



Apache PLX4Py

Roadmap

Agenda

- Introduction
- Define Milestones
 - What needs to be done?
Following the path of C Implementation
 - Upstream API
 - Downstream API
- High-Level Roadmap

Plan (1/2)

1. Setup Basic Directory Structure - Integratable

- a. Build-able by our Build Server

2. Distributed Python Module -> Deploy it to Pypi

- a. Goal: pip Install PLC4Py -> Dependency Log File
- b. Poetry - Manage Dependencies
- c. Sandbox already uses setup.py
- d. Linking into Build Pipeline

<https://python-poetry.org/>

3. Basic API Design

- a. Implement some dummy driver
- b. Load Driver - Read, Write, Subscription
- c. Return random values for testing the Basic API Usage
- d. Make it Pythonic **Guideline 1** - Natural API for Python Developers / Users
- e. Py-Test Drivers -> Tests Infrastructure as you go
- f. Write tests as we go! - **Guideline 2**

<https://www.python.org/dev/peps/pep-0001/>

Plan (2/2)

1. Language Template Project
 - a. **Create Dummy Implementation of Read and Write Buffer**
 - i. Enough of the Base API to Test if this what we are building is working
 - ii. Refactoring iteratively as we go
 - iii. XML Testsuites -> Test ressources per Protocol Module
 - b. *Look what we have in Java - Templating*
 - c. *In C started with an Empty Template (Empty struct with a name)*
 - d. Create an Empty Template Project -> Freemarker Templates
 - i. Copy from 2 Java classes (Spec Templates, Complex Types, Enums, DataIO) and Empty it
 - e. Template Helper -> Because of Debugging directly in Python (outside of Freemarker)
 - i. Language Based Freemarker
 - f. Then filling it with life
 - i. **Here starts the bigger refactoring for the Read and Write**
 - ii. Write manual Testbed
2. Code Generation
 - a. Create Framework / Generate the Logic for the Drivers
 - b. Write Sample Driver (e.g. Modbus)
- (optional) Sidething create an MSPEC to test Code Generation -> Good Regression Test -> Use it for all the parsers
 - Look into the Build Util tools

DataIO - Do the mapping - how something is encoded in the protocol
Start with Complex Types
2 Files created in Java (Pojo)
Depends on the number of entries in the list

Setup Basic Directory Structure

Plp install plc4py

Import plc4py

Start with poetry (<https://python-poetry.org/docs/>)

- Handles namespaces
- Handles dependencies

Type Hints: <https://www.python.org/dev/peps/pep-0484/>

- Check code quality

Create feature-branch - One Feature Branch (PLC4Py + Codegeneration)

- Pull Requests for everything to improve cooperation
- Do PR into that branch - so everyone can see and read the changes
- Define PLC4Py Development Guide
 - Use docstrings for classes and function
 - How do Pull Requests? (Who does the review?) - Assign specific person for Review
- Figure out Python Package delivery within the Apache Foundation
(as part of the Maven build stuff) - Ask ASF Infrastructure for Credentials etc.

Next steps - 2020.08.16

Initialize Project:

<https://issues.apache.org/jira/browse/PLC4X-232>

Update Website / Documentation

- Where do we document?
- Do we use Confluence for Brainstorming?

Apache Licence Topic for Python

- Licence Requirement for the Codefile
- Notice
- RAT Check? Possible in Python -> Check other Apache Python Projects for comparison

IDE

<https://blog.codota.com/python-plugins-for-intellij-idea/>

- PyCharm
- VSCode





