (*hpycc.Connection*) – HPCC Connection instance, see also *Connection*.
- **script** (*str*) – Path of script to execute.
- **syntax_check** (*bool, optional*) – Should the script be syntax checked before execution? True by default.
- **delete_workunit** (*bool, optional*) – Delete workunit once completed. True by default.
- **stored** (*dict or None, optional*) – Key value pairs to replace stored variables within the script. Values should be str, int or bool. None by default.

Returns

Return type pandas.DataFrame of the first output of *script*.

Raises SyntaxError: – If script fails syntax check.

See also:

get_outputs(), save_output(), Connection.syntax_check()

Examples

```
>>> import hpycc
>>> conn = hpycc.Connection("user")
>>> with open("example.ecl", "r+") as file:
...     file.write("OUTPUT(2);")
>>> hpycc.get_output(conn, "example.ecl")
    Result_1
0          2
```

```
>>> import hpycc
>>> conn = hpycc.Connection("user")
>>> with open("example.ecl", "r+") as file:
>>>     file.write("OUTPUT(2);OUTPUT(3);")
>>> hpycc.get_output(conn, "example.ecl")
    Result_1
0          2
```

```
>>> import hpycc
>>> conn = hpycc.Connection("user")
>>> with open("example.ecl", "r+") as file:
...     file.write(
...     "a:= DATASET([{'1', 'a'}],"
...     "{STRING col1; STRING col2});",
...     "OUTPUT(a);")
>>> hpycc.get_output(conn, "example.ecl")
   col1 col2
0     1    a
```

```
>>> import hpycc
>>> conn = hpycc.Connection("user")
>>> with open("example.ecl", "r+") as file:
...     file.write(
...     "a:= DATASET([{'a', 'a'}],"
...     "{STRING col1;});",
...     "OUTPUT(a(col1 != a));")
>>> hpycc.get_output(conn, "example.ecl")
Empty DataFrame
Columns: []
Index: []
```

hpycc.get.**get_outputs**(*connection*, *script*, *syntax_check=True*, *delete_workunit=True*, *stored=None*)

Return all outputs of an ECL script.

Note that whilst attempts are made to preserve the datatypes of the result, anything with an ambiguous type will revert to a string.

Parameters

- **connection** (*hpycc.Connection*) – HPCC Connection instance, see also *Connection*.
- **script** (*str*) – Path of script to execute.
- **syntax_check** (*bool, optional*) – Should the script be syntax checked before execution? True by default.
- **delete_workunit** (*bool,*) – Delete the workunit once completed. True by default.
- **stored** (*dict or None, optional*) – Key value pairs to replace stored variables within the script. Values should be str, int or bool. None by default.

Returns **as_dict** – Outputs of *script* in the form {output_name: pandas.DataFrame}

Return type dict of pandas.DataFrames

Raises SyntaxError: – If script fails syntax check.

See also:

get_output(), save_outputs(), Connection.syntax_check()

Examples

```
>>> import hpycc
>>> conn = hpycc.Connection("user")
>>> with open("example.ecl", "r+") as file:
...     file.write("OUTPUT(2);")
>>> hpycc.get_outputs(conn, "example.ecl")
{Result_1:
    Result_1
0          2
}
```

```
>>> import hpycc
>>> conn = hpycc.Connection("user")
>>> with open("example.ecl", "r+") as file:
...     file.write(
...     "a:= DATASET([{'1', 'a'}],"
...     "{STRING col1; STRING col2});",
...     "OUTPUT(a);")
>>> hpycc.get_outputs(conn, "example.ecl")
{Result_1:
   col1 col2
0     1    a
}
```

```
>>> import hpycc
>>> conn = hpycc.Connection("user")
>>> with open("example.ecl", "r+") as file:
...     file.write(
...     "a:= DATASET([{'1', 'a'}],"
...     "{STRING col1; STRING col2});",
...     "OUTPUT(a);"
...     "OUTPUT(a);")
>>> hpycc.get_outputs(conn, "example.ecl")
{Result_1:
   col1 col2
```

(continues on next page)

(continued from previous page)

```
0     1    a,
Result_2:
   col1 col2
0     1    a
}
```

```
>>> import hpycc
>>> conn = hpycc.Connection("user")
>>> with open("example.ecl", "r+") as file:
...     file.write(
...     "a:= DATASET([{'1', 'a'}],"
...     "{STRING col1; STRING col2});",
...     "OUTPUT(a);"
...     "OUTPUT(a, NAMED('ds_2'));")
>>> hpycc.get_outputs(conn, "example.ecl")
{Result_1:
   col1 col2
0     1    a,
ds_2:
   col1 col2
0     1    a
}
```

hpycc.get.**get_thor_file**(*connection*, *thor_file*, *max_workers=10*, *chunk_size='auto'*, *max_attempts=3*, *max_sleep=60*, *dtype=None*)

Return a thor file as a pandas.DataFrame.

Note: Ordering of the resulting DataFrame is not deterministic and may not be the same as on the HPCC cluster.

Parameters

- **connection** (*hpycc.Connection*) – HPCC Connection instance, see also *Connection*.
- **thor_file** (*str*) – Name of thor file to be downloaded.
- **max_workers** (*int, optional*) – Number of concurrent threads to use when downloading file. Warning: too many may cause instability! 10 by default.
- **chunk_size** (*int, optional*) – Size of chunks to use when downloading file. If auto this is rows / workers (bounded between 100,000 and 400,000). If give then no limits are enforced.
- **max_attempts** (*int, optional*) – Maximum number of times a chunk should attempt to be downloaded in the case of an exception being raised. 3 by default.
- **max_sleep** (*int, optional*) – Minimum time, in seconds, to sleep between attempts. The true sleep time is a random int between *max_sleep* and *max_sleep* * 0.75.
- **dtype** (*type name or dict of col -> type, optional*) – Data type for data or columns. E.g. {'a': np.float64, 'b': np.int32}. If converters are specified, they will be applied INSTEAD of dtype conversion. If None, or columns are missing from the provided dict, they will be converted to one of bool, str or int based on the HPCC datatype. None by default.

Returns **df** – Thor file as a pandas.DataFrame.

Return type pandas.DataFrame

See also:

```
save_thor_file()
```

Examples

```
>>> import hpycc
>>> import pandas
>>> conn = hpycc.Connection("user")
>>> df = pandas.DataFrame({"col1": [1, 2, 3]})
>>> df.to_csv("example.csv", index=False)
>>> hpycc.spray_file(conn,"example.csv","example")
>>> hpycc.get_thor_file(conn, "example")
    col1
0      1
1      2
2      3
```

```
>>> import hpycc
>>> import pandas
>>> conn = hpycc.Connection("user")
>>> df = pandas.DataFrame({"col1": [1, 2, 3]})
>>> df.to_csv("example.csv", index=False)
>>> hpycc.spray_file(conn,"example.csv","example")
>>> hpycc.get_thor_file(conn, "example", dtype=str)
    col1
0     '1'
1     '2'
2     '3'
```

2.6 hpycc.run module

Function to run an ECL script

This module provides a function, *run_script*, to run an ECL script using an existing *Connection*. This can be used to run a script, saving a logical file which can then be accessing with *get_thor_file()*. The advantage of giving the download task to *get_thor_file()* is that it is able to multi-thread, something which functions in *get_output*, *get_outputs*, *save_output* and *save_outputs* cannot do.

2.6.1 Functions

- *run_script* – Run an ECL script.

hpycc.run.**run_script**(*connection*, *script*, *syntax_check=True*, *delete_workunit=True*, *stored=None*)
Run an ECL script.

This function runs an ECL script using a *Connection* object. It does not return the result.

Parameters

- **connection** (*hpycc.Connection*) – HPCC Connection instance, see also *Connection*.
- **script** (*str*) – Path of script to execute.
- **syntax_check** (*bool, optional*) – Should the script be syntax checked before execution? True by default.

- **delete_workunit** (*bool, optional*) – Delete workunit once completed. True by default.
- **stored** (*dict or None, optional*) – Key value pairs to replace stored variables within the script. Values should be str, int or bool. None by default.

Returns

Return type True

Raises SyntaxError: – If script fails syntax check.

2.7 hpycc.save module

TEMPORARILY DEPRICATED! Just use get and save teh result. Trust us, it's cleaner

Functions to get data out of an HPCC instance and save them to disk.

This modules functions closely mirror those in *get*. In fact all they really do is wrap *get*'s functions around csv writing tasks. The first input to all functions is an instance of *Connection*.

2.7.1 Functions

- *save_output* – Save the first output of an ECL script.
- *save_outputs* – Save all outputs of an ECL script.
- *save_thor_file* – Save the contents of a thor file.

hpycc.save.**save_output**(*connection*, *script*, *path_or_buf=None*, *syntax_check=True*, *delete_workunit=True*, *stored=None*, ***kwargs*)

Save the first output of an ECL script as a csv. See save_outputs() for saving multiple outputs to file and get_output() for returning as a DataFrame.

Parameters

- **connection** (*Connection*) – HPCC Connection instance, see also *Connection*.
- **script** (*str*) – Path of script to execute.
- **path_or_buf** (*string or file handle, default None*) – File path or object, if None is provided the result is returned as a string.
- **syntax_check** (*bool, optional*) – Should script be syntax checked before execution. True by default.
- **delete_workunit** (*bool, optional*) – Delete workunit once completed. True by default.
- **stored** (*dict or None, optional*) – Key value pairs to replace stored variables within the script. Values should be str, int or bool. None by default.
- **kwargs** – Additional parameters to be provided to pandas.DataFrame.to_csv().

Returns if path_or_buf is not None, else a string representation of the output csv.

Return type None or str

hpycc.save.**save_thor_file**(*connection*, *thor_file*, *path_or_buf=None*, *max_workers=15*, *chunk_size='auto'*, *max_attempts=3*, *max_sleep=60*, *dtype=None*, ***kwargs*)

Save a logical file to disk, see *get_thor_file()* for returning a DataFrame.

Parameters

- **connection** (*Connection*) – HPCC Connection instance, see also *Connection*.
- **thor_file** (*str*) – Logical file to be downloaded
- **path_or_buf** (*string or file handle, default None*) – File path or object, if None is provided the result is returned as a string.
- **max_workers** (*int, optional*) – Number of concurrent threads to use when downloading. Warning: too many will likely cause either your machine or your cluster to crash! 15 by default.
- **chunk_size** (*int, optional.*) – Size of chunks to use when downloading file. 10000 by default.
- **max_attempts** (*int, optional*) – Maximum number of times a chunk should attempt to be downloaded in the case of an exception being raised. 3 by default.
- **max_sleep** (*int, optional*) – Maximum time, in seconds, to sleep between attempts. The true sleep time is a random int between *max_sleep* and *max_sleep* * 0.75.
- **dtype** (*type name or dict of col -> type, optional*) – Data type for data or columns. E.g. {'a': np.float64, 'b': np.int32}. If converters are specified, they will be applied INSTEAD of dtype conversion. If None, or columns are missing from the provided dict, they will be converted to one of bool, str or int based on the HPCC datatype. None by default.
- **kwargs** – Additional parameters to be provided to pandas.DataFrame.to_csv().

Returns if path_or_buf is not None, else a string representation of the output csv.

Return type None or str

2.8 hpycc.spray module

The module contains functions to send files to HPCC.

2.8.1 Functions

- *spray_file* – Spray a given csv or pandas DataFrame to HPCC.

hpycc.spray.**spray_file**(*connection*, *source_file*, *logical_file*, *overwrite=False*, *expire=None*, *chunk_size=100000*, *max_workers=5*, *delete_workunit=True*)

Spray a file to a HPCC logical file, bypassing the landing zone.

Parameters

- **connection** (*Connection*) – HPCC Connection instance, see also *Connection*.
- **source_file** (*str, pd.DataFrame*) – A pandas DataFrame or the path to a csv.
- **logical_file** (*str*) – Logical file name on THOR.
- **overwrite** (*bool, optional*) – Should the file overwrite any pre-existing logical file. False by default.
- **chunk_size** (*int, optional*) – Size of chunks to use when spraying file. 100000 by default.

- **max_workers** (*int, optional*) – Number of concurrent threads to use when spraying. Warning: too many will likely cause either your machine or your cluster to crash! 3 by default.
- **expire** (*int*) – How long (days) until the produced logical file expires? None (ie no expiry) by default
- **delete_workunit** (*bool*) – Delete workunit once completed.

Returns

Return type None

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

h

hpycc.connection, 7
hpycc.delete, 9
hpycc.get, 10
hpycc.run, 14
hpycc.save, 15
hpycc.spray, 16
hpycc.utils, 7
hpycc.utils.docker_tools, 5
hpycc.utils.filechunker, 5
hpycc.utils.parsers, 6

Index

A

apply_custom_dtypes() (*in module hpycc.utils.parsers*), 6

C

check_syntax() (*hpycc.connection.Connection method*), 7
Connection (*class in hpycc.connection*), 7
create_passwords() (*hpycc.utils.docker_tools.HPCCContainer method*), 5

D

delete_logical_file() (*in module hpycc.delete*), 9
delete_workunit() (*in module hpycc.delete*), 10

G

get_chunk_from_hpcc() (*hpycc.connection.Connection method*), 7
get_logical_file_chunk() (*hpycc.connection.Connection method*), 7
get_output() (*in module hpycc.get*), 10
get_outputs() (*in module hpycc.get*), 11
get_python_type_from_ecl_type() (*in module hpycc.utils.parsers*), 6
get_thor_file() (*in module hpycc.get*), 13

H

HPCCContainer (*class in hpycc.utils.docker_tools*), 5
hpycc.connection (*module*), 7
hpycc.delete (*module*), 9
hpycc.get (*module*), 10
hpycc.run (*module*), 14
hpycc.save (*module*), 15
hpycc.spray (*module*), 16
hpycc.utils (*module*), 7
hpycc.utils.docker_tools (*module*), 5
hpycc.utils.filechunker (*module*), 5
hpycc.utils.parsers (*module*), 6

M

make_chunks() (*in module hpycc.utils.filechunker*), 6

P

parse_schema_from_xml() (*in module hpycc.utils.parsers*), 6
parse_wuid_from_failed_response() (*in module hpycc.utils.parsers*), 6
parse_wuid_from_xml() (*in module hpycc.utils.parsers*), 6
parse_xml() (*in module hpycc.utils.parsers*), 6
pull_image() (*hpycc.utils.docker_tools.HPCCContainer method*), 5
put_archive() (*hpycc.utils.docker_tools.HPCCContainer method*), 5

R

run_ecl_script() (*hpycc.connection.Connection method*), 8
run_ecl_string() (*hpycc.connection.Connection method*), 8
run_script() (*in module hpycc.run*), 14
run_url_request() (*hpycc.connection.Connection method*), 9

S

save_output() (*in module hpycc.save*), 15
save_thor_file() (*in module hpycc.save*), 15
setup_hpcc() (*hpycc.utils.docker_tools.HPCCContainer method*), 5
spray_file() (*in module hpycc.spray*), 16
start_container() (*hpycc.utils.docker_tools.HPCCContainer method*), 5
start_hpcc() (*hpycc.utils.docker_tools.HPCCContainer method*), 5

stop_container() (*hpycc.utils.docker_tools.HPCCContainer method*), 5

T

test_connection() (*hpycc.connection.Connection method*), 9