# pocketutils

*Release 0.9.1*

**Douglas Myers-Turnbull**

**Aug 13, 2022**

# CONTENTS

**Warning:** A lot of this documentation is out of date.

CHAPTER

# ONE

# CORE UTILITY CLASSES

A couple of other things were imported, including `DevNull`, `DelegatingWriter`, and `TieredIterator`.

You can also make a Pandas DataFrame with pretty display and convenience functions using `TrivialExtendedDataFrame`.

`LazyWrap` creates lazy classes, extremely useful in some cases:

```
from datetime import datetime
from pocketutils.core import LazyWrap


def fetch_datetime():
    return datetime.now()


RemoteTime = LazyWrap.new_type("RemoteTime", fetch_datetime)
now = RemoteTime()
# nothing happens until now:
print(now.get())
```

## 1.1 Exceptions and warnings

Sometimes certain modes of failure are expected (think: checked exceptions). We want callers to be able to handle and potentially recover from them, but granularity in exception types and relevant values are needed. For example, if we couldn't load a "resource" file, what was the path? If something was wrong with a database record, what was its ID? Examples of exceptions defined here are `LockedError`, `IncompatibleDataError`, ``HashValidationError`, `MissingEnvVarError`, `MultipleMatchesError`, `AlreadyUsedError`, and `IllegalPathError`.

```
import time
from pocketutils.core.exceptions import *

resource = Path("resources/mydata.dat")


def update_resource():
    if resource.with_suffix(".lockfile").exists():
        raise LockedError("Resource is locked and may be in use.", key=resource)
    # ... do stuff
```

(continues on next page)

(continued from previous page)

```
try:
    update_resource()
except LockedError as e:
    if e.key == resource:
        print(f"{e.key} is locked. Waiting 5s and trying again.")
        print(e.info())
        time.sleep(5.0)
        update_resource()
    else:
        raise e
```

CHAPTER

TWO

# TOOLS PACKAGE

The `Tools` class has various small utility functions:

```
def fn_to_try():
    raise ValueError("")


from pocketutils.full import *

Tools.git_description(".").tag  # the tag, or None
Tools.ms_to_minsec(7512000)  # "02:05:12"
Tools.fix_greek("beta,eta and Gamma")  # ", and "
Tools.pretty_function(lambda s: 55)  # "<(1)>"
Tools.pretty_function(list)  # "<list>"
Tools.strip_paired_brackets("(ab[cd)")  # "ab[cd"
Tools.iceilopt(None), Tools.iceilopt(5.3)  # None, 6
Tools.succeeds(fn_to_try)  # True or False
Tools.or_null(fn_to_try)  # None if it failed
Tools.only([1]), Tools.only([1, 2])  # 1, MultipleMatchesError
Tools.is_probable_null(np.nan)  # True
Tools.read_properties_file("abc.properties")  # returns a dict
important_info = (
    Tools.get_env_info()
)  # a dict of info like memory usage, cpu, host name, etc.
```

`Chars` contains useful Unicode characters that are annoying to type, plus some related functions:

```
from pocketutils.full import *

print(Chars.hairspace)  # hair space
print(Chars.range(1, 2))  # "1-2" (with en dash)
```

`Tools` actually subclasses from several Tools-like classes. You can import only the ones you want instead:

```
from pocketutils.tools.path_tools import PathTools

print(PathTools.sanitize_path("ABC|xyz"))  # logs a warning & returns "ABC_xyz"
print(PathTools.sanitize_path("COM1"))  # complains!! illegal path on Windows.
from pocketutils.tools.console_tools import ConsoleTools

if ConsoleTools.prompt_yes_no("Delete?"):
```

(continues on next page)

(continued from previous page)

```
    #  Takes 10s, writing Deleting my_dir.......... Done.
    ConsoleTools.slow_delete("my_dir", wait=10)
```

CHAPTER

THREE

# JUPYTER UTILITIES

## 3.1 `J` for Jupyter display

The class `J` has tools for display in Jupyter:

```
from pocketutils.j import *
J.red("This is bad.")            # show red text
if J.prompt("Really delete?"):   # ask the user
    J.bold("Deleting.")
```

## 3.2 Filling in templates

`MagicTemplate` can build and register a Jupyter magic function that fills the cell from a template. Ex:

```
import os
from pocketutils.support.magic_template import *
def get_current_resource():
    return "something dynamic"
template_text = """
# My notebook
<Write a description here>
**Datetime:      ${{datetime}}**
**Hostname:      ${{version}}**
**Resource name: ${{resource}}**
"""
MagicTemplate.from_text(template_text)\
    .add("hostname", os.hostname)\
    .add_datetime()\
    .add("resource", lambda: get_current_resource())\
    .register_magic("mymagic")
```

Now you can type in `%mymagic` to replace with the parsed template.

CHAPTER

FOUR

# BIOCHEM PACKAGE

`WB1` is a multiwell plate with 1-based coordinates (read *well base-1*).

```
from pocketutils.biochem.multiwell_plates import WB1
wb1 = WB1(8, 12)                # 96-well plate
print(wb1.index_to_label(13))  # prints "B01"
for well in wb1.block_range("A01", "H11"):
    print(well)                 # prints "A01", "A02", etc.
```

Getting tissue-specific expression data in humans:

```
from pocketutils.biochem.tissue_expression import TissueTable
tissues = TissueTable()
# returns a Pandas DataFrame of expression levels per cell type per gene for this tissue.
tissues.tissue("MKNK2")
```

CHAPTER

# FIVE

# PLOTTING

CHAPTER

SIX

# SUPPORT PACKAGE

These classes range from common to very obscure.

`PrettyRecordFactory` makes beautiful aligned log messages.

```
import logging
from pocketutils.support.log_format import *
logger = logging.getLogger("myproject")
log_factory = PrettyRecordFactory(7, 13, 5).modifying(logger)
```

Output from an analysis might then be…

```
[20191228:14:20:06] kale>    datasets      :77    INFO    | Downloading QC-DR...
[20191228:14:21:01] kale>    __init__      :185   NOTICE  | Downloaded QC-DR with 8 runs,
↪ 85 names, and 768 wells.
[20191229:14:26:04] kale>    __init__      :202   INFO    | Registered new type␣
↪RandomForestClassifier:n_jobs=4,n_estimators=8000
```

`TomlData` is a wrapper around toml dict data.

CHAPTER

SEVEN

# API REFERENCE

This page contains auto-generated API reference documentation[1].

## 7.1 pocketutils

Metadata for this project.

[1] Created with sphinx-autoapi

CHAPTER

EIGHT

# INTRODUCTION / GETTING STARTED

To install pocketutils and its optional packages via pip, run:

```
pip install pocketutils[all]
```

You can avoid installing some dependencies by installing only what you need. For example:

```
pip install pocketutils
pip install pocketutils[numeric]
```

The optional dependency sets are:

- tools
- plotting
- notebooks
- misc

You can import the most general-purpose parts of pocketutils like this:

```
from pocketutils.full import *

print(Tools)
```

This will load:

- `Tools`, containing various utility functions
- `Chars`, containing common Unicode characters
- `abcd`, containing decorators
- ~10 miscellaneous classes, such as `SmartEnum`
- a collection of exceptions such as `MultipleMatchesError` and `DataWarning`
- numpy as `np` and pandas as `pd`

# PYTHON MODULE INDEX

# INDEX