Welcome

PLCnext Control AXC F 2152における gRPC(Python)通信動作検証

注意事項

- 本記事は、特定の環境における PLCnext Control AXC F 2152 と gRPC（Python）通信の
  動作検証結果を紹介するものです。
  すべての環境での動作を保証するものではありません。

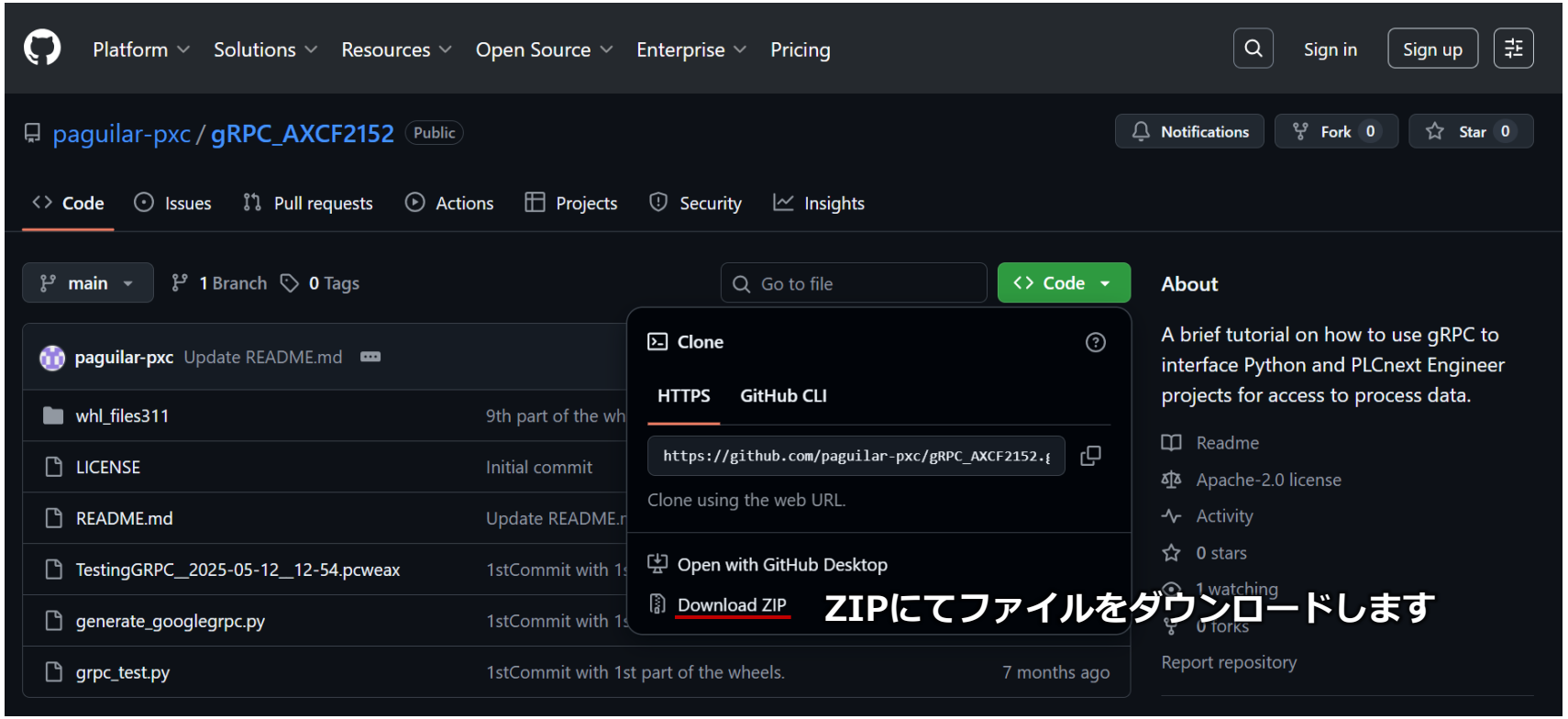
- 記事内の手順には root 権限での操作やコンテナ操作が含まれます。実施する場合は、
  検証環境で内容を十分に理解したうえで行ってください。

- 本記事の内容は、以下の PLCnext Community の記事を参考にしています。
  https://www.plcnext-community.net/makersblog/use-grpc-to-interface-python-scripts-with-plcnext-engineer-projects/

検証用ファイルのダウンロード（GitHub リポジトリ）

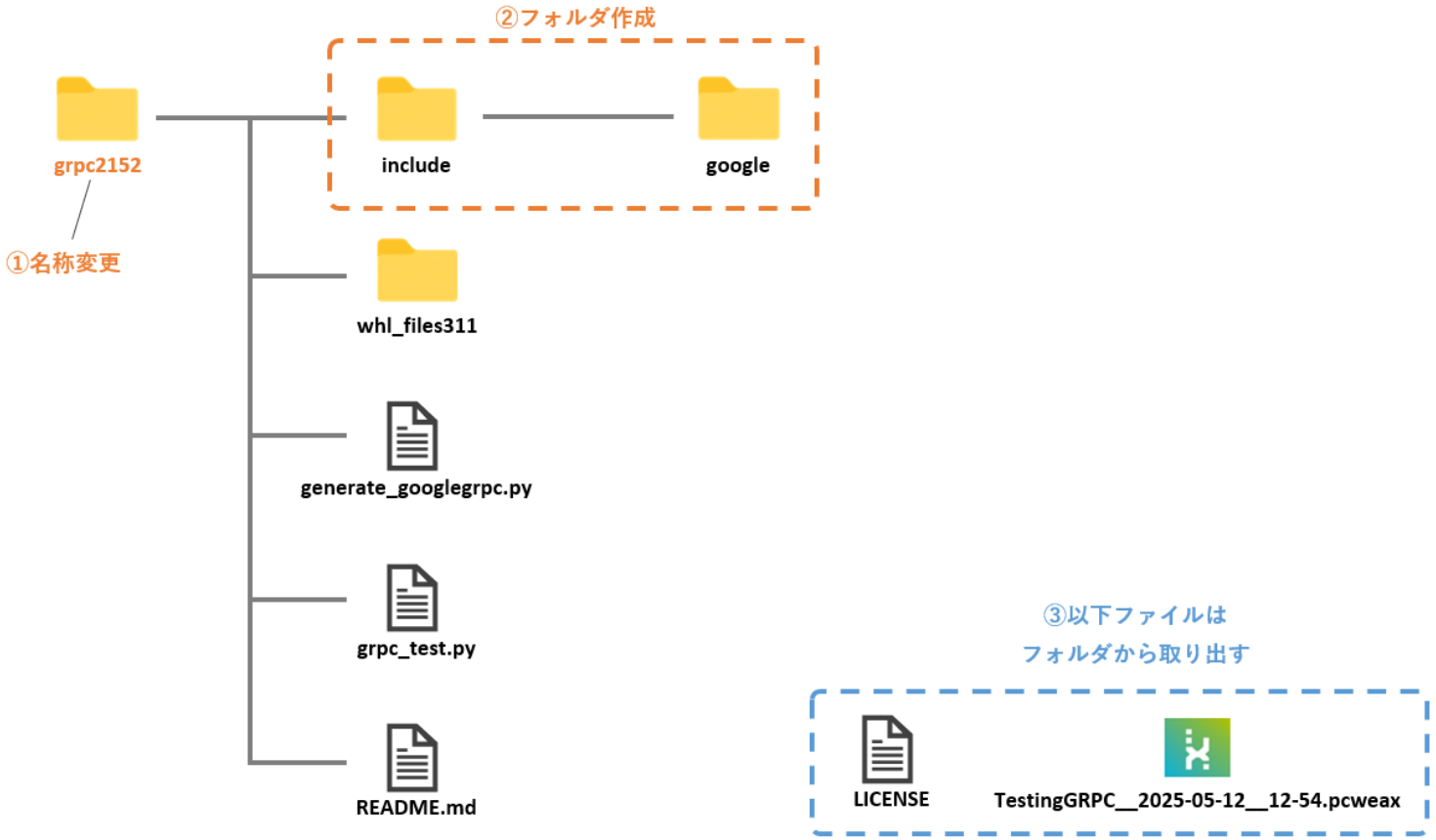
以下のリンクよりダウンロードを行います。

https://github.com/paguilar-pxc/gRPC_AXCF2152/tree/main



ダウンロードファイルの解凍と初期編集

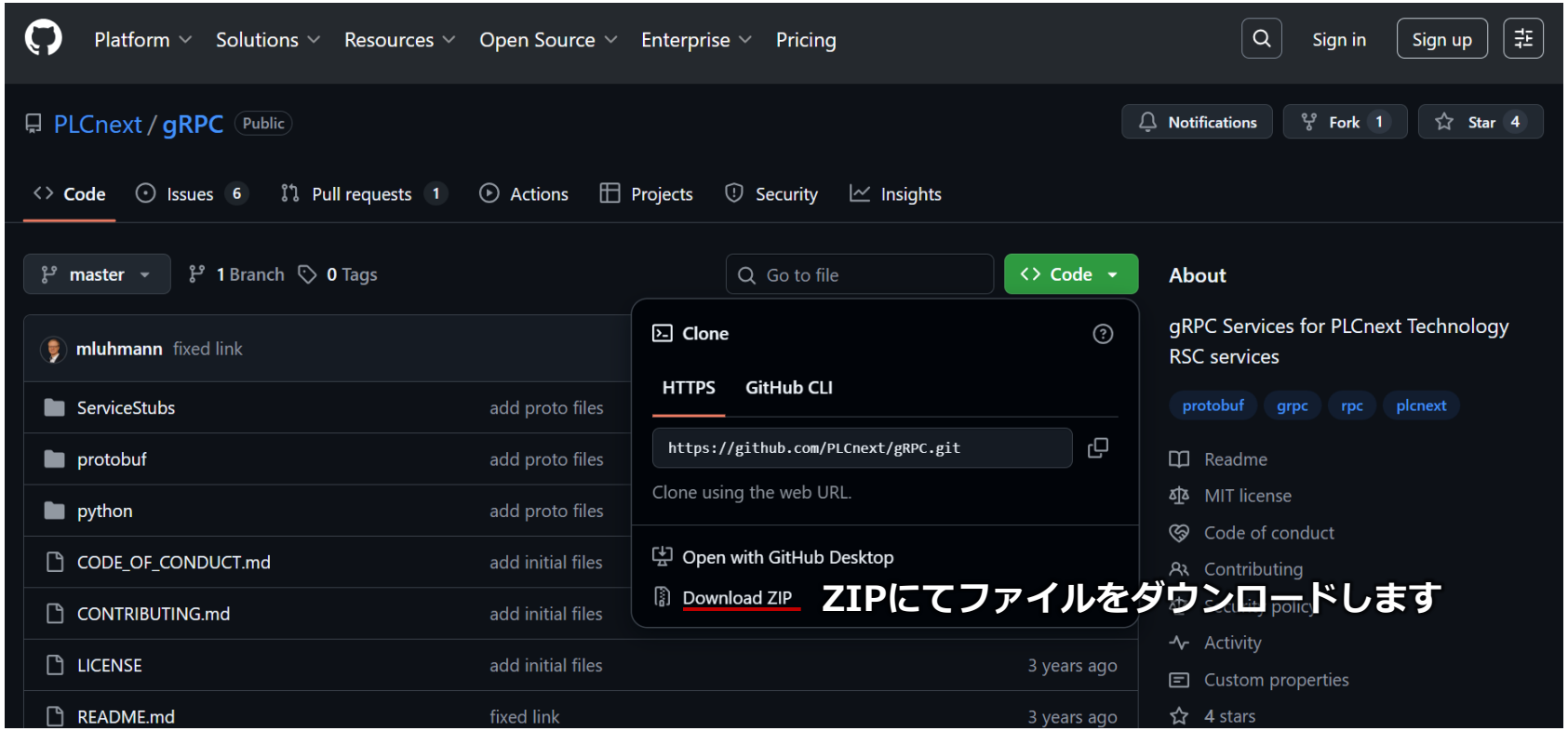
Zipファイルを解凍し、フォルダ構成を以下のように整理・編集します。



gRPC 定義ファイルのダウンロード（PLCnext公式）

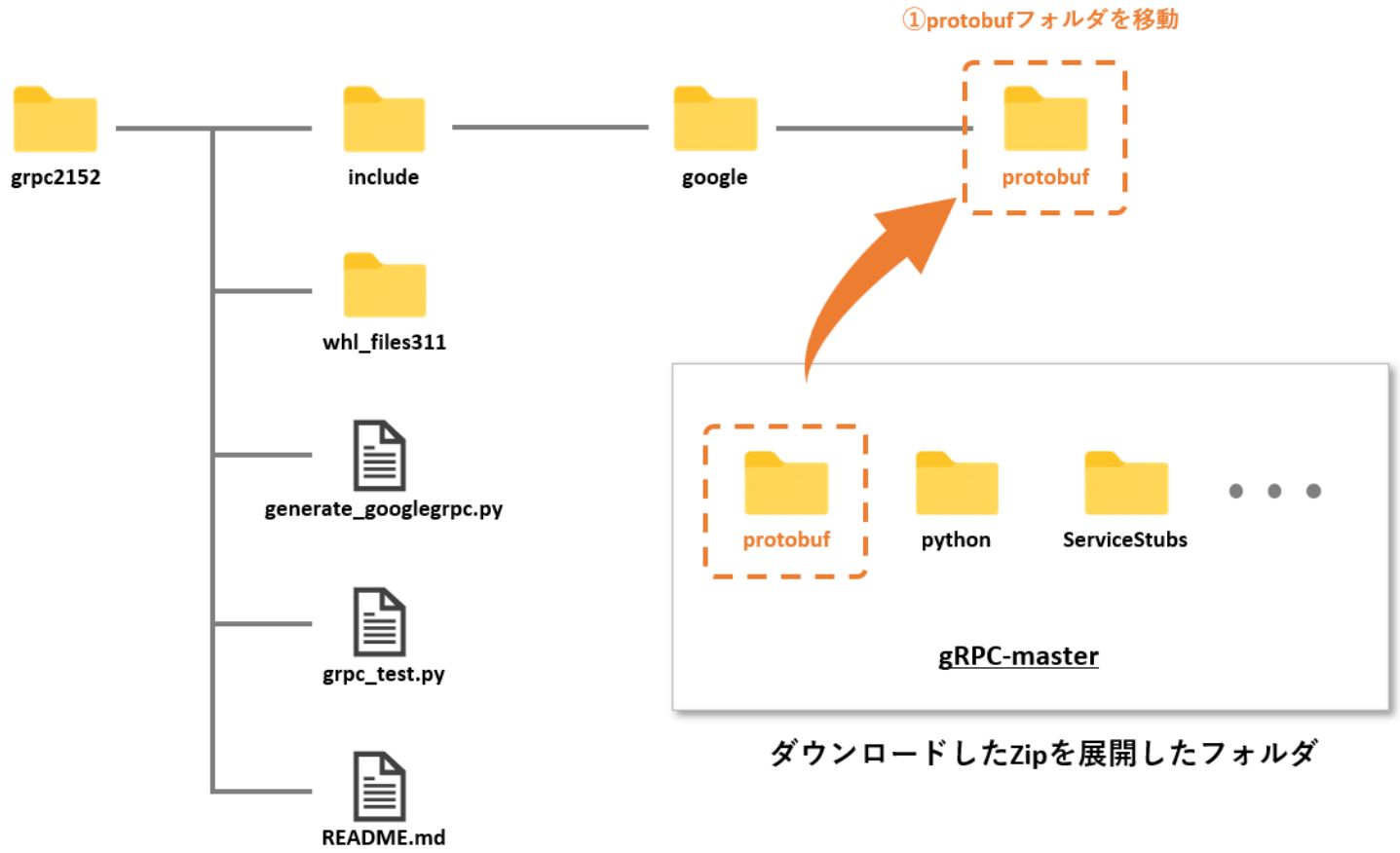
以下のリンクよりダウンロードを行います。

https://github.com/PLCnext/gRPC



protobuf フォルダの配置とフォルダ構成の調整

Zipファイルを解凍し以下の様にファイル内を編集します。



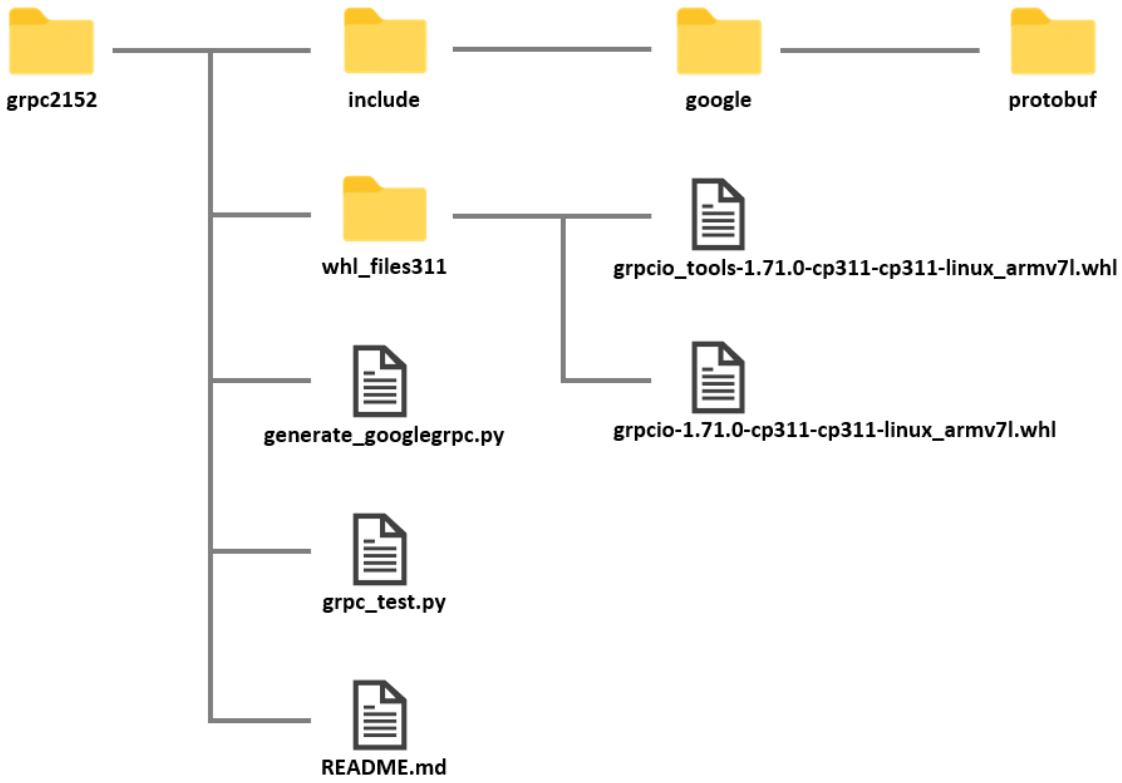
Python 用ライブラリ（.whl）の展開

Whl_files311フォルダの7zファイルを解凍します。

a

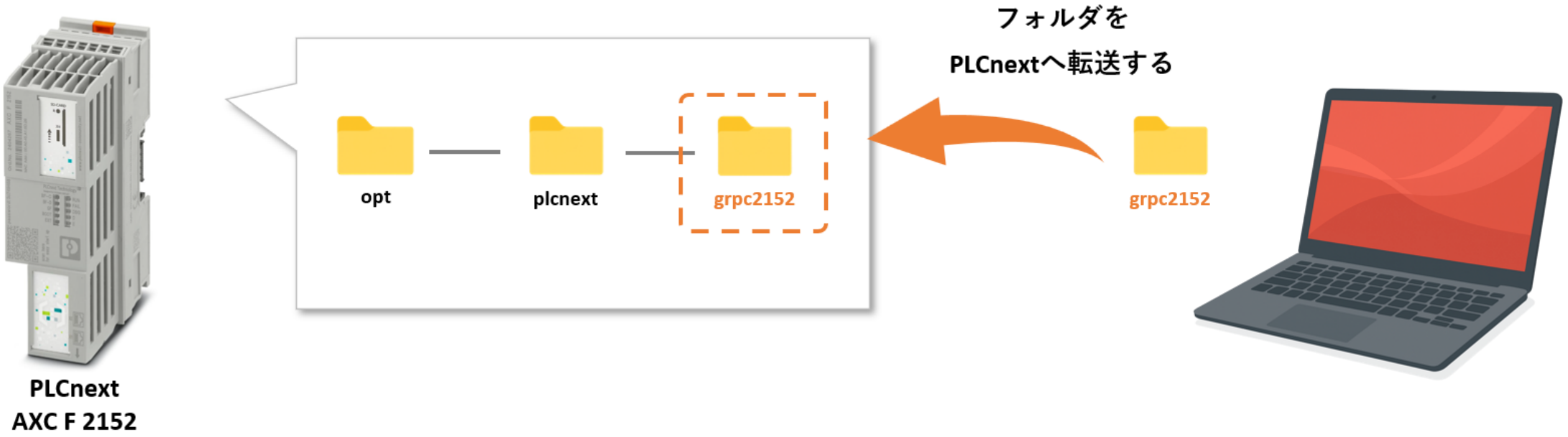
最終的なフォルダ構成の確認

最終的なフォルダ構成は以下となります。



検証フォルダの AXC F 2152 への配置

フォルダをAXC F 2152の指定ディレクトリ配下に配置します。



AXC F 2152 のネットワーク接続構成

AXC F 2152をインターネットに接続可能な状態にします。（Pythonコンテナイメージ取得のため）

【構成例】

AXC F 2152

LAN

EW50
(LTEルーター)

AXC F 2152 への SSH 接続と root 権限の取得

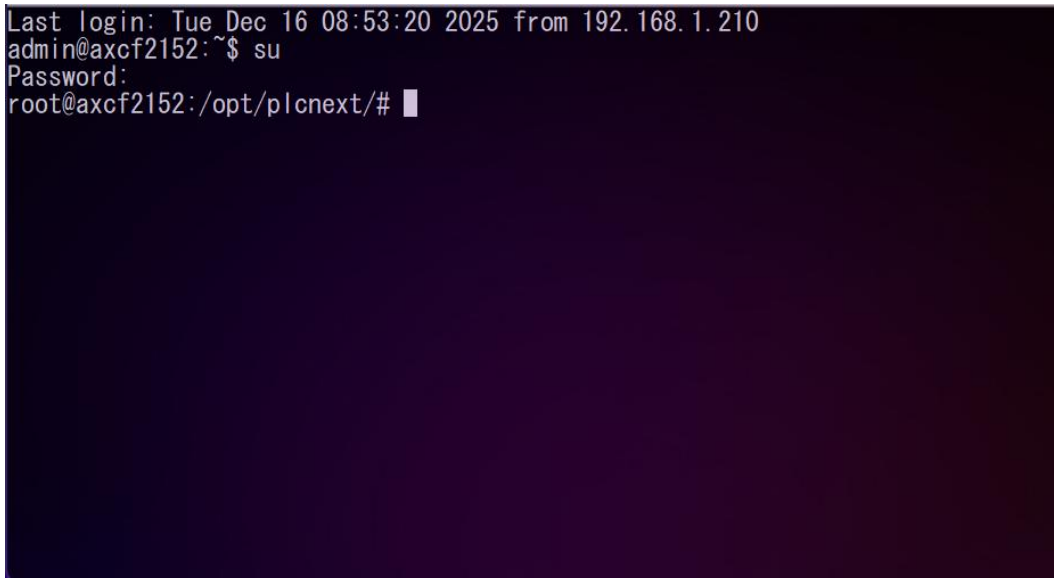
AXC F 2152にSSHアクセスを行いrootユーザーに切り替えます。

・ルートユーザーに切り替えます

$ su

rootユーザーのパスワードを入力

※rootユーザーに切り替えるにはあらかじめrootユーザーの
ログインを有効化し、パスワードを設定しておく必要が有ります。
方法は以下となります。
https://plcnext.jp/archives/1610



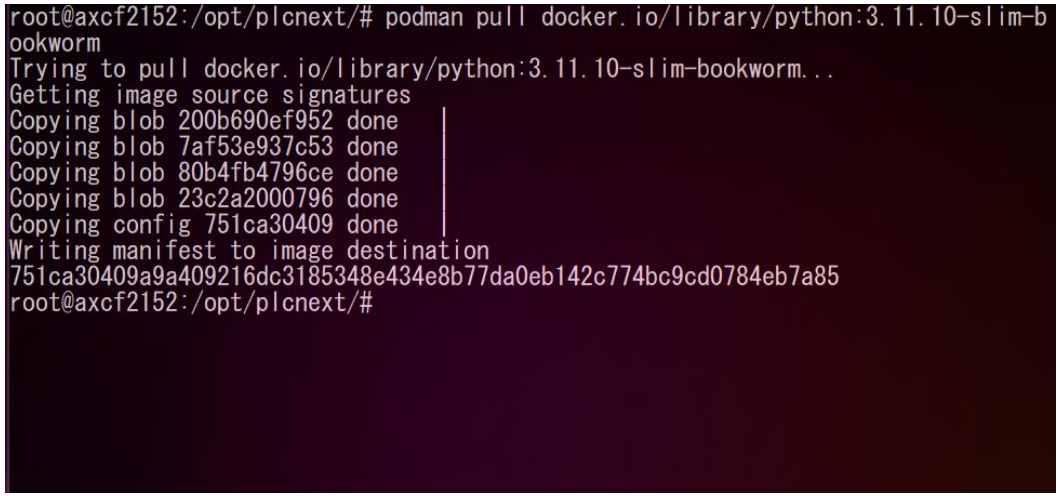
Python コンテナイメージの取得（Podman）

Podmanを使用してPythonのコンテナイメージをダウンロードします。

・コンテナイメージをダウンロードします

podman pull docker.io/library/python:3.11.10-slim-bookworm

※コンテナイメージをダウンロードする際は、AXC F 2152が
インターネットに接続されている必要があります。

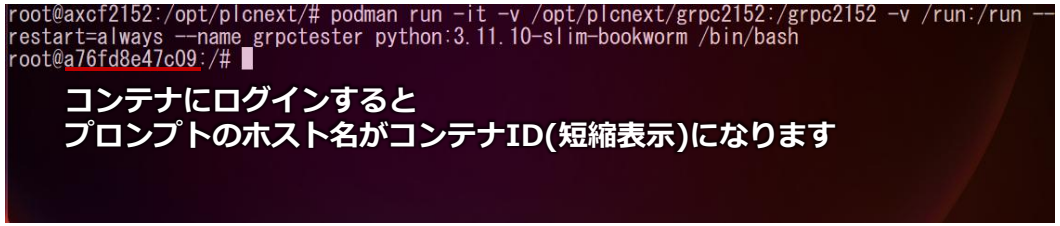


Python コンテナの起動とボリューム共有設定

Pythonコンテナを起動し、コンテナ内で作業できる状態にします。

・以下コマンドを実行します

podman run -it -v /opt/plcnext/grpc2152:/grpc2152
-v /run:/run --restart=always --name grpctester
python:3.11.10-slim-bookworm /bin/bash



Podmanを使用して、Python3.11.10のコンテナのイメージから新しいコンテナを起動します。
起動時にAXC F 2152上のディレクトリ「/opt/plcnext/grpc2152」をコンテナ内の「/grpc2152」と共有することで、
ホスト側のファイルをコンテナから参照できるようにしています。
また、コンテナ起動後にLinuxのbashを起動しコンテナ内にログインした状態でコマンド操作を行います

Python 追加ライブラリ（gRPC 関連）のインストール

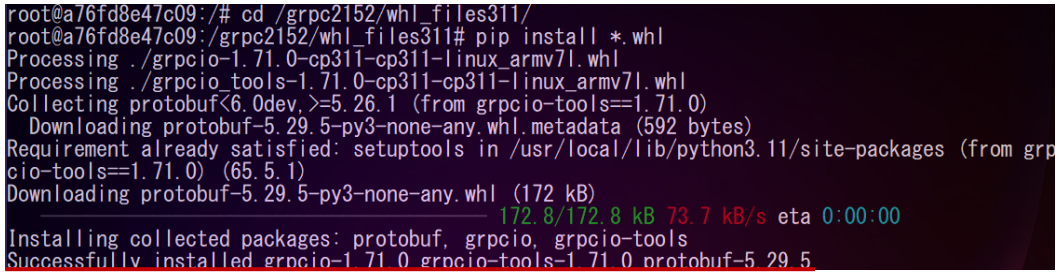
Pythonの追加ライブラリをインストールします。

・pythonライブラリが保存されたディレクトリに移動します

cd /grpc2152/whl_files311/

・pythonプログラムで使用するライブラリをインストールします

pip install *.whl



ライブラリのインストール成功

「/grpc2152/whl_files311」に移動し保存されているPythonライブラリ(.whlファイル)を
pipコマンドを使用してインストールします。
この操作によりgPRCなどのpythonプログラムで使用する追加ライブラリがコンテナ内の
python環境に登録されます。

gRPC 定義ファイル（.proto）から Python コードを生成

gRPC のインターフェース定義（.proto）から Python ファイルを生成します。

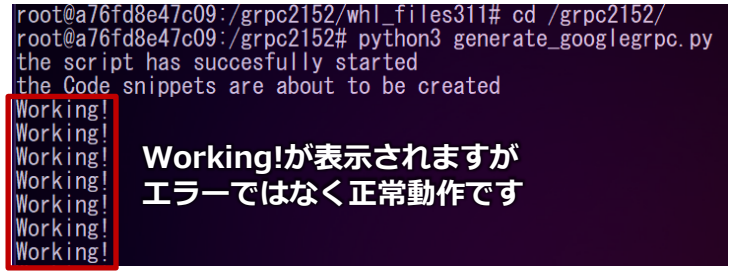
・pythonスクリプトがあるディレクトリに移動します

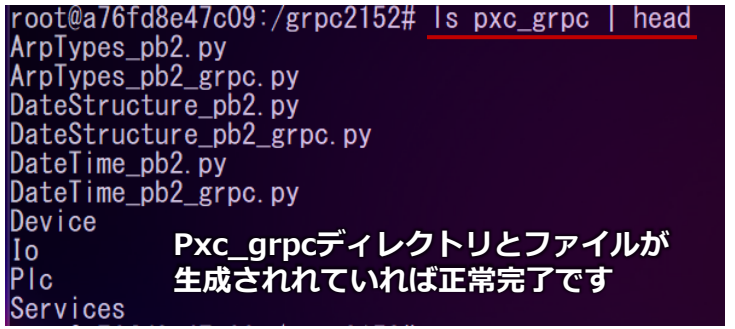
cd /grpc2152

・pythonスクリプトを実行します

python3 generate_googlegrpc.py

Generate_goooglegrpc.pyを実行し、include/google/protobuf配下にある
.protoファイルをもとにgRPC通信で使用するpythonの自動生成コードを作成します。
この処理により、pxc_grpcディレクトリ配下にpythonスクリプトなどのファイルが
生成され、pythonプログラムからRPCインターフェースを利用できるようになります。





PLCnext プロジェクトのコントローラへの書き込み

AXC F 2152にプロジェクトファイルを書き込みます。

プロジェクト書き込み

TestingGRPC__2025-05-12__12-54.pcweax

AXC F 2152

こちらのプロジェクトファイルは
https://github.com/paguilar-pxc/gRPC_AXCF2152/tree/main
のzipファイルに格納されていたプロジェクトファイルです

※プロジェクトのNWの設定やコントローラーのFWバージョンなど
については実機に合わせるように設定を行ってください。

既存 Python コンテナへの再接続方法

AXC F 2152 にSSHで再接続し、既存のコンテナに入ります。

・コンテナの確認を行います。

podman ps –a

・コンテナの状態が Exited の場合は、
停止しているため起動します。

podman start [コンテナ名]

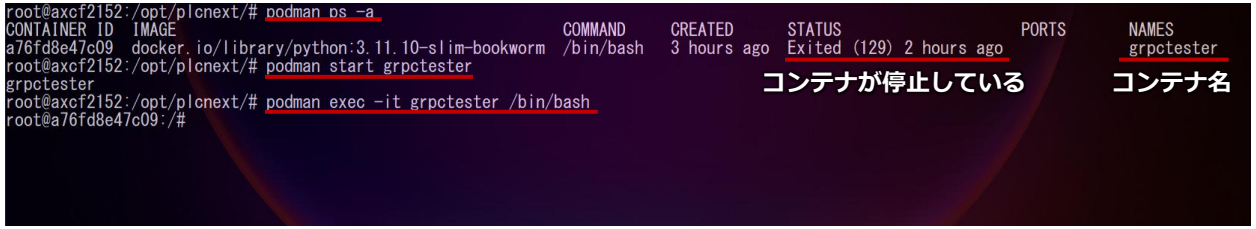
例) podman start grpctester

・起動中のコンテナの中に入る

podman exec -it [コンテナ名] /bin/bash

例) podman exec -it grpctester /bin/bash

※ podman exec は、起動中のコンテナにのみ使用できます。



podman ps -a でコンテナの状態を確認し、STATUS が Exited の場合は
コンテナが停止しているため podman exec で入ることはできません。

podman start でコンテナを起動してから、
podman exec でコンテナ内の bash を起動します。

gRPC 通信テスト用 Python スクリプトの実行

テストのpythonスクリプトを実行します。

・テスト用のpythonスクリプトがあるディレクトリに移動します

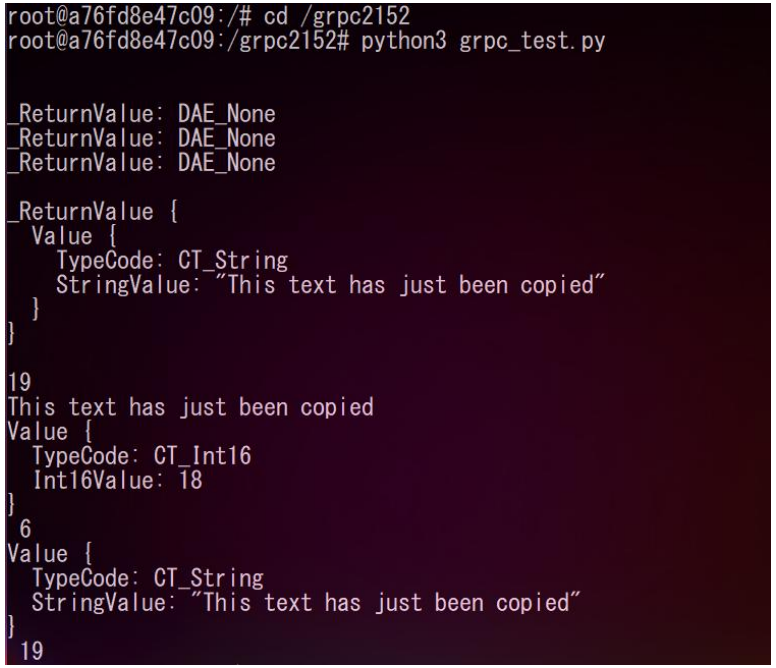
cd /grpc2152

・テスト用のpythonスクリプトを実行します

python3 grpc_test.py

このテスト用のpythonスクリプトはgRPCを使用してPLC変数の書き込み及び読み込みを行い、ReturnValueが「DAE_None」と表示され、読み込んだ値が表示されればgRPC通信は正常に動作しています。

(次のスライドでPLCnextEngineerより変数の値の確認を行います)



PLCnext Engineer による変数値の確認

PLCnextEngineerより変数の値の確認を行います。

Name	Value	Type	Usage	Translate	Comment	Init
Default						
strInput	'This text has just been copied'	STRING	Local			''
str1	''	STRING	Local			''
i1	0	INT	Local			INT#0
iInput	18	INT	Local			INT#0
rInput	0.0	REAL	Local			REAL#0.0
r1	0.0	REAL	Local			REAL#0.0
xOutput	FALSE	BOOL	Local			FALSE
x1	FALSE	BOOL	Local			FALSE
xInput	TRUE	BOOL	Local			FALSE
x2	FALSE	BOOL	Local			FALSE
strInput2	'The gRPC communication seems to work'	STRING	Local			''
str2	''	STRING	Local			''

TypeCode（CoreType）とデータ型の対応関係

TypeCode（CoreType）の定義は ArpTypes.proto に記載されています。
gRPC 通信では、この定義に基づいてデータ型が識別されます。


```
8   syntax = "proto3";
9   package Arp.Type.Grpc;
10  enum CoreType
11  {
12      option allow_alias = true;
13      CT_None = 0;
14      CT_End = 0;
15      CT_Void = 1;
16      CT_Boolean = 2;
17      CT_Char            = 3;
18      CT_Int8            = 4;
19      CT_Uint8           = 5;
20      CT_Int16           = 6;
21      CT_Uint16          = 7;
22      CT_Int32           = 8;
23      CT_Uint32          = 9;
24      CT_Int64           = 10;
25      CT_Uint64          = 11;
26      CT_Real32          = 12;
27      CT_Real64          = 13;
28      CT_Struct          = 18;
29      CT_String          = 19;
30      CT_Utf8String      = 19;
31      CT_Array           = 20;
```

```
68  message ObjectType
69  {
70      CoreType TypeCode = 1;
71      oneof Value
72      {
73          bool BoolValue = 2;
74          sint32 CharValue = 3;
75          sint32 Int8Value = 4;
76          uint32 Uint8Value = 5;
77          sint32 Int16Value = 6;
78          uint32 Uint16Value = 7;
79          sint32 Int32Value = 8;
80          uint32 Uint32Value = 9;
81          sint64 Int64Value = 10;
82          uint64 UIntValue = 11;
83          float  FloatValue = 12;
84          double DoubleValue = 13;
85          string StringValue = 14;
86
87          TypeStruct StructValue = 15;
88          TypeArray ArrayValue = 16;
89
90          sint32 EnumValue = 17;
91      }
92  }
```

PLCnext プロジェクトへのテスト変数の追加

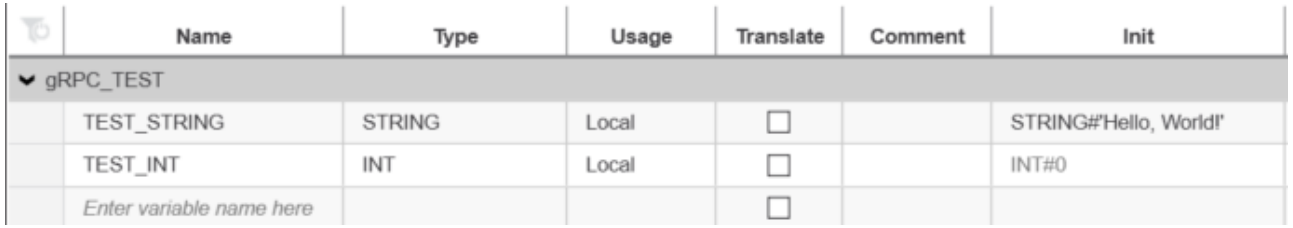
PLCnextに変数を追加してそちらのデータの読み書きを行います。

PLCnextEngineerにて2つの変数を追加

(変数名/変数型/初期値/読み込みor書き込み)

・TEST_STRING/STRING/Hello, World!/読み込み用

・TEST_INT/INT/0/書き込み用



Name	Type	Usage	Translate	Comment	Init
gRPC_TEST					
TEST_STRING	STRING	Local	☐		STRING#'Hello, World!'
TEST_INT	INT	Local	☐		INT#0
Enter variable name here			☐		

上記の変数を追加したプロジェクトをAXC F 2152へ書き込んでください。

gRPC テストスクリプトの修正（読み書き確認）

テスト用スクリプトの修正を行います。

grpc_test.pyに以下を追記します。

```
102     print("データ読み書きテスト")
103     r = read_single_value(stub, 'Arp.Plc.Eclr/MainInstance.TEST_STRING')
104     print(r._ReturnValue.Value.TypeCode)
105     print(r._ReturnValue.Value.StringValue)
106     print(write_single_int(stub, 'Arp.Plc.Eclr/MainInstance.TEST_INT', 100))
```

print("データ読み書きテスト")
r = read_single_value(stub, 'Arp.Plc.Eclr/MainInstance.TEST_STRING')
print(r._ReturnValue.Value.TypeCode)
print(r._ReturnValue.Value.StringValue)
print(write_single_int(stub, 'Arp.Plc.Eclr/MainInstance.TEST_INT', 100))

上記は TEST_STRING を読み込んで出力し、TEST_INT に 100 を書き込みます。

追加テストスクリプトの実行結果確認

テスト用スクリプトを実行します。

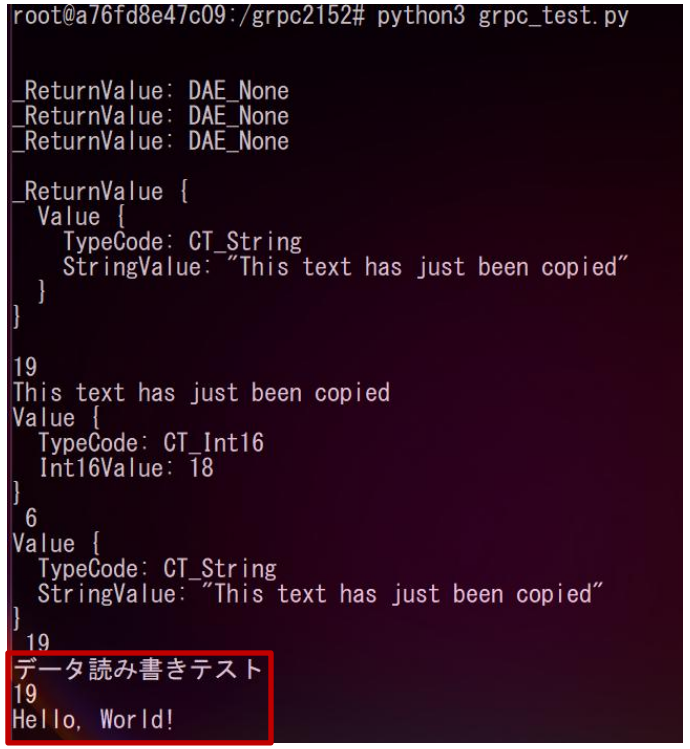
スクリプトを実行した際の出力の最後に

以下のような出力がされれば成功です。

データ読み書きテスト

19

Hello, World!

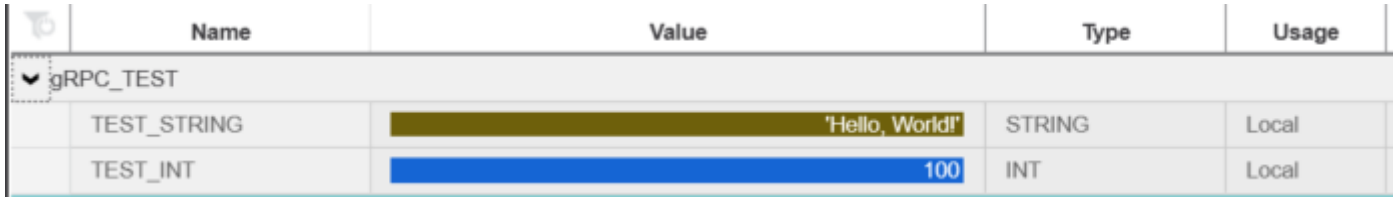


PLCnext Engineer での最終動作確認

PLCnextEngineerの変数の確認を行います。

PLCnextEngineerの変数の確認を行い

TEST_INTに100が書き込まれていれば成功です。



Name	Value	Type	Usage
gRPC_TEST			
TEST_STRING	'Hello, World!'	STRING	Local
TEST_INT	100	INT	Local

Thank you

PHOENIX CONTACT
INSPIRING INNOVATIONS