computer Documentation

Release

Author

September 16, 2016

Contents

CLI-Computer is a library created by the Climate Impact Lab team.

This code is open source and available on github.

We can add any additional information about the library here. Please make suggestions.

CHAPTER 1

Intro and Setup

This is where the introduction to our libraries will go. Ideally, this will be a short overview of the library, with one or two very simple use cases. It will, hopefully, answer the question: 'Why should I use this tool?'

1.1 Description

1.1.1 Paths

Servers have collections of *roots*, used to abstract away the location of files. Most methods take a *path* argument, which can take three forms:

- *Starting with a /*: An absolute path, not relative to *root*.
- *Containing a /*: The *root* is up to the first slash, and the remaining is a subpath.
- *Not containing a /*: Just a *root* directory.

1.1.2 Available Servers

Abstract Servers

- *DiskServer*: A server with a unique ID and a collection of named file root directories.
- *SizelessServer* (*DiskServer*): The main top-level server class, able to transfer data and run commands. Does not monitor or limit data usage (hence, "sizeless").
- *SizelessConnectableServer* (*SizelessServer*): A server that has a protocol for connecting to it.
- *SizelessLinuxServer* (*SizelessConnectableServer*): A standard Linux server, with everything except *run_command* implemented.
- *ParamikoServer* (*SizelessLinuxServer*): A server connected through ssh, using the Paramiko interface. The *self.session* variable needs to be set in the *connect* method before these functions will work.

Concrete Servers

- *LocalServer* (*SizelessServer*): A server representing the local machine, using python commands.
- *LoginServer* (*ParamikoServer*): A server using a Paramiko ssh connection, with a normal password login.

1.2 Installation Guide

```
git clone https://github.com/ClimateImpactLab/computer
```

CHAPTER 2

The API Documentation / Guide

If you are looking for information on a specific function, class, or method, this part of the documentation is for you.

2.1 Computer library

2.1.1 computer.linux_server module

class computer.linux_server.**RemoteLinuxProcess**(*server*, *pid*, *logfile*)
Bases: *computer.process.RemoteProcess*

clean()
Remove the log file

halt()
Temporarily stop a process.

is_running()
Returns bool.

kill()
Kill process if running

resume()
Resume a halted process.

class computer.linux_server.**SizelessLinuxServer**(*utup*, *cpus*, *roots*, *credentials*)
Bases: *computer.server.SizelessConnectableServer*

active_processes(*procname*)

check_connection()

cwd(*path*)

list_disk(*path*)
path: subdirectory

read_file(*filepath=None*)
Returns string.

start_process(*command*, *path=None*)

2.1.2 computer.local_server module

class computer.local_server.**LocalProcess**(*server*, *proc*, *stdout*, *logfile*)
Bases: *computer.process.SingleCPUProcess*

clean()

halt()
Temporarily stop a process.

is_running()

kill()

resume()
Resume a halted process.

class computer.local_server.**LocalServer**(*utup*, *cpus*, *roots*)
Bases: *computer.server.SizelessServer*

has_file(*path*)

list_disk(*path=None*)

read_file(*path*)

run_command(*command*, *path=None*)

start_process(*command*, *logfile*, *path=None*)

2.1.3 computer.login_server module

class computer.login_server.**LoginServer**(*utup*, *cpus*, *roots*, *credentials*)
Bases: *computer.paramiko_server.ParamikoServer*

connect()

get_cpu_util()
implement the idea of getting cpu_percents in psutil: 1. Get "cpu times" from /proc/stat once and wait a second before getting another reading of those. 2. Compare the two readings and define the utilization rate as:

(Non-idle and Non-iowait CPU time) / (total CPU time)

http://stackoverflow.com/questions/23367857/accurate-calculation-of-cpu-usage-given-in-percentage-in-linux

2.1.4 computer.manager module

class computer.manager.**DiskDependency**(*root*, *path*, *size*)
Bases: object

satisfy(*server*)

satisfy_cost(*server*)

class computer.manager.**Manager**
Bases: object

add_server(*server*, *priority*)

prepare_server(*dependencies*, *required=False*)
Finds a server with resources and minimal transfer requirements; transfers additional requirements as needed. Returns server.

process_check()

run_command(*root*, *path*, *command*, *dependencies*)

select_server(*dependencies*, *required*)

server_check()

submit(*root*, *path*, *command*, *dependencies*, *callback*)
Does not return process (so we can manage). callback only called on check().

2.1.5 computer.paramiko_server module

class computer.paramiko_server.**ParamikoServer**(*utup*, *cpus*, *roots*, *credentials*)
Bases: *computer.linux_server.SizelessLinuxServer*

disconnect()

receive()

receive_all()

receive_each()

run_command(*command*, *root=None*, *path=None*)
Returns (output, error) as strings.

run_command_each(*command*, *root=None*, *path=None*)

2.1.6 computer.process module

class computer.process.**RemoteProcess**(*server*, *pid*, *logfile*)
Bases: *computer.process.SingleCPUProcess*

class computer.process.**SingleCPUProcess**(*server*, *logfile=None*)
Bases: object

clean()
Remove the log file

close()

is_running()
Returns bool.

kill()
Kill process if running

2.1.7 computer.server module

class computer.server.**DiskServer**(*utup*, *roots*)
Bases: object

fullpath(*path*)

splitpath(*path*)

class computer.server.**SizelessConnectableServer**(*utup*, *cpus*, *roots*, *credentials*)
Bases: *computer.server.SizelessServer*

check_connection()

connect()

disconnect()

class computer.server.**SizelessServer**(*utup*, *cpus*, *roots*)
Bases: *computer.server.DiskServer*

has_file(*filepath*)
path: subdirectory

list_disk(*path*)
path: subdirectory

pull_disk(*path*, *localds*)
path: subdirectory local: local DiskServer

push_disk(*path*, *localds*)
path: subdirectory local: local DiskServer

read_file(*filepath*)
Returns string.

run_command(*command*, *path=None*)
Returns (output, error) as strings.

start_process(*command*, *path=None*)
Returns process.

2.1.8 computer.task module

class computer.task.**LambdaDependency**(*desc*, *check*, *install_func=None*, *atrun_func=None*)
Bases: *computer.task.OnceDependency*

atrun(*server*)

install(*server*)

raiseunimplemented()

satisfied(*server*)

class computer.task.**LambdaTask**(*dependencies*, *perform*)
Bases: *computer.task.SingleCPUTask*

selfrun(*server*)

class computer.task.**OnceDependency**(*desc*)
Bases: object

satisfied(*server*)

class computer.task.**SingleCPUTask**(*dependencies*)
Bases: object

fullrun(*server*, *doublecheck=True*)

prepare(*server*)

selfrun(*server*)

CHAPTER 3

The Developers Guide

If you want to contribute to the project, this part of the documentation is for you.

3.1 Contributor's Guide

This document lays out guidelines and advice for contributing to this project. If you're thinking of contributing, please start by reading this document and getting a feel for how contributing to this project works. If you have any questions, feel free to reach out to either James Rising, Mike Delgado, or Justin Simcock.

When contributing code, you'll want to follow this checklist:

1. Fork the repository on GitHub.
2. Run the tests to confirm they all pass on your system. If they don't, you'll need to investigate why they fail. If you're unable to diagnose this yourself, raise it as a bug report by following the guidelines in this document: *Bug Reports*.
3. Write tests that demonstrate your bug or feature. Ensure that they fail.
4. Make your change.
5. Run the entire test suite again, confirming that all tests pass *including the ones you just added*.
6. Send a GitHub Pull Request to the main repository's `master` branch. GitHub Pull Requests are the expected method of code collaboration on this project.

3.1.1 Documentation Links

The documentation files live in the `docs/` directory of the codebase. They're written in reStructuredText, and use Sphinx to generate the full suite of documentation.

When writing documentation, please do your best to follow the style of the documentation files.

Here is an reference example Python Module with resStructuredText in the docstring.

```
# -*- coding: utf-8 -*-
"""Example NumPy style docstrings.

This module demonstrates documentation as specified by the `NumPy
Documentation HOWTO`_. Docstrings may extend over multiple lines. Sections
are created with a section header followed by an underline of equal length.

Example
```

```
-------
Examples can be given using either the ``Example`` or ``Examples``
sections. Sections support any reStructuredText formatting, including
literal blocks::

    $ python example_numpy.py


Section breaks are created with two blank lines. Section breaks are also
implicitly created anytime a new section starts. Section bodies *may* be
indented:

Notes
-----
    This is an example of an indented section. It's like any other section,
    but the body is indented to help it stand out from surrounding text.

If a section is indented, then a section break is created by
resuming unindented text.

Attributes
----------
module_level_variable1 : int
    Module level variables may be documented in either the ``Attributes``
    section of the module docstring, or in an inline docstring immediately
    following the variable.

    Either form is acceptable, but the two should not be mixed. Choose
    one convention to document module level variables and be consistent
    with it.


.. _NumPy Documentation HOWTO:
   https://github.com/numpy/numpy/blob/master/doc/HOWTO_DOCUMENT.rst.txt

"""

module_level_variable1 = 12345

module_level_variable2 = 98765
"""int: Module level variable documented inline.

The docstring may span multiple lines. The type may optionally be specified
on the first line, separated by a colon.
"""


def function_with_types_in_docstring(param1, param2):
    """Example function with types documented in the docstring.

    `PEP 484`_ type annotations are supported. If attribute, parameter, and
    return types are annotated according to `PEP 484`_, they do not need to be
    included in the docstring:

    Parameters
    ----------
    param1 : int
        The first parameter.
```

```
    param2 : str
        The second parameter.

    Returns
    -------
    bool
        True if successful, False otherwise.

    .. _PEP 484:
        https://www.python.org/dev/peps/pep-0484/

    """


def function_with_pep484_type_annotations(param1: int, param2: str) -> bool:
    """Example function with PEP 484 type annotations.

    The return type must be duplicated in the docstring to comply
    with the NumPy docstring style.

    Parameters
    ----------
    param1
        The first parameter.
    param2
        The second parameter.

    Returns
    -------
    bool
        True if successful, False otherwise.

    """


def module_level_function(param1, param2=None, *args, **kwargs):
    """This is an example of a module level function.

    Function parameters should be documented in the ``Parameters`` section.
    The name of each parameter is required. The type and description of each
    parameter is optional, but should be included if not obvious.

    If \*args or \*\*kwargs are accepted,
    they should be listed as ``*args`` and ``**kwargs``.

    The format for a parameter is::

        name : type
            description

            The description may span multiple lines. Following lines
            should be indented to match the first line of the description.
            The ": type" is optional.

            Multiple paragraphs are supported in parameter
            descriptions.

    Parameters
```

```
    ----------
    param1 : int
        The first parameter.
    param2 : :obj:`str`, optional
        The second parameter.
    *args
        Variable length argument list.
    **kwargs
        Arbitrary keyword arguments.

    Returns
    -------
    bool
        True if successful, False otherwise.

        The return type is not optional. The ``Returns`` section may span
        multiple lines and paragraphs. Following lines should be indented to
        match the first line of the description.

        The ``Returns`` section supports any reStructuredText formatting,
        including literal blocks::

            {
                'param1': param1,
                'param2': param2
            }

    Raises
    ------
    AttributeError
        The ``Raises`` section is a list of all exceptions
        that are relevant to the interface.
    ValueError
        If `param2` is equal to `param1`.

    """
    if param1 == param2:
        raise ValueError('param1 may not be equal to param2')
    return True


def example_generator(n):
    """Generators have a ``Yields`` section instead of a ``Returns`` section.

    Parameters
    ----------
    n : int
        The upper limit of the range to generate, from 0 to `n` - 1.

    Yields
    ------
    int
        The next number in the range of 0 to `n` - 1.

    Examples
    --------
    Examples should be written in doctest format, and should illustrate how
    to use the function.
```

```
    >>> print([i for i in example_generator(4)])
    [0, 1, 2, 3]

    """
    for i in range(n):
        yield i


class ExampleError(Exception):
    """Exceptions are documented in the same way as classes.

    The __init__ method may be documented in either the class level
    docstring, or as a docstring on the __init__ method itself.

    Either form is acceptable, but the two should not be mixed. Choose one
    convention to document the __init__ method and be consistent with it.

    Note
    ----
    Do not include the `self` parameter in the ``Parameters`` section.

    Parameters
    ----------
    msg : str
        Human readable string describing the exception.
    code : :obj:`int`, optional
        Numeric error code.

    Attributes
    ----------
    msg : str
        Human readable string describing the exception.
    code : int
        Numeric error code.

    """

    def __init__(self, msg, code):
        self.msg = msg
        self.code = code


class ExampleClass(object):
    """The summary line for a class docstring should fit on one line.

    If the class has public attributes, they may be documented here
    in an ``Attributes`` section and follow the same formatting as a
    function's ``Args`` section. Alternatively, attributes may be documented
    inline with the attribute's declaration (see __init__ method below).

    Properties created with the ``@property`` decorator should be documented
    in the property's getter method.

    Attributes
    ----------
    attr1 : str
        Description of `attr1`.
    attr2 : :obj:`int`, optional
```

```
        Description of `attr2`.

    """

    def __init__(self, param1, param2, param3):
        """Example of docstring on the __init__ method.

        The __init__ method may be documented in either the class level
        docstring, or as a docstring on the __init__ method itself.

        Either form is acceptable, but the two should not be mixed. Choose one
        convention to document the __init__ method and be consistent with it.

        Note
        ----
        Do not include the `self` parameter in the ``Parameters`` section.

        Parameters
        ----------
        param1 : str
            Description of `param1`.
        param2 : :obj:`list` of :obj:`str`
            Description of `param2`. Multiple
            lines are supported.
        param3 : :obj:`int`, optional
            Description of `param3`.

        """
        self.attr1 = param1
        self.attr2 = param2
        self.attr3 = param3  #: Doc comment *inline* with attribute

        #: list of str: Doc comment *before* attribute, with type specified
        self.attr4 = ["attr4"]

        self.attr5 = None
        """str: Docstring *after* attribute, with type specified."""

    @property
    def readonly_property(self):
        """str: Properties should be documented in their getter method."""
        return "readonly_property"

    @property
    def readwrite_property(self):
        """:obj:`list` of :obj:`str`: Properties with both a getter and setter
        should only be documented in their getter method.

        If the setter method contains notable behavior, it should be
        mentioned here.
        """
        return ["readwrite_property"]

    @readwrite_property.setter
    def readwrite_property(self, value):
        value

    def example_method(self, param1, param2):
```

```
        """Class methods are similar to regular functions.

        Note
        ----
        Do not include the `self` parameter in the ``Parameters`` section.

        Parameters
        ----------
        param1
            The first parameter.
        param2
            The second parameter.

        Returns
        -------
        bool
            True if successful, False otherwise.

        """
        return True

    def __special__(self):
        """By default special members with docstrings are not included.

        Special members are any methods or attributes that start with and
        end with a double underscore. Any special member with a docstring
        will be included in the output, if
        ``napoleon_include_special_with_doc`` is set to True.

        This behavior can be enabled by changing the following setting in
        Sphinx's conf.py::

            napoleon_include_special_with_doc = True

        """
        pass

    def __special_without_docstring__(self):
        pass

    def _private(self):
        """By default private members are not included.

        Private members are any methods or attributes that start with an
        underscore and are *not* special. By default they are not included
        in the output.

        This behavior can be changed such that private members *are* included
        by changing the following setting in Sphinx's conf.py::

            napoleon_include_private_with_doc = True

        """
        pass

    def _private_without_docstring(self):
        pass
```

3.1.2 Bug Reports

Bug reports are hugely important! Before you raise one, though, please check through the GitHub issues, **both open and closed**, to confirm that the bug hasn't been reported before. Duplicate bug reports are a huge drain on the time of other contributors, and should be avoided as much as possible.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

C

A

C

D

F

G

H

I

K

L

M

O

P

R

S