fdi

Release v0.18c

Maohai Huang

Nov 22, 2022

CONTENTS:

FDI, known as SPDC before, is written in Python for integrating different types of data, and letting the integrated product take care of inter-platform compatibility, serialisation, persistence, and data object referencing that enables lazy-loading.

CHAPTER

ONE

FEATURES

With FDI one can pack data of different format into **modular** Data Products, together with annotation (description and units) and meta data (data about data). One can make arrays or tables of Products using basic data structures such as sets, sequences (Python `list`), mappings (Python `dict`), or custom-made classes. FDI accomodates nested and highly complex structures.

Access APIs of the components of 'FDIs' are convenient, making it easier for **scripting and data mining** directly 'on FDIs'.

All levels of FDI Products and their component (datasets or metadata) are portable (**serializable**) in human-friendly standard format (JSON implemented), allowing machine data processors on different platforms to parse, access internal components, or re-construct "an FDI". Even a human with a web browser can understand the data.

The `toString()` method of major containers classes outputs nicely formated text representation of complex data to help converting FDI to ASCII.

Most FDI Products and components implement **event sender and listener interfaces**, allowing **scalable data-driven** processing pipelines and visualizers of live data to be constructed.

FDI storage 'pools' (file based and memory based) are provided as references for 1) queryable data **storage** and, 2) for all persistent data to be referenced to with **URNs** (Universal Resource Names).

FDI provides *Context* type of product so that references of other products can become components of a Context, enabling **encapsulation of rich, deep, sophisticated, and accessible contextual data**, yet remain light weight.

For data processors, an HTML **server** with **RESTful APIs** is implemented (named Processing Node Server, PNS) to interface data processing modules. PNS is especially suitable for **Docker containers** in pipelines mixing **legacy software** or software of incompatible environments to form an integral data processing pipeline.

This package attempts to meet scientific observation and data processing requirements, and is inspired by data models of, and designs APIs as compatible as possible with, European Space Agency's Interactive Analysis package of Herschel Common Science System (written in Java, and in Jython for scripting).

CHAPTER

TWO

FDI PYTHON PACKAGES

- The base data model is defined in package *dataset*.
- Persistent data access, referencing, querying, and Universal Resource Names are defined in package *pal*.
- A reference REST API server designed to communicate with a data processing docker using the data model is in package *pns*.

2.1 Install FDI

2.1.1 for developers

```
FDIINSTDIR=/tmp   # change this to your installation dir
cd $FDIINSTDIR
git clone ssh://git@mercury.bao.ac.cn:9005/mh/fdi.git
cd fdi
pip3 install -e .
```

2.1.2 for users

```
cd /tmp
git clone http://mercury.bao.ac.cn:9006/mh/fdi.git
cd fdi
pip3 install -e .
```

to install in /tmp.

2.2 dataset: Model for Data Container

2.2.1 Rationale

A data processing task produces data products that are meant to be shared by other people. When someone receives a data 'product' besides datasets s/he woud expect explanation informaion associated with the product.

Many people tend to store data with no note of meaning attached to them. Without attach meaning of the collection of numbers, it is difficult for other people to fully understand or use the data. It could be difficult for even the data producer to recall the exact meaning of the numbers after a while.

This package implements a data product modeled after Herschel Common Software System (v15) products, taking other requirements of scientific observation and data processing into account. The APIs are kept as compatible with HCSS (written in Java, and in Jython for scripting) as possible.

2.2.2 Definitions

Product

A product has

- zero or more datasets: defining well described data entities (say images, tables, spectra etc…).
- history of this product: how was this data created,
- accompanying meta data – required information such as who created this product, what does the data reflect (say instrument) and so on; possible additional meta data specific to that particular product type.

Dataset

Three types of datasets are implemented to store potentially any data as a dataset. Like a product, all datasets may have meta data, with the distinction that the meta data of a dataset is related to that particular dataset only.

array dataset
a dataset containing array data (say a data vector, array, cube etc…) and may have a unit.

table dataset
a dataset containing a collection of columns. Each column contains array data (say a data vector, array, cube etc…) and may have a unit. All columns have the same number of rows. Together they make up the table.

composite dataset
a dataset containing a collection of datasets. This allows arbitrary complex structures, as a child dataset within a composite dataset may be a composite dataset itself and so on…

Metadata and Parameters

Meta data
data about data. Defined as a collection of parameters.

Parameter

named scalar variables.
This package provides the following parameter types:

- _Parameter_: Contains value (classes whitelisted in ParameterTypes)
- _NumericParameter_: Contains a number with a unit.

Apart from the value of a parameter you can ask it for its description and -if it is a numeric parameter- for its unit as well.

History

The history is a lightweight mechanism to record the origin of this product or changes made to this product. Lightweight means, that the Product data itself does not records changes, but external parties can attach additional information to the Product which reflects the changes.

The sole purpose of the history interface of a Product is to allow notably pipeline tasks (as defined by the pipeline framework) to record what they have done to generate and/or modify a Product.

Serializability

In order to transfer data across the network between heterogeneous nodes data needs to be serializable. JSON format is used considering to transfer serialized data for its wide adoption, availability of tools, ease to use with Python, and simplicity.

2.2.3 run tests

In the install directory:

```
make test
```

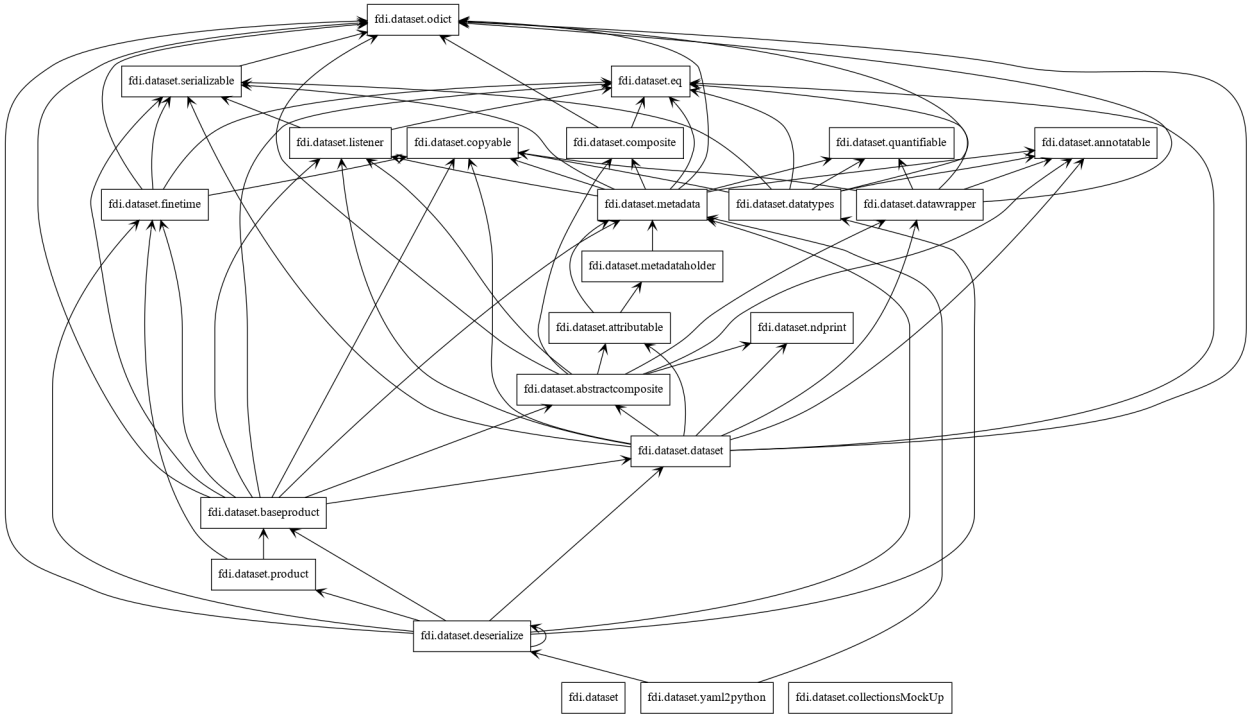
You can only test sub-package `dataset`, `pal`, `pns` or pns server self-test only, by changing `test` above to `test1`, `test2`, `test3`, `test4`, respectively. To pass command-line arguments to `pytest` do

```
make test T='-k Bas'
```

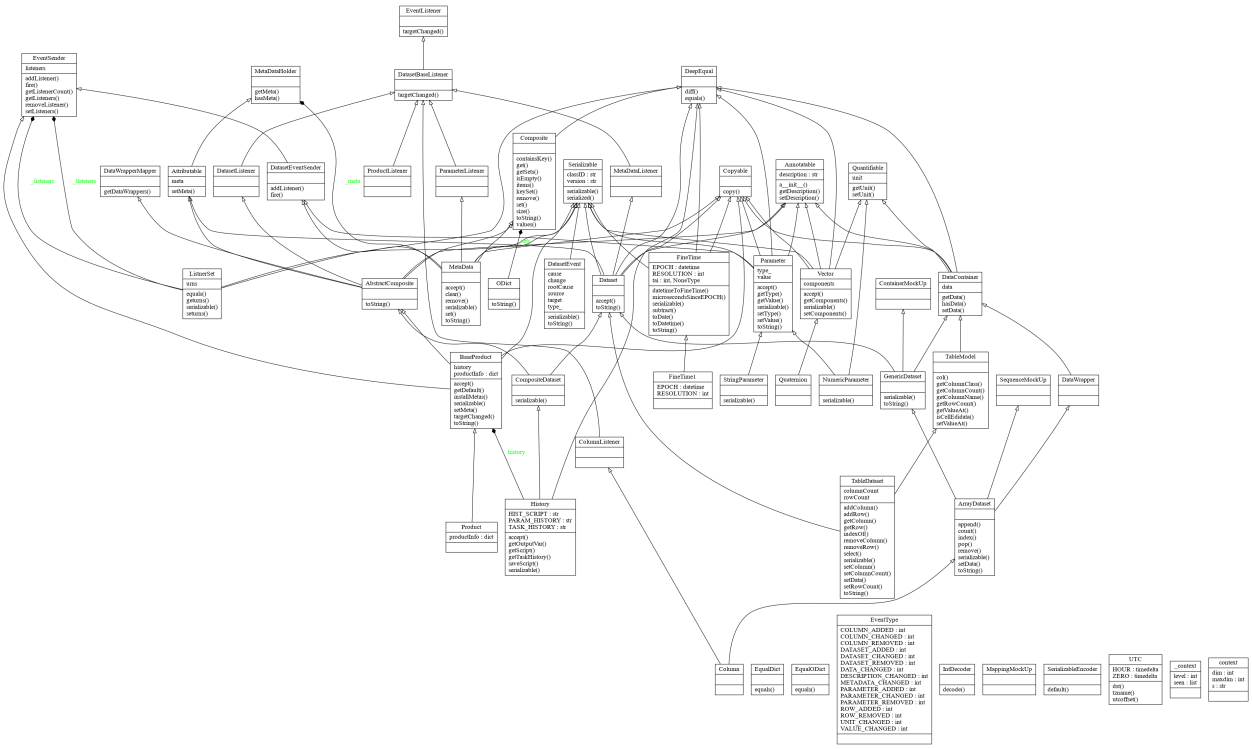
to test `BaseProduct` in sub-package `dataset`.

2.2.4 Design

Packages


fdi.dataset.odict
fdi.dataset.serializable
fdi.dataset.eq
fdi.dataset.listener
fdi.dataset.copyable
fdi.dataset.composite
fdi.dataset.quantifiable
fdi.dataset.annotatable
fdi.dataset.finetime
fdi.dataset.metadata
fdi.dataset.datatypes
fdi.dataset.datawrapper
fdi.dataset.metadataholder
fdi.dataset.attributable
fdi.dataset.ndprint
fdi.dataset.abstractcomposite
fdi.dataset.dataset
fdi.dataset.baseproduct
fdi.dataset.product
fdi.dataset.deserialize
fdi.dataset
fdi.dataset.yaml2python
fdi.dataset.collectionsMockUp

Classes


EventListener
targetChanged()
EventSender
listeners
addListener()
fire()
getListenerCount()
getListeners()
removeListener()
setListeners()
MetaDataHolder
getMeta()
hasMeta()
DatasetBaseListener
targetChanged()
DeepEqual
diff()
equals()
Composite
containsKey()
get()
getMeta()
getProp()
isEmpty()
keySet()
remove()
set()
size()
toString()
values()
Serializable
classID : str
version : str
serializable()
serialized()
MetaDataListener
Copyable
copy()
Annotatable
description : str
__init__()
getDescription()
setDescription()
Quantifiable
unit
getUnit()
setUnit()
DataWrapperMapper
getDataWrapper()
Attributable
meta
setMeta()
DatasetListener
DatasetEventSender
addListener()
fire()
ProductListener
ParameterListener
_listeners
_listeners
_meta
ListenerSet
uris
equals()
getUris()
serializable()
setUris()
AbstractComposite
toString()
MetaData
accept()
clear()
remove()
serializable()
set()
toString()
ODict
toString()
DatasetEvent
cause
change
rootCause
source
target
type_
serializable()
toString()
Dataset
accept()
toString()
FineTime
EPOCH : datetime
RESOLUTION : int
tai : int, NoneType
datetimeToFineTime()
microsecondsSinceEPOCH()
serializable()
subtract()
toDate()
toDatetime()
toString()
Parameter
type_
value
accept()
getTypes()
getValue()
serializable()
setType()
setValue()
toString()
Vector
components
accept()
getComponents()
serializable()
setComponents()
ContainerMockUp
DataContainer
data
getData()
hasData()
setData()
BaseProduct
history
productInfo : dict
accept()
getDefault()
initialMeta()
serializable()
setMeta()
targetChanged()
toString()
CompositeDataset
serializable()
FineTime1
EPOCH : datetime
RESOLUTION : int
StringParameter
serializable()
Quaternion
NumericParameter
serializable()
GenericDataset
serializable()
toString()
TableModel
ocs()
getColumnClass()
getColumnCount()
getColumnName()
getRowCount()
getValueAt()
isCellEditable()
setValueAt()
SequenceMockUp
DataWrapper
history
ColumnListener
TableDataset
columnCount
rowCount
addColumn()
addRow()
getColumn()
getRow()
indexOf()
removeColumn()
removeRow()
select()
serializable()
setColumn()
setColumnCount()
setData()
setRowCount()
toString()
ArrayDataset
append()
count()
index()
pop()
remove()
serializable()
setData()
toString()
Product
productInfo : dict
History
HIST_SCRIPT : str
PARAM_HISTORY : str
TASK_HISTORY : str
accept()
getOutputVar()
getScript()
getTaskHistory()
saveScript()
serializable()
EventType
COLUMN_ADDED : int
COLUMN_CHANGED : int
COLUMN_REMOVED : int
DATASET_ADDED : int
DATASET_CHANGED : int
DATASET_REMOVED : int
DATA_CHANGED : int
MEMBER_CHANGED : int
METADATA_CHANGED : int
PARAMETER_ADDED : int
PARAMETER_CHANGED : int
PARAMETER_REMOVED : int
ROW_ADDED : int
ROW_REMOVED : int
UNIT_CHANGED : int
VALUE_CHANGED : int
Column
EqualDict
equals()
EqualODict
equals()
InitDecoder
decode()
MappingMockUp
SerializableEncoder
default()
UTC
HOUR : timedelta
ZERO : timedelta
dst()
tzname()
utcoffset()
__context
level : int
seen : list
context
dtm : int
numdays : int
s : str

2.3 pal: Product Access Layer

Product Access Layer allows data stored logical "pools" to be accessed with light weight product refernces by data processers, data storage, and data consumers. A data product can include a context built with references of relevant data. A `ProductStorage` interface is provided to handle saving/retrieving/querying data in registered pools.

2.3.1 Rationale

In a data processing pipeline or network of processing nodes, data products are generated within a context which may include input data, reference data, and auxiliary data of many kind. It is often needed to have relevant context recorded with a product. However the context could have a large size so including their actual data as metadata of the product is often impractical.

Once FDI data are generated they can have a reference through which they can be accessed. The size of such references are typically less than a few hundred bytes, like a URL. In the product context only data references are recorded.

This package provides `MapContext`, `ProductRef`, `Urn`, `ProductStorage`, `ProductPool`, and `Query` classes (simplified but mostly API-compatible with Herschel Common Science System v15.0) for the storing, retrieving, tagging, and context creating of data product modeled in the dataset package.

2.3.2 Definitions

URN

The Universial Resource Name (URN) string has this format:

```
urn:poolname:resourceclass:serialnumber
```

where

resourceclass
full class name of the resource (product)

poolname
scheme + `://` + place + directory

scheme
`file`, `mem`, `http` … etc

place
`192.168.5.6:8080`, `c:`, an empty string … etc

directory

A label for the pool that is by default used as the full path where the pool is stored. ProductPool.transformpath() can used to change the directory here to other meaning.

- for `file` scheme: `/` + name + `/` + name + … + `/` + name
- for `mem` scheme: `/` + name + `/`

serialnumber
internal index. str(int).

ProductRef

This class not only holds the URN of the product it references to, but also records who (the _parents_) are keeping this reference.

Context and MapContext

Context is a Product that holds a set of `productRef` s that accessible by keys. The keys are strings for MapContext which usually maps names to product references.

ProductStorage

A centralized access place for saving/loading/querying/deleting data organized in conceptual pools. One gets a ProductRef when saving data.

ProductPool

An place where products can be saved, with a reference for the saved product generated. The product can be retrieved with the reference. Pools based on different media or networking mechanism can be implemented. Multiple pools can be registered in a ProductStorage front-end where users can do the saving, loading, querying etc. so that the pools are collectively form a larger logical storage.

Query

One can make queries to a ProductStorage and get back a list of references to products that satisfy search chriteria. Queries can be constructed using Python predicate expressions about a product and its metadata, or a function that returns True or False.

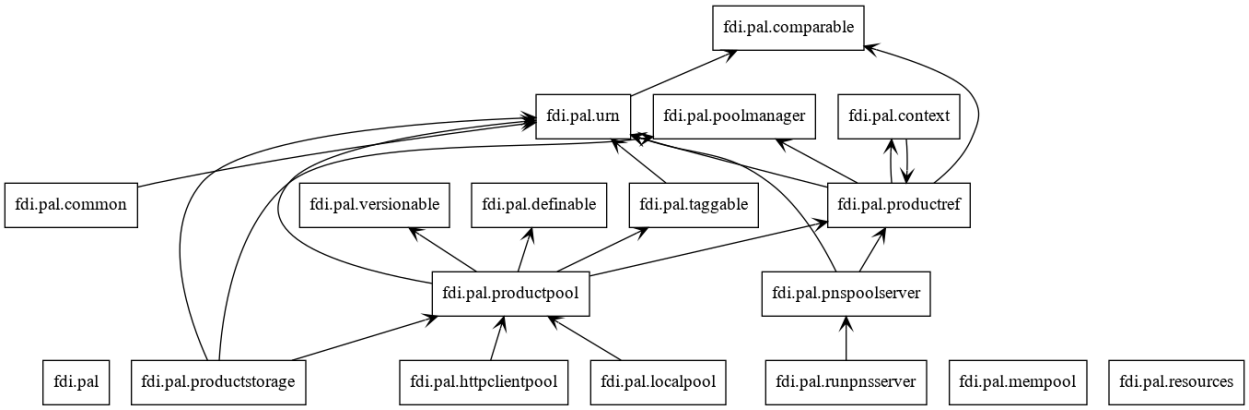
2.3.3 run tests

in the same directory:

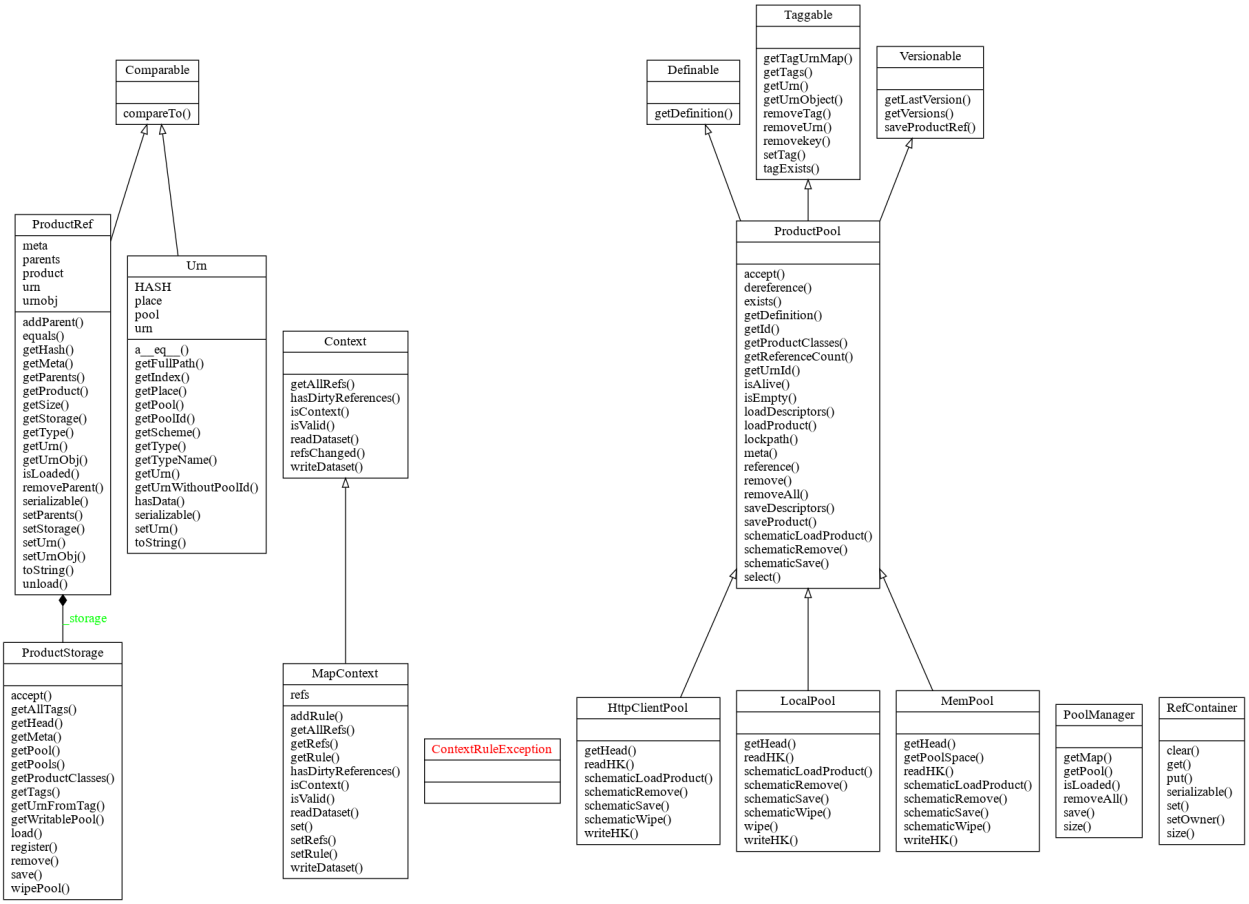
```
make test2
```

2.3.4 Design

Packages


fdi.pal.comparable
fdi.pal.urn
fdi.pal.poolmanager
fdi.pal.context
fdi.pal.common
fdi.pal.versionable
fdi.pal.definable
fdi.pal.taggable
fdi.pal.productref
fdi.pal.productpool
fdi.pal.pnspoolserver
fdi.pal
fdi.pal.productstorage
fdi.pal.httpclientpool
fdi.pal.localpool
fdi.pal.runpnsserver
fdi.pal.mempool
fdi.pal.resources

Classes


Comparable
compareTo()
Taggable
getTagUrnMap()
getTags()
getUrn()
getUrnObject()
removeTag()
removeUrn()
removekey()
setTag()
tagExists()
Definable
getDefinition()
Versionable
getLastVersion()
getVersions()
saveProductRef()
ProductRef
meta
parents
product
urn
urnobj
addParent()
equals()
getHash()
getMeta()
getParents()
getProduct()
getSize()
getStorage()
getType()
getUrn()
getUrnObj()
isLoaded()
removeParent()
serializable()
setParents()
setStorage()
setUrn()
setUrnObj()
toString()
unload()
Urn
HASH
place
pool
urn
a__eq__()
getFullPath()
getIndex()
getPlace()
getPool()
getPoolId()
getScheme()
getType()
getTypeName()
getUrn()
getUrnWithoutPoolId()
hasData()
serializable()
setUrn()
toString()
Context
getAllRefs()
hasDirtyReferences()
isContext()
isValid()
readDataset()
refsChanged()
writeDataset()
ProductPool
accept()
dereference()
exists()
getDefinition()
getId()
getProductClasses()
getReferenceCount()
getUrnId()
isAlive()
isEmpty()
loadDescriptors()
loadProduct()
lockpath()
meta()
reference()
remove()
removeAll()
saveDescriptors()
saveProduct()
schematicLoadProduct()
schematicRemove()
schematicSave()
select()
_storage
ProductStorage
accept()
getAllTags()
getHead()
getMeta()
getPool()
getPools()
getProductClasses()
getTags()
getUrnFromTag()
getWritablePool()
load()
register()
remove()
save()
wipePool()
MapContext
refs
addRule()
getAllRefs()
getRefs()
getRule()
hasDirtyReferences()
isContext()
isValid()
readDataset()
set()
setRefs()
setRule()
writeDataset()
ContextRuleException
HttpClientPool
getHead()
readHK()
schematicLoadProduct()
schematicRemove()
schematicSave()
schematicWipe()
writeHK()
LocalPool
getHead()
readHK()
schematicLoadProduct()
schematicRemove()
schematicSave()
schematicWipe()
wipe()
writeHK()
MemPool
getHead()
getPoolSpace()
readHK()
schematicLoadProduct()
schematicRemove()
schematicSave()
schematicWipe()
writeHK()
PoolManager
getMap()
getPool()
isLoaded()
removeAll()
save()
size()
RefContainer
clear()
get()
put()
serializable()
set()
setOwner()
size()

2.4 pns: Processing Node Server

2.4.1 Rationale

Many data processing pipelines need to run software that only runs on a specific combination of OS type, version, language, and library. These software could be impractical to replace or modify but need to be run side-by-side with software of incompatible environments/formats to form an integral data processing pipeline, each software being a "node" to perform a processing task. Docker containers are often the perfect solution to run software with incompatible dependencies.

PNS installed on a Docker container or a normal server allows such processing tasks to run in the PNS memory space, in a daemon process, or as an OS process receiving input and delivering output through a 'delivery man' protocol.

This Web API Server for a data processing pipeline/network node provides interfaces to configure the data processing task software (PTS) in a processing node, to make a run request, to deliver necessary input data, and to read results, all via web APIs.

The following commands are run from the fdi directory from installation.

2.4.2 Basic Configuration

When running Flask server, the host IP is `0.0.0.0` and port number `5000` by default. They are configurable in `pnsconfig.py`. Default configuration can be overridden by `~/.config/pnslocal.py`. Copy `pnsconfig.py` to `~/.config/pnslocal.py`

```
cp fdi/pns/pnsconfig.py ~/.config/pnslocal.py
```

and customize `~/.config/pnslocal.py`.

When in developement mode, set `dev` to `True` (`dev = True` or `dev = 1`) to run local server. The `serveruser` should be the name of the user of web server, usually your username if you run `make runserver`. This is the default if `dev` is true.

For production deployment the `dev` should be set false. Set `serveruser` depending which web server (e.g. `'apache'`).

The `ptsuser` is usually the user required by the processing software. It is set to `serveruser` by default. `ptsuser` must have write previlige to read and write `inputdir` and `outputdir`, which are owned by `serveruser` with mode `o0775`.

On the server side (or on your computer which can be both the server and the client) edit `Makefile` by changing the value of varible `PNSDIR` in `Makefile` the pnshome directory if you do not want the default (`~/pns`).

Then run the deployment command

```
make installpns
```

to create the pns home directory and copy the demo PTS script set.

2.4.3 Run the FLASK Server

Edit ~/.config/pnslocal.py if needed. Then

```
python3 fdi/pns/runflaskserver.py --username=<username> --password=<password> [--ip=
↪<host ip>] [--port=<port>]
```

Contents in [], like [--ip=<host ip>] [--port=<port>] above, are optional.

<> means you need to substitute with actual information (for example --port=<port> becomes --port=5000).

Or you can run

```
python3 fdi/pns/runflaskserver.py -u <username> -p <password> [-i <host ip>] [-o <port>]
```

in debugging mode:

```
python3 fdi/pns/runflaskserver.py --username=foo --password=bar -v
```

or just

```
make runserver
```

to use the defaults.

Do not run debugging mode for production use.

The username and password are used when making run requests.

2.4.4 Test and Verify Installation

To run all tests in one go:

```
make test3 [T='-u <username> -p <password> [-i <host ip>] [-o <port>] [options]']
```

Tests can be done step-by-step to pin-point possible problems:

2.4.5 1. Server Unit Test

Run this on the server host to verify that internal essential functions of the server work with current configuration. This runs without needing starting the server:

```
make test4
```

2.4.6 2. Local Flask Server Functional Tests

In ~/.config/pnslocal.py (see above for installation and customization), set dev=True and make sure the IP is local (0.0.0.0 or 127.0.0.1). Start the server fresh in one terminal (see above) and in another terminal (on the server host) run the following:

2a: test GET initPTS script to see if reading the init script back works:

```
make test3 T='getinit'
```

2b: test PUT initialization test:

```
make test3 T='-k putinittest'
```

2c1: If the test passes, you can Run all tests in one go:

```
make test3
```

2c2: Or keep on individual tests…

test POST In-server processing

```
make test3 T='-k _post'
```

test POST PTS processing

```
make test3 T='-k _run'
```

test DELETE Clean-up the server by removing the input and output dirs

```
make test3 T='-k deleteclean'
```

Now is a good time to …

2.4.7 3. Get public access APIs and information

Suppose the server address and port are `127.0.0.1` and `5000`, respectively:

Run the Flask server in a terminal (see above) and open this in a browser. The up-to-date URL is displayed in the server stating message:

http://127.0.0.1:5000/v0.6/

An online API documentation page similar to below is shown.


```
{
"APIs": {
"DELETE": [
{
"URL": "http://127.0.0.1:5000/v0.6/clean",
"description": " Removing traces of past runnings the Processing Task Software.\n    "
}
],
"GET": [
{
"URL": "http://127.0.0.1:5000/v0.6/init",
"description": "the initPTS file"
},
{
"URL": "http://127.0.0.1:5000/v0.6/config",
"description": "the configPTS file"
},
{
"URL": "http://127.0.0.1:5000/v0.6/run",
"description": "the file running PTS"
},
{
```


(continues on next page)

(continued from previous page)


```
"URL": "http://127.0.0.1:5000/v0.6/clean",
"description": "the cleanPTS file"
},
{
"URL": "http://127.0.0.1:5000/v0.6/input",
"description": " returns names and contents of all files in the dir, 'None' if dir not␣
↪existing. "
},
{
"URL": "http://127.0.0.1:5000/v0.6/output",
"description": " returns names and contents of all files in the dir, 'None' if dir not␣
↪existing. "
},
{
"URL": "http://127.0.0.1:5000/v0.6/pnsconfig",
"description": "PNS configuration"
}
],
"POST": [
{
"URL": "http://127.0.0.1:5000/v0.6/calc",
"description": " generates result product directly using data on PNS.\n    "
},
{
"URL": "http://127.0.0.1:5000/v0.6/testcalc",
"description": " generate post test product.\n    put the 1st input (see maketestdata in␣
↪test_all.py)\n    parameter to metadata\n    and 2nd to the product's dataset\n    "
},
{
"URL": "http://127.0.0.1:5000/v0.6/echo",
"description": "Echo"
},
{
"URL": "http://127.0.0.1:5000/v0.6/run",
"description": " Generates a product by running script defined in the config under 'run'.
↪ Execution on the server host is in the pnshome directory and run result and status␣
↪are returned.\n    "
},
{
"URL": "http://127.0.0.1:5000/v0.6/testrun",
"description": "  Run 'runPTS' for testing, and as an example.\n    "
}
],
"PUT": [
{
"URL": "http://127.0.0.1:5000/v0.6/init",
"description": " Initialize the Processing Task Software by running the init script␣
↪defined in the config. Execution on the server host is in the pnshome directory and␣
↪run result and status are returned. If input/output directories cannot be created with␣
↪serveruser as owner, Error401 will be given.\n    "
},
{
```


(continues on next page)

(continued from previous page)


```
"URL": "http://127.0.0.1:5000/v0.6/config",
"description": " Configure the Processing Task Software by running the config script.␣
↪Ref init PTS.\n    "
},
{
"URL": "http://127.0.0.1:5000/v0.6/pnsconf",
"description": " Configure the PNS itself by replacing the pnsconfig var\n    "
},
{
"URL": "http://127.0.0.1:5000/v0.6/inittest",
"description": "     Renames the 'init' 'config' 'run' 'clean' scripts to \"*.save\" and␣
↪points it to the '.ori' scripts.\n    "
}
]
},
"timestamp": 1566130779.0208821
}
```


Continue with tests…

2.4.8 4. Run tests from a remote client

Install pns on a remote host, configure IP and port, then run the tests above. This proves that the server and the client have connection and fire wall configured correctly.

2.4.9 Run the local tests with Apache

Set dev=False in ~/.config/pnslocal.py (see above) and set the IP and port. Suppose the server is on CentOS. Edit pns/resources/pns.conf according to local setup, then

```
cp pns/resources/pns.conf /etc/httpd/conf.d
systemctl restart httpd
systemctl status http -l
```

then run the above with correct IP and port (edit ~/.config/pnslocal.py or specifying in command line). Start the server and run all the tests:

```
make test3
```

2.4.10 PTS Configuration

To run a PTS shell script instead of the 'hello' demo, change the `run` parameter in the config file, e.g. to run the script named `runPTS.vvpp`

```
run=[join(h, 'runPTS.vvpp'), ''],
```

restart the server. run

```
make test4
```

2.4.11 PTS API

TBW

2.4.12 Return on Common Errors

400

```
{'error': 'Bad request.', 'timestamp': ts}
```

401

```
{'error': 'Unauthorized. Authentication needed to modify.', 'timestamp': ts}
```

404

```
{'error': 'Not found.', 'timestamp': ts}
```

409

```
{'error': 'Conflict. Updating.', 'timestamp': ts}
```

Resources

TBW

2.5 fdi Quick Start

Contents:

The following demostrates important dataset and pal functionalities. It was made by running `fdi/resources/example.py` with command `elpy-shell-send-group-and-step [c-c c-y c-g]` in `emacs`.

You can copy the code from code blocks by clicking the `copy` icon on the top-right, with he prompts and results removed.

```
>>> # import these first.
... import copy
... import getpass
... import os
... from datetime import datetime
... import logging
... from fdi.dataset.product import Product
... from fdi.dataset.metadata import Parameter, NumericParameter, MetaData
... from fdi.dataset.finetime import FineTime1, utcobj
... from fdi.dataset.dataset import ArrayDataset, TableDataset, Column
... from fdi.pal.context import Context, MapContext
... from fdi.pal.productref import ProductRef
... from fdi.pal.query import MetaQuery
... from fdi.pal.poolmanager import PoolManager, DEFAULT_MEM_POOL
... from fdi.pal.productstorage import ProductStorage
```

2.5.1 dataset

ArrayDataset

```
>>> # Creation
... a1 = [1, 4.4, 5.4E3, -22, 0xa2]      # a 1D array of data
... v = ArrayDataset(data=a1, unit='ev', description='5 elements')
... v
ArrayDataset{ [1, 4.4, 5400.0, -22, 162] <ev>, description = "5 elements", meta =␣
↪MetaData{[], listeners = []}}
```

```
>>> # data access
... v[2]
5400.0
```

```
>>> v.unit
'ev'
```

```
>>> v.unit = 'm'
... v.unit
'm'
```

```
>>> # iteration
... for m in v:
...     print(m)
1
4.4
5400.0
```

(continues on next page)

(continued from previous page)

```
-22
162
```

```
>>> [m**3 for m in v if m > 0 and m < 40]
[1, 85.18400000000003]
```

```
>>> # slice
... v[1:3]
[4.4, 5400.0]
```

```
>>> v[2:-1]
[5400.0, -22]
```

```
>>> v.data = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
... v[0:2]
[[1, 2, 3], [4, 5, 6]]
```

```
>>> # Run this to see a demo of the ``toString()`` function::
... # make a 4-D array: a list of 2 lists of 3 lists of 4 lists of 5 elements.
... s = [[[[i + j + k + l for i in range(5)] for j in range(4)]
...       for k in range(3)] for l in range(2)]
... x = ArrayDataset(data=s)
... print(x.toString())
```

```
# ArrayDataset
# description = "UNKNOWN"
# meta = MetaData{[], listeners = []}
# unit = "None"
# data =

0 1 2 3
1 2 3 4
2 3 4 5
3 4 5 6
4 5 6 7


1 2 3 4
2 3 4 5
3 4 5 6
4 5 6 7
5 6 7 8


2 3 4 5
3 4 5 6
4 5 6 7
5 6 7 8
6 7 8 9
```

(continues on next page)

(continued from previous page)

```
#=== dimension 4

1 2 3 4
2 3 4 5
3 4 5 6
4 5 6 7
5 6 7 8


2 3 4 5
3 4 5 6
4 5 6 7
5 6 7 8
6 7 8 9


3 4 5 6
4 5 6 7
5 6 7 8
6 7 8 9
7 8 9 10


#=== dimension 4
```

TableDataset

```
>>> # Creation
... a1 = [dict(name='col1', unit='eV', column=[1, 4.4, 5.4E3]),
...       dict(name='col2', unit='cnt', column=[0, 43.2, 2E3])
...       ]
... v = TableDataset(data=a1)
... v
TableDataset{ description = "UNKNOWN", meta = MetaData{[], listeners = []}, data = "OD{
↪'col1':Column{ [1, 4.4, 5400.0] <eV>, description = "UNKNOWN", meta = MetaData{[],␣
↪listeners = []}}, 'col2':Column{ [0, 43.2, 2000.0] <cnt>, description = "UNKNOWN",␣
↪meta = MetaData{[], listeners = []}}}"}
```

```
>>> # many other ways to create a TableDataset
... v3 = TableDataset(data=[('col1', [1, 4.4, 5.4E3], 'eV'),
...                         ('col2', [0, 43.2, 2E3], 'cnt')])
... v == v3
True
```

```
>>> # quick and dirty. data are list of lists without names or units
... a5 = [[1, 4.4, 5.4E3], [0, 43.2, 2E3]]
... v5 = TableDataset(data=a5)
... print(v5.toString())
```

```
# TableDataset
# description = "UNKNOWN"
# meta = MetaData{[], listeners = []}
# data =

# col1 col2
# None None
1 0
4.4 43.2
5400.0 2000.0
```

```
>>> # access
... # get names of all column
... v5.data.keys()
odict_keys(['col1', 'col2'])
```

```
>>> # get a list of all columns' data
... [c.data for c in v5.data.values()]   # == a5
[[1, 4.4, 5400.0], [0, 43.2, 2000.0]]
```

```
>>> # get column by name
... c_1 = v5['col1']
... c_1
Column{ [1, 4.4, 5400.0] <None>, description = "UNKNOWN", meta = MetaData{[], listeners␣
↪= []}}
```

```
>>> #  indexOf
... v5.indexOf('col1')  # == u.indexOf(c_1)
0
```

```
>>> v5.indexOf(c_1)
0
```

```
>>> # get a cell
... v5['col2'][1]    # 43.2
43.2
```

```
>>> # set cell value
... v5['col2'][1] = 123
... v5['col2'][1]    # 123
123
```

```
>>> v5.setValueAt(aValue=42, rowIndex=1, columnIndex=1)
... v5.getValueAt(rowIndex=1, columnIndex=1)    # 42
42
```

```
>>> # unit access
... v3['col1'].unit  # == 'eV'
'eV'
```

```
>>> # add, set, and replace columns and rows
... # column set / get
... u = TableDataset()
... c1 = Column([1, 4], 'sec')
... u.addColumn('col3', c1)
... u.columnCount        # 1
1
```

```
>>> # for non-existing names set is addColum.
... c2 = Column([2, 3], 'eu')
... u['col4'] = c2
... u['col4'][0]    # 2
2
```

```
>>> u.columnCount        # 2
2
```

```
>>> # replace column for existing names
... c3 = Column([5, 7], 'j')
... u['col4'] = c3
... u['col4'][0]    # c3.data[0]
5
```

```
>>> # addRow
... u.rowCount    # 2
2
```

```
>>> cc = copy.deepcopy(c1)
... c33, c44 = 3.3, 4.4
... cc.append(c33)
... u.addRow({'col4': c44, 'col3': c33})
... u.rowCount    # 3
3
```

```
>>> u['col3']    # cc
Column{ [1, 4, 3.3] <sec>, description = "UNKNOWN", meta = MetaData{[], listeners = []}}
```

```
>>> # removeRow
... u.removeRow(u.rowCount - 1)    # [c33, c44]
[3.3, 4.4]
```

```
>>> u.rowCount    # 2
2
```

```
>>> # syntax ``in``
... [c for c in u]  # list of column names ['col1', 'col2']
['col3', 'col4']
```

```
>>> # run this to see ``toString()``
... ELECTRON_VOLTS = 'eV'
```

(continues on next page)

(continued from previous page)

```
... SECONDS = 'sec'
... t = [x * 1.0 for x in range(10)]
... e = [2 * x + 100 for x in t]
... # creating a table dataset to hold the quantified data
... x = TableDataset(description="Example table")
... x["Time"] = Column(data=t, unit=SECONDS)
... x["Energy"] = Column(data=e, unit=ELECTRON_VOLTS)
... print(x.toString())
```

```
# TableDataset
# description = "Example table"
# meta = MetaData{[], listeners = []}
# data =

# Time Energy
# sec eV
0.0 100.0
1.0 102.0
2.0 104.0
3.0 106.0
4.0 108.0
5.0 110.0
6.0 112.0
7.0 114.0
8.0 116.0
9.0 118.0
```

Parameter

```
>>> # Creation
... # standard way -- with keyword arguments
... a1 = 'a test parameter'
... a2 = 300
... a3 = 'integer'
... v = Parameter(description=a1, value=a2, type_=a3)
... v.description   # == a1
'a test parameter'
```

```
>>> v.value   # == a2
300
```

```
>>> v.type_   # == a3
'integer'
```

```
>>> # with no argument
... v = Parameter()
... v.description   # == 'UNKNOWN# inherited from Anotatable
'UNKNOWN'
```

```
>>> v.value   # is None
```

```
>>> v.type_   # == ''
''
```

```
>>> # make a blank one then set attributes
... v = Parameter(description=a1)
... v.description   # == a1
'a test parameter'
```

```
>>> v.value    # is None
```

```
>>> v.type_   # == ''
''
```

```
>>> v.setValue(a2)
... v.setType(a3)
... v.description   # == a1
'a test parameter'
```

```
>>> v.value   # == a2
300
```

```
>>> v.type_   # == a3
'integer'
```

```
>>> # test equivalence of v.setXxxx(a) and v.xxx = a
... a1 = 'test score'
... a2 = 98
... v = Parameter()
... v.description = a1
... v.value = a2
... v.description   # == a1
'test score'
```

```
>>> v.value   # == a2
98
```

```
>>> # test equals
... b1 = ''.join(a1)  # make a new string copy
... b2 = a2 + 0  # make a copy
... v1 = Parameter(description=b1, value=b2)
... v.equals(v1)
True
```

```
>>> v == v1
True
```

```
>>> v1.value = -4
... v.equals(v1)   # False
False
```

```
>>> v != v1  # True
True
```

Metadata

```
>>> # Creation
... a1 = 'age'
... a2 = NumericParameter(description='since 2000',
...                       value=20, unit='year', type_='integer')
... v = MetaData()
... v.set(a1, a2)
... v.get(a1)   # == a2
NumericParameter{ 20 (year) <integer>, "since 2000"}
```

```
>>> # add more parameter
... a3 = 'Bob'
... v.set(name='name', newParameter=Parameter(a3))
... v.get('name').value   # == a3
'Bob'
```

```
>>> # access parameters in metadata
... v = MetaData()
... # a more readable way to set a parameter
... v[a1] = a2  # DRM doc case
... # a more readable way to get a parameter
... v[a1]   # == a2
NumericParameter{ 20 (year) <integer>, "since 2000"}
```

```
>>> v.get(a1)   # == a2
NumericParameter{ 20 (year) <integer>, "since 2000"}
```

```
>>> v['date'] = Parameter(description='take off at',
...                       value=FineTime1.datetimeToFineTime(datetime.now(tz=utcobj)))
... # names of all parameters
... [n for n in v]   # == [a1, 'date']
['age', 'date']
```

```
>>> print(v.toString())
MetaData{[age = NumericParameter{ 20 (year) <integer>, "since 2000"}, date = Parameter{␣
↪108120221290 <integer>, "take off at"}, ], listeners = []}
```

```
>>> # remove parameter
... v.remove(a1)  # inherited from composite
... print(v.size())  # == 1
1
```

Product

```
>>> # Creation:
... x = Product(description="product example with several datasets",
...             instrument="Crystal-Ball", modelName="Mk II")
... x.meta['description'].value  # == "product example with several datasets"
'product example with several datasets'
```

```
>>> x.instrument  # == "Crystal-Ball"
'Crystal-Ball'
```

```
>>> # ways to add datasets
... i0 = 6
... i1 = [[1, 2, 3], [4, 5, i0], [7, 8, 9]]
... i2 = 'ev'                  # unit
... i3 = 'image1'     # description
... image = ArrayDataset(data=i1, unit=i2, description=i3)
... x["RawImage"] = image
... x["RawImage"].data  # == [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

```
>>> # no unit or description. different syntax but same function as above
... x.set('QualityImage', ArrayDataset(
...     [[0.1, 0.5, 0.7], [4e3, 6e7, 8], [-2, 0, 3.1]]))
... x["QualityImage"].unit  # is None
```

```
>>> # add a tabledataset
... s1 = [('col1', [1, 4.4, 5.4E3], 'eV'),
...       ('col2', [0, 43.2, 2E3], 'cnt')]
... x["Spectrum"] = TableDataset(data=s1)
... print(x["Spectrum"].toString())
```

```
# TableDataset
# description = "UNKNOWN"
# meta = MetaData{[], listeners = []}
# data =

# col1 col2
# eV cnt
1 0
4.4 43.2
5400.0 2000.0
```

```
>>> # mandatory properties are also in metadata
... # test mandatory BaseProduct properties that are also metadata
... x.creator = ""
... a0 = "Me, myself and I"
... x.creator = a0
... x.creator   # == a0
'Me, myself and I'
```

```
>>> # metada by the same name is also set
... x.meta["creator"].value   # == a0
'Me, myself and I'
```

```
>>> # change the metadata
... a1 = "or else"
... x.meta["creator"] = Parameter(a1)
... # metada changed
... x.meta["creator"].value   # == a1
'or else'
```

```
>>> # so did the property
... x.creator   # == a1
'or else'
```

```
>>> # Demo ``toString()`` function. The result should be ::
... print(x.toString())
```

```
# Product
# description = "product example with several datasets"
# meta = MetaData{[description = Parameter{ product example with several datasets
↪<string>, "Description of this product"}, type = Parameter{ Product <string>, "Product␣
↪Type identification. Fully qualified Python class name or CARD."}, creator = Parameter
↪{ or else <string>, "UNKNOWN"}, creationDate = Parameter{ 2017-01-01T00:00:00.000000␣
↪TAI(0) <finetime>, "Creation date of this product"}, rootCause = Parameter{ UNKOWN
↪<string>, "Reason of this run of pipeline."}, schema = Parameter{ 0.3 <string>,
↪"Version of product schema"}, startDate = Parameter{ 2017-01-01T00:00:00.000000 TAI(0)
↪<finetime>, "Nominal start time  of this product."}, endDate = Parameter{ 2017-01-
↪01T00:00:00.000000 TAI(0) <finetime>, "Nominal end time  of this product."},␣
↪instrument = Parameter{ Crystal-Ball <string>, "Instrument that generated data of this␣
↪product"}, modelName = Parameter{ Mk II <string>, "Model name of the instrument of␣
↪this product"}, mission = Parameter{ _AGS <string>, "Name of the mission."}, ],␣
↪listeners = []}
# History
# description = "UNKNOWN"
# meta = MetaData{[], listeners = []}
# data =

# data =


# [ RawImage ]
# ArrayDataset
# description = "image1"
# meta = MetaData{[], listeners = []}
# unit = "ev"
# data =

1 4 7
2 5 8
3 6 9
```

(continues on next page)

(continued from previous page)

```
# [ QualityImage ]
# ArrayDataset
# description = "UNKNOWN"
# meta = MetaData{[], listeners = []}
# unit = "None"
# data =

0.1 4000.0 -2
0.5 60000000.0 0
0.7 8 3.1


# [ Spectrum ]
# TableDataset
# description = "UNKNOWN"
# meta = MetaData{[], listeners = []}
# data =

# col1 col2
# eV cnt
1 0
4.4 43.2
5400.0 2000.0
```

2.5.2 pal

Store a Product in a Pool and Get a Reference Back

Create a product and a productStorage with a pool registered

```
>>> # disable debugging messages
... logger = logging.getLogger('')
... logger.setLevel(logging.WARNING)
```

```
>>> # a pool for demonstration will be create here
... demopoolpath = '/tmp/demopool_' + getpass.getuser()
... demopool = 'file://' + demopoolpath
... # clean possible data left from previous runs
... os.system('rm -rf ' + demopoolpath)
... PoolManager.getPool(DEFAULT_MEM_POOL).removeAll()
... PoolManager.removeAll()
```

```
>>> # create a prooduct and save it to a pool
... x = Product(description='in store')
... # add a tabledataset
... s1 = [('energy', [1, 4.4, 5.6], 'eV'), ('freq', [0, 43.2, 2E3], 'Hz')]
... x["Spectrum"] = TableDataset(data=s1)
... # create a product store
... pstore = ProductStorage(pool=demopool)
```

(continues on next page)

(continued from previous page)

```
... pstore
ProductStorage { pool= OD{'file:///tmp/demopool_mh':LocalPool { pool= file:///tmp/
↪demopool_mh }} }
```

```
>>> # save the product and get a reference
... prodref = pstore.save(x)
... print(prodref)
ProductRef{ ProductURN=urn:file:///tmp/demopool_mh:fdi.dataset.product.Product:0,␣
↪meta=MetaData{[description = Parameter{ in store <string>, "Description of this product
↪"}, type = Parameter{ Product <string>, "Product Type identificat...}
```

```
>>> # get the urn string
... urn = prodref.urn
... print(urn)    # urn:file:///tmp/demopool_mh:fdi.dataset.product.Product:0
urn:file:///tmp/demopool_mh:fdi.dataset.product.Product:0
```

```
>>> newp = ProductRef(urn).product
... # the new and the old one are equal
... print(newp == x)   # == True
True
```

Context: a Product with References

```
>>> # the reference can be stored in another product of Context class
... p1 = Product(description='p1')
... p2 = Product(description='p2')
... # create an empty mapcontext that can carry references with name labels
... map1 = MapContext(description='product with refs 1')
... # A ProductRef created from a lone product will use a mempool
... pref1 = ProductRef(p1)
... pref1
ProductRef{ ProductURN=urn:mem:///default:fdi.dataset.product.Product:0, meta=None}
```

```
>>> # A productStorage with a pool on disk
... pref2 = pstore.save(p2)
... pref2
ProductRef{ ProductURN=urn:file:///tmp/demopool_mh:fdi.dataset.product.Product:1,␣
↪meta=MetaData{[description = Parameter{ p2 <string>, "Description of this p...
```

```
>>> # how many prodrefs do we have? (do not use len() due to classID, version)
... map1['refs'].size()   # == 0
0
```

```
>>> len(pref1.parents)   # == 0
0
```

```
>>> len(pref2.parents)   # == 0
0
```

```
>>> # add a ref to the contex. every ref has a name in mapcontext
... map1['refs']['spam'] = pref1
... # add the second one
... map1['refs']['egg'] = pref2
... # how many prodrefs do we have? (do not use len() due to classID, version)
... map1['refs'].size()   # == 2
2
```

```
>>> len(pref2.parents)   # == 1
1
```

```
>>> pref2.parents[0] == map1
True
```

```
>>> pref1.parents[0] == map1
True
```

```
>>> # remove a ref
... del map1['refs']['spam']
... # how many prodrefs do we have? (do not use len() due to classID, version)
... map1.refs.size()   # == 1
1
```

```
>>> len(pref1.parents)   # == 0
0
```

```
>>> # add ref2 to another map
... map2 = MapContext(description='product with refs 2')
... map2.refs['also2'] = pref2
... map2['refs'].size()   # == 1
1
```

```
>>> # two parents
... len(pref2.parents)   # == 2
2
```

```
>>> pref2.parents[1] == map2
True
```

Query a ProductStorage

```
>>> # clean possible data left from previous runs
... defaultpoolpath = '/tmp/pool_' + getpass.getuser()
... newpoolpath = '/tmp/newpool_' + getpass.getuser()
... os.system('rm -rf ' + defaultpoolpath)
... os.system('rm -rf ' + newpoolpath)
... PoolManager.getPool(DEFAULT_MEM_POOL).removeAll()
... PoolManager.removeAll()
... # make a productStorage
```

(continues on next page)

(continued from previous page)

```
... defaultpool = 'file://'+defaultpoolpath
... pstore = ProductStorage(defaultpool)
... # make another
... newpoolname = 'file://' + newpoolpath
... pstore2 = ProductStorage(newpoolname)
```

```
>>> # add some products to both storages
... n = 7
... for i in range(n):
...     a0, a1, a2 = 'desc %d' % i, 'fatman %d' % (i*4), 5000+i
...     if i < 3:
...         x = Product(description=a0, instrument=a1)
...         x.meta['extra'] = Parameter(value=a2)
...     elif i < 5:
...         x = Context(description=a0, instrument=a1)
...         x.meta['extra'] = Parameter(value=a2)
... ...
...         x = MapContext(description=a0, instrument=a1)
...         x.meta['extra'] = Parameter(value=a2)
...         x.meta['time'] = Parameter(value=FineTime1(a2))
...     if i < 4:
...         r = pstore.save(x)
...     else:
...         r = pstore2.save(x)
...     print(r.urn)
... # Two pools, 7 products
... # [P P P C] [C M M]
urn:file:///tmp/pool_mh:fdi.dataset.product.Product:0
urn:file:///tmp/pool_mh:fdi.dataset.product.Product:1
urn:file:///tmp/pool_mh:fdi.dataset.product.Product:2
urn:file:///tmp/pool_mh:fdi.pal.context.Context:0
urn:file:///tmp/newpool_mh:fdi.pal.context.Context:0
urn:file:///tmp/newpool_mh:fdi.pal.context.MapContext:0
urn:file:///tmp/newpool_mh:fdi.pal.context.MapContext:1
```

```
>>> # register the new pool above to the  1st productStorage
... pstore.register(newpoolname)
... len(pstore.getPools())   # == 2
2
```

```
>>> # make a query on product metadata, which is the variable 'm'
... # in the query expression, i.e. ``m = product.meta; ...``
... # But '5000 < m["extra"]' does not work. see tests/test.py.
... q = MetaQuery(Product, 'm["extra"] > 5001 and m["extra"] <= 5005')
... # search all pools registered on pstore
... res = pstore.select(q)
... # [2,3,4,5]
... len(res)   # == 4
... [r.product.description for r in res]
['desc 2', 'desc 3', 'desc 4', 'desc 5']
```

```
>>> def t(m):
...     # query is a function
...     import re
...     return re.match('.*n.1.*', m['instrument'].value)
```

```
>>> q = MetaQuery(Product, t)
... res = pstore.select(q)
... # [3,4]
... [r.product.instrument for r in res]
['fatman 12', 'fatman 16']
```

```
>>> # same as above but query is on the product. this is slow.
... q = AbstractQuery(Product, 'p', '"n 1" in p.instrument')
... res = pstore.select(q)
... # [3,4]
... [r.product.instrument for r in res]
['fatman 12', 'fatman 16']
```

```
>>>
```

2.5.3 pns

See the installation and testing sections of the pns page.

CHAPTER

THREE

API DOCUMENT

3.1 API Reference

3.1.1 fdi.dataset package

Submodules

fdi.dataset.abstractcomposite module

class fdi.dataset.abstractcomposite.**AbstractComposite**(***kwds*)

Bases: *Attributable*, *Annotatable*, *Composite*, *DataWrapperMapper*, *DatasetListener*

an annotatable and attributable subclass of Composite.

toString(*matprint=None*, *trans=True*, *beforedata=''*)

fdi.dataset.annotatable module

class fdi.dataset.annotatable.**Annotatable**(*description='UNKNOWN'*, ***kwds*)

Bases: object

An Annotatable object is an object that can give a human readable description of itself.

a__init__(**args*, *description='UNKNOWN'*, ***kwds*)

getDescription()

gets the description of this Annotatable object.

setDescription(*newDescription*)

sets the description of this Annotatable object.

fdi.dataset.attributable module

class fdi.dataset.attributable.**Attributable**(*meta=None*, ***kwds*)

Bases: *MetaDataHolder*

An Attributable object is an object that has the notion of meta data.

property meta

setMeta(*newMetadata*)

Replaces the current MetaData with specified argument. mh: Product will override this to add listener whenevery meta is replaced

fdi.dataset.baseproduct module

class fdi.dataset.baseproduct.**BaseProduct**(*description='UNKOWN', type_='BaseProduct', creator='UNKOWN', creationDate=2017-01-01T00:00:00.000000 TAI(0), rootCause='UNKOWN', schema='0.3', **kwds*)

Bases: *AbstractComposite*, *Copyable*, *Serializable*, *EventSender*

A BaseProduct is a generic result that can be passed on between (standalone) processes.

In general a Product contains zero or more datasets, history, optional metadata as well as some required metadata fields. Its intent is that it can fully describe itself; this includes the way how this product was achieved (its history). As it is the result of a process, it should be able to save to and restore from an Archive device.

Many times a Product may contain a single dataset and for this purpose the first dataset entry can be accessed by the getDefault() method. Note that the datasets may be a composite of datasets by themselves.

mh: Built-in Attributes in productInfo['metadata'] can be accessed with e.g. p.creator or p.meta['description'].value: p.creator='foo' assert p.creatur=='foo' assert p.meta['creator']=='foo' p.meta['creator']=Parameter('bar') assert p.meta['creator']==Parameter('bar')

accept(*visitor*)

Hook for adding functionality to meta data object through visitor pattern.

getDefault()

Convenience method that returns the first dataset belonging to this product.

installMetas(*mtbi*)

put parameters in group in product metadata, and updates productInfo. values in mtbi override those default ones in group.

productInfo = {'metadata': {'creationDate': {'data_type': 'finetime', 'default': '0', 'description': 'Creation date of this product', 'fits_keyword': 'DATE', 'unit': 'None'}, 'creator': {'data_type': 'string', 'default': 'UNKOWN', 'description': 'Generator of this product. Example name of institute, organization, person, software, special algorithm etc.', 'fits_keyword': 'CREATOR', 'unit': 'None'}, 'description': {'data_type': 'string', 'default': 'UNKOWN', 'description': 'Description of this product', 'fits_keyword': 'DESCRIPT', 'unit': 'None'}, 'rootCause': {'data_type': 'string', 'default': 'UNKOWN', 'description': 'Reason of this run of pipeline.', 'fits_keyword': 'ROOTCAUS', 'unit': 'None'}, 'schema': {'data_type': 'string', 'default': '0.3', 'description': 'Version of product schema', 'fits_keyword': 'SCHEMA', 'unit': 'None'}, 'type': {'data_type': 'string', 'default': 'BaseProduct', 'description': 'Product Type identification. Fully qualified Python class name or CARD.', 'fits_keyword': 'TYPE', 'unit': 'None'}}}

serializable()

Can be encoded with serializableEncoder

setMeta(*newMetadata*)

Replaces the current MetaData with specified argument. mh: Product will override this to add listener whenevery meta is replaced

targetChanged(*event*)

Informs that an event has happened in a target of the specified type.

toString(*matprint=None*, *trans=True*, *beforedata=''*)

like AbstractComposite but with history

class fdi.dataset.baseproduct.**History**(*other=None*, ***kwds*)

Bases: *CompositeDataset*, *DeepEqual*

Public interface to the history dataset. Contains the main methods for retrieving a script and copying the history.

accept(*visitor*)

Hook for adding functionality to meta data object through visitor pattern.

getOutputVar()

Returns the final output variable of the history script.

getScript()

Creates a Jython script from the history.

getTaskHistory()

Returns a human readable formatted history tree.

saveScript(*file*)

Saves the history script to a file.

serializable()

Can be encoded with serializableEncoder

fdi.dataset.baseproduct.**addMandatoryProductAttrs**(*cls*)

mh: Add MPAs to a class so that although they are metadata, they can be accessed by for example, product-foo.creator. dynamic properties see https://stackoverflow.com/a/2584050 https://stackoverflow.com/a/1355444

fdi.dataset.classes module

class fdi.dataset.classes.**Classes**

Bases: object

A dictionary of class names and their class objects that are allowed to be deserialized. A fdi package built-in dictionary (in the format of locals() output) is kept internally. Users who need add more deserializable class can for example: def myclasses():

from foo.bar import Baz ...

Classes.classes = myClasses

class fdi.dataset.classes.**Classes_meta**(**args*, ***kwds*)

Bases: type

metaclass for 'classproperty'. # https://stackoverflow.com/a/1800999

makePackageClasses()

The set of fdi package-wide deserializable classes is maintained by hand. Do nothing if the classes mapping is already made so repeated calls will not cost lots more time.

property mapping

Returns the dictionary of classes allowed for deserialization, including the fdi built-ins and user added classes.

updateMapping(*c={}*)

Updates classes mapping. Make the package mapping if it has not been made.

fdi.dataset.classes.**logger = <Logger fdi.dataset.classes (WARNING)>**

Note: this has to be in a different file where other interface classes are defined to avoid circular dependency (such as , Serializable.

fdi.dataset.collectionsMockUp module

class fdi.dataset.collectionsMockUp.**ContainerMockUp**

Bases: object

class fdi.dataset.collectionsMockUp.**MappingMockUp**

Bases: object

class fdi.dataset.collectionsMockUp.**SequenceMockUp**

Bases: object

fdi.dataset.composite module

class fdi.dataset.composite.**Composite**(***kwds*)

Bases: *DeepEqual*

A container of named Datasets.

This container can hold zero or more datasets, each of them stored against a unique name. The order of adding datasets to this Composite is important, that is: the keySet() method will return a set of labels of the datasets in the sequence as they were added. Note that replacing a dataset with the same name, will keep the order.

containsKey(*name*)

Returns true if this map contains a mapping for the specified name.

get(*name*)

Returns the dataset to which this composite maps the specified name. If the attitute does not exist, return None. This is an OrderedDict behavior.

getSets()

Provide access to the Map < String, Dataset > . mh: api from CompositeDataset

isEmpty()

Returns true if this map contains no key - value mappings.

items()

Enable pairs = [(v, k) for (k, v) in d.items()].

keySet()

Returns a set view of the keys contained in this composite.

remove(*name*)

Removes the mapping for this name from this composite. mh: returns None if name is None or item does not exist.

set(*name*, *dataset*)

Associates the specified dataset with the specified key in this map(optional operation). If the map previously contained a mapping for this key, the old dataset is replaced by the specified dataset. this composite does not permit null or empty keys.

size()

Returns the number of key - value mappings in this map.

toString(*matprint=None*, *trans=True*)

values()

Enable pairs = zip(d.values(), d.keys())

fdi.dataset.copyable module

class fdi.dataset.copyable.**Copyable**

Bases: object

Interface for objects that can make a copy of themselves.

copy()

Makes a deep copy of itself.

fdi.dataset.dataset module

class fdi.dataset.dataset.**ArrayDataset**(**args*, ***kwds*)

Bases: *DataWrapper*, *GenericDataset*, Sequence

Special dataset that contains a single Array Data object. mh: If omit the parameter names during instanciation, e.g. ArrayDataset(a, b, c), the assumed order is data, unit, description. mh: contains a sequence which provides methods count(), index(), remove(), reverse(). A mutable sequence would also need append(), extend(), insert(), pop() and sort().

append(**args*, ***kwargs*)

appends to data.

count(**args*, ***kwargs*)

returns size.

index(**args*, ***kwargs*)

returns the index of a value.

pop(**args*, ***kwargs*)

revomes and returns value

remove(**args*, ***kwargs*)

removes value at first occurrence.

serializable()

Can be encoded with serializableEncoder

setData(*data*)

toString(*matprint=None*, *trans=True*)

matprint: an external matrix print function trans: print 2D matrix transposed. default is True.

class fdi.dataset.dataset.**Column**(**args*, ***kwds*)

Bases: *ArrayDataset*, *ColumnListener*

A Column is a the vertical cut of a table for which all cells have the same signature. It contains raw ArrayData, and optionally a description and unit. example:

```
table = TableDataset()
table.addColumn("Energy",Column(data=[1,2,3,4],description="desc",unit='eV'))
```

class fdi.dataset.dataset.**CompositeDataset**(***kwds*)

Bases: *AbstractComposite*, *Dataset*

An CompositeDataset is a Dataset that contains zero or more named Datasets. It allows to build arbitrary complex dataset structures.

It also defines the iteration ordering of its children, which is the order in which the children were inserted into this dataset.

serializable()

Can be encoded with serializableEncoder

class fdi.dataset.dataset.**Dataset**(***kwds*)

Bases: *Attributable*, *Annotatable*, *Copyable*, *Serializable*, *DeepEqual*, *MetaDataListener*

Attributable and annotatable information data container that can be be part of a Product.

developer notes: The intent is that developers do not derive from this interface

directly. Instead, they should inherit from one of the generic datasets that this package provides:

mh: GenericDataset, ArrayDataset. TableDataset or CompositeDataset.

accept(*visitor*)

Hook for adding functionality to object through visitor pattern.

toString()

class fdi.dataset.dataset.**GenericDataset**(***kwds*)

Bases: *Dataset*, *DataContainer*, Container

mh: Contains one data item.

serializable()

Can be encoded with serializableEncoder

toString(*matprint=None*, *trans=True*)

matprint: an external matrix print function trans: print 2D matrix transposed. default is True.

class fdi.dataset.dataset.**TableDataset**(***kwds*)

Bases: *Dataset*, *TableModel*

Special dataset that contains a single Array Data object. A TableDataset is a tabular collection of Columns. It is optimized to work on array data.. The column-wise approach is convenient in many cases. For example, one has an event list, and each algorithm is adding a new field to the events (i.e. a new column, for example a quality mask).

Although mechanisms are provided to grow the table row-wise, one should use these with care especially in performance driven environments as this orthogonal approach (adding rows rather than adding columns) is expensive.

General Note:

For reasons of flexibility, memory consumption and performance, this class is not checking whether all columns are of the same length: this is the responsibility of the user/developer. See also the library documentation for more information about this.

Note on column names:

If a column is added without specifying a name, the name ColumnX is created, where X denotes the index of that column. Column name duplicity is not allowed.

Developers:

See "Writing special datasets or products" at the developer's documentation also.

Please see also this selection example.

addColumn(*name*, *column*)

Adds the specified column to this table, and attaches a name to it. If the name is null, a dummy name is created, such that it can be accessed by getColumn(str).

Duplicated column names are not allowed.

Parameters: name - column name. column - column to be added.

addRow(*row*)

Adds the specified map as a new row to this table. mh: row is a dict with names as keys

property columnCount

getColumn(*key*)

return colmn if given string as name or int as index. returns name if given column.

getRow(*rowIndex*)

Returns a list containing the objects located at a particular row.

indexOf(*key*)

Returns the index of specified column; if the key is a Column, it looks for equal references (same column objects), not for equal values. If the key is a string, Returns the index of specified Column name. mh: Or else returns the key itself.

removeColumn(*key*)

removeRow(*rowIndex*)

Removes a row with specified index from this table. mh: returns removed row.

property rowCount

select(*selection*)

Select a number of rows from this table dataset and return a new TableDataset object containing only the selected rows.

serializable()

Can be encoded with serializableEncoder

setColumn(*key*, *value*)

Replaces a column in this table with specified name to specified column if key exists, or else add a new coolumn.

setColumnCount(*columnCount*)

cannot do this.

setData(*data*)

sets name-column pairs from [{ 'name':str,'column':Column}] or {str:Column} or [[num]] or [(str, [], 'str')] form of data, Existing data will be discarded except when the provided data is a list of lists, where existing column names and units will remain but data replaced, and extra data items will form new columns named 'col[index]' (index counting from 1) with unit None.

setRowCount(*rowCount*)

cannot do this.

toString(*matprint=None*, *trans=True*)

class fdi.dataset.dataset.**TableModel**(***kwds*)

Bases: *DataContainer*

to interrogate a tabular data model

col(*columIndex*)

returns a tuple of (name, column) at the column index.

getColumnClass(*columnIndex*)

Returns the most specific superclass for all the cell values in the column.

getColumnCount()

Returns the number of columns in the model.

getColumnName(*columnIndex*)

Returns the name of the column at columnIndex.

getRowCount()

Returns the number of rows in the model.

getValueAt(*rowIndex*, *columnIndex*)

Returns the value for the cell at columnIndex and rowIndex.

isCellEdidata(*rowIndex*, *columnIndex*)

Returns true if the cell at rowIndex and columnIndex is edidata.

setValueAt(*aValue*, *rowIndex*, *columnIndex*)

Sets the value in the cell at columnIndex and rowIndex to aValue.

fdi.dataset.datatypes module

class fdi.dataset.datatypes.**Quaternion**(*components=[0, 0, 0, 0]*, ***kwds*)

Bases: *Vector*

Quaternion with a unit.

class fdi.dataset.datatypes.**Vector**(*components=[0, 0, 0]*, *description='UNKNOWN'*, *unit=''*, ***kwds*)

Bases: *Annotatable*, *Copyable*, *DeepEqual*, *Quantifiable*, *Serializable*

Three dimensional vector with a unit.

accept(*visitor*)

Adds functionality to classes of this components.

property components

for property getter

getComponents()

Returns the actual components that is allowed for the components of this vector.

serializable()

Can be encoded with serializableEncoder

setComponents(*components*)

Replaces the current components of this vector.

fdi.dataset.datawrapper module

class fdi.dataset.datawrapper.**DataContainer**(*data=None*, ***kwds*)

Bases: *Annotatable*, *Quantifiable*, *Copyable*, *DeepEqual*

A DataContainer is a composite of data and description. mh: note that There is no metadata. Implemented partly from AbstractDataWrapper.

property data

getData()

Returns the data in this dw

hasData()

Returns whether this data wrapper has data.

setData(*data*)

Replaces the current DataData with specified argument. mh: subclasses can override this to add listener whenevery data is replaced

class fdi.dataset.datawrapper.**DataWrapper**(*data=None*, ***kwds*)

Bases: *DataContainer*

A DataWrapper is a composite of data, unit and description. mh: note that all data are in the same unit. There is no metadata. Implemented from AbstractDataWrapper.

class fdi.dataset.datawrapper.**DataWrapperMapper**

Bases: `object`

Object holding a map of data wrappers.

getDataWrappers()

Gives the data wrappers, mapped by name.

fdi.dataset.deserialize module

class fdi.dataset.deserialize.**IntDecoder**(**, *object_hook=None*, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *strict=True*, *object_pairs_hook=None*)

Bases: `JSONDecoder`

adapted from https://stackoverflow.com/questions/45068797/how-to-convert-string-int-json-into-real-int-with-json-loads modified to also convert keys in dictionaries.

decode(*s*)

Return the Python representation of `s` (a `str` instance containing a JSON document).

`fdi.dataset.deserialize.constructSerializableClassID(obj, lgb=None, debug=False)`

mh: reconstruct object from the output of jason.loads(). Recursively goes into nested class instances that are not encoded by default by JSONEncoder, instantiate and fill in variables. Objects to be deserialized must have their classes loaded. ClassID cannot have module names in it (e.g. dataset.Product) or locals()[classname] or globals()[classname] will not work. See alternative in https://stackoverflow.com/questions/452969/does-python-have-an-equivalent-to-java-class-forname

`fdi.dataset.deserialize.deserializeClassID(js, lgb=None, debug=False, usedict=False)`

Loads classes with ClassID from the results of serializeClassID.

if usedict is True dict insted of ODict will be used.

`fdi.dataset.deserialize.imakedesables()`

makes a class dictionary for instanciation.

`fdi.dataset.deserialize.lls(s, length=80)`

length-limited string

`fdi.dataset.deserialize.logger = <Logger fdi.dataset.deserialize (WARNING)>`

Note: this has to be in a different file where other interface classes are defined to avoid circular dependency (such as , Serializable.

fdi.dataset.eq module

`class fdi.dataset.eq.DeepEqual`

Bases: `object`

mh: Can compare key-val pairs of another object with self. False if compare with None or exceptions raised, e.g. obj does not have items()

`diff(obj, seenlist)`

recursively compare components of list and dict. until meeting equality. seenlist: a list of classes that has been seen. will not descend in to them.

`equals(obj)`

`class fdi.dataset.eq.EqualDict`

Bases: `object`

mh: Can compare key-val pairs of another object with self. False if compare with None or exceptions raised, e.g. obj does not have items()

`equals(obj)`

`class fdi.dataset.eq.EqualODict`

Bases: `object`

mh: Can compare order and key-val pairs of another object with self. False if compare with None or exceptions raised, e.g. obj does not have items()

`equals(obj)`

`fdi.dataset.eq.deepcmp(obj1, obj2, seenlist=None, verbose=False)`

Recursively descends into obj1's every member, which may be set, list, dict, ordereddict, (or ordereddict subclasses) and any objects with '__class__' attribute, compares every member found with its counterpart in obj2. Returns None if finds no difference, a string of explanation otherwise. Detects cyclic references.

fdi.dataset.finetime module

class fdi.dataset.finetime.**FineTime**(*date=None*, ***kwds*)

Bases: *Copyable*, *DeepEqual*, *Serializable*

Atomic time(SI seconds) elapsed since the TAI epoch of 1 January 1958 UT2. The resolution is one microsecond and the allowable range is: epoch + /-290, 000 years approximately.

This has the following advantages, compared with the standard class:

It has better resolution(microseconds) Time differences are correct across leap seconds It is immutable

EPOCH = datetime.datetime(1958, 1, 1, 0, 0, tzinfo=<fdi.dataset.finetime.UTC object>)

RESOLUTION = 1000000

classmethod datetimeToFineTime(*dtm*)

Return given Python Datetime as a FineTime.

equals(*obj*)

can compare TAI directly

microsecondsSinceEPOCH()

Return the rounded integer number of microseconds since the epoch: 1 Jan 1958.

serializable()

Can be encoded with serializableEncoder

subtract(*time*)

Subract the specified time and return the difference in microseconds.

toDate()

Return this time as a Python Datetime.

classmethod toDatetime(*tai*)

Return given FineTime as a Python Datetime.

toString()

Returns a String representation of this object. prints like 2019 - 02 - 17T12: 43: 04.577000 TAI

class fdi.dataset.finetime.**FineTime1**(*date=None*, ***kwds*)

Bases: *FineTime*

Same as FineTime but Epoch is 2017-1-1 0 UTC

EPOCH = datetime.datetime(2017, 1, 1, 0, 0, tzinfo=<fdi.dataset.finetime.UTC object>)

RESOLUTION = 1000

class fdi.dataset.finetime.**UTC**

Bases: tzinfo

https://docs.python.org/2.7/library/datetime.html?highlight=datetime#datetime.tzinfo

HOUR = datetime.timedelta(0, 3600)

ZERO = datetime.timedelta(0)

dst(*dt*)

datetime -> DST offset in minutes east of UTC.

tzname(*dt*)

datetime -> string name of time zone.

utcoffset(*dt*)

datetime -> timedelta showing offset from UTC, negative values indicating West of UTC

fdi.dataset.listener module

class fdi.dataset.listener.**ColumnListener**

Bases: *DatasetBaseListener*

Listener for events occuring in a Column.

Available types:

```
* DESCRIPTION_CHANGED
* UNIT_CHANGED
* DATA_CHANGED
```

Cause is always null.

class fdi.dataset.listener.**DatasetBaseListener**

Bases: *EventListener*

Generic interface for listeners that will listen to events happening on a target of a specific type. Java Warning: The listener must be a class field in order to make an object hard reference.

targetChanged(*event*)

Informs that an event has happened in a target of the specified type.

class fdi.dataset.listener.**DatasetEvent**(*source*, *target*, *type_*, *change*, *cause*, *rootCause*, ***kwds*)

Bases: *Serializable*

serializable()

Can be encoded with serializableEncoder

toString()

class fdi.dataset.listener.**DatasetEventSender**(***kwds*)

Bases: *EventSender*

addListener(*listener*, *cls=<class 'fdi.dataset.listener.DatasetBaseListener'>*)

Adds a listener to this.

fire(*event*)

class fdi.dataset.listener.**DatasetListener**

Bases: *DatasetBaseListener*

Listener for events occuring in MetaData. Available types:

```
* DESCRIPTION_CHANGED, METADATA_CHANGED (all datasets)
* DATA_CHANGED, UNIT_CHANGED (ArrayDataset)
* COLUMN_ADDED, COLUMN_REMOVED, COLUMN_CHANGED, ROW_ADDED, VALUE_CHANGED␣
```

(continues on next page)

(continued from previous page)

```
↪(TableDataset)
* DATASET_ADDED, DATASET_REMOVED, DATASET_CHANGED (CompositeDataset)
```

Possible causes:

```
* not null (METADATA_CHANGED, COLUMN_CHANGED, DATASET_CHANGED)
* null (rest)
```

Warning: The listener handler must be a class attribute in order to create an object hard reference. See DatasetBaseListener.

class fdi.dataset.listener.**EventListener**

Bases: object

Generic interface for listeners that will listen to anything

targetChanged(**args*, ***kwargs*)

Informs that an event has happened in a target of any type.

class fdi.dataset.listener.**EventSender**(***kwds*)

Bases: object

adapted from Peter Thatcher's https://stackoverflow.com/questions/1092531/event-system-in-python/1096614#1096614

addListener(*listener*, *cls=<class 'fdi.dataset.listener.EventListener'>*)

Adds a listener to this.

fire(**args*, ***kwargs*)

getListenerCount()

getListeners()

Returns the current Listeners.

property listeners

removeListener(*listener*)

Removes a listener from this.

setListeners(*listeners*)

Replaces the current Listeners with specified argument.

class fdi.dataset.listener.**EventType**

Bases: object

COLUMN_ADDED = 0

COLUMN_CHANGED = 1

COLUMN_REMOVED = 2

DATASET_ADDED = 4

DATASET_CHANGED = 5

DATASET_REMOVED = 6

DATA_CHANGED = 3

DESCRIPTION_CHANGED = 7

METADATA_CHANGED = 8

PARAMETER_ADDED = 9

PARAMETER_CHANGED = 10

PARAMETER_REMOVED = 11

ROW_ADDED = 12

ROW_REMOVED = 13

UNIT_CHANGED = 14

VALUE_CHANGED = 15

class fdi.dataset.listener.**ListnerSet**(***kwds*)

Bases: *Serializable*, *DeepEqual*, list

Mutable collection of Listeners of an EvenSender.

equals(*obj*)

compares with another one.

geturns(*remove=None*)

Returns the current urns.

serializable()

Can be encoded with serializableEncoder

seturns(*urns*)

Replaces the current urn with specified argument.

property urns

class fdi.dataset.listener.**MetaDataListener**

Bases: *DatasetBaseListener*

Listener for events occuring in MetaData. Available types:

```
* PARAMETER_ADDED
* PARAMETER_REMOVED
* PARAMETER_CHANGED
```

Possible causes: not null (for PARAMETER_CHANGED, if parameter internally changed) null (for PARAMETER_CHANGED, when set is called with a previous existing parameter, and rest)

Warning: The listener handler must be a class attribute in order to create an object hard reference. See DatasetBaseListener.

class fdi.dataset.listener.**ParameterListener**

Bases: *DatasetBaseListener*

Listener for events occuring in a Parameter. Available types:

```
* DESCRIPTION_CHANGED
* UNIT_CHANGED
* VALUE_CHANGED
```

Cause is always null.

Warning: The listener handler must be a class attribute in order to create an object hard reference. See DatasetBaseListener.

class fdi.dataset.listener.**ProductListener**

Bases: *DatasetBaseListener*

Listener for events occuring in Product. Available types:

```
* METADATA_CHANGED
* DATASET_ADDED
* DATASET_REMOVED
* DATASET_CHANGED
```

Possible causes:

```
* not null (METADATA_CHANGED, DATASET_CHANGED)
* null (METADATA_CHANGED, DATASET_REMOVED, DATASET_CHANGED)
```

Warning: The listener handler must be a class attribute in order to create an object hard reference. See DatasetBaseListener.

fdi.dataset.metadata module

class fdi.dataset.metadata.**MetaData**(*copy=None*, ***kwds*)

Bases: *Composite*, *Copyable*, *Serializable*, *ParameterListener*, *DatasetEventSender*

A container of named Parameters. A MetaData object can have one or more parameters, each of them stored against a unique name. The order of adding parameters to this container is important, that is: the keySet() method will return a set of labels of the parameters in the sequence as they were added. Note that replacing a parameter with the same name, will keep the order.

accept(*visitor*)

Hook for adding functionality to meta data object through visitor pattern.

clear()

Removes all the key - parameter mappings.

remove(*name*)

add eventhandling

serializable()

Can be encoded with serializableEncoder

set(*name*, *newParameter*)

Saves the parameter and add eventhandling. Raises TypeError if not given Parameter (sub) class object.

toString()

class fdi.dataset.metadata.**NumericParameter**(***kwds*)

Bases: *Parameter*, *Quantifiable*

has a number as the value and a unit.

serializable()

Can be encoded with serializableEncoder

class fdi.dataset.metadata.**Parameter**(*value=None*, *description='UNKNOWN'*, *type_=''*, ***kwds*)

Bases: *Annotatable*, *Copyable*, *DeepEqual*, *DatasetEventSender*, *Serializable*

Parameter is the interface for all named attributes in the MetaData container. It can have a value and a description.

accept(*visitor*)

Adds functionality to classes of this type.

equals(*obj*)

can compare value

getType()

Returns the actual type that is allowed for the value of this Parameter.

getValue()

Gets the value of this parameter as an Object.

serializable()

Can be encoded with serializableEncoder

setType(*type_*)

Replaces the current type of this parameter. Unsuported parameter types will get a NotImplementedError.

setValue(*value*)

Replaces the current value of this parameter. If given/current **type_** is '' and arg value's type is in ParameterTypes both value and type are updated to the suitable one in ParameterDataTypes; or else TypeError is raised. If value type and given/current **type_** are different.

Incompatible value and **type_** will get a TypeError.

toString()

property type_

for property getter

property value

for property getter

class fdi.dataset.metadata.**StringParameter**(***kwds*)

Bases: *Parameter*

has a unicode string as the value.

serializable()

Can be encoded with serializableEncoder

fdi.dataset.metadata.**logger = <Logger fdi.dataset.metadata (WARNING)>**

Allowed Parameter types and the corresponding classes. The keys are mnemonics for humans; the values are type(x).__name__.

fdi.dataset.metadataholder module

class fdi.dataset.metadataholder.**MetaDataHolder**(***kwds*)

Bases: object

Object holding meta data. mh: object for compatibility with python2

getMeta()

Returns the current MetaData container of this object. Cannot become a python property because setMeta is in Attributable

hasMeta()

whether the metadata holder is present. During initialization subclass of MetaDataHolder may need to know if the metadata holder has been put in place with is method.

fdi.dataset.ndprint module

fdi.dataset.ndprint.**ndprint**(*data*, *trans=True*, *table=True*)

makes a formated string of an N-dimensional array for printing. The fastest changing index is the innerest list. E.g. A 2 by 3 matrix is [[1,2],[3,4],[5,6]] written as 1 2 3 4 5 6 But if the matrix is a table, the cells in a column change the fastest, and the columns are written vertically. So to print a matrix as a table, whose columns are the innerest list, set trans = True (default) then the matrix needs to be printed transposed: 1 3 5 2 4 6

fdi.dataset.odict module

class fdi.dataset.odict.**ODict**

Bases: OrderedDict

OrderedDict with a better __repre__.

toString(*matprint=None*, *trans=True*)

fdi.dataset.odict.**bstr**(*x*, *tostr=True*, *quote="'"*, ***kwds*)

returns the best string representation. if the object is a string, return single-quoted; if has toString(), use it; else returns str().

fdi.dataset.product module

class fdi.dataset.product.**Product**(*description='UNKOWN'*, *type_='Product'*, *creator='UNKOWN'*, *creationDate=2017-01-01T00:00:00.000000 TAI(0)*, *rootCause='UNKOWN'*, *schema='0.4'*, *startDate=2017-01-01T00:00:00.000000 TAI(0)*, *endDate=2017-01-01T00:00:00.000000 TAI(0)*, *instrument='UNKOWN'*, *modelName='UNKOWN'*, *mission='_AGS'*, ***kwds*)

Bases: *BaseProduct*

Product class (level ALL) version 0.5 inheriting BaseProduct. Automatically generated from fdi/dataset/resources/Product.yml on 2020-06-19 12:49:05.502009.

Generally a Product (inheriting BaseProduct) has project-wide attributes and can be extended to define a plethora of specialized products.

```
productInfo = {'metadata': {'creationDate': {'data_type': 'finetime', 'default':
'0', 'description': 'Creation date of this product', 'fits_keyword': 'DATE',
'unit': 'None'}, 'creator': {'data_type': 'string', 'default': 'UNKOWN',
'description': 'Generator of this product. Example name of institute, organization,
person, software, special algorithm etc.', 'fits_keyword': 'CREATOR', 'unit':
'None'}, 'description': {'data_type': 'string', 'default': 'UNKOWN',
'description': 'Description of this product', 'fits_keyword': 'DESCRIPT', 'unit':
'None'}, 'endDate': {'data_type': 'finetime', 'default': '0', 'description':
'Nominal end time of this product.', 'fits_keyword': 'DATE_END', 'unit': 'None',
'valid': ''}, 'instrument': {'data_type': 'string', 'default': 'UNKOWN',
'description': 'Instrument that generated data of this product', 'fits_keyword':
'INSTRUME', 'unit': 'None', 'valid': ''}, 'mission': {'data_type': 'string',
'default': '_AGS', 'description': 'Name of the mission.', 'fits_keyword':
'TELESCOP', 'unit': 'None', 'valid': ''}, 'modelName': {'data_type': 'string',
'default': 'UNKOWN', 'description': 'Model name of the instrument of this
product', 'fits_keyword': 'MODEL', 'unit': 'None', 'valid': ''}, 'rootCause':
{'data_type': 'string', 'default': 'UNKOWN', 'description': 'Reason of this run
of pipeline.', 'fits_keyword': 'ROOTCAUS', 'unit': 'None'}, 'schema':
{'data_type': 'string', 'default': '0.4', 'description': 'Version of product
schema', 'fits_keyword': 'SCHEMA', 'unit': 'None', 'valid': ''}, 'startDate':
{'data_type': 'finetime', 'default': '0', 'description': 'Nominal start time of
this product.', 'fits_keyword': 'DATE_OBS', 'unit': 'None', 'valid': ''}, 'type':
{'data_type': 'string', 'default': 'Product', 'description': 'Product Type
identification. Fully qualified Python class name or CARD.', 'fits_keyword':
'TYPE', 'unit': 'None', 'valid': ''}}}
```

fdi.dataset.quantifiable module

class fdi.dataset.quantifiable.**Quantifiable**(*unit=None*, ***kwds*)

Bases: object

A Quantifiable object is a numeric object that has a unit. $ x.unit = ELECTRON_VOLTS $ print x.unit eV [1.60218E-19 J]

getUnit()

Returns the unit related to this object.

setUnit(*unit*)

Sets the unit of this object.

property unit

fdi.dataset.serializable module

class fdi.dataset.serializable.**Serializable**(***kwds*)

Bases: object

mh: Can be serialized. Has a ClassID and version instance property to show its class and version information.

serializable()

returns an odict that has all state info of this object. Subclasses should override this function.

serialized(*indent=None*)

class fdi.dataset.serializable.**SerializableEncoder**(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *default=None*)

Bases: JSONEncoder

can encode parameter and product etc such that they can be recovered with deserializeClassID. Python 3 treats string and unicode as unicode, encoded with utf-8, byte blocks as bytes, encoded with utf-8. Python 2 treats string as str and unicode as unicode, encoded with utf-8, byte blocks as str, encoded with utf-8

default(*obj*)

Implement this method in a subclass such that it returns a serializable object for o, or calls the base implementation (to raise a TypeError).

For example, to support arbitrary iterators, you could implement default like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
    # Let the base class default method raise the TypeError
    return JSONEncoder.default(self, o)
```

fdi.dataset.serializable.**serializeClassID**(*o*, *indent=None*)

return JSON using special encoder SerializableEncoder

fdi.dataset.serializable.**serializeHipe**(*o*)

return JSON using special encoder SerializableHipeEncoder

fdi.dataset.yaml2python module

fdi.dataset.yaml2python.**mkinfo**(*attrs*, *indent*, *demo*, *onlyInclude*)

make productInfo string from attributes given.

3.1.2 fdi.pal package

Subpackages

fdi.pal.resources package

Submodules

fdi.pal.common module

fdi.pal.common.**getObjectbyId**(*idn*, *lgbv*)

lgb is from deserializing caller's globals().values() locals().values() and built-ins

fdi.pal.common.**getProductObject**(*urn*, *lgb=None*)

Returns a product from URN. returns object.

fdi.pal.comparable module

class fdi.pal.comparable.**Comparable**(***kwds*)

Bases: object

compareTo(*o*)

fdi.pal.context module

class fdi.pal.context.**Context**(***kwds*)

Bases: *Product*

A special kind of Product that can hold references to other Products.

This abstract product introduces the lazy loading and saving of references to Products or ProductRefs that it is holding. It remembers its state. http://herschel.esac.esa.int/hcss-doc-15.0/load/hcss_drm/api/herschel/ia/pal/Context.html

getAllRefs(*recursive*, *includeContexts*)

Provides a set of the unique references stored in this context.

hasDirtyReferences(*storage*)

Returns a logical to specify whether this context has dirty references or not.

static isContext(*cls*)

Yields true if specified class belongs to the family of contexts.

isValid()

Provides a mechanism to ensure whether it is valid to store this context in its current state.

readDataset(*storage*, *table*, *defaultPoolId*)

Reads a dataset with information within this context that is normally not accessible from the normal Product interface.

refsChanged()

Indicates that the references have been changed in memory, which marks this context as dirty.

writeDataset(*storage*)

Creates a dataset with information within this context that is normally not accessible from the normal Product interface.

exception fdi.pal.context.**ContextRuleException**

Bases: ValueError

class fdi.pal.context.**MapContext**(***kwds*)

Bases: *Context*

Allows grouping Products into a map of (String, ProductRef) pairs. New entries can be added if they comply to the adding rules of this context. The default behaviour is to allow adding any (String,ProductRef).

An example:

```
image     = ImageProduct(description="hi")
spectrum  = SpectrumProduct(description="there")
simple    = Product(description="everyone")

context=MapContext()
```

(continues on next page)

(continued from previous page)

```
context.refs.put("x",ProductRef(image))
context.refs.put("y",ProductRef(spectrum))
context.refs.put("z",ProductRef(simple))
print context.refs.size() # 3
print context.refs.get('x').product.description # hi
print context.refs.get('y').product.description # there
print context.refs.get('z').product.description # everyone
```

It is possible to insert a ProductRef at a specific key in the MapContext. The same insertion behaviour is followed as for a Python dict, in that if there is already an existing ProductRef for the given key, that ProductRef is replaced with the new one:

```
product4=SpectrumProduct(description="everybody")
context.refs.put("y", ProductRef(product4))
product5=SpectrumProduct(description="here")
context.refs.put("a", ProductRef(product5))

print context.refs.get('x').product.description # hi
print context.refs.get('y').product.description # everybody
print context.refs.get('z').product.description # everyone
print context.refs.get('a').product.description # here
```

Note that the rules are only applied when putting an entry to the map!

Be aware that

1. the put() method of the map view may throw a ContextRuleException if the data added to the context violates the rules applied to the context.
2. the put() method of the map view may throw a ValueError if either of the arguments to the put() method are null.

addRule(*rule*)

Add to the rule that controls the products to be added into the context. The new rule will be old rule AND added rule.

getAllRefs(*recursive=False*, *includeContexts=True*, *seen=None*)

Provides a set of the unique references stored in this context. This includes references that are contexts, but not the contents of these subcontexts. This is equivalent to getAllRefs(recursive=false, includeContexts=true). recursive - if true, include references in subcontexts includeContexts - if true, include references to contexts, not including this one

getRefs()

Returns the reference container mapping

getRule()

Get the rule that controls the products to be added into the context.

hasDirtyReferences(*storage*)

Returns a logical to specify whether this context has dirty references or not.

static isContext(*cls*)

Yields true if specified class belongs to the family of contexts.

isValid()

Provides a mechanism to ensure whether it is valid to store this context in its current state.

readDataset(*storage*, *table*, *defaultPoolId*)

Reads a dataset with information within this context that is normally not accessible from the normal Product interface.

property refs

Property

set(*name*, *refs*)

add owner to RefContainer

setRefs(*refs*)

Changes/Adds the mapping container that holds references.

setRule(*rule*)

Set the rule that controls the products to be added into the context.

writeDataset(*storage*)

Creates a dataset with information within this context that is normally not accessible from the normal Product interface.

class fdi.pal.context.**RefContainer**(**kwds*)

Bases: *Serializable*, *ODict*

A map where Rules of a Context are applied when put(k,v) is called, and the owner MapContext's ID can be put to v's parents list.

Implemwnted using dataset.Composite so that RefContainer has a ClassID when json.loads'ed. A MapContext has a _sets, which has a refs:RefContainer, which has a _sets, which has a name:ProductRef. when used as context.refs.get('x').product.description, the RefContainer is called with get() or __getitem__(), which calls superclass composite's _set's __getitem__()

clear()

remove all productRefs

get(*key*)

put(*key*, *ref*)

set label-ref pair after validating then add parent to the ref

serializable()

Can be encoded with serializableEncoder

set(*key*, *ref*)

setOwner(*owner*)

records who owns this container

size()

fdi.pal.context.**applyrules**(*key*, *ref*, *rules*)

fdi.pal.definable module

class fdi.pal.definable.**Definable**(**kwds)

Bases: object

for items being able to be defined with a Definition.

getDefinition()

Returns the definition associated to this definable item. mh: adopting http://herschel.esac.esa.int/hcss-doc-15.0/load/hcss_drm/api/index.html?herschel/ia/pal/ProductRef.html but parameterize definition does not seem worth it given the few numbers of implemented definitions

fdi.pal.httpclientpool module

class fdi.pal.httpclientpool.**HttpClientPool**(**kwds)

Bases: *ProductPool*

the pool will save all products on a remote server.

getHead(*ref*)

Returns the latest version of a given product, belonging to the first pool where the same track id is found.

readHK()

loads and returns the housekeeping data

schematicLoadProduct(*resourcename*, *indexstr*, *urn*)

does the scheme-specific part of loadProduct.

schematicRemove(*typename*, *serialnum*, *urn*)

does the scheme-specific part of removal.

schematicSave(*typename*, *serialnum*, *data*, *urn*, *tag=None*)

does the media-specific saving to remote server save metadata at localpool

schematicWipe()

does the scheme-specific remove-all

transformpath(*path*)

override this to changes the output from the input one (default) to something else.

writeHK(*fp0*)

save the housekeeping data to disk

fdi.pal.httpclientpool.**writeJsonwithbackup**(*fp*, *data*)

write data in JSON after backing up the existing one.

fdi.pal.localpool module

class fdi.pal.localpool.**LocalPool**(**kwds)

Bases: *ProductPool*

the pool will save all products in local computer.

getHead(*ref*)

Returns the latest version of a given product, belonging to the first pool where the same track id is found.

readHK()

loads and returns the housekeeping data

schematicLoadProduct(*resourcename*, *indexstr*, *urn=None*)

does the scheme-specific part of loadProduct.

schematicRemove(*typename*, *serialnum*)

does the scheme-specific part of removal.

schematicSave(*typename*, *serialnum*, *data*, *urn=None*, *tag=None*)

does the media-specific saving

schematicWipe()

does the scheme-specific remove-all

transformpath(*path*)

override this to changes the output from the input one (default) to something else.

writeHK(*fp0*)

save the housekeeping data to disk

fdi.pal.localpool.**writeJsonwithbackup**(*fp*, *data*)

write data in JSON after backing up the existing one.

fdi.pal.mempool module

class fdi.pal.mempool.**MemPool**(***kwds*)

Bases: *ProductPool*

the pool will save all products in memory.

getHead(*ref*)

Returns the latest version of a given product, belonging to the first pool where the same track id is found.

getPoolSpace()

returns the map of this memory pool.

readHK()

loads and returns the housekeeping data

schematicLoadProduct(*resourcename*, *indexstr*)

does the scheme-specific part of loadProduct. note that the index is given as a string.

schematicRemove(*typename*, *serialnum*)

does the scheme-specific part of removal.

schematicSave(*typename*, *serialnum*, *data*)

does the media-specific saving

schematicWipe()

does the scheme-specific remove-all

writeHK(*fp0*)

save the housekeeping data to mempool

fdi.pal.pnspoolserver module

fdi.pal.pnspoolserver.**bad_request**(*error*)

fdi.pal.pnspoolserver.**calc**(*d*)

generates result product directly using data on PNS.

fdi.pal.pnspoolserver.**calcresult**(*cmd*, *ops=''*)

fdi.pal.pnspoolserver.**checkpath**(*path*)

fdi.pal.pnspoolserver.**cleanPTS**(*d*)

Removing traces of past runnings the Processing Task Software.

fdi.pal.pnspoolserver.**cleanup**(*cmd*)

DELETE is used to clean up the Processing Task Software (PST) to its initial configured state.

fdi.pal.pnspoolserver.**configPNS**(*d=None*)

Configure the PNS itself by replacing the pnsconfig var

fdi.pal.pnspoolserver.**configPTS**(*d=None*)

Configure the Processing Task Software by running the config script. Ref init PTS.

fdi.pal.pnspoolserver.**defaultprocessinput**(*data*)

puts all undecoded json to every files.

fdi.pal.pnspoolserver.**defaultprocessoutput**(*filemode*)

reads each of the files and returns the contents in a filename indexed dict.

fdi.pal.pnspoolserver.**dosleep**(*indata*, *ops*)

run 'sleep [ops]' in the OS. ops is 3 if not given.

fdi.pal.pnspoolserver.**filesin**(*dir*)

returns names and contents of all files in the dir, 'None' if dir not existing.

fdi.pal.pnspoolserver.**genposttestprod**(*d*)

generate post test product. put the 1st input (see maketestdata in test_pns.py) parameter to metadata and 2nd to the product's dataset

fdi.pal.pnspoolserver.**get_apis**()

fdi.pal.pnspoolserver.**getinfo**(*cmd*)

returns init, config, run input, run output.

fdi.pal.pnspoolserver.**initPTS**(*d=None*)

Initialize the Processing Task Software by running the init script defined in the config. Execution on the server host is in the pnshome directory and run result and status are returned. If input/output directories cannot be created with serveruser as owner, Error401 will be given.

fdi.pal.pnspoolserver.**makepublicAPI**(*ops*)

fdi.pal.pnspoolserver.**not_found**(*error*)

fdi.pal.pnspoolserver.**run**(*d*, *processinput=None*, *processoutput=None*)

Generates a product by running script defined in the config under 'run'. Execution on the server host is in the pnshome directory and run result and status are returned.

`fdi.pal.pnspoolserver.setup(cmd, ops='')`

PUT is used to initialize or configure the Processing Task Software (PST).

`fdi.pal.pnspoolserver.testinit(d=None)`

Renames the 'init' 'config' 'run' 'clean' scripts to '.save' and points it to the '.ori' scripts.

`fdi.pal.pnspoolserver.testrun(d)`

`fdi.pal.pnspoolserver.unauthorized(error)`

`fdi.pal.pnspoolserver.uploadScript(op, d=None)`

`fdi.pal.pnspoolserver.verify(username, password)`

This function is called to check if a username / password combination is valid.

fdi.pal.poolmanager module

class `fdi.pal.poolmanager.PoolManager`

Bases: `object`

> This class provides the means to reference ProductPool objects without having to hard-code the type of pool. For example, it could be desired to easily switch from one pool type to another.

This is done by calling the getPool(String) method, which will return an existing pool or create a new one if necessary.

classmethod `getMap()`

classmethod `getPool(poolurn)`

returns an instance of pool according to urn.

create the pool if it does not already exist. the same pool-URN always get the same pool.

classmethod `isLoaded(poolurn)`

Whether an item with the given id has been loaded (cached).

classmethod `removeAll()`

deletes all pools from the pool list, pools unwiped

classmethod `save(poolurn, poolobj)`

classmethod `size()`

Gives the number of entries in this manager.

fdi.pal.productpool module

class `fdi.pal.productpool.ProductPool(poolurn=None, use_basepoolpath=True, **kwds)`

Bases: `Definable`, `Taggable`, `Versionable`

A mechanism that can store and retrieve Products.

A product pool should not be used directly by users. The general user should access data in a ProductPool through a ProductStorage instance.

When implementing a ProductPool, the following rules need to be applied:

1. Pools must guarantee that a Product saved via the pool saveProduct(Product) method is stored persistently, and that method returns a unique identifier (URN). If it is not possible to save a Product, an IOException shall be raised.

2. A saved Product can be retrieved using the loadProduct(Urn) method, using as the argument the same URN that assigned to that Product in the earlier saveProduct(Product) call. No other Product shall be retrievable by that same URN. If this is not possible, an IOException or GeneralSecurityException is raised.
3. Pools should not implement functionality currently implemented in the core paclage. Specifically, it should not address functionality provided in the Context abstract class, and it should not implement versioning/cloning support.

accept(*visitor*)

Hook for adding functionality to object through visitor pattern.

dereference(*ref*)

Decrement the reference count of a ProductRef.

exists(*urn*)

Determines the existence of a product with specified URN.

getDefinition()

Returns pool definition info which contains pool type and other pool specific configuration parameters

getId()

Gets the identifier of this pool.

getProductClasses()

Returns all Product classes found in this pool. mh: returns an iterator.

getReferenceCount(*ref*)

Returns the reference count of a ProductRef.

getUrnId()

Get the identifier of this pool used to build URN, usually it's same as id returned by getId().

isAlive()

Test if the pool is capable of responding to commands.

isEmpty()

Determines if the pool is empty.

loadDescriptors(*urn*)

Loads the descriptors belonging to specified URN.

loadProduct(*urn*)

Loads a Product belonging to specified URN.

lockpath()

meta(*urn*)

Loads the meta-data belonging to the product of specified URN.

mfilter(*q*, *cls=None*, *reflist=None*, *urnlist=None*, *snlist=None*)

returns filtered collection using the query.

q is a MetaQuery valid inputs: cls and ns list; productref list; urn list

pfilter(*q*, *cls=None*, *reflist=None*, *urnlist=None*, *snlist=None*)

returns filtered collection using the query.

q is a AbstractQuery. valid inputs: cls and ns list; productref list; urn list

`reference`(*ref*)

Increment the reference count of a ProductRef.

`remove`(*urn*)

Removes a Product belonging to specified URN.

`removeAll`()

Remove all pool data (self, products) and all pool meta data (self, descriptors, indices, etc.).

`saveDescriptors`(*urn*, *desc*)

Save/Update descriptors in pool.

`saveProduct`(*product*, *tag=None*, *geturnobjs=False*)

Saves specified product and returns the designated ProductRefs or URNs. Saves a product or a list of products to the pool, possibly under the supplied tag, and return the reference (or a list of references is the input is a list of products), or Urns if geturnobjs is True.

Pool:!!dict

_classes:!!odict

$product0_class_name:!!dict

currentSN:!!int $the serial number of the latest added prod to the pool

sn:!!list

- $serial number of a prod
- $serial number of a prod
- …

$product1_class_name:!!dict …

_urns:!!odict

$URN0:!!odict

meta:!!MetaData $prod.meta tags:!!list

- $tag
- $tag
- …

_tags:!!odict

urns:!!list

- $urn
- $urn
- …

$urn:!!$serialized product

`schematicLoadProduct`(*resourcename*, *indexstr*)

to be implemented by subclasses to do the scheme-specific loading

`schematicRemove`(*typename*, *serialnum*)

to be implemented by subclasses to do the scheme-specific removing

schematicSave(*typename*, *serialnum*, *data*)

to be implemented by subclasses to do the scheme-specific saving

select(*query*, *results=None*)

Returns a list of references to products that match the specified query.

fdi.pal.productref module

class fdi.pal.productref.**ProductRef**(*urn=None*, *poolurn=None*, *product=None*, *meta=None*, ***kwds*)

Bases: *MetaDataHolder*, *Serializable*, *Comparable*

A lightweight reference to a product that is stored in a ProductPool or in memory.

addParent(*parent*)

add a parent

equals(*o*)

true if o is a non-null ProductRef, with the same Product type than this one, and:

urns and products are null in both refs, or nurs are equal and products are null, or # <– mh urns are null in both refs, and their products are equal, or urns and products are equal in both refs

getHash()

Returns a code number for the product; actually its MD5 signature. This allows checking whether a product already exists in a pool or not.

getMeta()

Returns the metadata of the product.

getParents()

Return the in-memory parent context products of this reference. That is, the contexts in program memory that contain this product reference object. A context that contains a different product reference object pointing to the same URN is not a parent of this product reference.

Furthermore, it should be understood that this method does not return the parent contexts of the product pointed to by this reference as stored in any underlying pool or storage.

Returns: the parents

getProduct()

Get the product that this reference points to.

If the product is a Context, it is kept internally, so further accesses don't need to ask the storage for loading it again. Otherwise, the product is returned but the internal reference remains null, so every call to this method involves a request to the storage.

This way, heavy products are not kept in memory after calling this method, thus maintaining the ProductRef a lighweight reference to the target product.

In case of a Context, if it is wanted to free the reference, call unload().

Returns: the product

getSize()

Returns the estimated size(in bytes) of the product in memory. Useful for providing this information for a user that wants to download the product from a remote site. Returns: the size in bytes

getStorage()

Returns the product storage associated.

getType()

Specifies the Product class to which this Product reference is pointing to.

getUrn()

Returns the Uniform Resource Name (URN) of the product.

getUrnObj()

Returns the URN as an object.

isLoaded()

Informs whether the pointed product is already loaded.

property meta

Property

property parents

property

property product

removeParent(*parent*)

remove a parent

serializable()

Can be encoded with serializableEncoder

setParents(*parents*)

Sets the in-memory parent context products of this reference.

setStorage(*storage*)

Sets the product storage associated.

setUrn(*urn*)

setUrnObj(*urnobj*, *poolurn=None*, *meta=None*)

sets urn Poolurn if given overrides the pool URN in urn, and causes metadata to be loaded from pool. A productref created from a single product will result in a memory pool urn, and the metadata won't be loaded. If meta is given, it will be used instead of that from poolurn.

toString(*matprint=None*, *trans=True*)

unload()

Clear the cached meta and frees internal reference to the product, so it can be garbage collected.

property urn

Property

property urnobj

Property

fdi.pal.productstorage module

class fdi.pal.productstorage.**ProductStorage**(*pool=None*, ***kwds*)

Bases: object

Logical store created from a pool or a poolURN.

Every instanciation with the same pool will result in a new instance of ProdStorage.

accept(*visitor*)

Hook for adding functionality to object through visitor pattern.

getAllTags()

Get all tags defined in the writable pool.

getHead(*ref*)

Returns the latest version of a given product, belonging to the first pool where the same track id is found.

getMeta(*urn*)

Get the metadata belonging to the writable pool that associated to a given URN. returns an ODict.

getPool(*poolurn*)

mh: returns the pool object from poolurn

getPools()

Returns the set of ProductPools registered. mh: in a list of poolurns

getProductClasses(*poolurn*)

Yields all Product classes found in this pool.

getTags(*urn*)

Get the tags belonging to the writable pool that associated to a given URN. returns an iterator.

getUrnFromTag(*tag*)

Get the URN belonging to the writable pool that is associated to a given tag.

getWritablePool()

returns the poolurn of the first pool, which is the only writeable pool.

load(*urnortag*)

Loads a product with a URN or a list of products with a tag, from the (writeable) pool. It always creates new ProductRefs. returns productref(s). urnortag: urn or tag

register(*pool*)

Registers the given pools to the storage.

remove(*urn*)

removes product of urn from the writeable pool

save(*product*, *tag=None*, *poolurn=None*, *geturnobjs=False*)

saves to the writable pool if it has been registered. if not, registers and saves. product can be one or a list of prpoducts. Returns: one or a list of productref with storage info. mh: or UrnObjs if geturnobjs is True.

select(*query*, *previous=None*)

Returns a list of URNs to products that match the specified query.

Parameters: query - the query object previous - results to be refined Returns: the set of return eferences to products matching the supplied query.

wipePool(*poolurn*)

fdi.pal.query module

class fdi.pal.query.**AbstractQuery**(*product=<class 'fdi.dataset.product.Product'>*, *variable='p'*, *where=''*, *allVersions=False*, ***kwds*)

Bases: *StorageQuery*

provides default implementations for the pal storage query.

hashCode()

toString()

class fdi.pal.query.**MetaQuery**(*product=<class 'fdi.dataset.product.Product'>*, *where=''*, *allVersions=False*, ***kwds*)

Bases: *AbstractQuery*

Meta data query formulates a query on the meta data of a Product.

Typically this type of query is faster than a full query on the Product Access Layer.

class fdi.pal.query.**StorageQuery**(***kwds*)

Bases: *Serializable*

Query on a ProductStorage.

accept(*visitor*)

Hook for adding functionality to object through visitor pattern.

getType()

Get the class used in the query.

getVariable()

Get the variable name used in the query expression, eg "p".

getWhere()

Get the query expression to be evaluated.

retrieveAllVersions()

Are all versions to be retrieved, or just the latest?

fdi.pal.runpnsserver module

fdi.pal.taggable module

class fdi.pal.taggable.**Taggable**(***kwds*)

Bases: object

Definition of services provided by a product storage supporting versioning.

getTagUrnMap()

Get the full tag->urn mappings. mh: returns an iterator

getTags(*urn=None*)

Get all of the tags that map to a given URN. Get all known tags if urn is not specified. mh: returns an iterator.

getUrn(*tag*)

Gets the URNs corresponding to the given tag.

getUrnObject(*tag*)

Gets the URNobjects corresponding to the given tag.

removeTag(*tag*)

Remove the given tag from the tag and urn maps.

removeUrn(*urn*)

Remove the given urn from the tag and urn maps.

removekey(*key*, *themap*, *thename*, *othermap*, *othername*)

Remove the given key.

setTag(*tag*, *urn*)

Sets the specified tag to the given URN.

tagExists(*tag*)

Tests if a tag exists.

fdi.pal.urn module

class fdi.pal.urn.**Urn**(*urn=None*, *pool=None*, *cls=None*, *index=None*, ***kwds*)

Bases: *DeepEqual*, *Serializable*, *Comparable*

The object representation of the product URN string. The memory consumed by sets of this object are much less than sets of URN strings.

Only when the class types in URN string are not in classpath, the urn object will consume equals or a little more than URN string as the object has to hold the original urn string. However this should be considered as exceptional cases.

Using this object representation also help to avoid parsing cost of URN string. mh: URN format:

```
urn:poolname:resourceclass:serialnumber
```

where

Resourceclass

(fully qualified) class name of the resource (product)

Poolname

scheme + `://` + place + directory

Scheme

`file`, `mem`, `http` … etc

Place

`192.168.5.6:8080`, `c:`, an empty string … etc

Directory

A label for the pool that is by default used as the full path where the pool is stored. ProductPool.transformpath() can used to change the directory here to other meaning.

- for `file` scheme: `/` + name + `/` + name + … + `/` + name
- for `mem` scheme: `/` + name

Serialnumber

internal index. str(int).

URN is immutable.

a__eq__(*o*)

mh: compare urn string after the first 4 letters.

static getFullPath(*urn*)

returns the place+poolname+resource directory of the urn

getIndex()

Returns the product index.

getPlace()

Returns the netloc in this

getPool()

Returns the pool name in this

getPoolId()

Returns the pool URN in this

getScheme()

Returns the urn scheme.

getType()

Returns class type of Urn

getTypeName()

Returns class type name of Urn.

getUrn()

Returns the urn in this

getUrnWithoutPoolId()

hasData()

Returns whether this data wrapper has data.

property place

property pool

returns the pool URN.

serializable()

Can be encoded with serializableEncoder

setUrn(*urn*)

parse urn to get scheme, place, pool, resource, index.

toString()

property urn

property

fdi.pal.urn.**makeUrn**(*poolname*, *typename*, *index*)

assembles a URN with infos of the pool, the resource type, and the index

fdi.pal.urn.**parseUrn**(*urn*)

Checks the URN string is valid in its form and splits it. Pool URN is in the form of a URL that does not have ':' in its path part. Product URNs are more complicated. For example if the urn is `urn:file://c:/tmp/mypool/proj1.product:322` into poolname `file://c:/tmp/mypool`, resource type (usually class) name `proj1.product`, serial number in string `'322'`, scheme `file`, place `c:` (with ip and port if given), and poolpath `c:/tmp/mypool` Poolname is also called poolURN or poolID.

fdi.pal.versionable module

class fdi.pal.versionable.**Versionable**(***kwds*)

Bases: object

for items being able to be defined with a Definition.

getLastVersion(*ref*)

Returns the latest version of the given ProductRef.

getVersions(*ref*)

Returns all the versions of the given ProductRef.

saveProductRef(*ref*)

Saves the product referenced and returns the designated URN.

3.1.3 fdi.pns package

Subpackages

fdi.pns.resources package

Submodules

fdi.pns.jsonio module

fdi.pns.jsonio.**deleteJsonObj**(*url*, *obj*, *headers*)

deletes object from url. Returns None if fails.

fdi.pns.jsonio.**getJsonObj**(*url*, *headers=None*, *usedict=False*)

return object from url. url can be http or file. translate keys and values from string to number if applicable. Raise exception if fails. Not using requests.get() as it cannot open file:// w/o installing https://pypi.python.org/pypi/requests-file

fdi.pns.jsonio.**getJsonObj1**(*url*, *headers=None*, *usedict=False*)

return object from url. url can be http or file. translate keys and values from string to number if applicable. Return None if fails. Not using requests.get() as it cannot open file:// w/o installing https://pypi.python.org/pypi/requests-file

fdi.pns.jsonio.**jsonREST**(*url*, *obj*, *headers*, *cmd*)

generic RESTful command handler for POST, PUT, and DELETE.

fdi.pns.jsonio.**postJsonObj**(*url*, *obj*, *headers*)

posts object to url. Returns None if fails.

fdi.pns.jsonio.**putJsonObj**(*url*, *obj*, *headers*)

puts object to url. Returns None if fails.

fdi.pns.jsonio.**writeJsonObj**(*o*, *fn*)

Write an object to file fn in json safely Return True if successful else False

fdi.pns.logdict module

fdi.pns.pnsconfig module

fdi.pns.runflaskserver module

fdi.pns.runflaskserver.**setuplogging**()

fdi.pns.server module

fdi.pns.server.**bad_request**(*error*)

fdi.pns.server.**calc**(*d*)

generates result product directly using data on PNS.

fdi.pns.server.**calcresult**(*cmd*, *ops=''*)

fdi.pns.server.**checkpath**(*path*)

Checks the directories for data exchange between pns server and pns PTS.

fdi.pns.server.**cleanPTS**(*d*)

Removing traces of past runnings the Processing Task Software.

fdi.pns.server.**cleanup**(*cmd*)

DELETE is used to clean up the Processing Task Software (PST) to its initial configured state.

fdi.pns.server.**configPNS**(*d=None*)

Configure the PNS itself by replacing the pnsconfig var

fdi.pns.server.**configPTS**(*d=None*)

Configure the Processing Task Software by running the config script. Ref init PTS.

fdi.pns.server.**defaultprocessinput**(*data*)

puts all undecoded json to every files.

fdi.pns.server.**defaultprocessoutput**(*filemode*)

reads each of the files and returns the contents in a filename indexed dict.

fdi.pns.server.**dosleep**(*indata*, *ops*)

run 'sleep [ops]' in the OS. ops is 3 if not given.

fdi.pns.server.**filesin**(*dir*)

returns names and contents of all files in the dir, 'None' if dir not existing.

fdi.pns.server.**genposttestprod**(*d*)

generate post test product. put the 1st input (see maketestdata in test_pns.py) parameter to metadata and 2nd to the product's dataset

fdi.pns.server.**getConfig**(*conf='pns'*)

Imports a dict named [conf]config defined in ~/.config/[conf]local.py

fdi.pns.server.**getUidGid**(*username*)

returns the UID and GID of the named user.

fdi.pns.server.**get_apis**()

`fdi.pns.server.getinfo(cmd)`

returns init, config, run input, run output.

`fdi.pns.server.initPTS(d=None)`

Initialize the Processing Task Software by running the init script defined in the config. Execution on the server host is in the pnshome directory and run result and status are returned. If input/output directories cannot be created with serveruser as owner, Error401 will be given.

`fdi.pns.server.makepublicAPI(ops)`

`fdi.pns.server.not_found(error)`

`fdi.pns.server.run(d, processinput=None, processoutput=None)`

Generates a product by running script defined in the config under 'run'. Execution on the server host is in the pnshome directory and run result and status are returned.

`fdi.pns.server.setOwnerMode(p, username)`

makes UID and GID set to those of serveruser given in the config file. This function is usually done by the initPTS script.

`fdi.pns.server.setup(cmd, ops='')`

PUT is used to initialize or configure the Processing Task Software (PST).

`fdi.pns.server.testinit(d=None)`

Renames the 'init' 'config' 'run' 'clean' scripts to '.save' and points it to the '.ori' scripts.

`fdi.pns.server.testrun(d)`

`fdi.pns.server.unauthorized(error)`

`fdi.pns.server.uploadScript(op, d=None)`

`fdi.pns.server.verify(username, password)`

This function is called to check if a username / password combination is valid.

3.1.4 fdi.utils package

Submodules

fdi.utils.checkjson module

`fdi.utils.checkjson.checkjson(obj, dbg=0)`

seriaizes the given object and deserialize. check equality.

fdi.utils.common module

`fdi.utils.common.fullname(obj)`

full class name with module name.

https://stackoverflow.com/a/2020083/13472124

`fdi.utils.common.pathjoin(*p)`

join path segments with given separater (default '/'). Useful when '' is needed.

`fdi.utils.common.str2md5(string)`

`fdi.utils.common.trbk(e)`

trace back

`fdi.utils.common.trbk2(e)`

fdi.utils.options module

`fdi.utils.options.opt(ops)`

Example: ops = [{'long':'help', 'char':'h', 'default': false, 'description':'print help'}, {'long':'verbose', 'char':'v', 'default': false, 'description':'print info'}, {'long':'username=', 'char':'u', 'default': 'foo', 'description':'non-empty user name/ID'},
{'long':'password=', 'char':'p', 'default': 'bar', 'description':'password'}, {'long':'host=', 'char':'i', 'default': '0.0.0.0', 'description':'host IP/name'}, {'long':'port=', 'char':'o', 'default': 5000, 'description':'port number'}
]

fdi.utils.ydump module

class `fdi.utils.ydump.MyRepresenter(default_style=None, default_flow_style=None, dumper=None)`

Bases: `RoundTripRepresenter`

```
yaml_representers = {<class 'NoneType'>:  <function
RoundTripRepresenter.represent_none>, <class 'str'>:  <function
SafeRepresenter.represent_str>, <class 'bytes'>:  <function
SafeRepresenter.represent_binary>, <class 'bool'>:  <function
SafeRepresenter.represent_bool>, <class 'int'>:  <function
SafeRepresenter.represent_int>, <class 'float'>:  <function
SafeRepresenter.represent_float>, <class 'list'>:  <function
SafeRepresenter.represent_list>, <class 'tuple'>:  <function
SafeRepresenter.represent_list>, <class 'dict'>:  <function
SafeRepresenter.represent_dict>, <class 'set'>:  <function
SafeRepresenter.represent_set>, <class 'ruamel.yaml.compat.ordereddict'>:  <function
SafeRepresenter.represent_ordereddict>, <class 'collections.OrderedDict'>:
<function RoundTripRepresenter.represent_dict>, <class 'datetime.date'>:  <function
SafeRepresenter.represent_date>, <class 'datetime.datetime'>:  <function
SafeRepresenter.represent_datetime>, None:  <function
SafeRepresenter.represent_undefined>, <class
'ruamel.yaml.scalarstring.LiteralScalarString'>:  <function
RoundTripRepresenter.represent_literal_scalarstring>, <class
'ruamel.yaml.scalarstring.FoldedScalarString'>:  <function
RoundTripRepresenter.represent_folded_scalarstring>, <class
'ruamel.yaml.scalarstring.SingleQuotedScalarString'>:  <function
RoundTripRepresenter.represent_single_quoted_scalarstring>, <class
'ruamel.yaml.scalarstring.DoubleQuotedScalarString'>:  <function
RoundTripRepresenter.represent_double_quoted_scalarstring>, <class
'ruamel.yaml.scalarstring.PlainScalarString'>:  <function
RoundTripRepresenter.represent_plain_scalarstring>, <class
'ruamel.yaml.scalarint.ScalarInt'>:  <function
RoundTripRepresenter.represent_scalar_int>, <class
'ruamel.yaml.scalarint.BinaryInt'>:  <function
RoundTripRepresenter.represent_binary_int>, <class
'ruamel.yaml.scalarint.OctalInt'>:  <function
RoundTripRepresenter.represent_octal_int>, <class 'ruamel.yaml.scalarint.HexInt'>:
<function RoundTripRepresenter.represent_hex_int>, <class
'ruamel.yaml.scalarint.HexCapsInt'>:  <function
RoundTripRepresenter.represent_hex_caps_int>, <class
'ruamel.yaml.scalarfloat.ScalarFloat'>:  <function
RoundTripRepresenter.represent_scalar_float>, <class
'ruamel.yaml.scalarbool.ScalarBoolean'>:  <function
RoundTripRepresenter.represent_scalar_bool>, <class
'ruamel.yaml.comments.CommentedSeq'>:  <function
RoundTripRepresenter.represent_list>, <class 'ruamel.yaml.comments.CommentedMap'>:
<function RoundTripRepresenter.represent_dict>, <class
'ruamel.yaml.comments.CommentedOrderedMap'>:  <function
SafeRepresenter.represent_ordereddict>, <class 'ruamel.yaml.comments.CommentedSet'>:
<function RoundTripRepresenter.represent_set>, <class
'ruamel.yaml.comments.TaggedScalar'>:  <function
RoundTripRepresenter.represent_tagged_scalar>, <class
'ruamel.yaml.timestamp.TimeStamp'>:  <function
RoundTripRepresenter.represent_datetime>}
```

class fdi.utils.ydump.**MyYAML**(*, *typ=None*, *pure=False*, *output=None*, *plug_ins=None*)

Bases: YAML

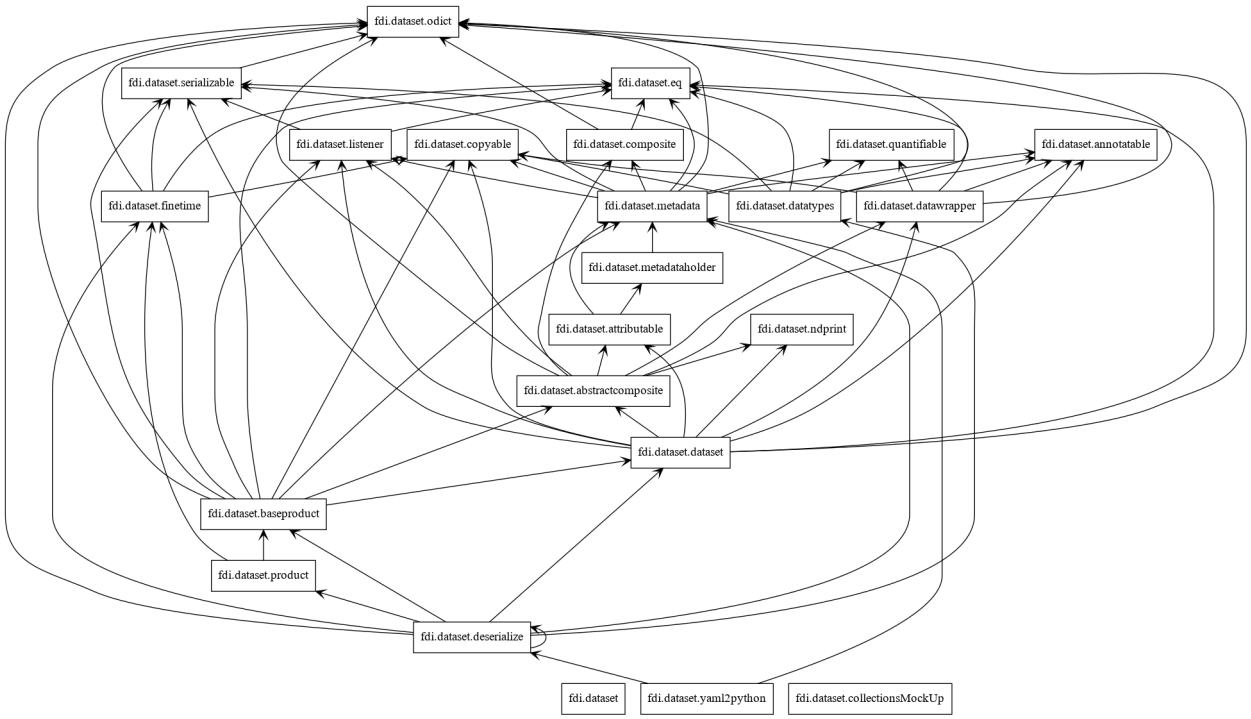
dump(*data*, *stream=None*, ***kw*)

`fdi.utils.ydump.`**`ydump`**(*od*, *stream=None*)

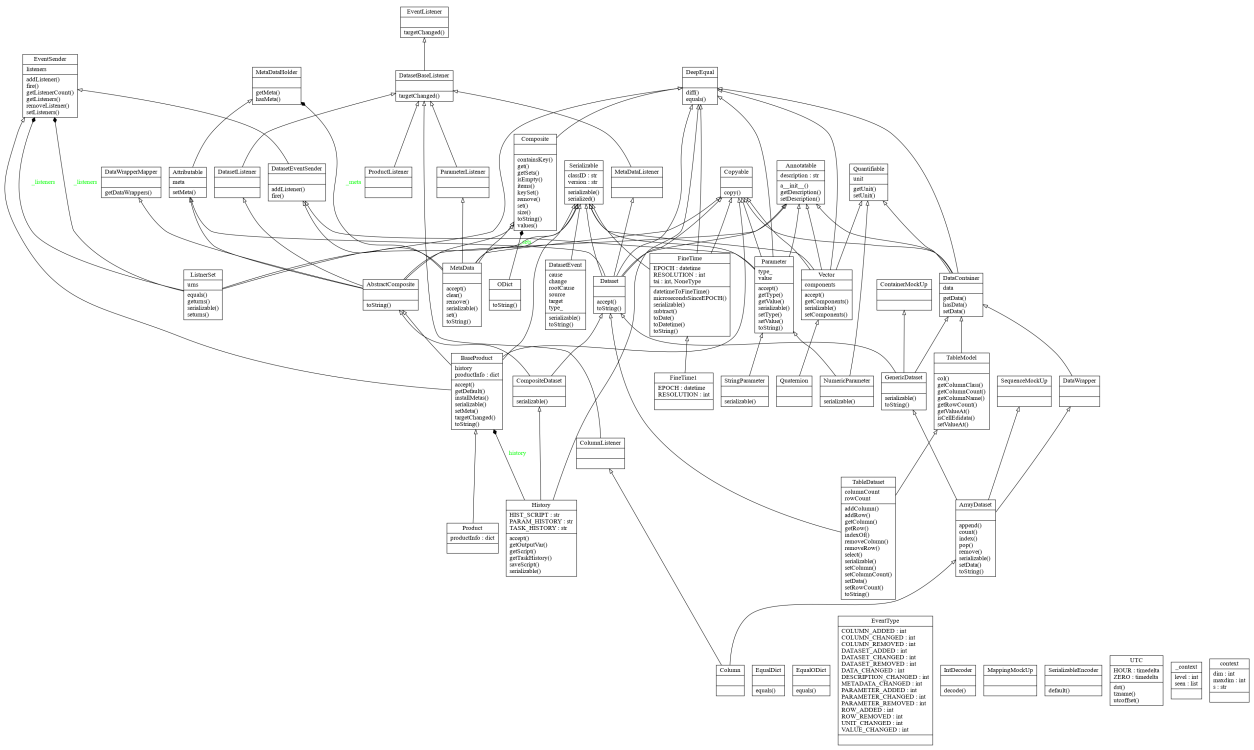
YAML dump that outputs OrderedDict like dict.

3.1.5 Diagrams

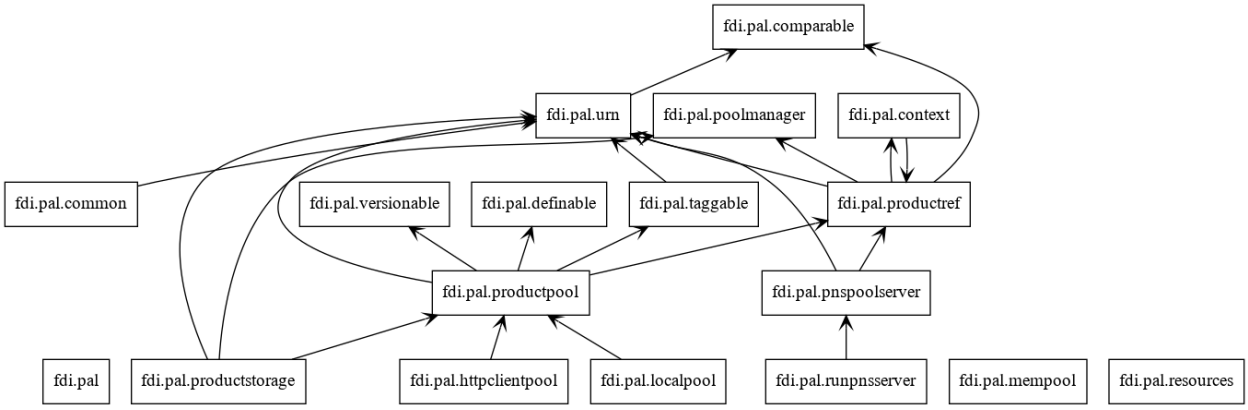
packages_dataset.png



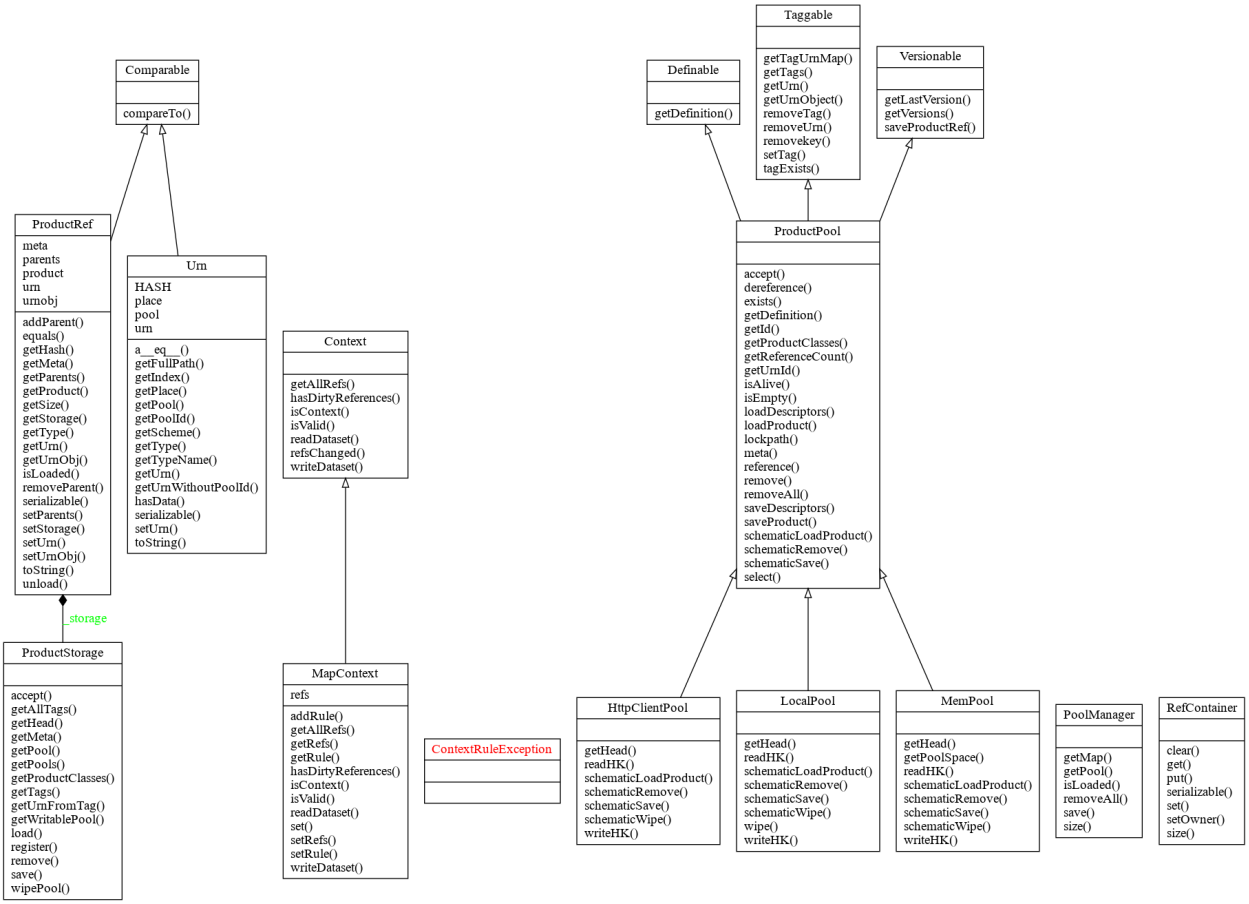
classes_dataset.png



packages_pal.png



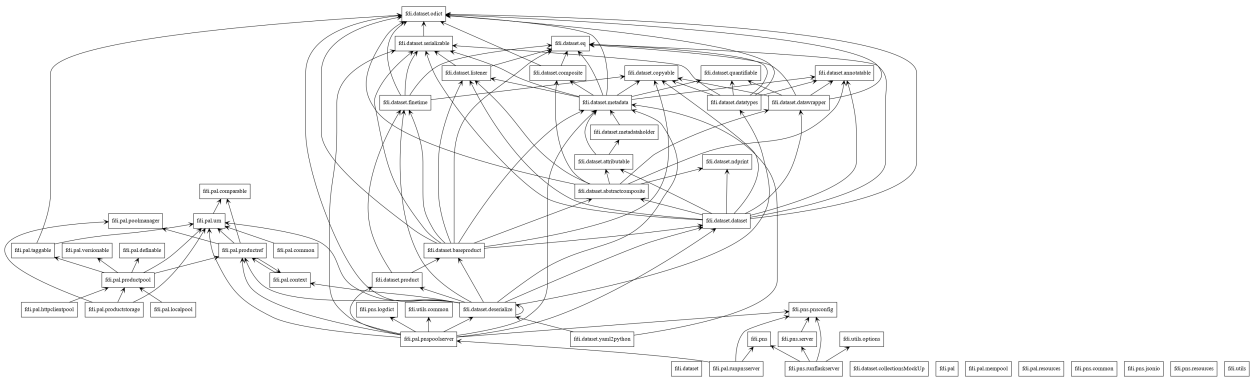
classes_pal.png



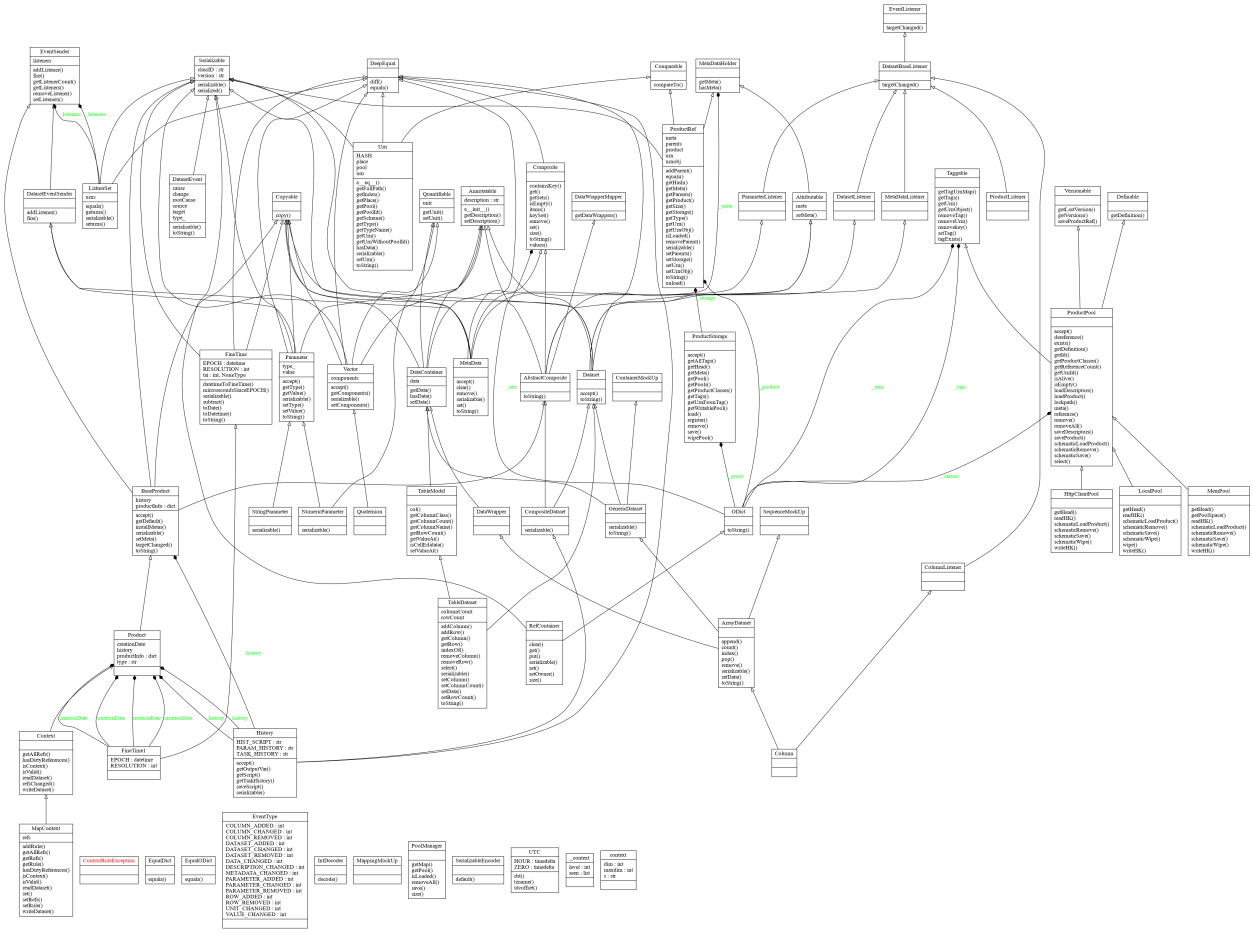
packages_pns.png

classes_pns.png

packages_all.png

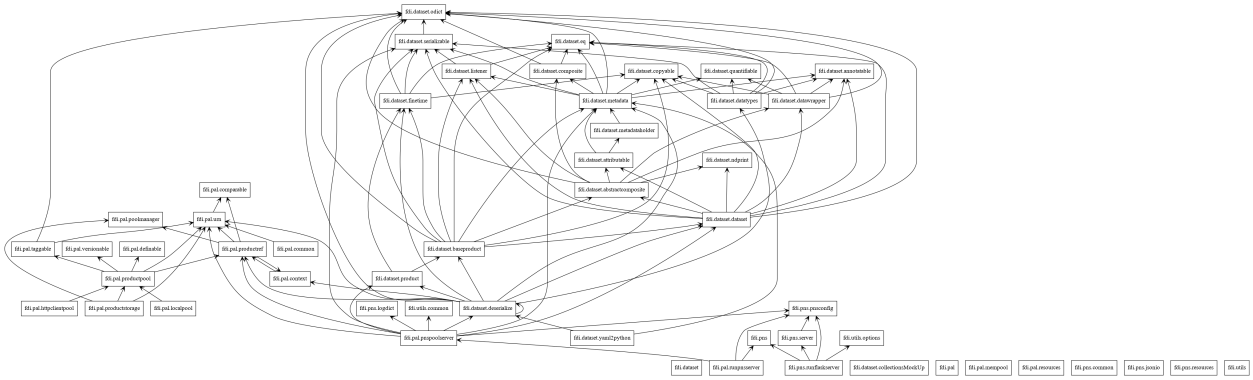


classes_all.png



3.1.6 Indices and tables

- genindex
- modindex
- search



CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

f

INDEX

A

B

C

D

E

F

G

N

O

P

Q

R

S

T

U

V

W

Y

Z