SHTK

Release v0.9.2

Jon Roose

Feb 14, 2022

CONTENTS

CHAPTER

ONE

SHELLS

Shells are used to abstract away the inner workings of shtk, and are meant to be the primary interface end users interact with. shtk.Shell instances serve as the standard mechanism for building and running command pipelines.

1.1 shtk.Shell

class shtk.**Shell**(*cwd=None*, *env=None*, *umask=None*, *stdin=None*, *stdout=None*, *stderr=None*, *exceptions=True*, *user=None*, *group=None*)

Bases: `object`

A shell object tracks pre-requisite information (e.g. cwd and environment variables) necessary to run commands as subprocesses. A shell is also a context manager that exposes environment variables and other info to subshells and subprocesses, while also setting itself as the default shell within managed code.

Parameters

- **cwd** (*str, pathlib.Path*) – Current working directory for subprocesses.
- **env** (*bool*) – If provided Shell will use these key-value pairs as environment variables. Otherwise Shell inherits from the currently active Shell, or _default_shell.
- **umask** (*int*) – Controls the default umask for subprocesses
- **stdin** (*file-like*) – Default stdin stream for subprocesses (defaults to sys.stdin)
- **stdout** (*file-like*) – Default stdout stream for subprocesses (defaults to sys.stdout)
- **stderr** (*file-like*) – Default stderr stream for subprocesses (defaults to sys.stderr)
- **exceptions** (*bool*) – Whether exceptions should be raised when non-zero exit codes are returned by subprocesses.
- **user** (*None, int, str*) – Run subprocesses as the given user. If None, run as same user as this process. If int, run as user with uid=user. If str, run as user with name=user. Requires Python >= 3.9.
- **group** (*None, int, str*) – Run subprocesses as the given group. If None, run as same group as this process. If int, run as group with gid=group. If str, run as group with name=group. Requires Python >= 3.9.

command(*name*)

Creates a PipelineProcessFactory suitable for executing a command

Parameters **name** (*str or pathlib.Path*) – Name or path of the command to run. If an absolute or relative path is provided (must contain a '/' character) then the command will be

loaded from the specified location. Otherwise the locations specified by the $PATH environment variable will be checked for suitable executable and readable files with the appropriate name.

If name is an str, then the name will be passed through os.path.expanduser prior to lookup.

Returns A PipelineProcessFactory node representing the command to be executed.

Return type *PipelineProcessFactory*

Raises

- **RuntimeError** – command cannot be found
- **RuntimeError** – command filepath is not readable
- **RuntimeError** – command filepath is not executable

cd(*path*)

Changes the default current working directory for subprocesses built by the Shell

Parameters **path** (*str or pathlib.Path*) – Changes directory to provided path such that managed subprocesses will use this directory as their default current working directory. If '-' is provided, returns to previous working directory.

Raises **RuntimeError** – raised if path is not a directory

cd_manager(*new_wd*)

Contextmanager for Shell.cd() returns to previous dir after exit

Parameters **new_wd** (*str or pathlib.Path*) – directory to change to

Yields *pathlib.Path* – The new self.cwd

export(***env*)

Sets environment variables passed as keyword arguments

Parameters ****env** (*dict*) – List of key-value pairs that will set as environment variables for the Shell()

getenv(*name*)

Gets the value of an environment variable within the Shell

Parameters **name** (*str*) – Name of the environment variable to evaluate

Returns The value of the named environment variable

Return type str

classmethod get_shell()

Gets the current active shell from the shell stack

Returns The most recently entered shell context

Return type *Shell*

run(**pipeline_factories*, *exceptions=None*, *wait=True*, *close_fds=True*)

Executes a series of PipelineNodeFactory nodes as subprocesses

Parameters

- ***pipeline_factories** – The PipelineNodeFactory nodes to execute. If multiple arguments are provided, then the commands will run in parallel.
- **exceptions** – Whether or not to raise exceptions for non-zero exit codes (Default value = None, meaning inherited)

- **wait** – Whether the call should block waiting for the subprocesses to exit (Default value = True)
- **close_fds** (*bool*) – If true, close_fds will be passed to the equivalent of subprocess.Popen() (Default value = True).

Returns Job instances representing individual pipelines. The length of the list will always be equal to the len(pipeline_factories)

Return type list of Job

evaluate(*pipeline_factory*, *, *exceptions=None*)

Executes a PipelineNodeFactory and returns the stdout text

Parameters

- **pipeline_factory** (PipelineNodeFactory) – the pipeline to instantiate and execute
- **exceptions** – Whether or not to raise exceptions when subprocesses return non-zero return codes (Default value = None)

Returns A string generated by the text that the final subprocess writes to stdout

Return type str or bytes

source(*filepath*, *, *exceptions=None*)

Use /bin/sh to source a file, then import the resulting environment as the shtk.Shell's new environment.

This method uses (approximately) the following kludge:

```
q = shlex.quote
exec = str(sys.exec or 'python3')
kludge = '''
import json, os, sys;
fout = os.fdopen(int(sys.argv[1]), "w");
json.dump(dict(os.environ), fout);
fout.close();
'''
args = (".", q(path), '&&', q(exec), '-c', q(kludge), str(int(pipe_fd)))
pipeline_factory = shcmd('-c', " ".join(args))
...
```

Parameters

- **filepath** (*str or pathlib.Path*) – path to file to be sourced. It will be converted to an absolute path before sourcing for compatability with picky shells (e.g. dash).
- **exceptions** (*bool*) – Whether or not to raise exceptions when subprocesses return non-zero return codes (Default value = None)

Returns A string generated by the text that the final subprocess writes to stdout

Return type str or bytes

CHAPTER

TWO

JOBS

Jobs represent and control running pipelines. They are returned by shtk.Shell.run() and used by other internal methods of shtk.Shell, such as shtk.Shell.evaluate(). Provided functionality includes starting, killing, and awaiting completion of process pipelines.

The Job.run() method is also responsible for instantiating shtk.PipelineNodeFactory templates to create shtk.PipelineNode instances, as well as instantiating StreamFactory templates to create shtk.Stream instances.

2.1 shtk.Job

class shtk.**Job**(*pipeline_factory*, *cwd=None*, *env=None*, *event_loop=None*, *user=None*, *group=None*, *close_fds=True*)

Bases: object

Instantiates PipelienNodeFactory instances to run subprocesses.

Job objects instantiate a PipelineNodeFactory template to create PipelineNode's that run and track the progress of a command pipeline.

Parameters

- **pipeline_factory** (PipelineNodeFactory) – The command pipeline template that will be instantiated by the Job instance.
- **cwd** (*str or Pathlib.Path*) – The current working directory in which to run the pipeline processes.
- **env** (*dict*) – The environment variables to pass to processes run within the pipleine
- **event_loop** (*None or asyncio.AbstractEventLoop*) – The event loop to use for asyncio based processing. If None is passed a new event loop is created from asyncio.new_event_loop() instead.
- **user** (*None, int, or str*) – The user that will be used to setreuid() any child processes. The behavior is the same as that of the user arg to subprocess.Popen().
- **group** (*None, int, or str*) – The user that will be used to setregid() any child processes. The behavior is the same as that of the group arg to subprocess.Popen().
- **close_fds** (*bool*) – If true, close_fds will be passed to the equivalent of subprocess.Popen() (Default value = True).

cwd

The current working directory in which to run the pipeline processes.

Type str or Pathlib.Path

environment
The environment variables to pass to processes run within the pipleine

Type dict

event_loop
The event loop to use for asyncio based processing. If None is passed a new event loop is created from asyncio.new_event_loop() instead.

Type None or asyncio.AbstractEventLoop

pipeline
The running PipelineNode generated by PipelineFactory.build in run().

Type None, *PipelineNode*

pipeline_factory
The command pipeline template that will be instantiated by the Job instance.

Type *PipelineNodeFactory*

user
The user that will be used to setreuid() any child processes. The behavior is the same as that of the user arg to subprocess.Popen().

Type None, int, or str

group
The user that will be used to setregid() any child processes. The behavior is the same as that of the group arg to subprocess.Popen().

Type None, int, or str

close_fds
If true, close_fds will be passed to the equivalent of subprocess.Popen().

Type bool

property stdin
Returns the pipeline's stdin_stream.writer(), or None

Returns Pipeline's stdin_stream.writer(), or None if no pipeline is running.

property stdout
Returns the pipeline's stdout_stream.reader(), or None

Returns Pipeline's stdout_stream.reader(), or None if no pipeline is running.

property stderr
Returns the pipeline's stderr_stream.reader(), or None

Returns Pipeline's stderr_stream.reader(), or None if no pipeline is running.

async run_async(*stdin_factory*, *stdout_factory*, *stderr_factory*)
Creates and runs a new pipeline

Instantiates and runs a pipeline based on the PipelineNodeFactory provided to the Job's constructor.

Parameters

- **stdin_factory** (StreamFactory) – the StreamFactory to instantiate to create the job's default stdin stream.
- **stdout_factory** (StreamFactory) – the StreamFactory to instantiate to create the job's default stdout stream.

- **stderr_factory** (StreamFactory) – the StreamFactory to instantiate to create the job's default stderr stream.

run(*stdin_factory*, *stdout_factory*, *stderr_factory*)
Creates and runs a new pipeline

Synchronously wrapper for run_async.

Parameters

- **stdin_factory** (StreamFactory) – the StreamFactory to instantiate to create the job's default stdin stream.
- **stdout_factory** (StreamFactory) – the StreamFactory to instantiate to create the job's default stdout stream.
- **stderr_factory** (StreamFactory) – the StreamFactory to instantiate to create the job's default stderr stream.

async wait_async(*pipeline_node=None*, *exceptions=True*)
Waits for all processes in the pipeline to complete

Waits for all processes in the pipleine to complete checks the return codes of each command.

Parameters

- **pipeline_node** (PipelineNode *or None*) – The pipeline node to wait for
- **exceptions** (*Boolean*) – When true returns an exception when processes exit with non-zero return codes

Returns A tuple of exit codes from the completed proceses

Raises

- ***NonzeroExitCodeException*** – When a process returns a non-zero return code
- **RuntimeError** – When called on a Job that has not invoked Job.run()

wait(*pipeline_node=None*, *exceptions=True*)
Synchronous wrapper for the wait_async() method.

Waits for all processes in the pipleine to complete checks the return codes of each command.

Parameters

- **pipeline_node** (PipelineNode *or None*) – The pipeline node to wait for
- **exceptions** (*Boolean*) – When true returns an exception when processes exit with non-zero return codes

Returns A tuple of exit codes from the completed proceses

Raises ***NonzeroExitCodeException*** – When a process returns a non-zero return code

send_signal(*signum*)
Sends a signal to all child ProcessNode processes.

Parameters **signum** (*int*) – the signal to send.

terminate()
Sends a signal.SIGTERM to all child ProcessNode processes.

kill()
Sends a signal.SIGKILL to all child ProcessNode processes.

CHAPTER

THREE

STREAM FACTORIES

shtk.StreamFactory subclasses are templates used to define the properties of shtk.Stream instances. shtk.Stream instances are pairs of file-like objects (one for reading data from the process, one for writing data to the process) used for communication with running processes.

shtk.StreamFactory instances are typically constructed via calls to shtk.PipelineNodeFactory.stdin(), shtk.PipelineNodeFactory.stdout(), and shtk.PipelineNodeFactory.stderr().

3.1 shtk.StreamFactory

class shtk.**StreamFactory**

Bases: abc.ABC

Base class for templates creating associated Stream instances

abstract build(*job*)

Instantiates the Stream instance

Parameters **job** (Job) – job to use for current working directory and environment variables.

Returns The constructed Stream instance.

Return type *Stream*

3.2 shtk.FileStreamFactory

class shtk.**FileStreamFactory**(*partial_path*, *mode*)

Bases: *shtk.StreamFactory.StreamFactory*

Creates a template for FileStream instances

Parameters

- **partial_path** (*str or pathlib.Path*) – The absolute or job.cwd relative path to the file.
- **mode** (*str*) – The mode to pass to open() when instantiating the FileStream.

partial_path

The absolute or job.cwd relative path to the file.

Type pathlib.Path

mode

The mode to pass to open() when instantiating the FileStream.

Type str

build(*job*)

Instantiates the FileStream instance

Parameters **job** (Job) – job to use for current working directory and environment variables.

Returns The constructed FileStream instance.

Return type *FileStream*

3.3 shtk.ManualStreamFactory

class shtk.**ManualStreamFactory**(*fileobj_r=None*, *fileobj_w=None*)

Bases: *shtk.StreamFactory.StreamFactory*

Creates a template for ManualStream instances

Parameters

- **fileobj_r** (*file-like or None*) – A file-like object suitable for reading. None implies os.devnull should be used (Default value = None).
- **fileobj_w** (*file-like or None*) – A file-like object suitable for writing. None implies os.devnull should be used (Default value = None).

fileobj_r

A file-like object suitable for reading. None implies os.devnull should be used.

Type file-like or None

fileobj_w

A file-like object suitable for writing. None implies os.devnull should be used.

Type file-like or None

build(*job*)

Instantiates the ManualStream instance

Parameters **job** (Job) – job to use for current working directory and environment variables.

Returns The constructed ManualStream instance.

Return type *ManualStream*

3.4 shtk.NullStreamFactory

class shtk.**NullStreamFactory**

Bases: *shtk.StreamFactory.StreamFactory*

Creates a template for NullStream instances

build(*job*)

Instantiates the NullStream instance

Parameters **job** (Job) – job to use for current working directory and environment variables.

Returns The constructed NullStream instance.

Return type *NullStream*

3.5 shtk.PipeStreamFactory

class shtk.**PipeStreamFactory**(*flags=0*)

Bases: *shtk.StreamFactory.StreamFactory*

Creates a template for PipeStream instances

Parameters **flags** (*int*) – flags to pass to PipeStream constructor (Default value = 0).

flags

flags to pass to PipeStream constructor

Type int

build(*job*)

Instantiates the PipeStream instance

Parameters **job** (Job) – job to use for current working directory and environment variables.

Returns The constructed PipeStream instance.

Return type *PipeStream*

CHAPTER

FOUR

STREAMS

shtk.Stream instances are pairs of file-like objects (one for reading data from the process, one for writing data to the process) used for communication with running processes. If a stream is one way (e.g. FileStream) then the underlying file-like objects reader or writer may be handles to os.devnull.

shtk.Stream instances are usually constructed internally within SHTK, rather than being directly instantiated by the end user.

4.1 shtk.Stream

class shtk.**Stream**(*fileobj_r=None*, *fileobj_w=None*)

Bases: object

Base class for other Stream classes.

Wraps file-like objects to couple readers and writers to the same streams (where it makes sense) and more tightly control closure of the stream. Also functions as a context manager (yielding self) that calls self.close() upon exit.

Parameters

- **fileobj_r** (*file-like or None*) – A file-like object suitable for reading.
- **fileobj_w** (*file-like or None*) – A file-like object suitable for writing.

fileobj_r

A file-like object suitable for reading.

Type file-like or None

fileobj_w

A file-like object suitable for writing.

Type file-like or None

reader()

Returns fileobj_r

Returns self.fileobj_r

Return type file-like

writer()

Returns fileobj_w

Returns self.fileobj_w

Return type file-like

close_reader()
Closes self.fileobj_r if it's not None, then set it to None

close_writer()
Closes self.fileobj_w if it's not None, then set it to None

close()
Calls self.close_reader() and self.close_writer()

4.2 shtk.FileStream

class shtk.**FileStream**(*path*, *mode*, *user=None*, *group=None*)
Bases: *shtk.Stream.Stream*

Opens a file for reading or writing

Parameters

- **path** (*str or pathlib.Path*) – The path of the file to open.
- **mode** (*str*) – Mode passed to open() when opening the file. If mode contains 'r' then the file will be opened for reading. If the mode contains 'w' or 'a' it will be opened for writing.
- **user** (*None, int, str*) – The user that will own the file (if 'w' in mode). If user is an int, the file will be chown'd to the user whose uid=user. If user is an str, the file will be chown'd to the user whose name=user.
- **group** (*None, int, str*) – The group that will own the file (if 'w' in mode). If group is an int, the file will be chown'd to the group whose gid=group. If group is an str, the file will be chown'd to the group whose name=group.

close()
Calls self.close_reader() and self.close_writer()

close_reader()
Closes self.fileobj_r if it's not None, then set it to None

close_writer()
Closes self.fileobj_w if it's not None, then set it to None

reader()
Returns fileobj_r

Returns self.fileobj_r

Return type file-like

writer()
Returns fileobj_w

Returns self.fileobj_w

Return type file-like

4.3 shtk.ManualStream

class shtk.**ManualStream**(*fileobj_r=None*, *fileobj_w=None*)

Bases: *shtk.Stream.Stream*

Uses provided file-like objects for fileobj_r and fileobj_w.

Note: The files will not be manually closed even when close_reader() or close_writer() are called. Closing the files is the responsibility of the caller.

Parameters

- **fileobj_r** (*file-like*) – The file-like object to use for self.fileobj_r.
- **fileobj_w** (*file-like*) – The file-like object to use for self.fileobj_w.

close_r

Whether the reader should be closed when close_reader() is called.

Type boolean

close_w

Whether the writer should be closed when close_writer() is called.

Type boolean

close_reader()

Close the reader only if it wasn't provided at instantiation.

close_writer()

Close the writer only if it wasn't provided at instantiation.

close()

Calls self.close_reader() and self.close_writer()

reader()

Returns fileobj_r

Returns self.fileobj_r

Return type file-like

writer()

Returns fileobj_w

Returns self.fileobj_w

Return type file-like

4.4 shtk.NullStream

class shtk.**NullStream**
Bases: *shtk.Stream.Stream*

Opens os.devnull for both reading and writing

close()
Calls self.close_reader() and self.close_writer()

close_reader()
Closes self.fileobj_r if it's not None, then set it to None

close_writer()
Closes self.fileobj_w if it's not None, then set it to None

reader()
Returns fileobj_r

Returns self.fileobj_r

Return type file-like

writer()
Returns fileobj_w

Returns self.fileobj_w

Return type file-like

4.5 shtk.PipeStream

class shtk.**PipeStream**(*binary=False*, *flags=0*)
Bases: *shtk.Stream.Stream*

Creates an os.pipe2() suitable for communicating between processes

Parameters

- **binary** (*boolean*) – Whether the streams should be opened in binary mode (Default value = False).
- **flags** (*int*) – Flags to pass to os.pipe2 in addition to os.O_CLOEXEC (Default value = 0).
- **user** (*None, int, str*) – The user that will own the pipe. If user is an int, the file will be chown'd to the user whose uid=user. If user is an str, the file will be chown'd to the user whose name=user.
- **group** (*None, int, str*) – The group that will own the pipe. If group is an int, the file will be chown'd to the group whose gid=group. If group is an str, the file will be chown'd to the group whose name=group.

close()
Calls self.close_reader() and self.close_writer()

close_reader()
Closes self.fileobj_r if it's not None, then set it to None

close_writer()
Closes self.fileobj_w if it's not None, then set it to None

`reader()`
Returns fileobj_r

Returns self.fileobj_r

Return type file-like

`writer()`
Returns fileobj_w

Returns self.fileobj_w

Return type file-like

CHAPTER

FIVE

PIPELINE NODE FACTORIES

shtk.PipelineNodeFactory subclasses are templates used to define the properties of shtk.PipelineNode instances. shtk.PipelineNode instances are nodes within a directed acyclic graph (DAG) that represent a process pipeline.

shtk.PipelineNodeFactory instances are useful because they enable (1) running process pipelines multiple times (2) displaying the commands that will be run as part of the pipeline prior to executing the command and (3) composing partial pipelines into more complex process pipelines.

PipelineNodeFactory instances are usually constructed by a shtk.Shell.command(), or by using the pipe operator to connect multiple PipelineProcessFactory instances together into a single process pipeline.

5.1 shtk.PipelineNodeFactory

class shtk.**PipelineNodeFactory**(*stdin_factory=None*, *stdout_factory=None*, *stderr_factory=None*)

Bases: abc.ABC

Abstract base class defining a template for building PipelineNode's

Parameters

- **stdin_factory** (*None or StreamFactory*) – Template for stdin Stream's (Default value: None)
- **stdout_factory** (*None or StreamFactory*) – Template for stdout Stream's (Default value: None)
- **stderr_factory** (*None or StreamFactory*) – Template for stderr Stream's (Default value: None)

stdin_factory

Template for stdin Stream's (Default value: None)

Type None or *StreamFactory*

stdout_factory

Template for stdout Stream's (Default value: None)

Type None or *StreamFactory*

stderr_factory

Template for stderr Stream's (Default value: None)

Type None or *StreamFactory*

children

templates for children

Type list of PipelineNodeFactory

stdin(*arg*, *mode='r'*)
Sets the stdin stream factory (in-place)

Parameters

- **arg** (*str, pathlib.Path, StreamFactory, or None*) – If arg is an str or pathlib.Path, it is treated as a filename and stdin will be read from that file.

  If arg is a StreamFactory it is used directly to create streams for stdin.

  If None, stdin reads from os.devnull
- **mode** – The mode in which to open the file, if opened. Only relevant if arg is a str or pathlib.Path. Must be one of ('r', 'rb'). (Default value = 'r')

Returns Altered self

Return type *PipelineNodeFactory*

stdout(*arg*, *mode='w'*)
Sets the stdout stream factory (in-place)

Parameters

- **arg** (*str, pathlib.Path, StreamFactory, or None*) – If arg is an str or pathlib.Path, it is treated as a filename and stdout will write to that file.

  If arg is a StreamFactory it is used directly to create streams for stdout.

  If None, stdout writes to os.devnull
- **mode** – The mode in which to open the file, if opened. Only relevant if arg is a str or pathlib.Path. Must be one of ('w', 'wb', 'a', 'ab'). (Default value = 'w')

Returns Altered self

Return type *PipelineNodeFactory*

stderr(*arg*, *mode='w'*)
Sets the stderr stream factory (in-place)

Parameters

- **arg** (*str, pathlib.Path, StreamFactory, or None*) – If arg is an str or pathlib.Path, it is treated as a filename and stderr will write to that file.

  If arg is a StreamFactory it is used directly to create streams for stderr.

  If None, stderr writes to os.devnull
- **mode** – The mode in which to open the file, if opened. Only relevant if arg is a str or pathlib.Path. Must be one of ('w', 'wb', 'a', 'ab'). (Default value = 'w')

Returns Altered self

Return type *PipelineNodeFactory*

async build(*job*, *stdin_stream=None*, *stdout_stream=None*, *stderr_stream=None*)
Creates and executes PipelineNode's and self-defined StreamFactories

If self.std{in,out,err}_factory is defined, it is pased to the child as the preferred stream. Otherwise the std{in,out,err}_stream parameters are used.

Parameters

- **job** (Job) – job from which to pull environment variables, current working directory, etc.
- **stdin_stream** (Stream) – Stream instance to pass to PipelineNode as stdin

- **stdout_stream** (Stream) – Stream instance to pass to PipelineNode as stdout
- **stderr_stream** (Stream) – Stream instance to pass to PipelineNode as stderr

Returns The constructed PipelineNode instance

Return type *PipelineNode*

abstract async build_inner(*job*, *stdin_stream*, *stdout_stream*, *stderr_stream*)

Abstract method used for instantiating PipelineNodes. This method is wrapped by build() which handles stream management prior to passing them to build_inner().

Parameters

- **job** (Job) – The job from which to pull the current working directory and environment variables for subprocesses.
- **stdin_stream** (Stream) – The Stream instance to be used as the PipelineNode's stdin_stream.
- **stdout_stream** (Stream) – The Stream instance to be used as the PipelineNode's stdout_stream.
- **stderr_stream** (Stream) – The Stream instance to be used as the PipelineNode's stderr_stream.

Returns An instantiated PipelineNode.

Return type *PipelineNode*

5.2 shtk.PipelineChannelFactory

class shtk.**PipelineChannelFactory**(*left*, *right*, ***kwargs*)

Bases: *shtk.PipelineNodeFactory.PipelineNodeFactory*

PipelineChannelFactory is a template for creating PipelineChannel.

PipelineChannelFactory creates PipelineChannel instances representing a chain of subprocesses with each feeding stdout to the next subprocess's stdin.

Parameters

- **left** (PipelineNodeFactory) – A PipelineNodeFactory that will create a series of processes that should write to stdout.
- **right** (PipelineNodeFactory) – A PipelineNodeFactory that will create a series of processes that should read from stdin.

left

A PipelineNodeFactory that will create a series of processes that should write to stdout.

Type *PipelineNodeFactory*

right

A PipelineNodeFactory that will create a series of processes that should read from stdin.

Type *PipelineNodeFactory*

children

[left, right]

Type list of PipelineNodeFactory

pipe_stream
The PipeStreamFactory that will be used to redirect left's stdout to right's stdin.

Type *PipeStreamFactory*

async build_inner(*job*, *stdin_stream*, *stdout_stream*, *stderr_stream*)
Instantiates a PipelineChannel

async build(*job*, *stdin_stream=None*, *stdout_stream=None*, *stderr_stream=None*)
Creates and executes PipelineNode's and self-defined StreamFactories

If self.std{in,out,err}_factory is defined, it is pased to the child as the preferred stream. Otherwise the std{in,out,err}_stream parameters are used.

Parameters

- **job** (Job) – job from which to pull environment variables, current working directory, etc.
- **stdin_stream** (Stream) – Stream instance to pass to PipelineNode as stdin
- **stdout_stream** (Stream) – Stream instance to pass to PipelineNode as stdout
- **stderr_stream** (Stream) – Stream instance to pass to PipelineNode as stderr

Returns The constructed PipelineNode instance

Return type *PipelineNode*

stderr(*arg*, *mode='w'*)
Sets the stderr stream factory (in-place)

Parameters

- **arg** (*str, pathlib.Path,* StreamFactory*, or None*) – If arg is an str or pathlib.Path, it is treated as a filename and stderr will write to that file.

  If arg is a StreamFactory it is used directly to create streams for stderr.

  If None, stderr writes to os.devnull

- **mode** – The mode in which to open the file, if opened. Only relevant if arg is a str or pathlib.Path. Must be one of ('w', 'wb', 'a', 'ab'). (Default value = 'w')

Returns Altered self

Return type *PipelineNodeFactory*

stdin(*arg*, *mode='r'*)
Sets the stdin stream factory (in-place)

Parameters

- **arg** (*str, pathlib.Path,* StreamFactory*, or None*) – If arg is an str or pathlib.Path, it is treated as a filename and stdin will be read from that file.

  If arg is a StreamFactory it is used directly to create streams for stdin.

  If None, stdin reads from os.devnull

- **mode** – The mode in which to open the file, if opened. Only relevant if arg is a str or pathlib.Path. Must be one of ('r', 'rb'). (Default value = 'r')

Returns Altered self

Return type *PipelineNodeFactory*

stdout(*arg*, *mode='w'*)
Sets the stdout stream factory (in-place)

Parameters

- **arg** (*str, pathlib.Path, StreamFactory, or None*) – If arg is an str or pathlib.Path, it is treated as a filename and stdout will write to that file.

  If arg is a StreamFactory it is used directly to create streams for stdout.

  If None, stdout writes to os.devnull

- **mode** – The mode in which to open the file, if opened. Only relevant if arg is a str or pathlib.Path. Must be one of ('w', 'wb', 'a', 'ab'). (Default value = 'w')

Returns Altered self

Return type *PipelineNodeFactory*

5.3 shtk.PipelineProcessFactory

class shtk.**PipelineProcessFactory**(**args*, *env=None*, *cwd=None*, *close_fds=True*)

Bases: *shtk.PipelineNodeFactory.PipelineNodeFactory*

Template for a PipelineProcess which runs a command as a subprocess

Parameters

- ***args** (*list of str or pathlib.Path*) – The command to run and its arguments for the instantiated PipelineProcess instances.
- **environment** (*dict*) – The environment variables to use for the instantiated PipelineProcess instances (Default value = {}).
- **cwd** (*str or pathlib.Path*) – The current working directory for the instantiated PipelineProcess instances (Default value = None).
- **close_fds** (*bool*) – If true, close_fds will be passed to the equivalent of subprocess.Popen().

async build(*job*, *stdin_stream=None*, *stdout_stream=None*, *stderr_stream=None*)

Creates and executes PipelineNode's and self-defined StreamFactories

If self.std{in,out,err}_factory is defined, it is pased to the child as the preferred stream. Otherwise the std{in,out,err}_stream parameters are used.

Parameters

- **job** (Job) – job from which to pull environment variables, current working directory, etc.
- **stdin_stream** (Stream) – Stream instance to pass to PipelineNode as stdin
- **stdout_stream** (Stream) – Stream instance to pass to PipelineNode as stdout
- **stderr_stream** (Stream) – Stream instance to pass to PipelineNode as stderr

Returns The constructed PipelineNode instance

Return type *PipelineNode*

async build_inner(*job*, *stdin_stream*, *stdout_stream*, *stderr_stream*)

Instantiates a PipelineProcess

stderr(*arg*, *mode='w'*)

Sets the stderr stream factory (in-place)

Parameters

- **arg** (*str, pathlib.Path, StreamFactory, or None*) – If arg is an str or pathlib.Path, it is treated as a filename and stderr will write to that file.

  If arg is a StreamFactory it is used directly to create streams for stderr.

  If None, stderr writes to os.devnull

- **mode** – The mode in which to open the file, if opened. Only relevant if arg is a str or pathlib.Path. Must be one of ('w', 'wb', 'a', 'ab'). (Default value = 'w')

Returns Altered self

Return type *PipelineNodeFactory*

stdin(*arg*, *mode='r'*)
Sets the stdin stream factory (in-place)

Parameters

- **arg** (*str, pathlib.Path, StreamFactory, or None*) – If arg is an str or pathlib.Path, it is treated as a filename and stdin will be read from that file.

  If arg is a StreamFactory it is used directly to create streams for stdin.

  If None, stdin reads from os.devnull

- **mode** – The mode in which to open the file, if opened. Only relevant if arg is a str or pathlib.Path. Must be one of ('r', 'rb'). (Default value = 'r')

Returns Altered self

Return type *PipelineNodeFactory*

stdout(*arg*, *mode='w'*)
Sets the stdout stream factory (in-place)

Parameters

- **arg** (*str, pathlib.Path, StreamFactory, or None*) – If arg is an str or pathlib.Path, it is treated as a filename and stdout will write to that file.

  If arg is a StreamFactory it is used directly to create streams for stdout.

  If None, stdout writes to os.devnull

- **mode** – The mode in which to open the file, if opened. Only relevant if arg is a str or pathlib.Path. Must be one of ('w', 'wb', 'a', 'ab'). (Default value = 'w')

Returns Altered self

Return type *PipelineNodeFactory*

CHAPTER

SIX

PIPELINE NODES

shtk.PipelineNodeFactory subclasses are templates used to define the properties of shtk.PipelineNode instances. shtk.PipelineNode instances are nodes within a directed acyclic graph (DAG) that represent a process pipeline.

The leaf nodes of this DAG are always PipelineProcess instances representing an individual process that is part of the pipeline.

PipelineNode instances can be used to communicate, start, and stop individual processes within a process pipeline.

PipelineNode classes are typically instantiated by other elements of SHTK, rather than being manually instantiated by the end user.

6.1 shtk.PipelineNode

class shtk.**PipelineNode**(*event_loop*)

Bases: abc.ABC

Abstract base class for subprocess management nodes

children

children of this node

Type list of PipelineNode

stdin_stream

Stream to use for stdin

Type None or *Stream*

stdout_stream

Stream to use for stdout

Type None or *Stream*

stderr_stream

Stream to use for stderr

Type None or *Stream*

async classmethod create(**args*, ***kwargs*)

Instantiates and runs the node

Parameters

- ***args** – passed to the constructor
- ****kwargs** – passed to the constructor

Returns The instantiated and run node.

Return type *PipelineNode*

`async run()`
Runs the process

`flatten_children()`
Flattens the PipelineNode DAG into a list of PipelineProcess objects using a depth-first search.

Returns All child PipelineProcess nodes

Return type list of PipelineProcess

`send_signal(signum)`
Sends a signal to all child ProcessNode processes.

Parameters **signum** (*int*) – the signal to send.

`terminate()`
Sends a signal.SIGTERM to all child ProcessNode processes.

`kill()`
Sends a signal.SIGKILL to all child ProcessNode processes.

`async poll_async(ret)`
Gets the return codes of all child ProcessNodes

Parameters **ret** (*list of [int, None]*) – a list that will be modified to contain a collection of return codes from flattened child ProcessNodes. Child processes that have exited will be represented by their return code. Child processes that have not exited will be represented by None.

`poll(timeout=1e-06)`
Synchronous wrapper for poll_async(). Gets the return codes of all child ProcessNodes.

Returns

A list containing return codes from flattened child ProcessNodes. Child processes that have exited will be represented by their integer return code. Child processes that have not exited will be represented by None.

Return type list of (int or None)

`async wait_async()`
Waits for and retrieves the return codes of all child ProcessNodes.

Returns A list of return codes from a flattened collection of child processes.

Return type list of int

`wait()`
Synchronous wrapper for wait_async().

Returns A list of return codes from a flattened collection of child processes.

Return type list of int

6.2 shtk.PipelineChannel

class shtk.**PipelineChannel**(*event_loop*, *left*, *right*)

Bases: *shtk.PipelineNode.PipelineNode*

Represents a pipeline of commands

Parameters

- **left** (PipelineNode) – A PipelineNode whose stdout is (usually) fed to right
- **right** (PipelineNode) – A PipelineNode whose stdin is (usually) read from left

left

The left PipelineNode

Type *PipelineNode*

right

The right PipelineNode

Type *PipelineNode*

async classmethod **create**(**args*, ***kwargs*)

Instantiates and runs the node

Parameters

- ***args** – passed to the constructor
- ****kwargs** – passed to the constructor

Returns The instantiated and run node.

Return type *PipelineNode*

flatten_children()

Flattens the PipelineNode DAG into a list of PipelineProcess objects using a depth-first search.

Returns All child PipelineProcess nodes

Return type list of PipelineProcess

kill()

Sends a signal.SIGKILL to all child ProcessNode processes.

poll(*timeout=1e-06*)

Synchronous wrapper for poll_async(). Gets the return codes of all child ProcessNodes.

Returns

A list containing return codes from flattened child ProcessNodes. Child processes that have exited will be represented by their integer return code. Child processes that have not exited will be represented by None.

Return type list of (int or None)

async **poll_async**(*ret*)

Gets the return codes of all child ProcessNodes

Parameters **ret** (*list of [int, None]*) – a list that will be modified to contain a collection of return codes from flattened child ProcessNodes. Child processes that have exited will be represented by their return code. Child processes that have not exited will be represented by None.

async run()
Runs the process

send_signal(*signum*)
Sends a signal to all child ProcessNode processes.

Parameters **signum** (*int*) – the signal to send.

terminate()
Sends a signal.SIGTERM to all child ProcessNode processes.

wait()
Synchronous wrapper for wait_async().

Returns A list of return codes from a flattened collection of child processes.

Return type list of int

async wait_async()
Waits for and retrieves the return codes of all child ProcessNodes.

Returns A list of return codes from a flattened collection of child processes.

Return type list of int

6.3 shtk.PipelineProcess

class shtk.**PipelineProcess**(*event_loop*, *cwd*, *args*, *env*, *stdin_stream*, *stdout_stream*, *stderr_stream*, *user=None*, *group=None*, *close_fds=True*)
Bases: *shtk.PipelineNode.PipelineNode*

An interface representing subprocesses.

Parameters

- **cwd** (*str or pathlib.Path*) – The current working directory
- **args** (*list of str or pathlib.Path*) – The arguments for the process (including the base command).
- **env** (*dict of str*) – The environment variables for the process
- **stdin_stream** (*Stream*) – The Stream whose .reader() is used as stdin
- **stdout_stream** (*Stream*) – The Stream whose .writer() is used as stdout
- **stderr_stream** (*Stream*) – The Stream whose .writer() is used as stderr
- **user** (*None, int, or str*) – The user to pass to asyncio.create_subprocess_exec(). Requires Python >= 3.9.
- **group** (*None, int, or str*) – The group to pass to asyncio.create_subprocess_exec(). Requires Python >= 3.9.
- **close_fds** (*bool*) – If true, close_fds will be passed to the equivalent of subprocess.Popen().

Raises **AssertionError** – When len(args) <= 0

async run()
Runs the process using asyncio.create_subprocess_exec()

async classmethod create(**args*, ***kwargs*)
Instantiates and runs the node

Parameters

- ***args** – passed to the constructor
- ****kwargs** – passed to the constructor

Returns The instantiated and run node.

Return type *PipelineNode*

flatten_children()
Flattens the PipelineNode DAG into a list of PipelineProcess objects using a depth-first search.

Returns All child PipelineProcess nodes

Return type list of PipelineProcess

kill()
Sends a signal.SIGKILL to all child ProcessNode processes.

poll(*timeout=1e-06*)
Synchronous wrapper for poll_async(). Gets the return codes of all child ProcessNodes.

Returns

A list containing return codes from flattened child ProcessNodes. Child processes that have exited will be represented by their integer return code. Child processes that have not exited will be represented by None.

Return type list of (int or None)

async poll_async(*ret*)
Gets the return codes of all child ProcessNodes

Parameters **ret** (*list of [int, None]*) – a list that will be modified to contain a collection of return codes from flattened child ProcessNodes. Child processes that have exited will be represented by their return code. Child processes that have not exited will be represented by None.

send_signal(*signum*)
Sends a signal to all child ProcessNode processes.

Parameters **signum** (*int*) – the signal to send.

terminate()
Sends a signal.SIGTERM to all child ProcessNode processes.

wait()
Synchronous wrapper for wait_async().

Returns A list of return codes from a flattened collection of child processes.

Return type list of int

async wait_async()
Waits for and retrieves the return codes of all child ProcessNodes.

Returns A list of return codes from a flattened collection of child processes.

Return type list of int

CHAPTER

SEVEN

EXCEPTIONS

shtk defines certain custom exceptions to indicate when certain events occur.

7.1 shtk.NonzeroExitCodeException

class shtk.**NonzeroExitCodeException**(*processes*)

Bases: Exception

Raised when a process within an SHTK job exits with a nonzero return code.

Parameters **processes** (*list*) – The PipelineProcess instances to include in the error message

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

INDEX