

---

# **DKB Documentation**

**DKB team**

**Jun 13, 2020**



**CONTENTS:**

<b>1</b>	<b>pyDKB package</b>	<b>1</b>
1.1	Quickstart guide . . . . .	1
1.2	Subpackages . . . . .	3
<b>2</b>	<b>Stages</b>	<b>27</b>
2.1	Stage 055 . . . . .	27
<b>3</b>	<b>Indices and tables</b>	<b>33</b>
	<b>Python Module Index</b>	<b>35</b>
	<b>Index</b>	<b>37</b>



## PyDKB package

Common library for Data Knowledge Base Dataflow stages development.

**Dataflow** ETL process (extract-transform-load) for populating internal DKB storages and keeping them up to date

**Dataflow stage** Logical step of ETL process, implemented as standalone executable program (worker)

Dataflow stages are standalone programs, but can be combined into a pipeline by means of Kafka-based supervising program. For details about program compatibility with the supervisor please check documentation for the Metadata Integration Topology Management System (MInT MS) workers<sup>1</sup>. Worker program can be written in any language; pyDKB is intended to simplify this process for Python.

**Warning:** There are three types of stages corresponding three types of ETL operations: *source connector* (data extraction), *processor* (transformation) and *sink connector* (load to internal DKB storage). Currently pyDKB library can be used only for *processor* stages, but in future versions *connector* stages will also be supported.

### 1.1 Quickstart guide

To create simple processor stage application first decide input and output data format. In following examples we will work with data in JSON format (for the full list of supported formats check [pyDKB.dataflow.communication.messages module](#) section of this documentation).

Now let's start writing example processor `welcome.py` and implement message handler – functional part of the stage (operations to be performed on data flow units):

```
from pyDKB.dataflow.communication.messages import JSONMessage

def my_process(stage, message):
    """ Single message processing. """
    input_data = message.content()
    name = input_data.get('name')
    if name:
        out_data = {'message': "Welcome, %s!" % name}
        out_message = JSONMessage(out_data)
        stage.output(out_message)
    return True
```

Function must take two arguments: `stage` (stage context object) and `message` (input message, which should be transformed by our stage). Message is a smallest data unit in the data flow running through the processor, and every message is to be processed independently of previous or following ones. `message.content()` and

---

<sup>1</sup> WIP

`JSONMessage(out_data)` statements are used to decode/encode message to/from Python dict object. Message, passed to the function, is taken from the input data flow; to write new message(s) to the output data flow, `stage.output(out_message)` is used. It can be used as many times as many output messages were generated (or once with the list of messages). In our example, messages without key 'name' will produce no output messages, so `stage.output()` will not be called at all. In terms of data flow it means that the input message is filtered out and will not reach the *sink connector*.

Boolean return value of `my_process()` indicates if the processing was successful or not. If processing failed (False is returned), produced output messages will be dropped to avoid loading sketchy information into the DKB storages.

Now as we have processing logic implemented, we need to turn it into fully functional application. Add following lines to `welcome.py`:

```
import sys
from pyDKB.dataflow.stage import ProcessorStage
from pyDKB.dataflow import messageType
from pyDKB.dataflow.communication.messages import JSONMessage

def my_process(stage, message):
    <...function code...>

if __name__ == '__main__':
    stage = ProcessorStage()
    stage.set_input_message_type(messageType.JSON)
    stage.set_output_message_type(messageType.JSON)
    stage.process = my_process
    stage.configure(sys.argv[1:])
    stage.run()
```

First we create stage object: `stage = JSONProcessorStage()`; then indicate that input and output message format is JSON: `stage.set_{input,output}_message_type(messageType.JSON)` (for full list of message types check [pyDKB.dataflow.communication.messages module](#) section of this documentation); then set stage processing function to our function `my_process()`, parse command line arguments and configure stage instance according to it (`stage.configure(sys.argv[1:])`). Now we ready to start the stage execution.

Easy, right?

It's time to run our application. Create input data sample `input.ndjson` with following lines:

```
{"name": "James", "city": "New York"}
{"user": "Jonathan", "role": "support"}
{"name": "John Smith"}
```

and type:

```
$ python welcome.py --dest s input.ndjson
{"message": "Welcome, James!"}
{"message": "Welcome, John Smith!"}
```

`--dest s` indicates that output destination is (s)tdout (default destination is file). For full information about modes in which the stage application can be used, run `python welcome.py -h`.

That's it, your first application is ready to be integrated into an ETL process as data processing node. For details about ETL process creation check *MInT Supervisor*<sup>2</sup> documentation.

---

<sup>2</sup> WIP

## 1.2 Subpackages

### 1.2.1 pyDKB.common package

Common modules.

#### Submodules

#### pyDKB.common.LoggableObject module

pyDKB.common.LoggableObject

**class** pyDKB.common.LoggableObject.**LoggableObject**  
Bases: object

Common ancestor for all classes that need 'log' method.

**classmethod** **log** (*message*, *level=3*)  
Output log message with given log level.

#### Parameters

- **message** (*str*) – message to output
- **level** (pyDKB.common.types.logLevel member) – log level of the message

#### pyDKB.common.Type module

Abstract class for type definitions.

#### Example

```
>>> myType = Type("Orange", "Apple")
>>> myType.add("Plum")
>>> t = myType.Orange
>>> if t == myType.Orange:
...     print "Orange!"
... elif t == myType.member("Apple"):
...     print "Apple!"
...
Orange!
>>> if not myType.member("Walnut"):
...     print "Wrong type!"
...
Wrong type!
```

**class** pyDKB.common.Type.**Type** (*\*args*)  
Bases: object

Abstract class for type definitions.

Member names (*str*) are passed to the constructor as positional arguments.

**add** (*name*)  
Add member.

**Parameters** **name** (*str*) – name of the member to be added

**hasMember** (*val*)

Check if the member exists (by value).

**Parameters** **val** (*int*) – member to be checked

**Returns** True/False

**Return type** bool

**member** (*name*)

Check if the member exists (by name).

**Parameters** **name** (*str*) – name to be checked

**Returns** member value or False if member does not exist

**Return type** int, bool

**memberName** (*val*)

Return string name of the member.

**Parameters** **val** (*int*) – member to retrieve name for

**Returns** member name or False if member does not exist

**Return type** str, bool

## pyDKB.common.custom\_readline module

Implementation of “readline”-like functionality for custom separator.

---

**Todo:** make import of `fcntl` (or of this module) optional to avoid errors when library is used under Windows.

---

`pyDKB.common.custom_readline.custom_readline` (*f*, *newline*)

Read lines with custom line separator.

Construct generator with readline-like functionality: with every call of `next()` method it will read data from *f* until the *newline* separator is found; then yields what was read.

<b>Warning:</b> the last line can be incomplete, if the input data flow is interrupted in the middle of data writing.
---

To check if iteration is not over without reading next value, one may `send(True)` to the generator: it will return `True` if there is another message to yield or raise `StopIteration` if nothing left.

**Parameters**

- **f** (*file*) – readable file object
- **newline** (*str*) – delimiter to be used instead of `\n`

**Returns** iterable object

**Return type** generator

---

**Todo:**

- make last “line” handling more strict: `no newline == no line`;
- rethink function name (as “line” is actually a “message”);



- move functionality to `pyDKB.dataflow.communication`<sup>1</sup> submodule)

## pyDKB.common.exceptions module

Definition of common modules exceptions

**exception** `pyDKB.common.exceptions.HDFSException`

Bases: `exceptions.RuntimeError`

Base Exception for HDFS module.

## pyDKB.common.hdfs module

Utils to interact with HDFS.

`pyDKB.common.hdfs.File` (*fname*)

Get and open temporary local copy of HDFS file

Return value: open file object (TemporaryFile).

`pyDKB.common.hdfs.basename` (*p*)

Return file name without path.

`pyDKB.common.hdfs.check_stderr` (*proc, timeout=None, max\_lines=1*)

Wait till the end of the subprocess and send its STDERR to STDERR.

Output only MAX\_LINES of the STDERR to the current STDERR; if MAX\_LINES == None, output all the STDERR.

Return value is the subprocess' return code.

`pyDKB.common.hdfs.dirname` (*p*)

Return dirname without filename.

`pyDKB.common.hdfs.getfile` (*fname*)

Download file from HDFS.

Return value: file name (without directory)

`pyDKB.common.hdfs.join` (*p, \*args*)

Join given paths.

`pyDKB.common.hdfs.listdir` (*dirname, mode='a'*)

List files and/or subdirectories of HDFS directory.

**Parameters:** *dirname* – directory to list *mode* – ‘a’: list all objects

‘f’: list files ‘d’: list subdirectories

`pyDKB.common.hdfs.makedirs` (*dirname*)

Try to create directory (with parents).

`pyDKB.common.hdfs.movefile` (*fname, dest*)

Move local file to HDFS.

`pyDKB.common.hdfs.putfile` (*fname, dest*)

Upload file to HDFS.

<sup>1</sup> <https://github.com/PanDAWMS/dkb/pull/129>

## pyDKB.common.json\_utils module

Utils to work with JSON (dict) objects.

`pyDKB.common.json_utils.nestedKeys (key)`  
Transform STRING with nested keys into LIST.

### Parameters:

**STRING key – dot-separated list of nested keys.** If a key contains dot itself, the key must be put between quotation marks.

`pyDKB.common.json_utils.valueByKey (json_data, key)`  
Return value by a chain (list) of nested keys.

**Parameters:** DICT `json_data` – to search in STRING key – dot-separated list of nested keys

## pyDKB.common.misc module

`pyDKB.common.misc`

Miscellaneous utility functions.

`pyDKB.common.misc.ensure_argparse_arg_name (args, kwargs)`  
Ensure cmgline argument name for `argparse.add_argument()`.

Returns the expected argument name and updates `kwargs[ 'dest ' ]` value.

When **python:func:‘argparse.add\_argument()‘** is called, `argparse` decides how to name the new argument basing on the passed parameters (to use this name in `argparse.Namespace` object). This function returns the name, that (supposedly) will be used in `argparse`; but to make sure, `kwargs[ 'dest ' ]` is (re)set to the same value (if applicable).

For parameters description please refer to `argparse` documentation: <https://docs.python.org/2.7/library/argparse.html#the-add-argument-method>

### Parameters

- **args (list)** – list of `argparse.add_argument()` positional parameters
- **kwargs (dict)** – hash of `argparse.add_argument()` keyword parameters

**Returns** argument name

**Return type** string

`pyDKB.common.misc.log (message, level=3, *args)`  
Output log message with given log level.

In case of multiline messages or list of messages only first line (message) is prepended with provided prefixes and timestamp; in all the next lines (messages) they are replaced with special prefix ‘(==)’, representing that these lines belong to the same log record.

Empty lines and lines containing only whitespace symbols are ignored.

### Parameters

- **message (object)** – message to output (string, list of strings or any other object)
- **level (pyDKB.common.types.logLevel member)** – log level of the message
- **\*args** – additional prefixes (will be output between log level prefix and message body)

## pyDKB.common.types module

pyDKB.common.types

Definitions of types used across all the library modules.

## 1.2.2 pyDKB.dataflow package

Dataflow organization utils.

### Subpackages

#### pyDKB.dataflow.communication package

pyDKB.dataflow.communication

pyDKB.dataflow.communication.**Message** (*msg\_type*)  
Return class XXXMessage, where XXX is the passed type.

### Subpackages

#### pyDKB.dataflow.communication.consumer package

Consumer submodule init file.

```
class pyDKB.dataflow.communication.consumer.ConsumerBuilder (config={})
    Bases: object
    Constructor for Consumer instance.
    build (config={})
        Return constructed consumer.
    consumerClass = None
    setSource (source)
        Set data source for the consumer.
    setType (Type)
        Set message type for the consumer.
```

### Submodules

#### pyDKB.dataflow.communication.consumer.Consumer module

pyDKB.dataflow.communication.consumer.Consumer

```
class pyDKB.dataflow.communication.consumer.Consumer (config={})
    Bases: pyDKB.common.LoggableObject.LoggableObject
    Data consumer implementation.
    close ()
        Close opened data stream and data source.
```

**config** = None

**get\_message**()

Get new message from current source.

**Return values:** Message object False (failed to parse message) None (all input sources are empty)

**get\_source\_info**()

Return current source info.

**get\_stream**()

Get input stream linked to the current source.

**Return value:** InputStream None (no sources left to read from)

**init\_stream**()

Init input stream.

**message\_class**()

Return message class.

**message\_type** = None

**next**()

Return new Message, read from input stream.

**reconfigure**(*config={}*)

(Re)initialize consumer with stage config arguments.

**reset\_stream**()

Reset input stream to the current source.

**set\_message\_type**(*Type*)

Set input message type.

**stream\_is\_readable**()

Check if input data stream is readable.

**Returns** True – stream is initialized and not empty, False – stream is empty, None – stream is not initialized

**Return type** bool, NoneType

**exception** `pyDKB.dataflow.communication.consumer.Consumer.ConsumerException` (*message=*  
*rea-*  
*son=None*)

Bases: `pyDKB.dataflow.exceptions.DataflowException`

Dataflow Consumer exception.

## **pyDKB.dataflow.communication.consumer.FileConsumer module**

`pyDKB.dataflow.communication.consumer.FileConsumer`

Data consumer implementation for common (static) files.

**TODO: think about:**

- updatable files
- pipes (better, from the point of StreamConsumer)
- round-robin (for updatable sources)
- ...

```

class pyDKB.dataflow.communication.consumer.FileConsumer.FileConsumer (config={})
    Bases: pyDKB.dataflow.communication.consumer.Consumer.Consumer

    Data consumer implementation for HDFS data source.

    current_file = None

    get_source ()
        Get nearest non-empty source (current or next).

    get_source_info ()
        Return current source info.

    init_sources ()
        Initialize sources iterator if not initialized yet.

    next_source ()
        Reset $current_file to the next non-empty file.

        Return value: File descriptor of the new $current_file None (no files left)

    reconfigure (config={})
        (Re)initialize consumer with Stage configuration.

    source_is_readable ()
        Check if current source is readable.

        Returns None – no source, False – source is empty / fully read, True – source is defined and is
            not empty

        Return type bool, NoneType

```

### pyDKB.dataflow.communication.consumer.HDFSConsumer module

pyDKB.dataflow.communication.consumer.HDFSConsumer

```

class pyDKB.dataflow.communication.consumer.HDFSConsumer.HDFSConsumer (config={})
    Bases: pyDKB.dataflow.communication.consumer.FileConsumer.FileConsumer

    Data consumer implementation for HDFS data source.

    reconfigure (config={})
        Configure HDFS Consumer according to the config parameters.

```

### pyDKB.dataflow.communication.consumer.StreamConsumer module

pyDKB.dataflow.communication.consumer.StreamConsumer

Data consumer implementation for a single stream.

**TODO: think about multiple streams (like a number of named pipes, etc).** Perhaps, even merge this class with FileConsumer.

```

class pyDKB.dataflow.communication.consumer.StreamConsumer.StreamConsumer (config={})
    Bases: pyDKB.dataflow.communication.consumer.Consumer.Consumer

    Data consumer implementation for Stream data source.

    fd = None

    get_source ()
        Get Stream file descriptor.

```

```
get_source_info ()  
    Return current source info.  
  
next_source ()  
    Return None.  
  
    As currently we believe that there is only one input stream  
  
reconfigure (config={})  
    (Re)configure Stream consumer.
```

## pyDKB.dataflow.communication.producer package

Producer submodule init file.

```
class pyDKB.dataflow.communication.producer.ProducerBuilder (config={})  
    Bases: object  
  
    Constructor for Producer instance.  
  
    build (config={})  
        Return constructed producer.  
  
    message_type = None  
  
    producerClass = None  
  
    setDest (dest)  
        Set data destination for the producer.  
  
    setSourceInfoMethod (src_info)  
        Set method to get current source info.  
  
    setType (Type)  
        Set message type for the producer.  
  
    src_info = None
```

## Submodules

### pyDKB.dataflow.communication.producer.FileProducer module

pyDKB.dataflow.communication.producer.FileProducer

Data producer implementation for common (static) files.

**TODO: think about:**

- pipes (better, from the point of StreamProducer)
- multiple parallel dests
- ...

```
class pyDKB.dataflow.communication.producer.FileProducer.FileProducer (config={})  
    Bases: pyDKB.dataflow.communication.producer.Producer.Producer  
  
    Data producer implementation for local file data dest.  
  
    close ()  
        Close opened files and remove temporary one.
```

```

close_file()
    Close current file.

config_dir (config={})
    Configure output directory.

current_file = None

default_dir()
    Get default directory name.

dirname (dirname=None)
    Set/get preferable directory name.

ensure_dir()
    Ensure that current directory for output files exists.

file_info()
    Return output file metadata (name, directory, ...).

get_dest()
    Get destination file descriptor.

get_dest_info()
    Get current destination info.

get_dir()
    Get current directory for output files.

get_filename()
    Return filename, corresponding the source, or timestamp-based.

get_source_info()
    Set current data source, if any.

reconfigure (config={})
    (Re)configure producer according to the config hash.

reset_file()
    Resets current file according to the current source info.

Metadata include:
    • fd – open file descriptor
    • name – file name
    • dir – directory name
    • local_path – local path to the file

set_default_dir()
    Set default directory name.

subdir (base_dir, sub_dir="")
    Construct full path for $subdir of $base_dir.

```

## pyDKB.dataflow.communication.producer.HDFSProducer module

pyDKB.dataflow.communication.producer.HDFSProducer

Data producer implementation for common (static) files in HDFS.

**TODO: think about:**

- pipes (better, from the point of StreamProducer)
- multiple parallel dests
- ...

**class** pyDKB.dataflow.communication.producer.HDFSProducer.**HDFSProducer** (*config={}*)  
Bases: *pyDKB.dataflow.communication.producer.FileProducer.FileProducer*

Data producer implementation for HDFS data dest.

**close\_file** ()  
Close current file and move it to HDFS.

**config\_dir** (*config={}*)  
Configure output directory.

**ensure\_dir** ()  
Ensure that current directory for output files exists.

**file\_info** ()  
Return output file metadata (name, directory, ...).

**set\_default\_dir** ()  
Set default directory name.

**subdir** (*base\_dir, sub\_dir=""*)  
Construct full path for \$sub\_dir of \$base\_dir.

## pyDKB.dataflow.communication.producer.Producer module

pyDKB.dataflow.communication.producer.Producer

**class** pyDKB.dataflow.communication.producer.Producer.**Producer** (*config={}*)  
Bases: *pyDKB.common.LoggableObject.LoggableObject*

Data producer implementation.

**close** ()  
Close opened data stream and data dest.

**config** = **None**

**drop** ()  
Drop buffered messages.

**eop** ()  
Write EOP marker to the current dest.

**flush** ()  
Flush buffered messages to the current dest.

**get\_dest** ()  
Return current destination.

**get\_dest\_info** ()  
Return current dest info.

**get\_stream** (*actualize=True*)  
Get output stream linked to the current dest.

If \$actualize parameter set to True, will try to reset current stream destination; else will use last known destination or None.



```

init_stream()
    Init output stream (without real destination).

message_class()
    Return message class.

message_type = None

reconfigure (config={})
    (Re)initialize producer with stage config arguments.

reset_stream()
    Reset input stream to the current dest.

set_message_type (Type)
    Set input message type.

write (msg)
    Put new message to the current dest (buffer).

exception pyDKB.dataflow.communication.producer.Producer.ProducerException (message=",
                                                    rea-
                                                    son=None)

Bases: pyDKB.dataflow.exceptions.DataflowException

Dataflow Producer exception.

```

### pyDKB.dataflow.communication.producer.StreamProducer module

pyDKB.dataflow.communication.producer.StreamProducer

Data producer implementation for a single stream.

**TODO: think about multiple streams (like a number of named pipes, etc).** Perhaps, even merge this class with FileProducer.

```

class pyDKB.dataflow.communication.producer.StreamProducer.StreamProducer (config={})
    Bases: pyDKB.dataflow.communication.producer.Producer.Producer

    Data producer implementation for Stream data dest.

    fd = None

    get_dest()
        Get Stream file descriptor.

    get_dest_info()
        Return current dest info.

    reconfigure (config={})
        (Re)configure Stream producer.

```

### pyDKB.dataflow.communication.stream package

pyDKB.dataflow.communication.stream

```

class pyDKB.dataflow.communication.stream.StreamBuilder (fd, config={})
    Bases: object

    Constructor for Stream object.

```

```
build (config={})  
    Create instance of Stream.  
  
message_type = None  
  
setStream (stream)  
    Set stream type: 'input' or 'output'.  
  
setType (Type)  
    Set message type for the Stream.  
  
streamClass = None  
  
class pyDKB.dataflow.communication.stream.Stream (fd=None, config={})  
    Bases: pyDKB.common.LoggableObject.LoggableObject  
  
    Abstract class for input/output streams.  
  
    EOM = None  
  
    close ()  
        Close open file descriptors etc.  
  
    configure (config)  
        Stream configuration.  
  
    get_fd ()  
        Return open file descriptor or raise exception.  
  
    message_type ()  
        Get type of the messages in the stream.  
  
    reset (fd, close=True)  
        Reset file descriptor in operation.  
  
        Parameters fd – open file descriptor TODO: IOBase objects  
        Returns previous file descriptor (or None)  
  
    set_message_type (msg_type)  
        Set type of the messages in the stream.  
  
class pyDKB.dataflow.communication.stream.InputStream (fd=None, config={})  
    Bases: pyDKB.dataflow.communication.stream.Stream.Stream  
  
    Implementation of the input stream.  
  
    get_message ()  
        Get next message from the input stream.  
  
        Returns parsed next message, False – parsing failed, None – no messages left  
        Return type pyDKB.dataflow.communication.messages.AbstractMessage, bool, NoneType  
  
    is_readable ()  
        Check if current input stream is readable.  
  
        Returns None – not initialized, False – empty, True – not empty  
        Return type bool, NoneType  
  
    next ()  
        Get next message from the input stream.  
  
        Returns parsed next message, False – parsing failed or unexpected end of stream occurred  
        Return type pyDKB.dataflow.communication.messages.AbstractMessage, bool
```

**parse\_message** (*message*)

Verify and parse input message.

**Parameters** *message* (`pyDKB.dataflow.communication.messages.AbstractMessage`) – message to parse

**Returns** decoded message or False if parsing failed

**Return type** `pyDKB.dataflow.communication.messages.AbstractMessage`, bool

**reset** (*fd*, *close=True*, *force=False*)

Reset current stream with new file descriptor.

Overrides parent method to reset `__iterator` property.

**Parameters**

- **fd** (*file*) – new file descriptor
- **close** (*bool*) – if True, close the old file descriptor
- **force** (*bool*) – if True, force the reset of iterator

**Returns** old file descriptor

**Return type** file

**class** `pyDKB.dataflow.communication.stream.OutputStream` (*fd=None*, *config={}*)

Bases: `pyDKB.dataflow.communication.stream.Stream.Stream`

Implementation of the output stream.

**configure** (*config={}*)

Configure instance.

**drop** ()

Drop buffer without sending messages anywhere.

**eop** ()

Signalize Supervisor about end of process.

**flush** ()

Flush buffer to the output stream.

**msg\_buffer** = []

**write** (*message*)

Add message to the buffer.

## Submodules

### `pyDKB.dataflow.communication.stream.InputStream` module

`pyDKB.dataflow.communication.stream.InputStream`

**class** `pyDKB.dataflow.communication.stream.InputStream.InputStream` (*fd=None*, *config={}*)

Bases: `pyDKB.dataflow.communication.stream.Stream.Stream`

Implementation of the input stream.

**get\_message** ()

Get next message from the input stream.

**Returns** parsed next message, False – parsing failed, None – no messages left

**Return type** `pyDKB.dataflow.communication.messages.AbstractMessage`, bool, NoneType

**is\_readable()**

Check if current input stream is readable.

**Returns** None – not initialized, False – empty, True – not empty

**Return type** bool, NoneType

**next()**

Get next message from the input stream.

**Returns** parsed next message, False – parsing failed or unexpected end of stream occurred

**Return type** `pyDKB.dataflow.communication.messages.AbstractMessage`, bool

**parse\_message(message)**

Verify and parse input message.

**Parameters** **message** (`pyDKB.dataflow.communication.messages.AbstractMessage`) – message to parse

**Returns** decoded message or False if parsing failed

**Return type** `pyDKB.dataflow.communication.messages.AbstractMessage`, bool

**reset(fd, close=True, force=False)**

Reset current stream with new file descriptor.

Overrides parent method to reset `__iterator` property.

**Parameters**

- **fd** (*file*) – new file descriptor
- **close** (*bool*) – if True, close the old file descriptor
- **force** (*bool*) – if True, force the reset of iterator

**Returns** old file descriptor

**Return type** file

## pyDKB.dataflow.communication.stream.OutputStream module

`pyDKB.dataflow.communication.stream.OutputStream`

**class** `pyDKB.dataflow.communication.stream.OutputStream.OutputStream` (*fd=None*,  
*con-*  
*fig={}*)

Bases: `pyDKB.dataflow.communication.stream.Stream.Stream`

Implementation of the output stream.

**configure** (*config={}*)

Configure instance.

**drop()**

Drop buffer without sending messages anywhere.

**eop()**

Signalize Supervisor about end of process.

```
flush()
    Flush buffer to the output stream.

msg_buffer = []

write(message)
    Add message to the buffer.
```

## pyDKB.dataflow.communication.stream.Stream module

pyDKB.dataflow.communication.stream.Stream

**class** pyDKB.dataflow.communication.stream.Stream.**Stream**(fd=None, config={})

Bases: [pyDKB.common.LoggableObject.LoggableObject](#)

Abstract class for input/output streams.

**EOM = None**

```
close()
    Close open file descriptors etc.
```

```
configure(config)
    Stream configuration.
```

```
get_fd()
    Return open file descriptor or raise exception.
```

```
message_type()
    Get type of the messages in the stream.
```

```
reset(fd, close=True)
    Reset file descriptor in operation.
```

**Parameters** fd – open file descriptor TODO: IOBase objects

**Returns** previous file descriptor (or None)

```
set_message_type(msg_type)
    Set type of the messages in the stream.
```

## pyDKB.dataflow.communication.stream.exceptions module

pyDKB.dataflow.communication.stream.exceptions

**exception** pyDKB.dataflow.communication.stream.exceptions.**StreamException**(message="", reason=None)

Bases: [pyDKB.dataflow.exceptions.DataflowException](#)

Exception for Stream operations.

## Submodules

### pyDKB.dataflow.communication.messages module

pyDKB.dataflow.communication.messages

Definition of abstract message class and specific message classes

```
class pyDKB.dataflow.communication.messages.AbstractMessage (message=None)
    Bases: object

    Abstract message

    content ()
        Return message content.

    decode (code)
        Decode original from CODE to TYPE-specific format.

        Raises ValueError

    decoded = None

    encode (code)
        Encode original message from TYPE-specific format to CODE.

        Raises ValueError

    encoded = None

    classmethod extension ()
        Return file extension corresponding to this message type.

    getOriginal ()
        Return original message.

    incompl = None

    incomplete (status=None)
        Set message incomplete marker and/or get previous/current value.

        Parameters status (bool, NoneType) – new status (if not passed, current status is re-
            turned)

        Returns incomplete marker status (previous value, if reset)

        Return type bool

    msg_type = None

    native_types = []

    classmethod typeName ()
        Return message type name as string.

exception pyDKB.dataflow.communication.messages.DecodeUnknownType (code, cls)
    Bases: exceptions.NotImplementedError

    Exception to be thrown when message type is not decodable.

exception pyDKB.dataflow.communication.messages.EncodeUnknownType (code, cls)
    Bases: exceptions.NotImplementedError

    Exception to be thrown when message type is not encodable.

class pyDKB.dataflow.communication.messages.JSONMessage (message=None)
    Bases: pyDKB.dataflow.communication.messages.AbstractMessage

    Message in JSON format.

    decode (code=1)
        Decode original data as JSON.

    encode (code=1)
        Encode JSON as CODE.
```

```

    incompl_key = '_incomplete'
    msg_type = 2
    native_types = [<type 'dict'>, <type 'list'>, <type 'int'>, <type 'float'>]
pyDKB.dataflow.communication.messages.Message(msg_type)
    Return class XXXMessage, where XXX is the passed type.

class pyDKB.dataflow.communication.messages.TTLMessage(message=None)
    Bases: pyDKB.dataflow.communication.messages.AbstractMessage
    Messages in TTL format
    Single message = single TTL statement

    decode(code=1)
        Decode original data as TTL.

        Currently takes text as it is. TODO: check some formal matter to confirm the string is TTL.

    encode(code=1)
        Encode TTL as CODE.

    msg_type = 3
    native_types = [<type 'str'>, <type 'unicode'>]

```

## pyDKB.dataflow.stage package

Stage submodule init file.

```

class pyDKB.dataflow.stage.ProcessorStage(description='DKB Dataflow data processing
stage.')
    Bases: pyDKB.dataflow.stage.AbstractStage.AbstractStage
    Abstract class to implement Processor stages
    Processor stage – is a stage for data processing/transformation.
    Class/instance variable description:
        • communication.consumer.Consumer instance __input
        • Generator object for output file descriptor OR file descriptor (for (s)stream mode)
          __output
        • List of objects to be “stopped” __stoppable

    clear_buffer()
        Drop buffered output messages.

    configure(args=None)
        Configure stage according to the config parameters.

        If $args specified, arguments will be parsed anew.

    defaultArguments()
        Default parser configuration.

    flush_buffer()
        Flush message buffer to the output.

    forward()
        Send EOPMarker to the output stream.

```

**get\_source\_info()**

Get information about current source.

**input()**

Generator for input messages.

Returns iterable object. Every iteration returns single input message to be processed.

**input\_message\_class()**

Get input message class.

**output(message)**

Put the (list of) message(s) to the output buffer.

**output\_message\_class()**

Get output message class.

**static process(stage, input\_message)**

Transform input\_message -> output\_message.

To be implemented individually for every stage. Takes the stage as first argument to allow calling output() from inside the function.

**Return value:** True – processing successfully finished False – processing failed (skip the input message)

**run()**

Run process() for every input() message.

**set\_default\_arguments(ignore\_on\_skip=False, \*\*kwargs)**

Set (or overwrite) default values for arguments.

#### Parameters

- **ignore\_on\_skip** (*bool*) – ignore new default value when the stage is to be skipped
- **<arg\_name>** (*object*) – default value for argument <arg\_name>

**set\_input\_message\_type(Type=None)**

Set input message type.

**set\_output\_message\_type(Type=None)**

Set output message class.

**static skip\_process(stage, input\_message)**

Process input\_message in “skip” processing mode.

Skip mode is turned on with command line parameter *–skip*; in this mode stage is expected to skip its “semantic” part of the message transformation (like changes in data fields format, definition of additional data fields, etc) but perform actions required to keep the dataflow seamless.

The simplest way to achieve this is to send the input message to the output without changes (marking it as “incomplete”), and this is the default implementation of the method.

In some cases it may be necessary to re-implement it (just like *process()*) to keep the dataflow unbroken.

**NOTE: the output messages in “skip” mode MUST be marked as “incomplete”.**

**Parameters** **input\_message** (*pyDKB.messages.AbstractMessage*) – message to process

**Returns** True/False (success/failure)

**Return type** bool



**stop()**  
Finalize all the processes and prepare to exit.

## Submodules

### pyDKB.dataflow.stage.AbstractStage module

Definition of an abstract class for Dataflow Stages.

**class** pyDKB.dataflow.stage.AbstractStage.**AbstractStage** (*description='DKB Dataflow stage'*)

Bases: *pyDKB.common.LoggableObject.LoggableObject*

Class/instance variable description: \* Argument parser (argparse.ArgumentParser)

**\_\_parser**

- Original default argument values (dict)
- (filled by **add\_argument()** method) **\_default\_args**
- **Parsed arguments (argparse.Namespace)** **ARGS**
- **Stage config parser (ConfigParser.SafeConfigParser)** **\_\_config**
- **Stage custom config (defaultdict(defaultdict(str)))** **CONFIG**

**add\_argument** (\*args, \*\*kwargs)  
Add specific (not common) arguments.

**args\_error** (message)  
Output USAGE, error message and exit with code 2.

**config\_error** (message='Failed to read config file:')  
Output error message and exit with code 3.

**defaultArguments** ()  
Config argument parser with parameters common for all stages.

**log\_configuration** ()  
Log stage configuration.

**output\_error** (message=None, exc\_info=None)  
Output traceback of the passed (or last) error with *message*.

**parse\_args** (args)  
Parse arguments and set dependant arguments if needed.

**Exits in case of error with code:** 2 – failed to parse arguments 3 – failed to read config file

**print\_usage** (fd=<open file '<stderr>', mode 'w'>)  
Print usage message.

**read\_config** ()  
Read stage custom config file.

**Returns** (True|False)

**reset\_default\_arguments** (args=None)  
Reset default argument values to the original ones.

Original default value is a value passed to the **add\_argument()** method.

**Parameters** `args` (*list*, *NoneType*) – list of arguments to be reset. If not specified or set to `None`, all known arguments will be reset

**run** ()  
Run the stage.

**set\_default\_arguments** (*\*\*kwargs*)  
Set (or overwrite) default values for arguments.

**Parameters** `<arg_name>` (*object*) – default value for argument `<arg_name>`

**set\_error** (*err\_type*, *err\_val*, *err\_trace*)  
Set object `_err` variable from the last error info.

**stop** ()  
Stop running processes and output error information.

## pyDKB.dataflow.stage.ProcessorStage module

Definition of an abstract class for Dataflow Data Processing Stages.

**USAGE:** ProcessorStage [`<options>`] [`<input files>`]

**OPTIONS:**

<b>-s, --source</b>	{flsh} - where to get data from: local (f)iles, (s)tdin, (h)dfs
<b>-i, --input-dir</b>	DIR - base directory for relative input file names (for local and HDFS sources). If <code>&lt;input files&gt;</code> not specified, all files from the directory will be taken as the input.
<b>-d, --dest</b>	{flsh} - where to send data to: local (f)iles, (s)tdout, (h)dfs
<b>-o, --output-dir</b>	DIR - base directory for output files (for local and HDFS sources)
<b>--hdfs</b>	• equivalent to “ <code>--source h --dest h</code> ”
<b>-m, --mode</b>	MODE - MODE: (f)ile = <code>--source f</code>  <b><code>--dest f</code> (can be rewritten with ‘s’ or ‘h’) (s)tream = <code>--source s</code> (can be rewritten with ‘h’) <code>--dest s</code> (m)apreduce = <code>--source s</code> (can be rewritten with ‘h’) <code>--dest s</code></b>

**class** pyDKB.dataflow.stage.ProcessorStage.**ProcessorStage** (*description*=*'DKB  
Dataflow data processing  
stage.'*)

Bases: *pyDKB.dataflow.stage.AbstractStage.AbstractStage*

Abstract class to implement Processor stages

Processor stage – is a stage for data processing/transformation.

Class/instance variable description:

- **communication.consumer.Consumer instance** `__input`
- Generator object for output file descriptor OR file descriptor (for (s)stream mode)  
`__output`
- **List of objects to be “stopped”** `__stoppable`

**clear\_buffer()**  
Drop buffered output messages.

**configure** (*args=None*)  
Configure stage according to the config parameters.  
If \$args specified, arguments will be parsed anew.

**defaultArguments()**  
Default parser configuration.

**flush\_buffer()**  
Flush message buffer to the output.

**forward()**  
Send EOPMarker to the output stream.

**get\_source\_info()**  
Get information about current source.

**input()**  
Generator for input messages.  
Returns iterable object. Every iteration returns single input message to be processed.

**input\_message\_class()**  
Get input message class.

**output** (*message*)  
Put the (list of) message(s) to the output buffer.

**output\_message\_class()**  
Get output message class.

**static process** (*stage, input\_message*)  
Transform input\_message -> output\_message.  
To be implemented individually for every stage. Takes the stage as first argument to allow calling output() from inside the function.

**Return value:** True – processing successfully finished False – processing failed (skip the input message)

**run()**  
Run process() for every input() message.

**set\_default\_arguments** (*ignore\_on\_skip=False, \*\*kwargs*)  
Set (or overwrite) default values for arguments.

**Parameters**

- **ignore\_on\_skip** (*bool*) – ignore new default value when the stage is to be skipped
- **<arg\_name>** (*object*) – default value for argument <arg\_name>

**set\_input\_message\_type** (*Type=None*)  
Set input message type.

**set\_output\_message\_type** (*Type=None*)

Set output message class.

**static skip\_process** (*stage, input\_message*)

Process input\_message in “skip” processing mode.

Skip mode is turned on with command line parameter *–skip*; in this mode stage is expected to skip its “semantic” part of the message transformation (like changes in data fields format, definition of additional data fields, etc) but perform actions required to keep the dataflow seamless.

The simplest way to achieve this is to send the input message to the output without changes (marking it as “incomplete”), and this is the default implementation of the method.

In some cases it may be necessary to re-implement it (just like *process()*) to keep the dataflow unbroken.

**NOTE: the output messages in “skip” mode MUST be marked as “incomplete”.**

**Parameters** **input\_message** (*pyDKB.messages.AbstractMessage*) – message to process

**Returns** True/False (success/failure)

**Return type** bool

**stop** ()

Finalize all the processes and prepare to exit.

## Submodules

### pyDKB.dataflow.cds module

Extended CDSInvenioConnector allowing us to login via Kerberos

**class** `pyDKB.dataflow.cds.CDSInvenioConnector` (*\*args*)  
Bases: `invenio_client.contrib.cds.CDSInvenioConnector`

CDSInvenioConnector which closes the browser in most cases.

**delete** (*restore\_handlers=True*)

**handlers** = **False**

**kill** (*signum, frame*)

Run del and propagate signal.

**orig\_handlers** = {}

**class** `pyDKB.dataflow.cds.KerberizedCDSInvenioConnector` (*login='user', pass-*  
*word='password'*)

Bases: `pyDKB.dataflow.cds.CDSInvenioConnector`

Represents same CDSInvenioConnector, but this one is aware about SPNEGO: Simple and Protected GSSAPI Negotiation Mechanism

### pyDKB.dataflow.dkbID module

Utils to generate unique yet meaningful identifier for DKB objects.

`pyDKB.dataflow.dkbID.dkbID` (*json\_data, data\_type*)

Return unique identifier for object of TYPE based on DATA.

## pyDKB.dataflow.exceptions module

Definition of DKB Dataflow exceptions

**exception** pyDKB.dataflow.exceptions.**DataflowException** (*message=**"*, *reason=None*)

Bases: exceptions.Exception

Base Exception for Dataflow modules.

**reason = None**

## pyDKB.dataflow.types module

Type definitions for library objects.



## Stages

### 2.1 Stage 055

Stage for converting JSON documents (output of stage 015) into TTL documents (input for stage 060).

Initial JSON document should have the following structure:

```
{
  "GLANCE": {},
  "CDS" : {},
  "dkbID" : ...,
  "supporting_notes": [
    {
      "GLANCE": {},
      "CDS": {},
      "dkbID": ...,
    },
    {
      ...
    }
  ]
}
```

Some functions accept specific parts of this JSON - for example, if 'data' variable contains the initial JSON then "CDS' part of the initial JSON" means "data.get('CDS')".

Resulting TTL file has the following structure:

```
PAPER a atlas:Paper .
PAPER atlas:hasGLANCE_ID __ .
PAPER atlas:hasShortTitle __ .
PAPER atlas:hasFullTitle __ .
PAPER atlas:hasRefCode __ .
PAPER atlas:hasCreationDate __ .
PAPER atlas:hasCDSReportNumber __ .
PAPER atlas:hasCDSInternal __ .
PAPER atlas:hasCDS_ID __ .
PAPER atlas:hasAbstract __ .
PAPER atlas:hasArXivCode __ .
PAPER atlas:hasFullTitle __ .
PAPER atlas:hasDOI __ .
PAPER atlas:hasKeyword __ .
JOURNAL_ISSUE a atlas:JournalIssue .
JOURNAL_ISSUE atlas:hasTitle __ .
JOURNAL_ISSUE atlas:hasVolume __ .
```

```
JOURNAL_ISSUE atlas:hasYear __ .
JOURNAL_ISSUE atlas:containsPublication> PAPER .
SUPPORTING_DOCUMENT a atlas:SupportingDocument .
SUPPORTING_DOCUMENT atlas:hasGLANCE_ID __ .
SUPPORTING_DOCUMENT atlas:hasLabel __ .
SUPPORTING_DOCUMENT atlas:hasURL __ .
SUPPORTING_DOCUMENT atlas:hasCreationDate __ .
SUPPORTING_DOCUMENT atlas:hasCDSInternal __ .
SUPPORTING_DOCUMENT atlas:hasCDS_ID __ .
SUPPORTING_DOCUMENT atlas:hasAbstract __ .
SUPPORTING_DOCUMENT atlas:hasKeyword __ .
PAPER atlas:isBasedOn SUPPORTING_DOCUMENT .
```

**TODO: This module doesn't convert authors metadata.** This task is still under consideration.

055\_documents2TTL.documents2ttl.**abstract\_extraction**(data)

Extract abstract from JSON.

**Parameters** data (dict) – 'CDS' part of the initial JSON

**Returns** abstract or None if it was not found

**Return type** str or NoneType

055\_documents2TTL.documents2ttl.**arxiv\_extraction**(data)

Extract arXiv code from JSON.

**Parameters** data (dict) – 'CDS' part of the initial JSON

**Returns** arXiv code or None if it was not found

**Return type** str or NoneType

055\_documents2TTL.documents2ttl.**cds\_id\_extraction**(data)

Extract CDS id from JSON.

**Parameters** data (dict) – 'CDS' part of the initial JSON

**Returns** CDS id or None if it was not found

**Return type** int or NoneType

055\_documents2TTL.documents2ttl.**cds\_internal\_extraction**(data)

Extract CDS internal report number parameter from JSON.

**Parameters** data (dict) – 'CDS' part of the initial JSON

**Returns** CDS internal report number or None if it was not found

**Return type** unicode or NoneType

055\_documents2TTL.documents2ttl.**cds\_parameter\_extraction**(param\_name, json\_data)

Extract CDS parameter value from JSON.

**Parameters**

- **param\_name** (str) – name of the parameter, defined in \*\_CDS\_ATTRS dict
- **json\_data** (dict) – 'CDS' part of the initial JSON

**Returns** parameter value or None if it was not found

**Return type** int, str, NoneType

055\_documents2TTL.documents2ttl.**creation\_date\_extraction**(data)

Extract creation date from JSON.



**Parameters** **data** (*dict*) – ‘CDS’ part of the initial JSON

**Returns** creation date or None if it was not found

**Return type** str or NoneType

055\_documents2TTL.documents2ttl.**define\_globals** (*args*)

Define global variables for further usage in other functions.

Global variables GRAPH and ONTOLOGY are defined, their values are received from the command line arguments via argparse.

**Parameters** **args** (*argparse.Namespace*) – stage arguments

055\_documents2TTL.documents2ttl.**document\_cds** (*data, doc\_iri, cds\_attrs*)

Convert CDS metadata from JSON to TTL.

**Parameters**

- **data** (*dict*) – ‘CDS’ part of the initial JSON
- **doc\_iri** (*str*) – document IRI for current graph
- **cds\_attrs** (*list*) – PAPER\_CDS\_ATTRS | NOTE\_CDS\_ATTRS

**Returns** TTL string with CDS metadata

**Return type** str

055\_documents2TTL.documents2ttl.**document\_glance** (*data, doc\_iri, glance\_attrs*)

Convert GLANCE metadata from JSON to TTL.

**Parameters**

- **data** (*dict*) – ‘GLANCE’ part of the initial JSON
- **doc\_iri** (*str*) – document IRI for current graph
- **glance\_attrs** (*list*) – PAPER\_GLANCE\_ATTRS | NOTE\_GLANCE\_ATTRS

**Returns** TTL string with GLANCE metadata

**Return type** str

055\_documents2TTL.documents2ttl.**document\_links** (*data*)

Construct TTL sentences to link paper to its supporting documents.

The result looks as following:

```
PAPER atlas:isBasedOn SUPPORTING_DOCUMENT
```

**Parameters** **data** (*dict*) – initial JSON

**Returns** TTL string with links

**Return type** str

055\_documents2TTL.documents2ttl.**doi2ttl** (*doi, doc\_iri*)

Convert DOI parameter from JSON to TTL.

**Parameters**

- **doi** (*str, unicode or list*) – ‘doi’ part of the initial JSON
- **doc\_iri** (*str*) – document IRI for current graph

**Returns** TTL string with DOI

**Return type** str

055\_documents2TTL.documents2ttl.**fix\_list\_values** (*list\_vals*)

Apply fix\_string to each item in a list.

**Parameters** **list\_vals** (*list*) – list with strings to be fixed

**Returns** list with fixed strings

**Return type** list

055\_documents2TTL.documents2ttl.**fix\_string** (*wrong\_string*)

Fix escape sequences in a string.

**Parameters** **wrong\_string** (*object*) – string to be fixed, or any non-string object.

**Returns** fixed string, or unchanged non-string object

**Return type** object

055\_documents2TTL.documents2ttl.**generate\_journal\_id** (*journal\_dict*)

Generate a journal issue ID based on title, volume and year.

**Parameters** **journal\_dict** (*dict*) – journal parameters

**Returns** journal ID

**Return type** str

055\_documents2TTL.documents2ttl.**get\_document\_iri** (*doc\_id*)

Construct an IRI for a document.

**Parameters** **doc\_id** (*str*) – document id

**Returns** IRI

**Return type** str

055\_documents2TTL.documents2ttl.**glance\_parameter\_extraction** (*param\_name*,  
*json\_data*)

Extract a GLANCE parameter value from JSON.

**Parameters**

- **param\_name** (*str*) – name of the parameter
- **json\_data** (*dict*) – ‘GLANCE’ part of the initial JSON

**Returns** parameter value or None if it was not found

**Return type** str, unicode, NoneType

055\_documents2TTL.documents2ttl.**keywords2ttl** (*keywords*, *doc\_iri*)

Convert keywords from JSON to TTL.

**Parameters**

- **keywords** (*dict or list of dicts*) – ‘keywords’ part of the initial JSON
- **doc\_iri** (*str*) – document IRI for current graph

**Returns** TTL string with keywords

**Return type** str

055\_documents2TTL.documents2ttl.**main** (*argv*)

Parse command line arguments and run the stage.

**Parameters** **argv** (*list*) – arguments

`055_documents2TTL.documents2ttl.process(stage, msg)`

Transform input JSON message into TTL documents.

Implementation of `ProcessorStage.process()` for hooking the stage into DKB workflow. Output message containing the TTL result is generated in this function.

**Parameters**

- **stage** (`pyDKB.dataflow.stage.ProcessorStage`) – stage instance
- **msg** (`pyDKB.dataflow.communication.messages.JSONMessage`) – input message with initial JSON

**Returns** True

**Return type** bool

`055_documents2TTL.documents2ttl.process_journals(data, doc_iri)`

Convert journal data from JSON to TTL.

**Parameters**

- **data** (`list`, `dict`) – ‘CDS’ part of the initial JSON
- **doc\_iri** (`str`) – document IRI for current graph

**Returns** TTL string with journal issue with connection to paper

**Return type** str

`055_documents2TTL.documents2ttl.report_number_extraction(data)`

Extract report number from JSON.

**Parameters** **data** (`dict`) – ‘CDS’ part of the initial JSON

**Returns** report number or None if it was not found

**Return type** unicode or NoneType

`055_documents2TTL.documents2ttl.title_extraction(data)`

Extracting title from JSON.

**Parameters** **data** (`dict`) – ‘CDS’ part of the initial JSON

**Returns** title or None if it was not found

**Return type** str or NoneType



## Indices and tables

- `genindex`
- `modindex`
- `search`



## PYTHON MODULE INDEX

### 0

055\_documents2TTL.documents2ttl, [27](#)

### p

pyDKB, [1](#)

pyDKB.common, [3](#)

pyDKB.common.custom\_readline, [4](#)

pyDKB.common.exceptions, [5](#)

pyDKB.common.hdfs, [5](#)

pyDKB.common.json\_utils, [6](#)

pyDKB.common.LoggableObject, [3](#)

pyDKB.common.misc, [6](#)

pyDKB.common.Type, [3](#)

pyDKB.common.types, [7](#)

pyDKB.dataflow, [7](#)

pyDKB.dataflow.cds, [24](#)

pyDKB.dataflow.communication, [7](#)

pyDKB.dataflow.communication.consumer,  
[7](#)

pyDKB.dataflow.communication.consumer.Consumer,  
[7](#)

pyDKB.dataflow.communication.consumer.FileConsumer,  
[8](#)

pyDKB.dataflow.communication.consumer.HDFSConsumer,  
[9](#)

pyDKB.dataflow.communication.consumer.StreamConsumer,  
[9](#)

pyDKB.dataflow.communication.messages,  
[17](#)

pyDKB.dataflow.communication.producer,  
[10](#)

pyDKB.dataflow.communication.producer.FileProducer,  
[10](#)

pyDKB.dataflow.communication.producer.HDFSProducer,  
[11](#)

pyDKB.dataflow.communication.producer.Producer,  
[12](#)

pyDKB.dataflow.communication.producer.StreamProducer,  
[13](#)

pyDKB.dataflow.communication.stream, [13](#)

pyDKB.dataflow.communication.stream.exceptions,  
[17](#)

pyDKB.dataflow.communication.stream.InputStream,  
[15](#)

pyDKB.dataflow.communication.stream.OutputStream,  
[16](#)

pyDKB.dataflow.communication.stream.Stream,  
[17](#)

pyDKB.dataflow.dkbID, [24](#)

pyDKB.dataflow.exceptions, [25](#)

pyDKB.dataflow.stage, [19](#)

pyDKB.dataflow.stage.AbstractStage, [21](#)

pyDKB.dataflow.stage.ProcessorStage, [22](#)

pyDKB.dataflow.types, [25](#)





# INDEX

## Symbols

055\_documents2TTL.documents2ttl (module), 27

## A

abstract\_extraction() (in module 055\_documents2TTL.documents2ttl), 28

AbstractMessage (class in pyDKB.dataflow.communication.messages), 17

AbstractStage (class in pyDKB.dataflow.stage.AbstractStage), 21

add() (pyDKB.common.Type.Type method), 3

add\_argument() (pyDKB.dataflow.stage.AbstractStage.AbstractStage method), 21

args\_error() (pyDKB.dataflow.stage.AbstractStage.AbstractStage method), 21

arxiv\_extraction() (in module 055\_documents2TTL.documents2ttl), 28

## B

basename() (in module pyDKB.common.hdfs), 5

build() (pyDKB.dataflow.communication.consumer.ConsumerBuilder method), 7

build() (pyDKB.dataflow.communication.producer.ProducerBuilder method), 10

build() (pyDKB.dataflow.communication.stream.StreamBuilder method), 13

## C

cds\_id\_extraction() (in module 055\_documents2TTL.documents2ttl), 28

cds\_internal\_extraction() (in module 055\_documents2TTL.documents2ttl), 28

cds\_parameter\_extraction() (in module 055\_documents2TTL.documents2ttl), 28

CDSInvenioConnector (class in pyDKB.dataflow.cds), 24

check\_stderr() (in module pyDKB.common.hdfs), 5

clear\_buffer() (pyDKB.dataflow.stage.ProcessorStage method), 19

clear\_buffer() (pyDKB.dataflow.stage.ProcessorStage.ProcessorStage method), 23

close() (pyDKB.dataflow.communication.consumer.Consumer.Consumer method), 7

close() (pyDKB.dataflow.communication.producer.FileProducer.FileProducer method), 10

close() (pyDKB.dataflow.communication.producer.Producer.Producer method), 12

close() (pyDKB.dataflow.communication.stream.Stream method), 14

close() (pyDKB.dataflow.communication.stream.Stream.Stream method), 17

close\_file() (pyDKB.dataflow.communication.producer.FileProducer.FileProducer method), 10

close\_file() (pyDKB.dataflow.communication.producer.HDFSProducer.HDFSProducer method), 12

config (pyDKB.dataflow.communication.consumer.Consumer.Consumer attribute), 7

config (pyDKB.dataflow.communication.producer.Producer.Producer attribute), 12

config\_dir() (pyDKB.dataflow.communication.producer.FileProducer.FileProducer method), 11

config\_dir() (pyDKB.dataflow.communication.producer.HDFSProducer.HDFSProducer method), 12

config\_error() (pyDKB.dataflow.stage.AbstractStage.AbstractStage method), 21

configure() (pyDKB.dataflow.communication.stream.OutputStream.OutputStream method), 15

configure() (pyDKB.dataflow.communication.stream.OutputStream.OutputStream method), 16

configure() (pyDKB.dataflow.communication.stream.Stream method), 14

configure() (pyDKB.dataflow.communication.stream.Stream.Stream method), 17

configure() (pyDKB.dataflow.stage.ProcessorStage method), 19

configure() (pyDKB.dataflow.stage.ProcessorStage.ProcessorStage method), 23

Consumer (class in pyDKB.dataflow.communication.consumer.Consumer), 7

ConsumerBuilder (class in pyDKB.dataflow.communication.consumer.ConsumerBuilder), 7

consumerClass (pyDKB.dataflow.communication.consumer.Consumer), 7  
 ConsumerException, 8  
 content() (pyDKB.dataflow.communication.messages.AbstractMessage method), 16  
 creation\_date\_extraction() (in module 055\_documents2TTL.documents2ttl), 28  
 current\_file (pyDKB.dataflow.communication.consumer.FileConsumer.FileConsumer attribute), 9  
 current\_file (pyDKB.dataflow.communication.producer.FileProducer.FileProducer attribute), 11  
 custom\_readline() (in module py-DKB.common.custom\_readline), 4  
**D**  
 DataflowException, 25  
 decode() (pyDKB.dataflow.communication.messages.AbstractMessage method), 18  
 decode() (pyDKB.dataflow.communication.messages.JSONMessage method), 18  
 decode() (pyDKB.dataflow.communication.messages.TTLMessage method), 19  
 decoded (pyDKB.dataflow.communication.messages.AbstractMessage attribute), 18  
 DecodeUnknownType, 18  
 default\_dir() (pyDKB.dataflow.communication.producer.FileProducer.FileProducer method), 11  
 defaultArguments() (py-DKB.dataflow.stage.AbstractStage.AbstractStage method), 21  
 defaultArguments() (py-DKB.dataflow.stage.ProcessorStage method), 19  
 defaultArguments() (py-DKB.dataflow.stage.ProcessorStage.ProcessorStage method), 23  
 define\_globals() (in module 055\_documents2TTL.documents2ttl), 29  
 delete() (pyDKB.dataflow.cds.CDSInvenioConnector method), 24  
 dirname() (in module pyDKB.common.hdfs), 5  
 dirname() (pyDKB.dataflow.communication.producer.FileProducer.FileProducer method), 11  
 dkbID() (in module pyDKB.dataflow.dkbID), 24  
 document\_cds() (in module 055\_documents2TTL.documents2ttl), 29  
 document\_glance() (in module 055\_documents2TTL.documents2ttl), 29  
 document\_links() (in module 055\_documents2TTL.documents2ttl), 29  
 doi2ttl() (in module 055\_documents2TTL.documents2ttl), 29  
 drop() (pyDKB.dataflow.communication.producer.Producer.Producer method), 12  
 drop() (pyDKB.dataflow.communication.stream.OutputStream.OutputStream method), 15  
 drop() (pyDKB.dataflow.communication.stream.OutputStream.OutputStream method), 16  
**E**  
 encode() (pyDKB.dataflow.communication.messages.AbstractMessage method), 18  
 encode() (pyDKB.dataflow.communication.messages.JSONMessage method), 19  
 encode() (pyDKB.dataflow.communication.messages.TTLMessage method), 19  
 encoded (pyDKB.dataflow.communication.messages.AbstractMessage attribute), 18  
 EncodeUnknownType, 18  
 ensure\_argparse\_arg\_name() (in module py-DKB.common.misc), 6  
 ensure\_dir() (pyDKB.dataflow.communication.producer.FileProducer.FileProducer method), 11  
 ensure\_dir() (pyDKB.dataflow.communication.producer.HDFSProducer.HDFSProducer method), 12  
 EOM (pyDKB.dataflow.communication.stream.Stream attribute), 14  
 EOM (pyDKB.dataflow.communication.stream.Stream.Stream attribute), 17  
 eop() (pyDKB.dataflow.communication.producer.Producer.Producer method), 12  
 eop() (pyDKB.dataflow.communication.stream.OutputStream.OutputStream method), 15  
 eop() (pyDKB.dataflow.communication.stream.OutputStream.OutputStream method), 16  
 extension() (pyDKB.dataflow.communication.messages.AbstractMessage class method), 18  
**F**  
 fd (pyDKB.dataflow.communication.consumer.StreamConsumer.StreamConsumer attribute), 9  
 fd (pyDKB.dataflow.communication.producer.StreamProducer.StreamProducer attribute), 13  
 File() (in module pyDKB.common.hdfs), 5  
 file\_info() (pyDKB.dataflow.communication.producer.FileProducer.FileProducer method), 11  
 file\_info() (pyDKB.dataflow.communication.producer.HDFSProducer.HDFSProducer method), 12  
 FileConsumer (class in py-DKB.dataflow.communication.consumer.FileConsumer), 8  
 FileProducer (class in py-DKB.dataflow.communication.producer.FileProducer), 10  
 fix\_list\_values() (in module 055\_documents2TTL.documents2ttl), 29  
 fix\_list\_values() (in module 055\_documents2TTL.documents2ttl), 30



`InputStream` (class in `pyDKB.dataflow.communication.stream`), 14

`InputStream` (class in `pyDKB.dataflow.communication.stream.InputStream`), 15

`is_readable()` (`pyDKB.dataflow.communication.stream.InputStream` method), 14

`is_readable()` (`pyDKB.dataflow.communication.stream.InputStream.InputStream` method), 16

## J

`join()` (in module `pyDKB.common.hdfs`), 5

`JSONMessage` (class in `pyDKB.dataflow.communication.messages`), 18

## K

`KerberizedCDSInvenioConnector` (class in `pyDKB.dataflow.cds`), 24

`keywords2ttl()` (in module `055_documents2TTL.documents2ttl`), 30

`kill()` (`pyDKB.dataflow.cds.CDSInvenioConnector` method), 24

## L

`listdir()` (in module `pyDKB.common.hdfs`), 5

`log()` (in module `pyDKB.common.misc`), 6

`log()` (`pyDKB.common.LoggableObject.LoggableObject` class method), 3

`log_configuration()` (`pyDKB.dataflow.stage.AbstractStage.AbstractStage` method), 21

`LoggableObject` (class in `pyDKB.common.LoggableObject`), 3

## M

`main()` (in module `055_documents2TTL.documents2ttl`), 30

`makedirs()` (in module `pyDKB.common.hdfs`), 5

`member()` (`pyDKB.common.Type.Type` method), 4

`memberName()` (`pyDKB.common.Type.Type` method), 4

`Message()` (in module `pyDKB.dataflow.communication`), 7

`Message()` (in module `pyDKB.dataflow.communication.messages`), 19

`message_class()` (`pyDKB.dataflow.communication.consumer.Consumer` method), 8

`message_class()` (`pyDKB.dataflow.communication.producer.Producer` method), 13

`message_type` (`pyDKB.dataflow.communication.consumer.Consumer` attribute), 8

`message_type` (`pyDKB.dataflow.communication.producer.Producer` attribute), 13

`message_type` (`pyDKB.dataflow.communication.producer.ProducerBuilder` attribute), 10

`message_type` (`pyDKB.dataflow.communication.stream.StreamBuilder` attribute), 14

`message_type()` (`pyDKB.dataflow.communication.stream.Stream` method), 14

`message_type()` (`pyDKB.dataflow.communication.stream.Stream.Stream` method), 17

`movefile()` (in module `pyDKB.common.hdfs`), 5

`msg_buffer` (`pyDKB.dataflow.communication.stream.OutputStream` attribute), 15

`msg_buffer` (`pyDKB.dataflow.communication.stream.OutputStream.OutputStream` attribute), 17

`msg_type` (`pyDKB.dataflow.communication.messages.AbstractMessage` attribute), 18

`msg_type` (`pyDKB.dataflow.communication.messages.JSONMessage` attribute), 19

`msg_type` (`pyDKB.dataflow.communication.messages.TTLMessage` attribute), 19

## N

`native_types` (`pyDKB.dataflow.communication.messages.AbstractMessage` attribute), 18

`native_types` (`pyDKB.dataflow.communication.messages.JSONMessage` attribute), 19

`native_types` (`pyDKB.dataflow.communication.messages.TTLMessage` attribute), 19

`nestedKeys()` (in module `pyDKB.common.json_utils`), 6

`next()` (`pyDKB.dataflow.communication.consumer.Consumer.Consumer` method), 8

`next()` (`pyDKB.dataflow.communication.stream.InputStream` method), 14

`next()` (`pyDKB.dataflow.communication.stream.InputStream.InputStream` method), 16

`next_source()` (`pyDKB.dataflow.communication.consumer.FileConsumer.FileConsumer` method), 9

`next_source()` (`pyDKB.dataflow.communication.consumer.StreamConsumer.StreamConsumer` method), 10

## O

`orig_handlers` (`pyDKB.dataflow.cds.CDSInvenioConnector` attribute), 24

`output()` (`pyDKB.dataflow.stage.ProcessorStage` method), 20

`output()` (`pyDKB.dataflow.stage.ProcessorStage.ProcessorStage` method), 23

`output_message_class()` (`pyDKB.dataflow.stage.ProcessorStage` method), 23

`output_message_class()` (`pyDKB.dataflow.stage.ProcessorStage.ProcessorStage` method), 23

OutputStream (class in py-DKB.dataflow.communication.consumer.HDFSConsumer (module), 9)  
 OutputStream (class in py-DKB.dataflow.communication.consumer.StreamConsumer (module), 9)  
 16  
 pyDKB.dataflow.communication.messages (module), 17  
 pyDKB.dataflow.communication.producer (module), 10  
 pyDKB.dataflow.communication.producer.FileProducer (module), 10

## P

parse\_args() (pyDKB.dataflow.stage.AbstractStage.AbstractStage (module), 10)  
 method), 21  
 parse\_message() (pyDKB.dataflow.communication.stream.InputStream (module), 11)  
 method), 14  
 parse\_message() (pyDKB.dataflow.communication.stream.InputStream (module), 12)  
 method), 16  
 print\_usage() (pyDKB.dataflow.stage.AbstractStage.AbstractStage (module), 13)  
 method), 21  
 process() (in module 055\_documents2TTL.documents2ttl), 30  
 process() (pyDKB.dataflow.stage.ProcessorStage static (module), 15)  
 method), 20  
 process() (pyDKB.dataflow.stage.ProcessorStage.ProcessorStage static (module), 16)  
 static method), 23  
 process\_journals() (in module 055\_documents2TTL.documents2ttl), 31  
 ProcessorStage (class in pyDKB.dataflow.stage), 19  
 ProcessorStage (class in py-DKB.dataflow.stage.ProcessorStage), 22  
 Producer (class in py-DKB.dataflow.communication.producer.Producer (module), 12)  
 12  
 ProducerBuilder (class in py-DKB.dataflow.communication.producer), 10  
 10

producerClass (pyDKB.dataflow.communication.producer.ProducerBuilder (module), 21)  
 attribute), 10  
 ProducerException, 13  
 putfile() (in module pyDKB.common.hdfs), 5  
 pyDKB (module), 1  
 pyDKB.common (module), 3  
 pyDKB.common.custom\_readline (module), 4  
 pyDKB.common.exceptions (module), 5  
 pyDKB.common.hdfs (module), 5  
 pyDKB.common.json\_utils (module), 6  
 pyDKB.common.LoggableObject (module), 3  
 pyDKB.common.misc (module), 6  
 pyDKB.common.Type (module), 3  
 pyDKB.common.types (module), 7  
 pyDKB.dataflow (module), 7  
 pyDKB.dataflow.cds (module), 24  
 pyDKB.dataflow.communication (module), 7  
 pyDKB.dataflow.communication.consumer (module), 7  
 pyDKB.dataflow.communication.consumer.Consumer (module), 7  
 pyDKB.dataflow.communication.consumer.FileConsumer (module), 8  
 8  
 read\_config() (pyDKB.dataflow.stage.AbstractStage.AbstractStage (module), 21)  
 method), 21  
 reason (pyDKB.dataflow.exceptions.DataflowException (module), 25)  
 attribute), 25  
 reconfigure() (pyDKB.dataflow.communication.consumer.Consumer.Consumer (module), 8)  
 method), 8  
 reconfigure() (pyDKB.dataflow.communication.consumer.FileConsumer.FileConsumer (module), 9)  
 method), 9  
 reconfigure() (pyDKB.dataflow.communication.consumer.HDFSConsumer.HDFSConsumer (module), 9)  
 method), 9  
 reconfigure() (pyDKB.dataflow.communication.consumer.StreamConsumer.StreamConsumer (module), 10)  
 method), 10  
 reconfigure() (pyDKB.dataflow.communication.producer.FileProducer.FileProducer (module), 11)  
 method), 11  
 reconfigure() (pyDKB.dataflow.communication.producer.Producer.Producer (module), 13)  
 method), 13  
 reconfigure() (pyDKB.dataflow.communication.producer.StreamProducer.StreamProducer (module), 13)  
 method), 13  
 report\_number\_extraction() (in module 055\_documents2TTL.documents2ttl), 31  
 reset() (pyDKB.dataflow.communication.stream.InputStream (module), 15)  
 method), 15

## R



[reset\(\)](#) (pyDKB.dataflow.communication.stream.InputStream.InputStream method), 17  
[reset\(\)](#) (pyDKB.dataflow.communication.stream.Stream method), 14  
[reset\(\)](#) (pyDKB.dataflow.communication.stream.Stream.Stream method), 17  
[reset\\_default\\_arguments\(\)](#) (pyDKB.dataflow.stage.AbstractStage.AbstractStage method), 21  
[reset\\_file\(\)](#) (pyDKB.dataflow.communication.producer.FileProducer.FileProducer method), 11  
[reset\\_stream\(\)](#) (pyDKB.dataflow.communication.consumer.Consumer.Consumer method), 8  
[reset\\_stream\(\)](#) (pyDKB.dataflow.communication.producer.Producer.Producer method), 13  
[run\(\)](#) (pyDKB.dataflow.stage.AbstractStage.AbstractStage method), 22  
[run\(\)](#) (pyDKB.dataflow.stage.ProcessorStage.ProcessorStage method), 20  
[run\(\)](#) (pyDKB.dataflow.stage.ProcessorStage.ProcessorStage method), 23  
**S**  
[set\\_default\\_arguments\(\)](#) (pyDKB.dataflow.stage.AbstractStage.AbstractStage method), 22  
[set\\_default\\_arguments\(\)](#) (pyDKB.dataflow.stage.ProcessorStage.ProcessorStage method), 20  
[set\\_default\\_arguments\(\)](#) (pyDKB.dataflow.stage.ProcessorStage.ProcessorStage method), 23  
[set\\_default\\_dir\(\)](#) (pyDKB.dataflow.communication.producer.FileProducer.FileProducer method), 11  
[set\\_default\\_dir\(\)](#) (pyDKB.dataflow.communication.producer.HDFSProducer.HDFSProducer method), 12  
[set\\_error\(\)](#) (pyDKB.dataflow.stage.AbstractStage.AbstractStage method), 22  
[set\\_input\\_message\\_type\(\)](#) (pyDKB.dataflow.stage.ProcessorStage.ProcessorStage method), 20  
[set\\_input\\_message\\_type\(\)](#) (pyDKB.dataflow.stage.ProcessorStage.ProcessorStage method), 23  
[set\\_message\\_type\(\)](#) (pyDKB.dataflow.communication.consumer.Consumer.Consumer method), 8  
[set\\_message\\_type\(\)](#) (pyDKB.dataflow.communication.producer.Producer.Producer method), 13  
[set\\_message\\_type\(\)](#) (pyDKB.dataflow.communication.stream.Stream method), 14  
[set\\_message\\_type\(\)](#) (pyDKB.dataflow.communication.stream.Stream method), 14  
[set\\_output\\_message\\_type\(\)](#) (pyDKB.dataflow.stage.ProcessorStage.ProcessorStage method), 20  
[set\\_output\\_message\\_type\(\)](#) (pyDKB.dataflow.stage.ProcessorStage.ProcessorStage method), 23  
[set\\_dest\(\)](#) (pyDKB.dataflow.communication.producer.ProducerBuilder.ProducerBuilder method), 10  
[set\\_source\\_file\(\)](#) (pyDKB.dataflow.communication.consumer.ConsumerBuilder.ConsumerBuilder method), 7  
[set\\_source\\_info\(\)](#) (pyDKB.dataflow.communication.producer.ProducerBuilder.ProducerBuilder method), 10  
[set\\_stream\(\)](#) (pyDKB.dataflow.communication.stream.StreamBuilder.StreamBuilder method), 14  
[set\\_type\(\)](#) (pyDKB.dataflow.communication.consumer.ConsumerBuilder.ConsumerBuilder method), 7  
[set\\_type\(\)](#) (pyDKB.dataflow.communication.producer.ProducerBuilder.ProducerBuilder method), 10  
[set\\_type\(\)](#) (pyDKB.dataflow.communication.stream.StreamBuilder.StreamBuilder method), 14  
[skip\\_process\(\)](#) (pyDKB.dataflow.stage.ProcessorStage.ProcessorStage static method), 20  
[skip\\_process\(\)](#) (pyDKB.dataflow.stage.ProcessorStage.ProcessorStage static method), 24  
[source\\_is\\_readable\(\)](#) (pyDKB.dataflow.communication.consumer.FileConsumer.FileConsumer method), 9  
[src\\_info](#) (pyDKB.dataflow.communication.producer.ProducerBuilder.ProducerBuilder attribute), 10  
[stop\(\)](#) (pyDKB.dataflow.stage.AbstractStage.AbstractStage method), 22  
[stop\(\)](#) (pyDKB.dataflow.stage.ProcessorStage.ProcessorStage method), 20  
[stop\(\)](#) (pyDKB.dataflow.stage.ProcessorStage.ProcessorStage method), 24  
[Stream](#) (class in pyDKB.dataflow.communication.stream), 14  
[Stream](#) (class in pyDKB.dataflow.communication.stream.Stream), 17  
[stream\\_is\\_readable\(\)](#) (pyDKB.dataflow.communication.consumer.Consumer.Consumer method), 8  
[StreamBuilder](#) (class in pyDKB.dataflow.communication.stream), 13  
[streamClass](#) (pyDKB.dataflow.communication.stream.StreamBuilder.StreamBuilder attribute), 14  
[StreamConsumer](#) (class in pyDKB.dataflow.communication.consumer.StreamConsumer), 9  
[StreamException](#), 17  
[StreamProducer](#) (class in pyDKB.dataflow.communication.producer.StreamProducer), 9

[13](#)

subdir() (pyDKB.dataflow.communication.producer.FileProducer.FileProducer  
method), [11](#)

subdir() (pyDKB.dataflow.communication.producer.HDFSProducer.HDFSProducer  
method), [12](#)

## T

title\_extraction() (in module 055\_documents2TTL.documents2ttl), [31](#)

TTLMessage (class in pyDKB.dataflow.communication.messages),  
[19](#)

Type (class in pyDKB.common.Type), [3](#)

typeName() (pyDKB.dataflow.communication.messages.AbstractMessage  
class method), [18](#)

## V

valueByKey() (in module pyDKB.common.json\_utils), [6](#)

## W

write() (pyDKB.dataflow.communication.producer.Producer.Producer  
method), [13](#)

write() (pyDKB.dataflow.communication.stream.OutputStream  
method), [15](#)

write() (pyDKB.dataflow.communication.stream.OutputStream.OutputStream  
method), [17](#)