SPT NC Axis

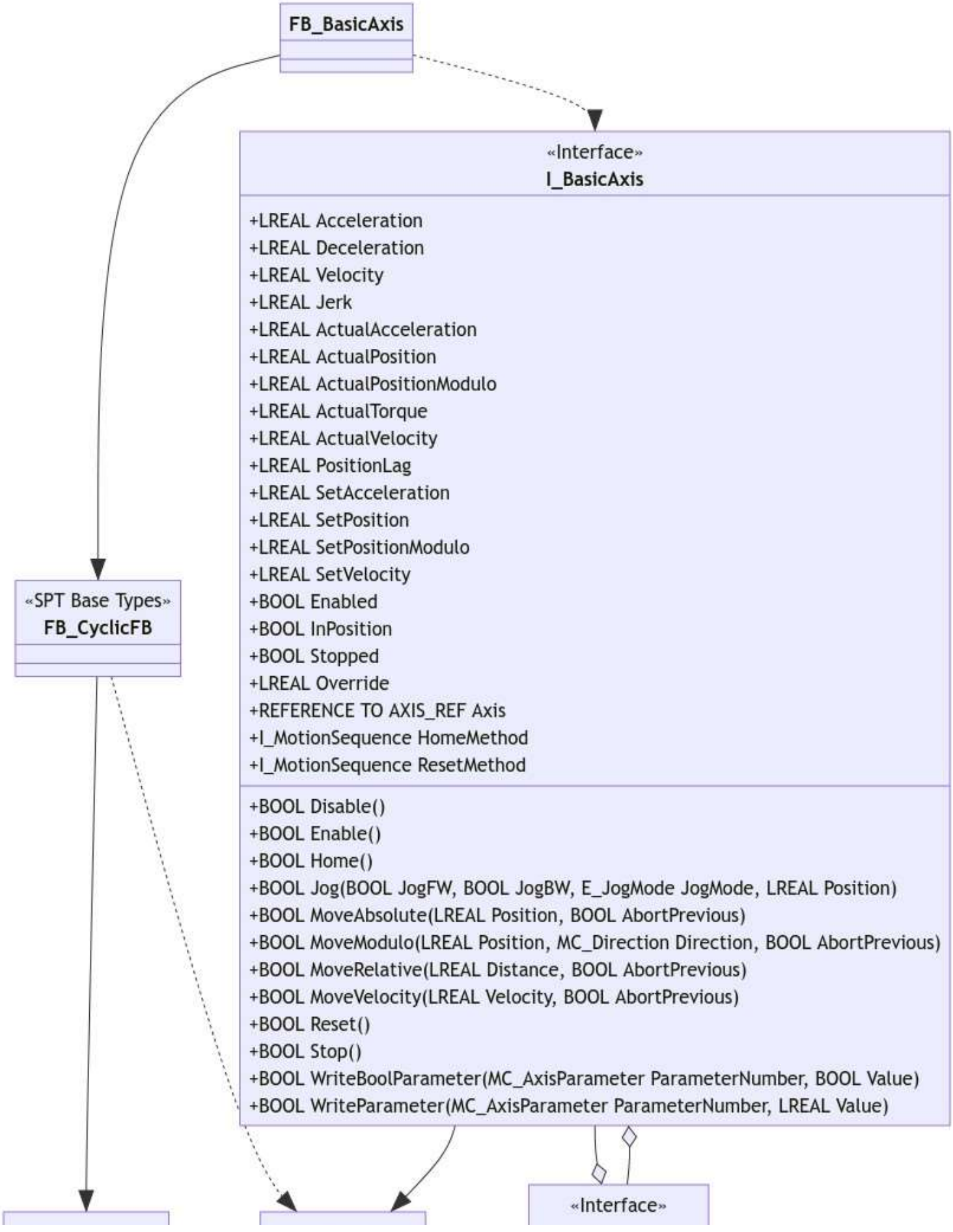
Last updated by | Nick Higgins | Oct 11, 2022 at 11:29 AM EDT

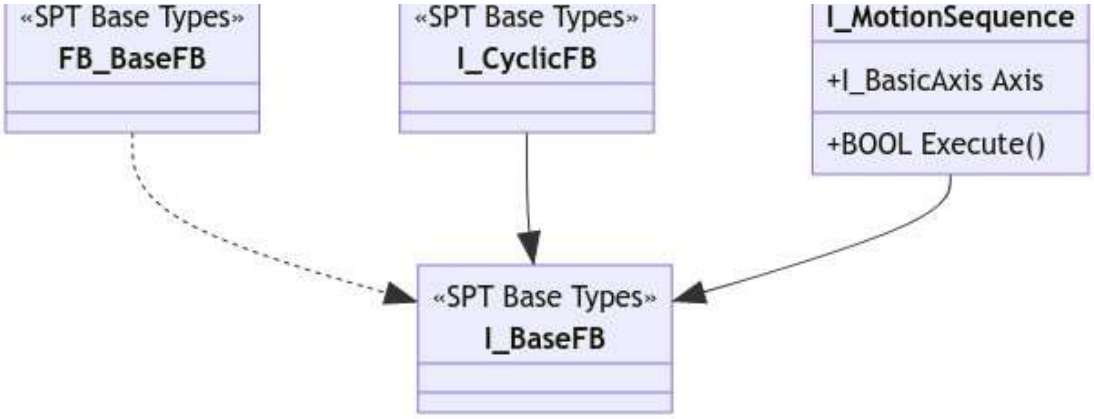
Contents

Overview

General-purpose NC axis wrapper function block. This function block can be used without any PackML-related functions and does not by itself implement any of the component model interfaces. Use FB_Component_BasicAxis for PackML-based projects.

Class Diagram





Interfaces

I_BasicAxis

(extends `I_CyclicFB`)

Defines basic required functionality for a basic NC axis (no coupling--see FB_BasicSlaveAxis/FB_CamSlaveAxis).

Properties

Property	Type	Access	Description
Axis	REFERENCE TO AXIS_REF	RO	Returns an AXIS_REF for use in external motion functions, etc.
Override	LREAL	RW	Get/Set axis override (100.0 = 100%)
HomeMethod	I_MotionSequence	WO	Assign a custom homing method
ResetMethod	I_MotionSequence	WO	Assign a custom reset method

Dynamics

Property	Type	Access	Description
Acceleration	LREAL	RW	Get/Set acceleration input to motion functions
Deceleration	LREAL	RW	Get/Set deceleration input to motion functions
Jerk	LREAL	RW	Get/Set jerk input to motion functions
Velocity	LREAL	RW	Get/Set velocity input to motion functions (exception: `MoveVelocity()`)

Feedback

Property	Type	Access	Description
ActualAcceleration	LREAL	RO	Get actual acceleration of axis
ActualPosition	LREAL	RO	Get actual position of axis
ActualPositionModulo	LREAL	RO	Get actual modulo position of axis
ActualTorque	LREAL	RO	Get actual torque of axis
ActualVelocity	LREAL	RO	Get actual velocity of axis
PositionLag	LREAL	RO	Get position lag of axis
SetAcceleration	LREAL	RO	Get setpoint acceleration of axis
SetPosition	LREAL	RO	Get setpoint position of axis
SetPositionModulo	LREAL	RO	Get setpoint modulo position of axis
SetVelocity	LREAL	RO	Get setpoint velocity of axis

Status

Property	Type	Access	Description
Enabled	BOOL	RO	Axis is enabled
InPosition	BOOL	RO	Axis is within target position window
Stopped	LREAL	BOOL	Axis is not moving

Methods

Method	Return Type	Access	Description
Disable	BOOL	PUBLIC	Disable axis
Enable	BOOL	PUBLIC	Enable axis
Home	BOOL	PUBLIC	Home axis
Jog	BOOL	PUBLIC	Jog axis
MoveAbsolute	BOOL	PUBLIC	Initiate an absolute move
MoveModulo	BOOL	PUBLIC	Initiate a modulo move
MoveRelative	BOOL	PUBLIC	Initiate a relative move
MoveVelocity	BOOL	PUBLIC	Initiate a continuous velocity move
Reset	BOOL	PUBLIC	Reset axis
WriteBoolParameter	BOOL	PUBLIC	Write a parameter of type BOOL
WriteParameter	BOOL	PUBLIC	Write a parameter of type LREAL

I_MotionSequence

(extends `I_BaseFB`)

Defines basic required functionality for external sequence of any sort.

See SPT NC Homing / SPT Drives libraries.

Properties

Property	Type	Access	Description
Axis	I_BasicAxis	RW	Get/Set axis to be controlled by the sequence

Methods

Method	Return Type	Access	Description
Execute	BOOL	PUBLIC	Start the sequence

Function Blocks

FB_BasicAxis

(extends `FB_CyclicFB`, implements `I_BasicAxis`)

Complete implementation of I_BasicAxis. For use as a PackML component, use `FB_Component_BasicAxis`.

Notes

- The `Initialize()` method checks to see if the underlying AXIS_REF has been linked to an NC axis. If not, `Initialize()` will not return `TRUE`.
  - This behavior can be inhibited by setting the library parameter `ALLOW_UNLINKED_NC_AXES` to `TRUE`.
- The `Initialize()` method automatically reads all of the axis' NC parameters, which can be accessed via the `NCParameters` property. (TODO: 1560 **NC Basic Axis - Axis Parameters is read into local but not exposed via property** To Do)
- `CyclicLogic()` should be called so that underlying function block calls are made
- `BOOL` return values from command methods should be interpreted as "COMMAND ACCEPTED". This is **not** "COMMAND COMPLETED". Monitor the `Busy` property to detect when a command has finished (see example below).
- Most commands are internally interlocked with `_Busy` such that a command will not be accepted (return `TRUE`) if a command is already in process. This can be bypassed (move interrupted) by passing `TRUE` to the `AbortPrevious` argument of the move method call. If the `AbortPrevious` argument is not included in the method signature, this should be interpreted to mean the command will take place immediately.

Properties

Axis

```
PROPERTY Axis : REFERENCE TO AXIS_REF
```

Returns a reference to the underlying NC axis for use in other axes or otherwise external motion functions.

Example

```
PROGRAM MAIN
VAR
        MyBasicAxis : FB_BasicAxis;
        MyBasicSlaveAxis : FB_BasicSlaveAxis;
        MySlaveAxis : AXIS_REF;
        MC_GearIn : MC_GearIn;
END_VAR

//Mixing basic MC2 and FB_BasicAxis
MC_GearIn(
        Master:= MyBasicAxis.Axis,
        Slave:= MySlaveAxis,
        Execute:= TRUE,
        RatioNumerator:= 1,
        RatioDenominator:= 1);

//Assigning a FB_BasicAxis to an FB_BasicSlaveAxis
MyBasicSlaveAxis.Master1 REF= MyBasicAxis.Axis;
```

HomeMethod

`PROPERTY HomeMethod : I_MotionSequence`

(see *SPT NC HOMING* library)

You can optionally assign an external sequence of events to occur when the `Home()` method is called. Assigning a function block which implements `I_MotionSequence` will automatically assign the axis' `AXIS_REF` for use in whatever function blocks you choose. The instantiation of custom home routine function block should take place in the parent of the `FB_BasicAxis` instance. **The default behavior is to set the current position as zero when this property is not set and the Home() method is called.**

ResetMethod

`PROPERTY ResetMethod : I_MotionSequence`

(see *SPT Drives* library)

You can optionally assign an external sequence of events to occur when the `Reset()` method is called. This can be useful, for instance, when using an AX5000 or some drive that needs a separate reset routine to be called in addition to MC_Reset. Assigning a function block which implements `I_MotionSequence` will automatically assign the axis' `AXIS_REF` for use in whatever function blocks you choose. The instantiation of custom reset routine function block should take place in the parent of the `FB_BasicAxis` instance. **The default behavior is to call MC_Reset when this property is not set and the Reset() method is called.**

Examples

Enabling Axis

```
MyAxis.CyclicLogic();

CASE State OF
  0:
    IF MyAxis.Enable() THEN
      State := State + 10;
    END_IF
  10:
    IF MyAxis.Enabled THEN
      State := State + 10;
    END_IF
  20:
    //Do things
END_CASE
```

Command a discrete move

```
CASE State OF
  0:
    MyAxis.Velocity := 10;
    IF MyAxis.MoveAbsolute(100, FALSE) THEN
      State := State + 10;
    END_IF

  10:
    IF NOT MyAxis.Busy THEN
      State := State + 10;
    END_IF

  20:
    //More things
END_CASE
```

Command a velocity move and then change velocity

```
CASE State OF
  0:
    IF MyAxis.MoveVelocity(100, FALSE) THEN
      State := State + 10;
    END_IF

  10:
    IF ChangeVelocity THEN
      MyAxis.MoveVelocity(10, TRUE);
      State := State + 10;
    END_IF

  20:
    //More things
END_CASE
```

Proper sequencing of multiple axes

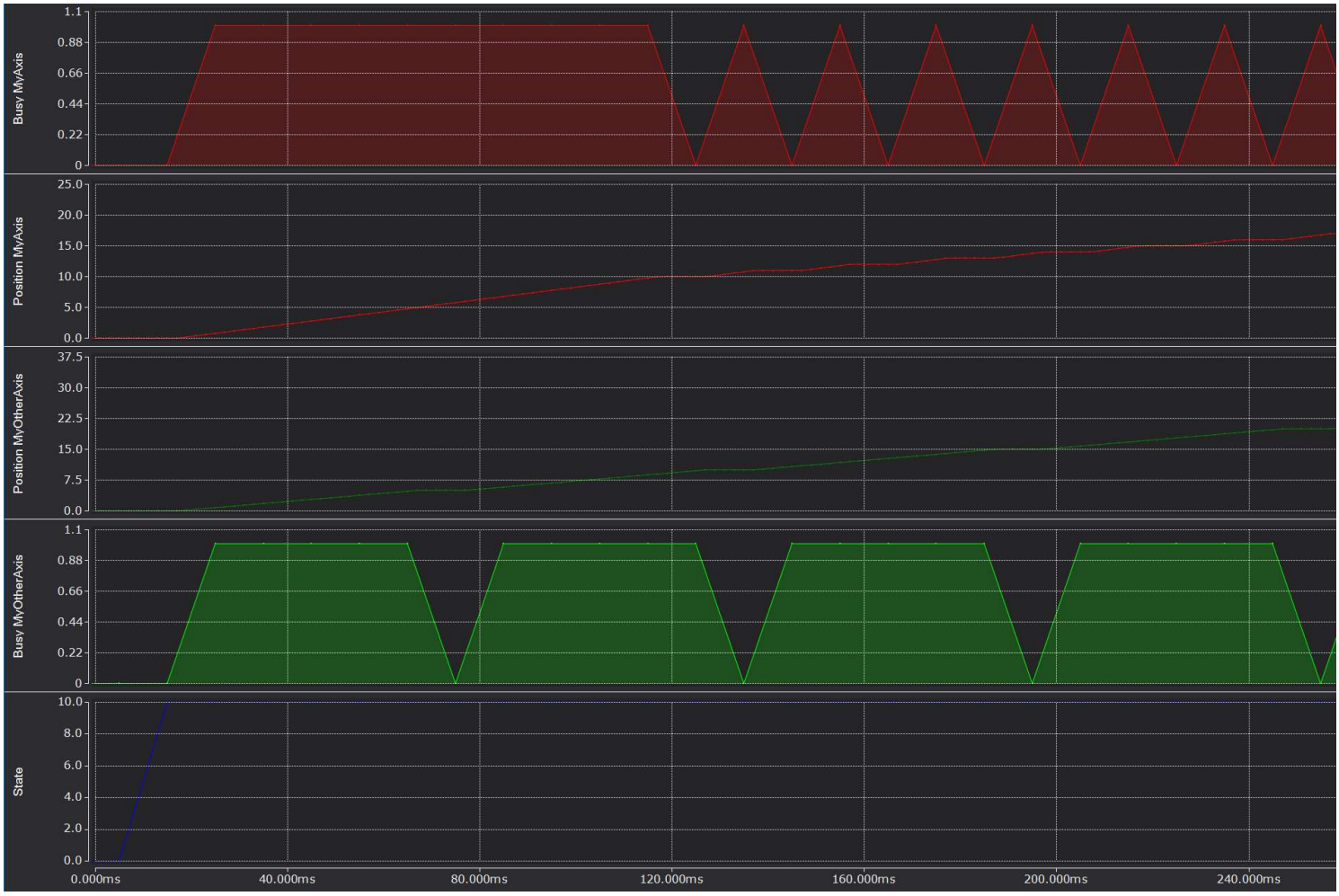
Resist the temptation to combine commands to multiple axes in the same state/line. Depending on the command, you can find yourself in a race condition.

Less typing, more headaches:

```
CASE State OF
  0:
    //Move is issued to both axes--okay
    IF MyAxis.MoveRelative(10, FALSE) AND MyOtherAxis.MoveRelative(5, FALSE) THEN
      State := State + 10;
    END_IF
  10:
    //MyOtherAxis finishes its move first and second command is issued
    //MyAxis is still busy at this point, so command is ignored until first move is complete
    //Commands end up flip-flopping and axes goes nowhere near where you intended
    IF MyAxis.MoveRelative(1, FALSE) AND MyOtherAxis.MoveRelative(5, FALSE) THEN
      State := State + 10;
    END_IF

  20:
    //You may or may not ever get here...
    IF NOT MyAxis.Busy AND NOT MyOtherAxis.Busy THEN
     State := State + 10;
    END_IF
END_CASE
```

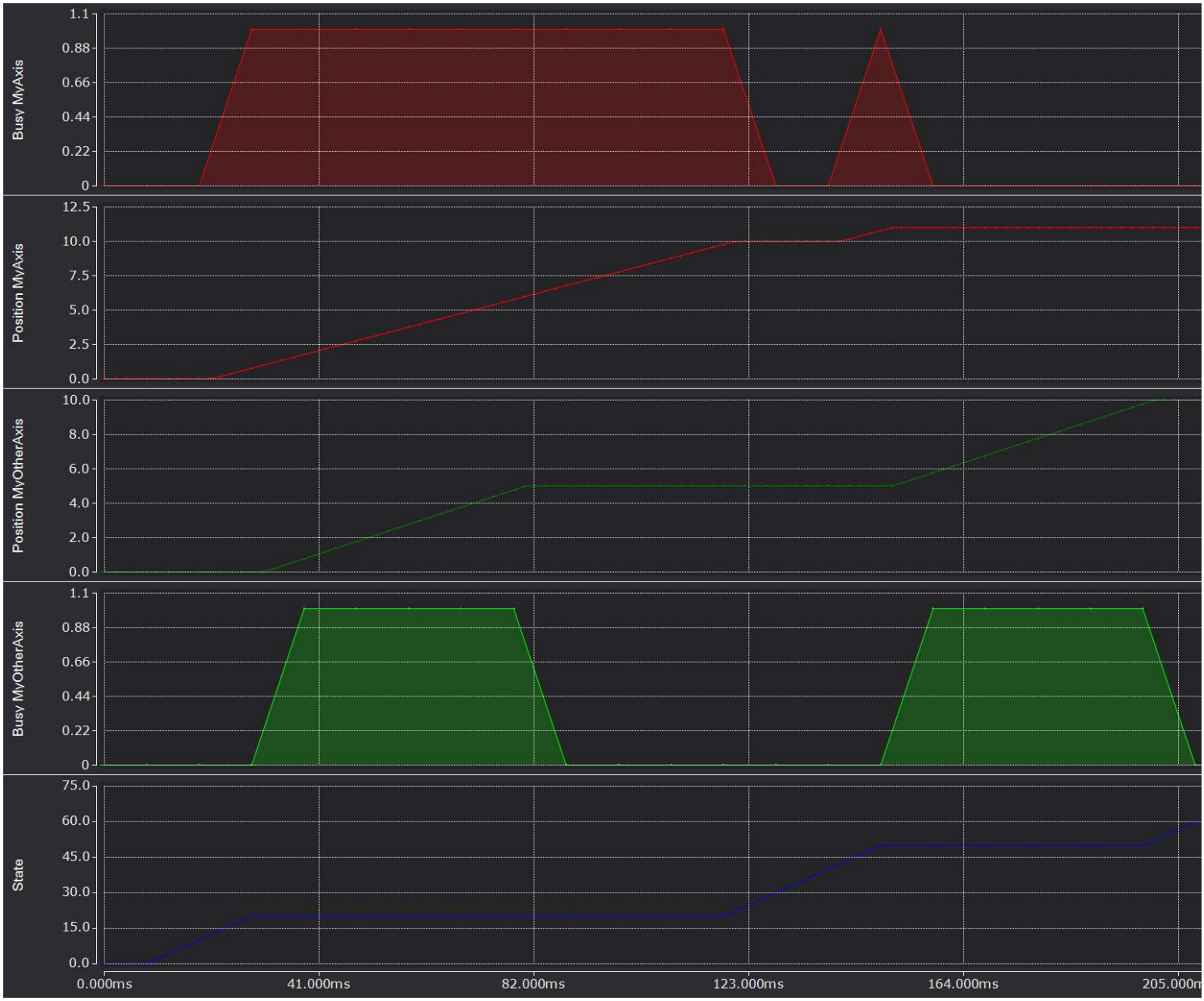
Result



More states, but axes go where you want them to:

```
CASE State OF
  0:
    IF MyAxis.MoveRelative(10, FALSE) THEN
      State := State + 10;
    END_IF
  10:
    IF MyOtherAxis.MoveRelative(5, FALSE) THEN
      State := State + 10;
    END_IF
  20:
    //Wait for both axes to be done
    IF NOT MyAxis.Busy AND NOT MyOtherAxis.Busy THEN
      State := State + 10;
    END_IF
  30:
    IF MyAxis.MoveRelative(1, FALSE) THEN
      State := State + 10;
    END_IF
  40:
    IF MyOtherAxis.MoveRelative(5, FALSE) THEN
      State := State + 10;
    END_IF
  50:
    IF NOT MyAxis.Busy AND NOT MyOtherAxis.Busy THEN
      State := State + 10;
    END_IF
END_CASE
```

Result


Busy MyAxis
1.1
0.88
0.66
0.44
0.22
0
Position MyAxis
12.5
10.0
7.5
5.0
2.5
0.0
Position MyOtherAxis
10.0
8.0
6.0
4.0
2.0
0.0
Busy MyOtherAxis
1.1
0.88
0.66
0.44
0.22
0
State
75.0
60.0
45.0
30.0
15.0
0.0
0.000ms
41.000ms
82.000ms
123.000ms
164.000ms
205.000m