pyats-06 - Managing logging and reporting capabilities

Syllabus

- logging concept
- default implementations
- banners and titles
- logging configuration
- plugins for `easypy`

Presentation

Logging concept

Python's logging system:

- each module/package can have its own `Logger`
- each `Logger` could be equipped with multiple `Handler`
- each `Handler` could have its own `Formatter` and `Filter`

Cisco log format

```
# the format itself, in Python logging formatter style
# ----------------------------------------------------

{seqnum}: {hostname}: {time}: %{appname}-{severity}-{msgname}: {tags}: {message}%

seqnum      - message sequence number, starts with 0 for each process
hostname    - current host name
time        - timestamp in yyyy-mm-ddThh:mm:ss format
appname     - the application generating this message
severity    - message severity, range 1-7
msgname     - name of the message
tags        - optional tags associated with this message
message     - message given by the user
```

Logging example:

```
# logs.py
import logging
from pyats.aetest import Testcase, test, main


_log = logging.getLogger(__name__)


class SmokeTest(Testcase):

    @test
    def test_1(self):
        _log.debug('debug message')
        _log.info('info message')
        _log.warning('warning message')
        _log.critical('critical message')
        _log.error('error message')


if __name__ == '__main__':
    main()
```

Default implementations

The sample below is integrated with `easypy`.

```
from pyats.log import ScreenHandler, TaskLogHandler, ScreenFormatter, TaskLogFormatter

help(ScreenHandler)
help(TaskLogHandler)
help(ScreenFormatter)
help(TaskLogFormatter)

import sys
import logging
logger = logging.getLogger(__name__)

# handler example
handler = ScreenHandler()
handler = ScreenHandler(sys.stdout)
logger.addHandler(handler)
logger.critical('a critical message')

# task log example
handler = TaskLogHandler('./TaskLog-A.log')
logger.addHandler(handler)
logger.setLevel(logging.INFO)
logger.info('an info message')
```

Banners and Titles

```
import logging
from pyats.log.utils import banner, title

logger = logging.getLogger(__name__)

# banners
logger.warning(banner('an informational message banner'))
logger.warning(banner('an error message\nwith newline'))
logger.warning(banner('aReallyLongMessageThatIsLongerThanMaxWidthIsChoppedUp', width = 40))

# titles
logger.warning(title('an informational message title'))
logger.warning(title('an error message\nwith newline'))
logger.warning(title('a message', margin='!'))
```

Logging configuration

There are two places for configuring logging's levels:

- `__main__` declaration in an `aetest` module
- inside `main` function in a job file

Custom package

Structure

```
.
└── moon
    ├── __init__.py
    ├── a.py
    └── b.py
```

Files

```
# moon/__init__.py
def template(log, prefix):
    log.debug(prefix + ': debug message')
    log.info(prefix + ': info message')
    log.warning(prefix + ': warning message')
    log.critical(prefix + ': critical message')
    log.error(prefix + ': error message')

# moon/a.py
import logging

from moon import template

_log = logging.getLogger(__name__)

def a():
    template(_log, 'a')

# moon/b.py
import logging

from moon import template

_log = logging.getLogger(__name__)

def b():
    template(_log, 'b')
```

Important note!
Don't set logging's level within your modules like

```
_log = logging.getLogger(__name__)
_log.setLevel(logging.DEBUG)
```

In this case, you won't be able to override defined configuration.

`aetest` module

```
# testcase.py
import logging

from pyats.aetest import Testcase, test, main
from moon import a, b, template

_log = logging.getLogger(__name__)


class SmokeTest(Testcase):

    @test
    def test_1(self):
        template(_log, __name__)
        a.a()
        b.b()

if __name__ == '__main__':
    logging.getLogger(__name__).setLevel(logging.DEBUG)
    logging.getLogger('moon').setLevel(logging.CRITICAL)
    logging.getLogger('moon.a').setLevel(logging.DEBUG)
    main()
```

```
python testcase.py
```

`easypy` job

```
# suit.py
import logging

from pyats.easypy import run


def main(runtime):
    logging.getLogger('pyats.aetest.testscript.testcase').setLevel(logging.ERROR)
    logging.getLogger('moon').setLevel(logging.CRITICAL)
    logging.getLogger('moon.a').setLevel(logging.DEBUG)

    run(testscript='testcase.py')
```

```
easypy suit.py
```

Plugins for `easypy`

Easypy plugin stages:

Stage	Description
`pre_job`	run before the jobfile starts
`pre_task`	run before the start of each Task, within the task process
`post_task`	run after the finish of each Task, within the task process
`post_job`	run after the jobfile finishes

Your plugin's class must inherit `easypy.plugins.bases.BasePlugin` class.

The task

```
# test.py
import os
from pyats.aetest import Testcase, test, setup, main


class Smoke(Testcase):

    @setup
    def create(self, directory):
        if not os.path.exists(directory):
            os.makedirs(directory)

    @test
    def foo(self, directory):
        with open(f'{directory}/foo.txt', 'w') as file:
            file.write('Some important info: foo')

    @test
    def bar(self, directory):
        with open(f'{directory}/bar.txt', 'w') as file:
            file.write('Some important info: bar')


if __name__ == '__main__':
    main(directory='./test-dir')

# job.py
from pyats.easypy import run


def main():
    run(testscript='test.py', directory='./txt')
```

```
easypy job.py
ls ./txt
unzip -l archive/...
```

Copy file plugin

```
# plugins/save.py
import logging
import shutil
from pyats.easypy.plugins.bases import BasePlugin

logger = logging.getLogger(__name__)
logger.setLevel(logging.INFO)


class CopyReport(BasePlugin):
    def __init__(self, directory, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self._source = directory
        self._dest='custom-data'

    def post_job(self, job):
        logger.info('The directory to copy: %s; destination: %s', self._source, self._dest)
        shutil.copytree(self._source, f'{job.runtime.directory}/{self._dest}')
```

`easypy` configuration

```
# plugins.yaml
plugins:
  CopyReport:
    directory: './txt'
    enabled: True
    module: save
    order: 1.0
```

```
export PYTHONPATH=${PWD}/plugins:${PYTHONPATH}
easypy -configuration plugins.yaml job.py
unzip -l archive/...
```

Additional materials

- https://docs.python.org/3.6/howto/logging.html#logging-advanced-tutorial
- https://pubhub.devnetcloud.com/media/pyats/docs/log
- https://pubhub.devnetcloud.com/media/pyats/docs/easypy/jobfile.html#log-levels
- https://pubhub.devnetcloud.com/media/pyats/docs/easypy/plugins.html
- https://github.com/CiscoDevNet/pyats-plugin-examples/tree/master/easypy_plugin_example

Control

Task description

There is a following testbed file:

```
testbed:
  name: "plugin homework"
  custom:
    directory: ./custom
    report-directory: ./data
```

You have to integrate it with the code from "plugins for `easypy`" section. So, the idea is to make all directories configurable via the testbed file.

Please perceive testbed file options like

1. `directory` is a place to store files in the tests
2. `directory` is a source directory in the plugin
3. `report-directory` is a destination directory within a report archive

Review items

Please send a zip archive with all required files and a command to run the execution.