ポストプロセッサー

https://robodk.com/
info@robodk.com
+ 1-855-692-7772

目次

ポストプロセッサー

ポストプロセッサは、特定のロボットコントローラのロボットプログラムを生成できるため、オフラインプログラミングの重要なステップです。ロボットプログラミングはベンダー固有のプログラミングルールに従う必要があります。これらのルールはポストプロセッサーに実装されています。ロボットポストプロセッサは、特定のロボットコントローラ用にロボットプログラムを生成する方法を定義します。

RoboDK シミュレーションから特定のロボットプログラムへの変換は、ポストプロセッサによって行われます。各ロボットは、特定のロボットプログラミングスタイルを定義するポストプロセッサにリンクされています。ポストプロセッサは、プログラムがオフラインで生成されるときに使用されます。プログラムを生成するセクション（プログラムを右クリックし、[Generate Robot Program]を選択）。

RoboDK には、多くのロボットコントローラをサポートする多くのポストプロセッサが付属しています。 利用可能なポストプロセッサーセクション。または、カスタマイズされたポストプロセッサーを作成したり、既存のポストプロセッサーを変更したりすることもできます。使用可能なすべてのポストプロセッサーは、C：/ RoboDK / Posts /フォルダーにあります。

1 つのポストプロセッサーは PY ファイルです（各ポストプロセッサーは Python スクリプトによって定義されます）。RoboDK の Posts フォルダー内のファイルを手動で追加、変更、または削除することができます。RoboDK ポストプロセッサー（PY ファイル）が提供されている場合は、それを Posts フォルダーに配置して、RoboDK から選択できるようにする必要があります。

このセクションでは、ポストプロセッサーを選択、編集、または作成し、RoboDK のロボットで使用する方法を示します。次のビデオで簡単な紹介をご覧いただけます。https://www.robodk.com/help#PostProcessor

注意：ほとんどのロボットブランドには、異なるコントローラーバージョンがあります。特定のロボットコントローラがサポートする適切なロボットプログラムを生成するには、適切なポストプロセッサを選択することが重要です。

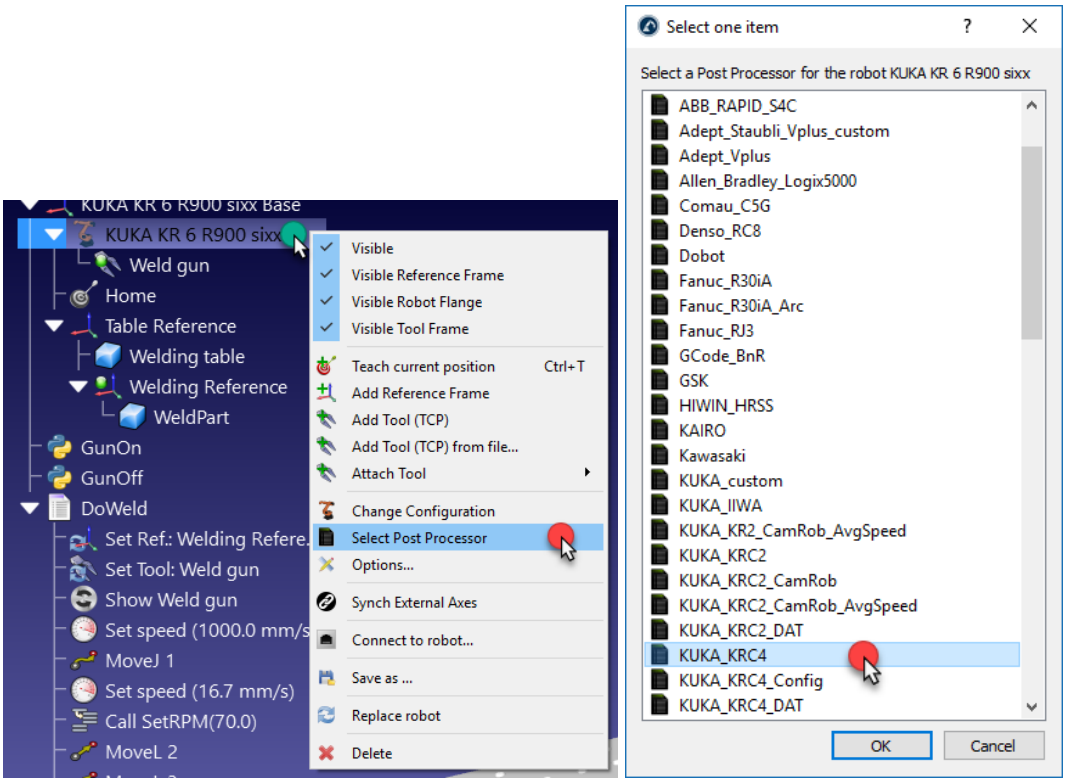
重要： 各ロボットには、デフォルトで特定のポストプロセッサーが割り当てられています。デフォルトの選択では、ロボットコントローラに適切なプログラムが生成されない場合があります。その場合、ロボットコントローラーでプログラムを実行するために、別のポストプロセッサを選択する必要がある場合があります。

ポストプロセッサーを選択する

ロボットやプログラムを右クリックすることで、簡単にポストプロセッサーを選択できます。ポストプロセッサの選択はロボットに割り当てられるため、プログラムに割り当てられているポストプロセッサを変更すると、同じロボットに割り当てられているすべてのプログラムで使用されるポストプロセッサが更新されます。

ロボットのポストプロセッサーを選択するには：

1. ロボットまたはプログラムを右クリック
2. ポストプロセッサーの選択
3. リストからポストプロセッサーを選択します
4. OK を選択します。



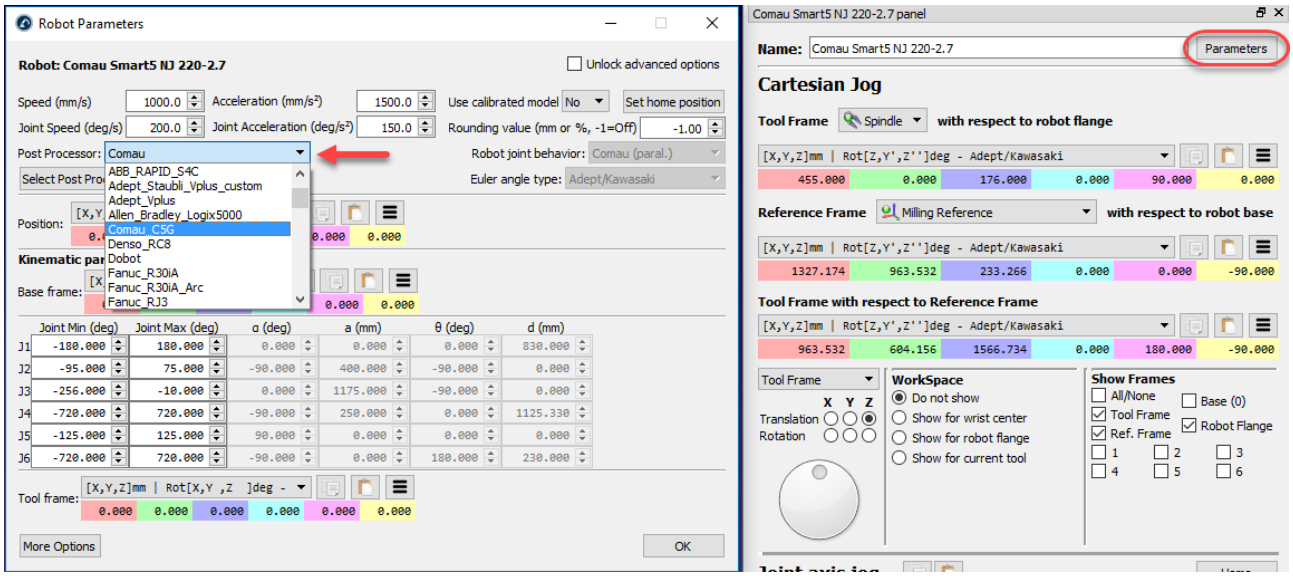
これで変更が適用され、プログラムを再度生成して結果を確認できます。

注意： ポストプロセッサーの選択は特定のロボットにリンクされています。プログラムのポストプロセッサを変更すると、同じロボットにリンクされているすべてのプログラムのポストプロセッサが更新されます。

別の方法として、次の手順に従ってポストプロセッサを選択することもできます。

1. ロボットパネルを開く（ロボットをダブルクリック）
2. パラメータを選択
3. 次の図に示すように、Robot ブランドボックスでポストプロセッサーを選択します。

ヒント：C：/ RoboDK / Posts /フォルダーにある PY ファイルをダブルクリックすることで、各ポストプロセッサーの出力をすばやくプレビューできます。詳細については、次のセクションで説明します。


Robot Parameters
Robot: Comau Smart5 NJ 220-2.7
Unlock advanced options
Speed (mm/s)
1000.0
Acceleration (mm/s²)
1500.0
Use calibrated model
No
Set home position
Joint Speed (deg/s)
200.0
Joint Acceleration (deg/s²)
150.0
Rounding value (mm or %, -1=Off)
-1.00
Post Processor:
Comau
Robot joint behavior:
Comau (paral.)
Euler angle type:
Adept/Kawasaki
ABB_RAPID_S4C
Adept_Staubli_Vplus_custom
Adept_Vplus
Allen_Bradley_Logix5000
Comau_C5G
Denso_RC8
Dobot
Fanuc_R30iA
Fanuc_R30iA_Arc
Fanuc_RJ3
Position:
Base frame:
Joint Min (deg)
Joint Max (deg)
α (deg)
a (mm)
θ (deg)
d (mm)
J1 -180.000 180.000 0.000 0.000 0.000 830.000
J2 -95.000 75.000 -90.000 400.000 -90.000 0.000
J3 -256.000 -10.000 0.000 1175.000 -90.000 0.000
J4 -720.000 720.000 -90.000 250.000 0.000 1125.330
J5 -125.000 125.000 90.000 0.000 0.000 0.000
J6 -720.000 720.000 -90.000 0.000 180.000 230.000
Tool frame:
[X,Y,Z]mm | Rot[X,Y ,Z]deg -
0.000 0.000 0.000 0.000 0.000 0.000
More Options
OK
Comau Smart5 NJ 220-2.7 panel
Name:
Comau Smart5 NJ 220-2.7
Parameters
Cartesian Jog
Tool Frame
Spindle
with respect to robot flange
[X,Y,Z]mm | Rot[Z,Y',Z'']deg - Adept/Kawasaki
455.000 0.000 176.000 0.000 90.000 0.000
Reference Frame
Milling Reference
with respect to robot base
[X,Y,Z]mm | Rot[Z,Y',Z'']deg - Adept/Kawasaki
1327.174 963.532 233.266 0.000 0.000 -90.000
Tool Frame with respect to Reference Frame
[X,Y,Z]mm | Rot[Z,Y',Z'']deg - Adept/Kawasaki
963.532 604.156 1566.734 0.000 180.000 -90.000
Tool Frame
X Y Z
Translation
Rotation
WorkSpace
Do not show
Show for wrist center
Show for robot flange
Show for current tool
Show Frames
All/None
Base (0)
Tool Frame
Robot Flange
Ref. Frame
1 2 3
4 5 6

ポストプロセッサーを編集する

既存のポストプロセッサーを変更したり、新しいポストプロセッサーを作成したりすることができます。ポストプロセッサーは、C：/ RoboDK / Posts /フォルダーに配置して、RoboDK で選択できるようにする必要があります。前のセクションでは、特定のロボットをポストプロセッサーにリンクする方法について説明しました。

各ポストプロセッサーは 1 つの PY ファイルです。ファイルの名前を変更するか、C：/ RoboDK / Posts /フォルダーとの間でファイルをコピーして、さまざまなポストプロセッサーを共有することができます。既存のポストプロセッサスクリプトを削除するには、Posts フォルダ内の対応する PY ファイルを削除するだけです。

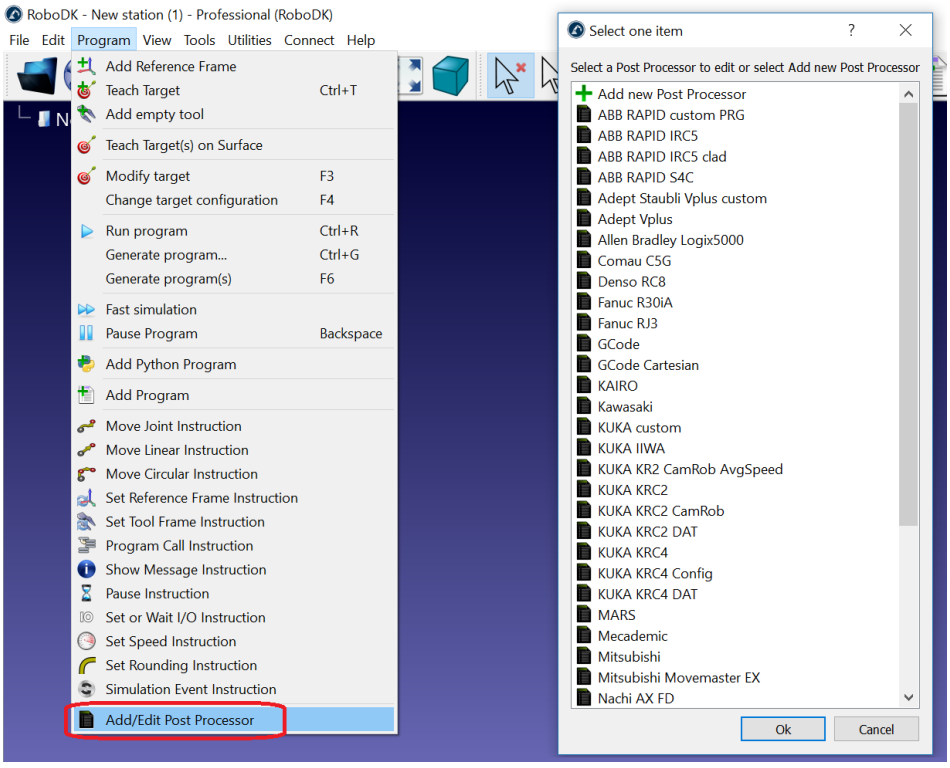
ポストプロセッサーは、任意のテキストエディターまたは Python エディター（Python IDLE）を使用して編集できます。Python エディターを使用すると、ファイルの最後にあるサンプルプログラムをすばやくデバッグして評価できます。

ビデオ：ポストプロセッサーの概要：https://www.youtube.com/watch?v=lbycCDjtOnE。

ポストプロセッサを適切にテストして使用するには、Python をインストールする必要があります（RoboDK では Python がデフォルトでインストールされます）。

次の手順に従って、既存のポストプロセッサーを変更します。

1. プログラムを選択→**ポストプロセッサーの追加/編集**
2. 既存のポストプロセッサーを選択します
3. OK を選択します。テキストエディタが開き、プログラムが起動します。
4. Go を選択します→**デバッグ（F5）**結果をプレビューします。必要に応じて変更します。使用するテキストエディターによっては、[実行]を選択する必要があります→**モジュールを実行（F5）** 代わりに。

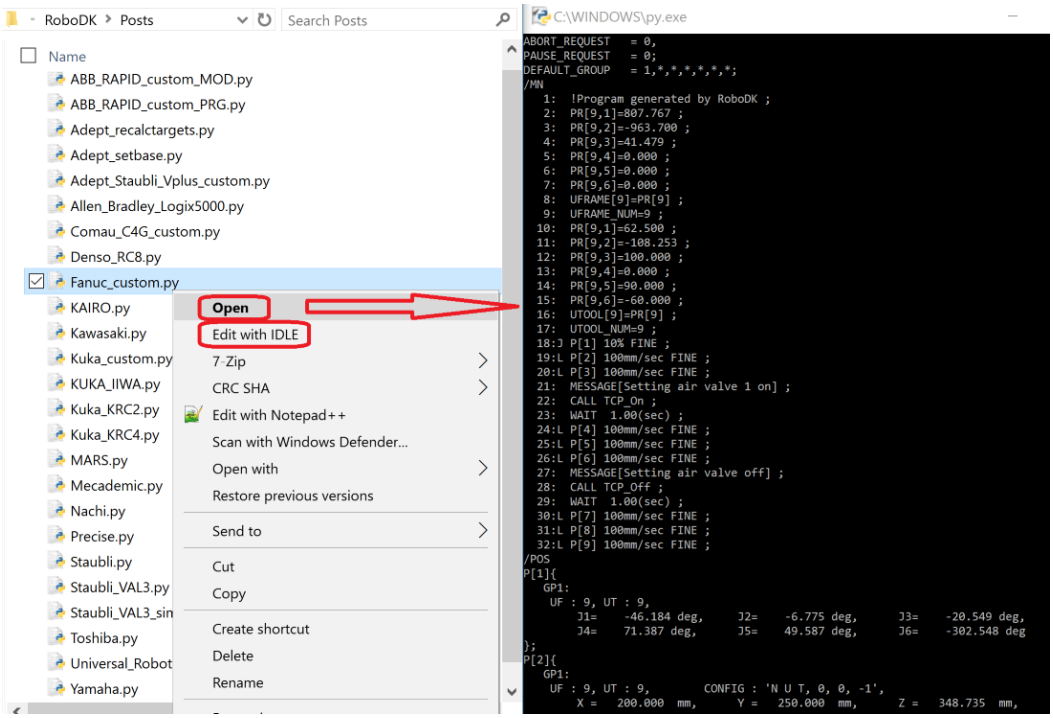


または、ポストプロセッサーを手動で編集することもできます。

1. ポストプロセッサーフォルダーに移動します：C：/ RoboDK / Posts /
2. Python IDLE で PY ファイルを開きます（右クリック → IDLE）または他のテキストエディターで編集します。

3. 必要な変更を行います。
4. ファイルを実行して結果をテストします。[実行]を選択します**→モジュールを実行**（デフォルトでは F5）Python IDLE から。

または、テキストエディターを使用してこのファイルを編集し、ダブルクリックして Python で実行することもできます。



ヒント： ツールを選択して、RoboDK のポストプロセッサーのデフォルトエディターを変更することができます。**→オプション→その他の** エディタパスを Python エディターコマンド。

重要：Notepad ++などのテキストエディタを使用する場合は、タブを 4 つのスペースに置き換えることが重要です。そうしないと、インデントでタブとスペースの使用に一貫性がないことを示す TabError が表示されます。

変更例

このセクションでは、既存のポストプロセッサーに小さな変更を加える方法を示します。ほとんどのポストプロセッサーには、ベンダー固有のロボットプログラミング機能を変更またはアクティブ化するために簡単に変更できるいくつかの変数があります。

例として、次の変更が既存の ABB および KUKA ポストプロセッサーに加えられます：

- 関節の動きを設定して、関節の角度情報を使用してモーションコマンドを生成します。
- 最高速度を 500 mm / s にオーバーライドします。ロボットがより速く動くようにプログラムされている場合でも、速度はポストプロセッサーで 500 mm / s に制限されます。
- 各プログラムを個別のファイルとして生成し、プログラムにプログラムあたり最大 3000 行のコードを強制します。大きなプログラムは小さなプログラムに分割され、順次呼び出されます。

前のセクションでは、既存のポストプロセッサーを開いて編集する方法を示しました。

1. プログラムを選択→**ポストプロセッサーの追加/編集**
2. ポストプロセッサーを選択します。たとえば、KUKA KRC4 の場合、KUKA_KRC4 を選択します。
3. OK を選択します。ポストプロセッサーは、Python IDLE エディターに表示されます。

次のセクションでは、テキストエディター（または Python IDLE）で提案された変更を行う方法を示します。

関節角度を使用して動きを課す

このセクションでは、デカルト値の代わりにジョイント値を使用して、軸移動を強制するように既存のポストプロセッサを変更する方法を示します。この例では、ABB IRC5 ロボットコントローラーにこの変更を適用する方法を示します。

1. ロボットコントローラのプログラミングマニュアルを見つけます。この例では、ABB IRC5 RAPID プログラミングマニュアルを使用します。
2. 関節運動の説明を探します。この場合、ABB の絶対関節移動コマンドは MoveAbsJ と呼ばれます。このコマンドには、関節軸を定義する jointtarget 変数が必要です。

1.90. MoveAbsJ - Moves the robot to an absolute joint position

Usage

`MoveAbsJ` (*Move Absolute Joint*) is used to move the robot and external axes to an absolute position defined in axes positions.

Basic examples

Basic examples of the instruction `MoveAbsJ` are illustrated below.

See also *More examples on page 233*.

Example 1

```
MoveAbsJ p50, v1000, z50, tool2;
```

The robot with the tool `tool2` is moved along a non-linear path to the absolute axis position, `p50`, with velocity data `v1000` and zone data `z50`.

Example 2

```
MoveAbsJ *, v1000\T:=5, fine, grip3;
```

The robot with the tool `grip3` is moved along a non-linear path to a stop point which is stored as an absolute axis position in the instruction (marked with an *). The entire movement takes `5` seconds.

Arguments

```
MoveAbsJ [\Conc] ToJointPos [\ID] [\NoEOffs] Speed [\V] | [\T]
    Zone [\Z] [\Inpos] Tool [\WObj]
```

3. プログラムを選択→**ポストプロセッサーの追加/編集** それを変更するために現在使用しているポストプロセッサーを選択します。現在カスタマイズ可能なポストプロセッサーを使用している場合は、デフォルトで選択されます。
4. ポストプロセッサ内で MoveJ 関数を探す必要があります。この関数は、ポストプロセッサーが関節運動コマンドを生成する方法を定義します。あるいは、RoboDK は直線移動に MoveL を使用します。次の図に示すように、Python プログラミング言語を使用して、プログラムファイルに追加された行を変更し、関節角度情報を提供します。

```
def MoveJ(self, pose, joints, conf_RLF=None):
    """Add a joint movement"""
    self.addline('MoveAbsJ [%s,%s],%s,%s,rdkTool,\WObj:=rdkWObj;'
                 % (angles_2_str(joints), extaxes_2_str(joints), self.SPEEDDATA, self.ZONEDATA))
```

5. 最後に、ポストプロセッサが関節角度またはポーズを文字列に変換する方法を変更する必要がある場合があります。関数 angles_2_str と pose_2_str は、それぞれ関節角度とポーズがテキストに変換される方法を定義します。

```
def pose_2_str(pose):
    # Convert the pose to XYZ position and q1,q2,q3,q4 quaternion values
    [x,y,z,q1,q2,q3,q4] = Pose_2_ABB(pose)
    return ('[%.3f, %.3f, %.3f],[%.8f, %.8f, %.8f, %.8f]' % (x,y,z,q1,q2,q3,q4))

def angles_2_str(angles):
    # Generate a string like:
    # [10,20,30,40,50,60]
    # with up to 6 decimals
    return '[%s]' % (','.join(format(ji, ".6f") for ji in angles[0:6]))
```

ほとんどの RoboDK ポストプロセッサは、関節の動きには関節データを使用し、直線的な動きにはデカルト座標を使用します。ジョイント座標を指定して、常に一連の直線移動の最初のポイントをジョイント移動として開始することをお勧めします。これにより、誤ったロボット構成でプログラムを開始したり、特異点や軸の制限に達したりすることが回避されます。

注意：すべての RoboDK ポストプロセッサは、デフォルトで robodk.py モジュールにリンクしています。このモジュールには、多くのロボットコントローラのポーズからオイラー角に変換するための便利なツールが含まれています。ロボットコントローラに対応する表記を使用します。詳細については、参照フレーム セクションと robodk.py モジュール。

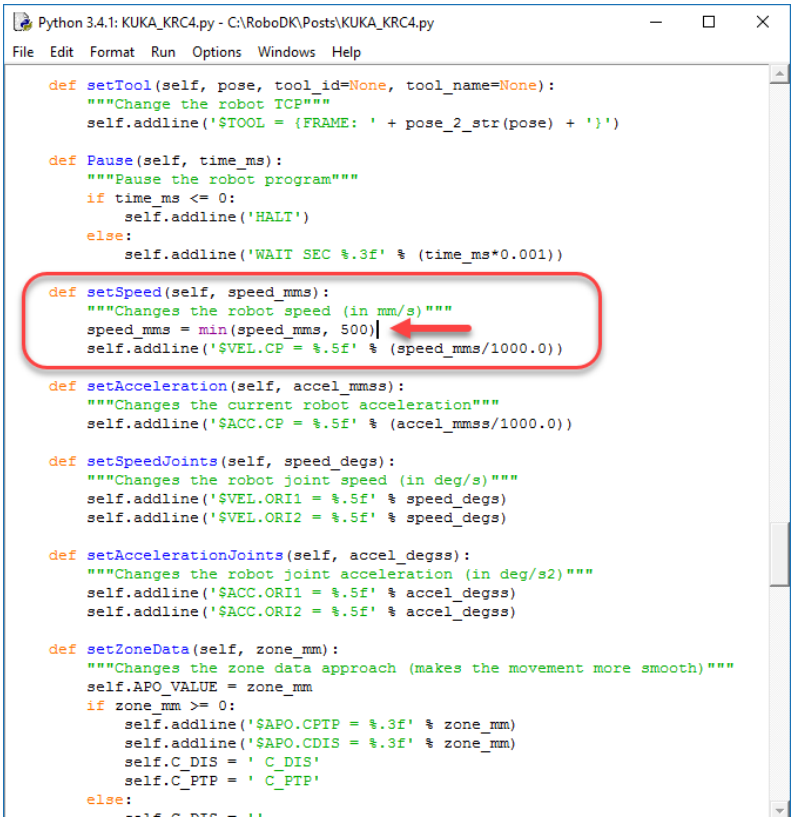
注意：複数のロボットメーカーが同じオイラー角表記を使用できます。たとえば、ファナックとモトマンはどちらも同じ X を使用しています→Y→Z 表記、ストーブリ、および Mecademic は同じ X を使用します→Y '→Z "表記など

重要：ポーズを使用して（たとえば、デカルト座標とオイラーまたはクォータニオン情報を使用して）動きを指定する場合、ロボットが達成する位置は、アクティブなツールと参照フレームによって異なります。ツールと参照フレームのポーズは、プログラムで適切に指定する必要があります。これは、それぞれ setTool 関数と setFrame 関数を介して行われます。

速度制限を強制する

RoboDK からプログラムを生成するとき、500 mm / s の速度制限を設定し、より高い速度を使用しないようにするには、次の手順に従ってください。この例では、KUKA KRC2 または KRC4 コントローラーを使用していると想定しています。

1. setSpeed 関数定義を見つけます（def setSpeed）
2. m / s 単位で速度を変更する$ VEL.CP 出力を生成する前に、次のコードを追加します。
   `speed_mms = min（speed_mms、500）`



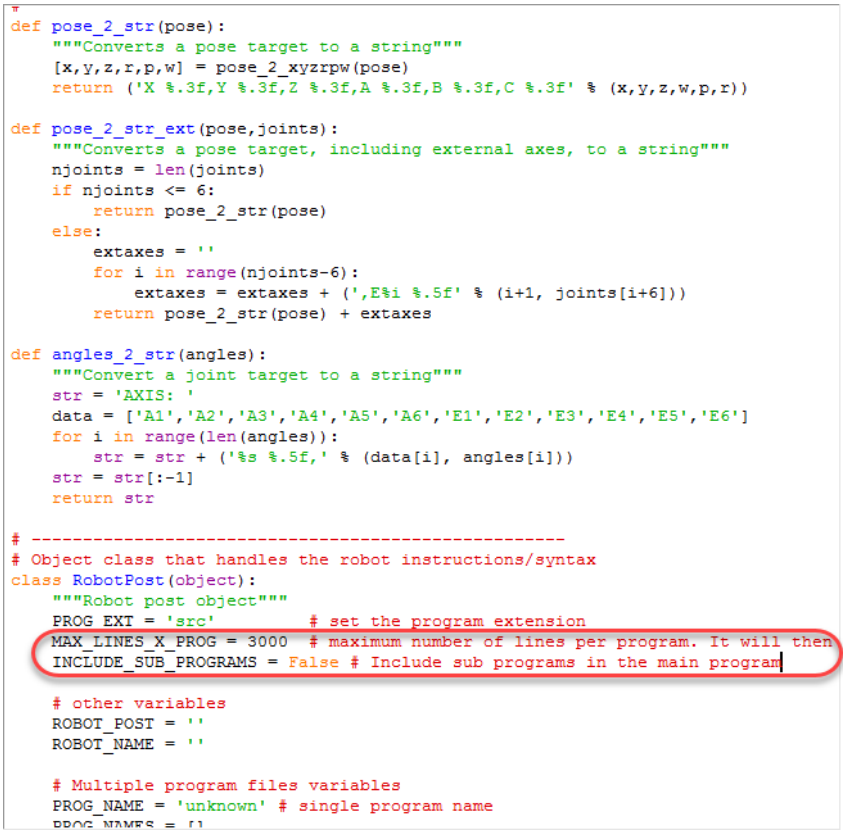
注意：200 mm / s のデフォルト速度は、ポストプロセッサーの上部にある HEADER 変数で定義されています（$ VEL.CP = 0.2）。このデフォルトの速度は、速度が RoboDK から変更されていない場合にも変更できます。

ファイルごとに 1 つのプログラムを生成する

次の変更を追加して、ファイルごとに複数のプログラムを追加することを避け、最大 3000 行のコードでプログラムを生成します。

1. 変数 MAX_LINES_X_PROG を 3000 に設定します
2. 変数 INCLUDE_SUB_PROGRAMS を False に設定します

注意： プログラムごとの最大行数が RoboDK から指定されている場合、変数 MAX_LINES_X_PROG はオーバーライドされる可能性があります プログラムオプションメニュー。ポストプロセッサーの__init__セクションを変更して、この変数を RoboDK 設定から変更しないようにします。



```
def pose_2_str(pose):
    """Converts a pose target to a string"""
    [x,y,z,r,p,w] = pose_2_xyzrpw(pose)
    return ('X %.3f,Y %.3f,Z %.3f,A %.3f,B %.3f,C %.3f' % (x,y,z,w,p,r))

def pose_2_str_ext(pose,joints):
    """Converts a pose target, including external axes, to a string"""
    njoints = len(joints)
    if njoints <= 6:
        return pose_2_str(pose)
    else:
        extaxes = ''
        for i in range(njoints-6):
            extaxes = extaxes + (',E%i %.5f' % (i+1, joints[i+6]))
        return pose_2_str(pose) + extaxes

def angles_2_str(angles):
    """Convert a joint target to a string"""
    str = 'AXIS: '
    data = ['A1','A2','A3','A4','A5','A6','E1','E2','E3','E4','E5','E6']
    for i in range(len(angles)):
        str = str + ('%s %.5f,' % (data[i], angles[i]))
    str = str[:-1]
    return str

# ----------------------------------------------------
# Object class that handles the robot instructions/syntax
class RobotPost(object):
    """Robot post object"""
    PROG_EXT = 'src'         # set the program extension
    MAX_LINES_X_PROG = 3000  # maximum number of lines per program. It will then
    INCLUDE_SUB_PROGRAMS = False # Include sub programs in the main program

    # other variables
    ROBOT_POST = ''
    ROBOT_NAME = ''

    # Multiple program files variables
    PROG_NAME = 'unknown' # single program name
    PROG_NAMES = []
```

利用可能なポストプロセッサ

RoboDK では、デフォルトで次のポストプロセッサーを使用できます。

- ABB RAPID IRC5：ABB IRC5 ロボットコントローラー用
- ABB RAPID S4C：ABB S4C ロボットコントローラー用
- Adept Vplus：Adept V +プログラミング言語用
- アレンブラッドリーLogix5000：アレンブラッドリーLogix5000 PLC 用
- Aubo：AUBO ロボットコントローラ用
- CLOOS：CLOOS ロボットコントローラ用
- Comau C5G：Comau C5G ロボットコントローラー用
- Denso PAC：Denso RC7（およびそれ以前）ロボットコントローラー（PAC プログラミング言語）
- Denso RC8：Denso RC8（以降）ロボットコントローラー（PacScript プログラミング言語）
- Dobot：教育用 Dobot ロボット用
- Doosan：Doosan 協調ロボット用
- Epson：Epson ロボットコントローラ用
- ファナック R30iA：ファナック R30iA および R30iB ロボットコントローラー用
- ファナック R30iA_Arc：ファナックアーク溶接用
- Fanuc RJ3：Fanuc RJ3 ロボットコントローラ用
- GCode BnR：B&R ロボットコントローラー用
- GSK：GSK ロボット用
- HCR：ハンファロボットコントローラー用
- HIWIN HRSS：HIWIN ロボット用
- KAIRO：Keba Kairo ロボットコントローラー用
- KUKA IIWA：Java での KUKA IIWA サンライズプログラミング用
- KUKA KRC2：KUKA KRC2 ロボットコントローラー用
- KUKA KRC2_CamRob：KUKA CamRob フライスオプション用
- KUKA KRC2_DAT：DAT データファイルを含む KUKA KRC2 ロボットコントローラー用
- KUKA KRC4：KUKA KRC4 ロボットコントローラー用
- KUKA KRC4_Config：KUKA KRC4 ロボットコントローラー用、各行に構成データ
- KUKA KRC4_DAT：DAT データファイルを含む KUKA KRC4 ロボットコントローラー用
- Kawasaki：Kawasaki AS ロボットコントローラ用
- Mecademic：Mecademic Meca500 ロボット用
- 三菱：三菱ロボットコントローラー用
- Motoman / Yaskawa：Inform II および Inform III（JBI）を使用するさまざまな Motoman ロボットコントローラー用
- Nachi AX FD：Nachi AX および FD ロボットコントローラ用
- Omron：Omron / Techman ロボットコントローラ用
- OTC：Daihen OTC ロボットコントローラー用
- Panasonic：Panasonic PRG プログラムの場合
- Precise：Precise Scara ロボット用
- Robostar：Robostar ロボットコントローラー用
- Siasun：Siasun ロボットコントローラー用
- Siemens_Sinumerik：Siemens Sinumerik ROBX ロボットコントローラー用
- Staubli VAL3：Staubli VAL3 ロボットプログラム用（CS8 コントローラ以降）
- Staubli VAL3_InlineMove：インライン移動データを含む Staubli VAL3 プログラムを生成します
- Staubli S6：Staubli S6 ロボットコントローラー用
- 東芝：東芝ロボット用
- Techman：Omron / Techman ロボットコントローラ用
- ユニバーサルロボット：UR ロボットの場合、ポーズターゲットとして直線運動を生成します

- Universal Robots URP：UR ロボットの場合、Polyscope（UR ロボットコントローラー）にロードおよび変更できる URP を生成します
- Universal Robots_RobotiQ：RobotiQ グリッパーのサポートを含む UR ロボット用
- Universal Robots_MoveP：UR ロボットの場合、MoveP コマンドとして直線運動を生成します
- ヤマハ：ヤマハロボット用

リファレンス

次のビデオは、RoboDK のポストプロセッサの概要を示しています。
https://robodk.com/help#PostProcessor

ポストプロセッサの各メソッドのリファレンスドキュメントはオンラインで入手できます。
http://robodk.com/doc/en/PythonAPI/postprocessor.html

すべてのポストプロセッサーは robodk.py モジュールを使用します：
http://robodk.com/doc/en/PythonAPI/robodk.html#robodk.Mat

robodk.py モジュールは、とりわけ、ポーズ操作（乗算、逆数など）や、ポーズ間のオイラー角への変換のためのツールを提供します。

注意： デフォルトでは、Python は RoboDK とともにインストールされ、robodk.py モジュールが Python パスに追加されます。

オンラインで Python プログラミングについて学習できるドキュメントを入手できます。
https://docs.python.org/3/

プログラムが生成されると、前処理済み/ユニバーサル Python プログラムが生成され、ローカルの一時フォルダーに保存されます。前処理されたプログラムは、適切なポストプロセッサ（RoboDK でユーザーが選択）にリンクされています。ポストプロセッサは、目的のコードを生成する RobotPost クラスを定義します。

プリコンパイルされたプログラムは Python で実行されます。

ヒント：Windows では、前処理された Python ファイルは一時フォルダーに保存されます（例：C：/ Users / username / AppData / Local / Temp フォルダー、または Windows ファイルエクスプローラーで%TEMP%と入力）。これらのプログラムは、新しいポストプロセッサーのデバッグにも使用できます。