pandora

Devel's Manual

Ver. 2017-04-30/2211

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Revision History

Date	Changes
2017-04-13	Created.

1 Glossary

Begriff:
Erklärung.

Begriff:
Erklärung.

2 Introduction

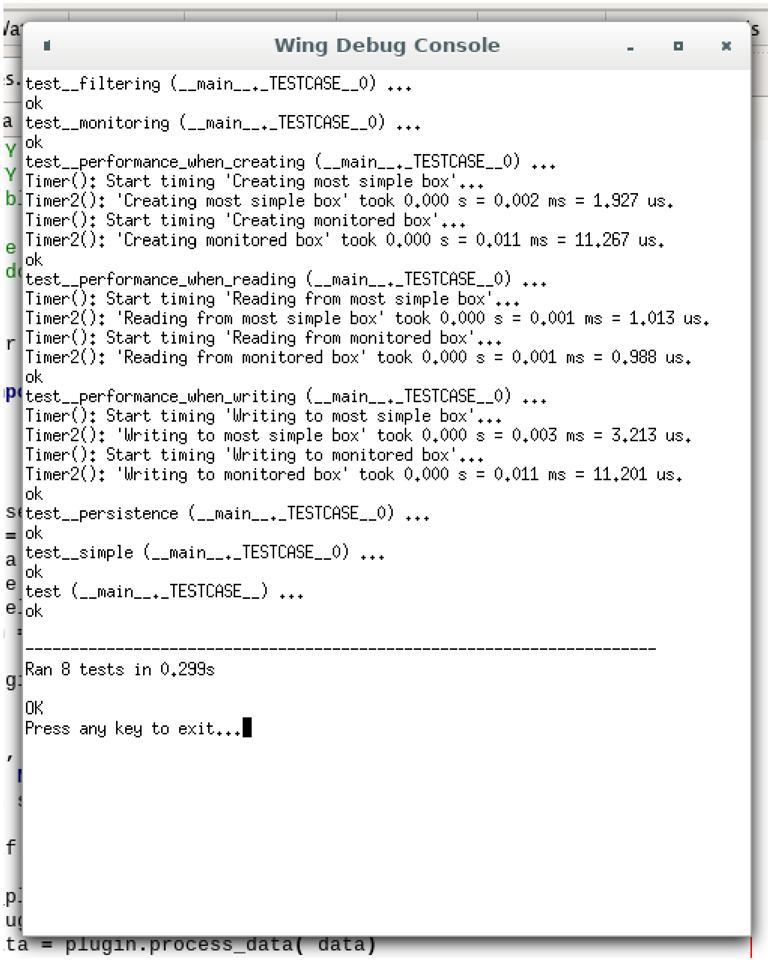
3 Design

A Box holds data, which can be an int, a float, a str or any other user defined type. Boxes may be customized by plugins. Plugins are supplied with the box's data and return processed data.

The amount of data attributes is not changed by plugins. This means, that e.g. timestamps must not be stored in the respective Box, but have to be stored in the respective plugin. If e.g. a filter needs timestamps, the filter plugin – and not the Box – is responsible for getting at them.

Obeying this concept keeps the Box's design clean and its performance decent.

4 Performance



```
test__filtering (__main__._TESTCASE__0) ...
ok
test__monitoring (__main__._TESTCASE__0) ...
ok
test__performance_when_creating (__main__._TESTCASE__0) ...
Timer(): Start timing 'Creating most simple box'...
Timer2(): 'Creating most simple box' took 0.000 s = 0.002 ms = 1.927 us.
Timer(): Start timing 'Creating monitored box'...
Timer2(): 'Creating monitored box' took 0.000 s = 0.011 ms = 11.267 us.
ok
test__performance_when_reading (__main__._TESTCASE__0) ...
Timer(): Start timing 'Reading from most simple box'...
Timer2(): 'Reading from most simple box' took 0.000 s = 0.001 ms = 1.013 us.
Timer(): Start timing 'Reading from monitored box'...
Timer2(): 'Reading from monitored box' took 0.000 s = 0.001 ms = 0.988 us.
ok
test__performance_when_writing (__main__._TESTCASE__0) ...
Timer(): Start timing 'Writing to most simple box'...
Timer2(): 'Writing to most simple box' took 0.000 s = 0.003 ms = 3.213 us.
Timer(): Start timing 'Writing to monitored box'...
Timer2(): 'Writing to monitored box' took 0.000 s = 0.011 ms = 11.201 us.
ok
test__persistence (__main__._TESTCASE__0) ...
ok
test__simple (__main__._TESTCASE__0) ...
ok
test (__main__._TESTCASE__) ...
ok

----------------------------------------------------------------------
Ran 8 tests in 0.299s

OK
Press any key to exit...
```

Fig. 1: Performance (2017-04-18)

5 Application

5.1 A Box W/O An Id

```
import pandora as p

    p_controlerror = p.Box( data=0.0, label="Control Error")
    p_controlerror.data( 42)
    controlerror = p_controlerror.data()
```

5.2 A Box W/ An Id

```
import pandora as p

    p_controlerror = p.Box( id="controlerror", data=0.0, label="Control Error")
    p_controlerror.data( 42)
    controlerror = p_controlerror.data()
```

5.3 A Box W/ A Label And A Dimension

```
import pandora as p

p_reference = p.Box( \
    label="Reference value", data=0.0, dim="U/min",
    plugins=(p.plugins.Monitor( _on_reference_changed_), )
    )
                                # _on_reference_changed_ defined elsewhere
```

5.4 A Box Monitoring Its Access

```
import pandora as p

p_reference = p.Box( \
    label="Reference value", data=0.0, dim="U/min",
    plugins=(p.plugins.Monitor( _on_reference_changed_), )
    )
                                # _on_reference_changed_ defined elsewhere
```

5.5 Plugins Available To Applications

- Clipper4Numbers
- Monitor

5.6 Create Your Own Plugins

See Chapter plugins.py on page 14.

Appendix: Source Code

__init__.py

```
#!/usr/bin/env python3
#   -*- coding: utf8 -*- #
#
#
#   Copyright (C) by p.oseidon@datec.at, 1998 - 2016
#
#   This file is part of pandora.
#
#   pandora is free software: you can redistribute it and/or modify
#   it under the terms of the GNU General Public License as published by
#   the Free Software Foundation, either version 3 of the License, or
#   (at your option) any later version.
#
#   pandora is distributed in the hope that it will be useful,
#   but WITHOUT ANY WARRANTY; without even the implied warranty of
#   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#   GNU General Public License for more details.
#
#   You should have received a copy of the GNU General Public License
#   along with pandora. If not, see <http://www.gnu.org/licenses/>.

from tau4.data.pandora._boxes import Box, Boxes
import tau4.data.pandora.plugins
```

_boxes.py

```
#!/usr/bin/env python3
#   -*- coding: utf8 -*- #
#
#
#   Copyright (C) by p.oseidon@datec.at, 1998 - 2016
#
#   This file is part of pandora.
#
#   pandora is free software: you can redistribute it and/or modify
#   it under the terms of the GNU General Public License as published by
#   the Free Software Foundation, either version 3 of the License, or
#   (at your option) any later version.
#
#   pandora is distributed in the hope that it will be useful,
#   but WITHOUT ANY WARRANTY; without even the implied warranty of
#   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#   GNU General Public License for more details.
#
#   You should have received a copy of the GNU General Public License
#   along with pandora. If not, see <http://www.gnu.org/licenses/>.

from collections import OrderedDict
import configparser as cp

from tau4.sweng import Singleton

from tau4.data.pandora.plugins import Plugins


class Box:

    def __init__( self, *, data, id=None, label="", dim="", plugins=None):
        self.__id = id
        self.__data = data
        self.__type = type( data)
        self.__label = label
        self.__dim = dim

        self.__plugins = Plugins()
        if plugins:
            for plugin in plugins:
                self.__plugins.append( plugin)

        return

    def data( self, data=None):
        if data is None:
            return self.__data

        data = self._typecast_( data)

        for plugin in self.__plugins.values():
            data = plugin.process_data( data)

        assert self._type_is_valid_( data)
        self.__data = data
        return self
    value = data # Compatibility w/ tau4data.flex
    def data2dict( self):
        return {"data": self.data()}

    def dict2data( self, d):
        return self.data( d[ "data"])

    def dim( self):
        return self.__dim

    def id( self):
        return self.__id

    def label( self):
        return self.__label

    def plugin_append( self, plugin):
        self.__plugins.append( plugin)
        return self

    def _typecast_( self, arg):
        return self.__type( arg)

    def _type_is_valid_( self, arg):
        return type( arg) is self.__type


class Boxes(metaclass=Singleton):

    def __init__( self):
        self.__boxes = {}
        self.__pathname = "./boxes.ini"

        self.__sectionname = "pandora.boxes"
        self.__cp = _ConfigParser( self.pathname_ini())
        return

    def add( self, p):
        if p.id() in self.__boxes:
            raise KeyError( "Box '%s' already in Boxes!" % p.id())

        self.__boxes[ p.id()] = p
        return

    def box( self, id):
        return self.__boxes[ id]

    def pathname_ini( self, pathname=None):
        if pathname is None:
            return str( self.__pathname)

        self.__pathname = pathname
        return self

    def restore_box( self, p):
        try:
            data = self.__cp.get( self.__sectionname, p.id())
            d = eval( data)
            p.dict2data( d)

        except cp.NoSectionError:
            self.__cp.add_section( self.__sectionname)
            self.store_box()

        except cp.NoOptionError:
            self.store_box()

        return self

    def store_box( self, p):
        try:
            self.__cp.set( self.__sectionname, p.id(), str( p.data2dict()))

        except cp.NoSectionError:
            self.__cp.add_section( self.__sectionname)
            self.__cp.set( self.__sectionname, p.id(), str( p.data2dict()))

        self.__cp.write()
        return self


class _ConfigParser(cp.ConfigParser):

    def __init__( self, pathname):
        super().__init__()

        self.__pathname = pathname
        return

    def pathname( self):
        return self.__pathname

    def read( self):
        return super().read( self.pathname())

    def write( self):
        with open( self.pathname(), "wt") as f:
            return super().write( f)
```

_pandora.py

```
#!/usr/bin/env python3
#   -*- coding: utf8 -*- #
#
#
#   Copyright (C) by p.oseidon@datec.at, 1998 - 2016
#
#   This file is part of pandora.
#
#   pandora is free software: you can redistribute it and/or modify
#   it under the terms of the GNU General Public License as published by
#   the Free Software Foundation, either version 3 of the License, or
#   (at your option) any later version.
#
#   pandora is distributed in the hope that it will be useful,
#   but WITHOUT ANY WARRANTY; without even the implied warranty of
#   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#   GNU General Public License for more details.
#
#   You should have received a copy of the GNU General Public License
#   along with pandora. If not, see <http://www.gnu.org/licenses/>.
```

plugins.py

```
#!/usr/bin/env python3
#   -*- coding: utf8 -*- #
#
#
#   Copyright (C) by p.oseidon@datec.at, 1998 - 2016
#
#   This file is part of pandora.
#
#   pandora is free software: you can redistribute it and/or modify
#   it under the terms of the GNU General Public License as published by
#   the Free Software Foundation, either version 3 of the License, or
#   (at your option) any later version.
#
#   pandora is distributed in the hope that it will be useful,
#   but WITHOUT ANY WARRANTY; without even the implied warranty of
#   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#   GNU General Public License for more details.
#
#   You should have received a copy of the GNU General Public License
#   along with pandora. If not, see <http://www.gnu.org/licenses/>.


import abc
from collections import OrderedDict
import sys

from tau4.sweng import PublisherChannel


class Plugins(OrderedDict):

    def __setitem__( self, id, plugin):
        if id in self:
            raise KeyError( "A plugin '%s' has been appended already!" % id)

        OrderedDict.__setitem__( self, id, plugin)
        return self

    def append( self, plugin):
        self[ plugin.id()] = plugin
        return self

    def remove( self, plugin_id):
        del self[ id]
        return self


class Plugin(metaclass=abc.ABCMeta):

    def __init__( self, id):
        self.__id = id
        self.__data = None
        return

    def data( self, data=None):
        if data is None:
            return self.__data

        self.__data = data
        return self

    def id( self):
        return self.__id

    @abc.abstractmethod
    def process_data( self, data):
        """Process data in plugin and - IMPORTANT - return the result.
        """
        pass


class Clipper4Numbers(Plugin):

    def __init__( self, *, id="clipper4numbers", min=sys.float_info.min, max=sys.float_info.max, callable=None):
        super().__init__( id)
        self.__min = min
        self.__max = max
        self.__data_unclipped = None
        if callable:
            self.__on_data_clipped = PublisherChannel.Synch( self)
            self.__on_data_clipped += callable

        return

    def data_unclipped( self):
        return self.__data_unclipped

    def process_data( self, data):
        _type = type( data)
        self.__data_unclipped = data

        is_clipped = False
        if not self.__min <= data:
            data = _type( self.__min)
            is_clipped = True

        if not data <= self.__max:
            data = _type( self.__max)
            is_clipped = True

        self.data( data)

        if is_clipped:
            if self.__on_data_clipped:
                self.__on_data_clipped()

        return data


class Monitor(Plugin):

    def __init__( self, *, id="monitor", callable):
        super().__init__( id)
        self.__on_data_changed = PublisherChannel.Synch( self)
        self.__on_data_changed += callable
        return

    def callable_append( self, callable):
        self.__on_data_changed += callable
        return self

    def callable_remove( self, callable):
        self.__on_data_changed -= callable
        return self

    def process_data( self, data):
        self.data( data)
        self.__on_data_changed()
        return data
```

test__data_pandora.py

```
#!/usr/bin/env python3
#   -*- coding: utf8 -*- #
#
#
#   Copyright (C) by p.oseidon@datec.at, 1998 - 2016
#
#   This file is part of pandora.
#
#   pandora is free software: you can redistribute it and/or modify
#   it under the terms of the GNU General Public License as published by
#   the Free Software Foundation, either version 3 of the License, or
#   (at your option) any later version.
#
#   pandora is distributed in the hope that it will be useful,
#   but WITHOUT ANY WARRANTY; without even the implied warranty of
#   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#   GNU General Public License for more details.
#
#   You should have received a copy of the GNU General Public License
#   along with pandora. If not, see <http://www.gnu.org/licenses/>.


import logging; _Logger = logging.getLogger()
import time
import unittest

from tau4.timing import Timer2

from tau4.data import pandora as p


class _TESTCASE__0(unittest.TestCase):

    def _on_reference_changed_( self, publisherchannel):
        return

    def _on_reference_clipped_( self, publisherchannel):
        return


    def test__simple( self):
        """
        """
        print()

        ##  A box w/o an id
        #
        p_controlerror = p.Box( data=0.0, label="Control Error")
        p_controlerror.data( 42)
        self.assertTrue( float is type( p_controlerror.data()))
        self.assertAlmostEqual( 42.0, p_controlerror.data())

        p_controlerror.data( p_controlerror.data() + 1)
        self.assertTrue( float is type( p_controlerror.data()))
        self.assertAlmostEqual( 43.0, p_controlerror.data())

        return

    def test__monitoring( self):
        """
        """
        print()

        p_reference = p.Box( \
            label="Reference value", data=0.0, dim="U/min",
            plugins=(p.plugins.Monitor( callable=self._on_reference_changed_), )
            )

        p_reference.data( 42)
        return

    def test__filtering( self):
        """
        """
        print()

        class Filter(p.plugins.Plugin):

            def __init__( self):
                super().__init__( id="filter")

            def process_data( self, data_new):
                data_old = self.data()
                data = data_new
                return data

        p_reference = p.Box( \
            label="Reference value", data=0.0, dim="U/min",
            plugins=(Filter(), p.plugins.Monitor( callable=self._on_reference_changed_), )
            )
        return

    def test__monitoring_and_clipping( self):
        """
        """
        print()

        p.Boxes().pathname_ini( "./boxes.ini")

        p_vel = p.Box( \
            id="velocity", label="v", data=42.0, dim="m/s",
            plugins=(\
                p.plugins.Clipper4Numbers( callable=self._on_reference_clipped_, min=-1, max=1),
                p.plugins.Monitor( callable=self._on_reference_changed_)
            )
        )

        p_vel.data( -p_vel.data())
        self.assertAlmostEqual( -1, p_vel.data())
        return

    def test__performance_when_creating( self):
        """
        """
        print()

        n = 10000
        with Timer2( "Creating most simple box") as t:
            for _ in range( n):
                p_aux = p.Box( data=0.0)
        print( t.results( timedivider=n))

        n = 10000
        with Timer2( "Creating monitored box") as t:
            for _ in range( n):
                p_aux = p.Box( data=0.0, plugins=(p.plugins.Monitor( callable=self._on_reference_changed_), ))

        print( t.results( timedivider=n))
        return

    def test__performance_when_writing( self):
        """
        """
        print()

        n = 10000
        p_aux = p.Box( data=0.0)
        with Timer2( "Writing to most simple box") as t:
            for _ in range( n):
                p_aux.data( 0.0)

        print( t.results( timedivider=n))

        i = n = 10000
        p_aux = p.Box( data=0.0, plugins=(p.plugins.Monitor( callable=self._on_reference_changed_), ))
        with Timer2( "Writing to monitored box") as t:
            while i:
                p_aux.data( 0.0)

                i -= 1

        print( t.results( timedivider=n))

        i = n = 10000
        p_aux = p.Box( \
            data=0.0,
            plugins=( \
                p.plugins.Monitor( callable=self._on_reference_changed_), p.plugins.Clipper4Numbers( min=-1, max=1, 
callable=self._on_reference_clipped_)
            )
        )
        with Timer2( "Writing to monitored clipping box") as t:
            while i:
                p_aux.data( i)

                i -= 1

        print( t.results( timedivider=n))
        return

    def test__performance_when_reading( self):
        """
        """
        print()

        n = 10000
        p_aux = p.Box( data=0.0)
        with Timer2( "Reading from most simple box") as t:
            for _ in range( n):
                p_aux.data()

        print( t.results( timedivider=n))

        n = 10000
        p_aux = p.Box( data=0.0, plugins=(p.plugins.Monitor( callable=self._on_reference_changed_), ))
        with Timer2( "Reading from monitored box") as t:
            for _ in range( n):
                p_aux.data()

        print( t.results( timedivider=n))
        return

    def test__persistence( self):
        """
        """
        print()

        p.Boxes().pathname_ini( "./boxes.ini")

        p_vel = p.Box( id="velocity", label="v", data=42.0, dim="m/s")
        p.Boxes().store_box( p_vel)
        p_vel.data( 43)
        self.assertAlmostEqual( 43, p_vel.data())
        self.assertIs( float, type( p_vel.data()))

        p.Boxes().restore_box( p_vel)
        self.assertAlmostEqual( 42, p_vel.data())
        self.assertIs( float, type( p_vel.data()))

        try:
            p.Boxes().box( "velocity")
            self.assertFalse( "Trapped!")

        except KeyError:
            p.Boxes().add( p_vel)
        self.assertIs( p_vel, p.Boxes().box( "velocity"))
        return


_Testsuite = unittest.makeSuite( _TESTCASE__0)


class _TESTCASE__(unittest.TestCase):

    def test( self):
        """
        """
        print()
        return


_Testsuite.addTest( unittest.makeSuite( _TESTCASE__))


def _lab_():
    return


def _Test_():
    unittest.TextTestRunner( verbosity=2).run( _Testsuite)


if __name__ == '__main__':
    _Test_()
    _lab_()
    input( u"Press any key to exit...")
```

Appendix: Document Index

Index