Zubax Python coding conventions

Coding conventions

Follow PEP8 with the following exceptions/additions:

- The maximum length of line is set to **120 characters**. In PEP8 the limit is 79 characters.
- Trailing whitespaces are not allowed. PEP8 provides a weak recommendation to avoid whitespaces, too. Configure your IDE to strip trailing whitespaces in the entire file on save.
- Usage of tabs is not allowed. If you're porting an existing codebase, replace all tabs with 4 spaces. PEP8 recommends to retain tabs when working with an existing codebase.

Every text file should contain exactly one empty line at the end.

Allowed end-of-line character sequence is Unix-style (`\n`, LF) only.

In CamelCase names, keep abbreviations capitalized. For example: `CANTransport`, not `CanTransport`; `IEEE802154Transport`, not `Ieee802154Transport`.

Name all entities with a leading underscore, including modules and packages, excepting those that are part of the package API. Re-export entities that should be public from their sub-package's `__init__.py`. See https://github.com/sphinx-doc/sphinx/issues/6574#issuecomment-511122156

When re-exporting entities from a package-level `__init__.py`, always use the form `import ... as ...` even if the name is not changed, to signal static analysis tools that the name is intended to be re-exported (unless the aliased name starts with an underscore). This is enforceable with MyPy.

Excepting the above described case of package-level API re-export, do not import specific entities; instead, import only the module itself and then use verbose references, as shown below. If you really need to import a specific entity, consider prefixing it with an underscore to prevent scope leakage. Exception applies to well-encapsulated submodules which are not part of the package API (i.e., prefixed with an underscore) – you can import whatever you want provided that the visibility scope of the module is sufficiently narrow. Example:

```
from pyuavcan.transport import Transport    # Don't do this.
import pyuavcan.transport                   # Good. Use like:
pyuavcan.transport.Transport
```

Semantic and behavioral conventions

Do not raise exceptions from properties. Generally, a property should always return its value. If the availability of the value is conditional, consider using a getter method instead.

Static type checking

Annotate types of all functions, methods, and properties, whether private or public. Enforce type correctness with MyPy using the following (or stricter) settings (example shown is an excerpt from `setup.cfg`):

MyPy static type checking options

```
[mypy]
warn_return_any = True
warn_unused_configs = True
disallow_untyped_defs = True
check_untyped_defs = True
no_implicit_optional = True
warn_redundant_casts = True
warn_unused_ignores = True
```

Also, use the following command-line options with MyPy: `--strict --strict-equality --no-implicit-reexport`.

Documentation

All public entities should be documented in ReST, unless they are simple enough for their purpose to be evident without explicit documentation. Do not write in prose what can be expressed in code; for example, avoid specifying type information in the docstrings – there are type annotations for that. Never spell out obvious things in the docs.

An example that shows the adopted documentation convention is provided below.

```
def write(self, data: typing.Union[bytes, str], timeout:
typing.Optional[float] = None) -> int:
    """
    This method can be invoked concurrently from multiple threads, even if
there are other threads blocked on
    any of the reading methods.
    :param data:        Data to transmit.
    :param timeout:     Timeout in seconds, None for infinity.
    :return:            Number of bytes written -- always either len(data)
or 0 on timeout.
    """
    if isinstance(data, str):
        data = data.encode()

    if not isinstance(data, bytes):
        raise ValueError('Invalid data type: %r' % type(data))

    try:
        if len(data) > 0:
            self._txq.put(data, timeout=timeout)
    except queue.Full:
        return 0
    else:
        return len(data)
```

Testing

Aim to cover 100% of the code in the branch coverage mode.

Keep simple test cases as close to the tested code as possible: either in doctest or in test functions; in the latter case, use the function name pattern `_unittest_*`. Move all complex test code outside of the main codebase into a dedicated package (usually named `tests`).

Misc

Write all new code in Python 3.7.

The recommended IDE for Python development is **JetBrains PyCharm**.