# PLUTO to Python Parser

Artur Scholz (artur.scholz@librecube.org), Christoph Buchner (christoph.buchner@gmail.com), Vidit Jain (viditjain5598@gmail.com)

## What is PLUTO?

PLUTO stands for “Procedure language for users in test and operations” and is an open and free standard published by the European Committee for Space Standardization (ECSS, https://ecss.nl/). It defines a domain specific language for testing and operation. It was developed for use in the space domain, but can be applied to any domain where the control of machines or instruments is needed.

## Why use PLUTO?

PLUTO is a domain specific language (DSL), which means that it is specialised to a particular domain of interest. This is in contrast to a general purpose language like Python, which is generic and can be used for any kind of problems. The advantage of a DSL is that operators do not need to learn the (many) specifics of a programming language, but only the rules of the DSL. Also, PLUTO is tested and was designed by experts for use in space applications. Moreover, the syntax of PLUTO is human readable and can be parsed by computer as well. That makes it particular useful for automation.

You may want to use PLUTO to write automated test scripts and for flight operations.

## Why this Parser?

PLUTO is currently used by a few missions of ESA and DLR, among others, mostly in the context of automated operations. Unfortunately, the existing PLUTO procedure parsers and executors are neither open source, nor available to third party users. Therefore we launched the development of this parser to demonstrate the capabilities of PLUTO language, and to make it easily and freely accessible to anyone. Most of the development was done in the frame of a Google Summer of Code project in summer 2019.

## How to use it?

Download/clone the repository and install it. You can then use the commands **pluto_convert** to parse a PLUTO script into a Python script (see example on the right). The generated Python code shall then be run by an executor. An example of such is contained in the command **pluto_run,** but you are free to produce your own executor and/or automation system on top of it.

To interact with the system (that is, your CubeSat, your test machine, etc.) you will need to create a system model of it first. This model acts as sink for activity calls and as source for reporting data, both of which are referenced to in the PLUTO scripts.

**Fig 1: The general structure of a procedure (from ECSS-E-ST-70-32C)**

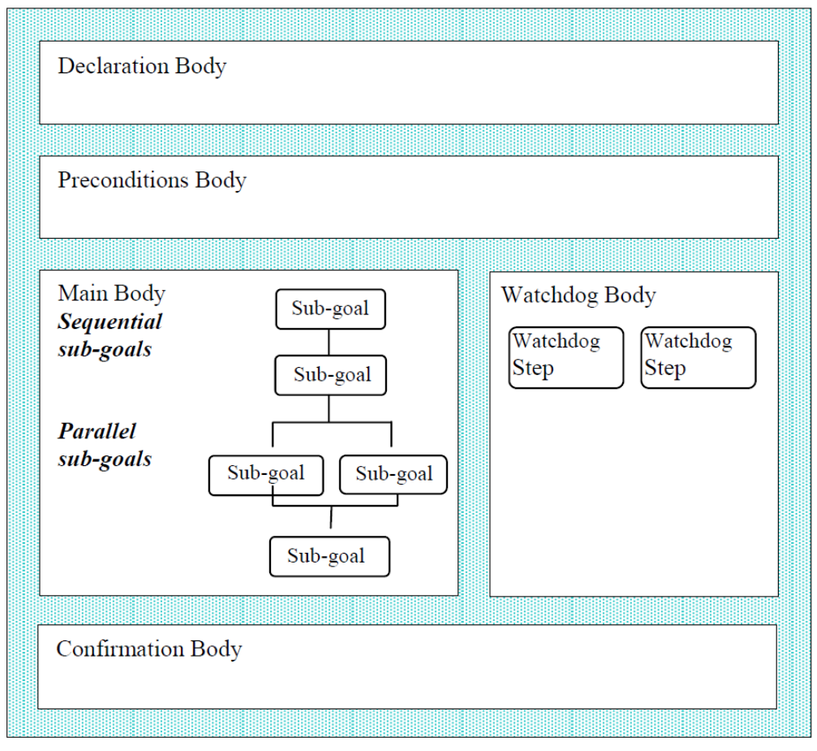


**Fig 2: Example of PLUTO script (as input)**

```
procedure
    preconditions
        wait until value of GyroTemp > 60 degC
    end preconditions
    main
        initiate and confirm SwitchOnGyroConverter;
        initiate and confirm SwitchOnGyro5;
        initiate and confirm Gyro5FineMode;
    end main
    confirmation
        wait until value of Gyro5 < 0.2 deg/h
    end confirmation
end procedure
```

**Fig 3: Example of generated Python code**

```
from pluto import *
from model import *


class Procedure_test0921(Procedure):

    def preconditions(self):
        if self.wait_until_expression(
                lambda: GyroTemp.get_value() > ureg('60degC')) is False:
            return False

    def main(self):
        act = ActivityCall(self, SwitchOnGyroConverter)
        if self.initiate_and_confirm_activity(act) is False: return False
        act = ActivityCall(self, SwitchOnGyro5)
        if self.initiate_and_confirm_activity(act) is False: return False
        act = ActivityCall(self, Gyro5FineMode)
        if self.initiate_and_confirm_activity(act) is False: return False

    def confirmation(self):
        if self.wait_until_expression(
                lambda: Gyro5.get_value() < ureg('0.2deg')/h) is False:
            return False
```

## Further information

The project is still in prototype phase. You can find it at the link below. Please contact the authors for more information or if you would like to contribute.

https://gitlab.com/librecube/prototypes/python-pluto